# CS231A Course Project Proposal
# Fully automated trimap generation for image matting with Kinect

Karen Cheng
kycheng@gmail.com

Buu-Minh Ta
bmta@stanford.edu

## Abstract

*This paper proposes a new way to perform digital matting, a process that extracts the foreground from the background. Typically, the estimation of foreground, background, and unknown pixels (trimap) is all done with RGB data. In the first part of the paper, we introduce a novel way of generating a trimap, by using a 3D camera. The depth map is mapped to the RGB image using intrinsic and extrinsic parameters obtained from stereo calibration. Then, we apply K-means segmentation on the depth map, and apply additional assumptions to create the trimaps. The second part of our paper presents our results and explains how they are evaluated. Our results were compared with results obtained with manually-generated trimaps. Our algorithm performed well when the size of the unknown zone is bigger than the precision of the Kinect, and when the details of the object is fully captured by the Kinect's resolution.*

## Future Distribution Permission

## 1. Introduction

Digital matting is a process that extracts the foreground from the background. A commonly-used method expresses each pixel value as a function of foreground color $F$, background color $B$, and the opacity of the foreground object $\alpha$, summarized by the *compositing equation*:

$$C = \alpha F + (1 - \alpha)B$$

Several approaches, such as the Bayesian or the learning based approach require a trimap to provide an initial guess for foreground, background, and unknown regions. The limitation of this approach is that the trimap is often generated manually, and would be highly inefficient if one were to implement matting on a video stream, and unfeasible if you wanted to do it in real-time.

This problem could be circumvented with depth map from a 3D camera – the Kinect, for example. In this paper, we propose different methods that we will evaluate computation of trimaps with the Kinect. They all rely on the depth map to do a segmentation. The methods we consider applying are based on k-means and Ncuts. In this paper, we will demonstrate that depth data could provide us with a more robust method to automate the generation of trimap.

## 2. Related work

Curless & al. [1] and Zheng & al. [2] have come up with two different methods to do digital matting, both using a trimap. The first one maximizes the likelihood given the image, and assuming Gaussian distributions of the foreground and the background. The second treats the problem as a semi-supervised learning task in machine learning. They both have satisfying results, but the user need to provide a trimap so the algorithm knows where to start. Finger & al. [3] introduces the use of a depth map to generate a trimap, and then use the Bayesian technique to solve the problem.

## 3. Approach

### 3.1. Calibration and image alignment

Data were collected using our own Kinect. Before the mapping of depth values to RGB pixels takes place, the cameras were calibrated to obtain the intrinsic and extrinsic parameters. The intrinsic parameters define the optical and geometrical properties of both cameras. The extrinsic parameters are what define the relationship between the camera coordinates. Calibration is done based on Zhengyou Zhang's flexible calibration method implemented in OpenCV, utilizing images of a checkerboard pattern at 8 different orientations. To increase OpenCV's cvFindChessboardCorners' accuracy in locating the corners, the checkerboard pattern was uniformly illuminated by a floodlight, and the IR projector was covered to block the speckles it projects to the scene. The results obtained were comparable to findings from an online source [8].

#### 3.1.1 Correspondence between RGB and depth

We determine the correspondence between depth and RGB pixels by the first transforming 2D image points from the depth camera, $[u, v]^T$, to 3D depth coordinate points $[X_c, Y_c, Z_c]^T$ :

The 3D depth points then undergo rotation and translation to transform into RGB space, and are then projected onto the RGB image plane. Finally, the depth value at $[u, v]_{RGB}$ are matched with the RGB values at $[u, v]_{RGB}$. Depth values are obtained by converting the raw 11-bit disparity values to distance values in centimeters.

#### 3.1.2 Image alignment

In a case where the foreground can be easily separated from the background, as on figure 1, we can see that the correspondence depth is not correct. The shape of the extracted foreground object (using k-means, as explained in next section) is correct, but is not at the correct location. This is an issue we have not managed to solve, and we could not figure its reason. The intrinsics paramaters seem to be good, since the size of the shape is correct (it is not before readujstment).

An ad-hoc solution was to try to manually align the pictures from the Kinect. This allows us to have a good enough correspondence, but makes our procedure not fully automated.

#### 3.1.3 Shadows from the Kinect projector

Another issue we ran into is unknown zones where the Kinect cannot give information about the depth. This is due to the way the Kinect works, using a IR projector and a sensor. It might happen that an object is occluding another object from the projector, but looking from the sensor, the second object might not be occluded. The problem is that the sensor does not receive any information from the projector on this zone, which makes 'holes' in the depth map.

There are different ways of interpreting these zones in the process of generating the trimap. We can basically say the are part of the foreground, or part of the background. Generally, these zones are around the object in the foreground we want to extract, and attributing them to the foreground would make it bigger than it really is. That's why we decided to 'project' these zones as for as possible (concretely, set their value at 1 once the depth has been normalized). However, in some cases, we can see that those unknown zones actually give details about the foreground object. So this choice make us lose some details, but is justified by the fact that the depth map is not that precise, so those details would not be relevant.

### 3.2. Trimap generation by segmentation

The goal is to generate a trimap from the data we acquired from the Kinect. This is basically a segmentation problem, as we want to find 3 clusters: a zone where we're sure it's the foreground, a zone where we're sure it's the background, and in the middle, a unknown zone where we apply a matting algorithm. We tried two approaches, one based on k-means, the other based on NCuts.

#### 3.2.1 With k-means

We used k-means to find two clusters. We define the distance between two pixels as the difference of their depth. This is basically equivalent to project the points in one dimension, the z-axis, and segment using their depth as their position.

If the foreground is separable from the background, then k-means should find two clusters that do not overlap. This is the easiest case, and it worked as expected, as illustrated on figure 1. The non trivial, and more interesting case is when the depth of background from which we try to separate the object overlaps with the foreground. This might be the case if you're in a corridor, or if you're in an angle. In those cases, the walls have continuous depth, form 'far away', to even closer than the object we want to extract.

What we can do is to project the clusters onto the image space, ie look at the cluster where each pixel falls in the (x,y) dimensions. In the problematic situations described above, the cluster corresponding to the foreground would be discontinuous, and separated by the background. We can assume that the object we're interested in is near the center of the image, and that it is continuous. That way, we can correctly adjust the foreground cluster by taking only the part that satisfy those conditions.

More precisely, we look for the closest pixel to the center of the image $P$ that belongs to the foreground cluster. The distance is the Euclidian distance using pixel coordinates. This assumption in kind of restrictive, but is also natural, as the picture is most likely center around the object you're interested in. We then extract the pixels that fall into the foreground cluster one by one, starting from $P$. We explore the continuous component, by visiting neighbors than point we've already extracted; we basically perform a BFS. We add one more restriction on the neighbors, which is that they must be close enough in depth. This is also natural, as the subject should also be continuous in depth, so its corresponding pixels should not make big jumps in that dimension (although we are obviously limited by the resolution of the depth map from the Kinect).

**Trimap creation** Once we have two clusters, we have to build the trimap, and thus define the confidence we have in each of the clusters. K-means does not really allow us to do that, because it does not give any probability of a point belonging to a cluster. One way to build the unknown zone in the trimap is to take the frontier between the background and foreground from our cluster in image space, and expand it to give it some width. This method is not very precise, but it follows the intuition that the clusters might be off by some pixels because of the segmentation, or that there might be some noise in the data. The only parameter we have in our method is the size of this unknown region we generate. We will discuss the impact of this parameter further down.

#### 3.2.2 Improving k-means?

In the previous section, the method we described relied on using k-means with the depth information we have, but we do not use any other geometric information. An idea is to segment in a 3-d space. It could be the 3-d positions of the points, that we can have as we do the calibration process, or a simplified position with x and y being the pixel coordinates, and z the depth of the pixel.

Let us consider the situation we described above, where the object lies between two walls, and shown on the second picture of figure 5. Basically, the top left corner is far away, and the bottom right corner is at the same depth as the object, or even closer. Using the distance in 3-d space cannot make those two points belong to the same cluster. Indeed, the bottom right corner is much closer to the object than to the top left corner, and the object is closer to the top left corner than the bottom right corner. This means that the object is approximately between those two interest zones, so they cannot belong to the same cluster, without the object in the cluster too, which we obviously do not want.

This proves that in difficult situtations, adding more geometric information does not really help. Besides, k-means tends to give circular clusters, which might result in the object being separated between the two clusters. Thus, the first method we described with k-means is better for our use.

#### 3.2.3 With NCuts

Our depth-based image segmentation was based on the Normalized Cuts algorithm formulated by Shi and Malik in [5]. The idea of N-cuts is to interpret image pixels as nodes, and the connections between nodes as edges. Weight values are assigned to the edges as a measure of similarity between nodes. For an $m \times n$ image, the weights are stored in an $m \times n$ affinity matrix, $W = \{w_{ij}\}$, where $w_{ij}$ is the weight value between node $i$ and node $j$ [5].

According to Shi and Malik, an optimal partition can be determined by solving a generalized eigenvalue system:

$$(D - W)y = \lambda D y$$

where $D$ is an $N \times N$ diagonal matrix with $d_i = \sum_j w_{ij}$ where $N$ is the number of pixels in the image. $y$ is a binary indicator vector that indicates whether the pixel belongs to group A or B [6].

Our depth-based image segmentation approach expresses the affinity matrix W as a function of depth. The code was adapted from Shi's Normalized Cut image segmentation MATLAB package online [7]:

$$w_{ij} = \begin{cases} e^{\frac{-|z_i - z_j|^2}{\sigma^2}} \text{ if } |z_i - z_j| < r \\ 0 \text{ otherwise} \end{cases}$$

**Trimap creation** The trimap was generated by taking the eigenvector corresponding to the largest eigenvalue, as it showed the most detailed segmentation of the foreground from the background. In a simple case with one foreground object, the background pixels (one cluster) are identified as pixels with eigenvector entry $\leq -0.90$. To further distinguish foreground pixels from unknown pixels, we define the foreground pixel to be $\mu_{FG+\text{unknown pixels}} \pm \sigma_{FG+\text{unknown pixels}}$, and the remaining unclassified pixels to be our unknown pixels.

One of the biggest limitations of the N-cuts algorithm in MATLAB is its inability to process large images due to memory restriction. We were only able to process at most a 6,400-pixel image. N-cuts is effective at defining clusters with well-defined shapes, however, our approach breaks down with a large number of segments, and with overlapping foreground and background depth data. In those instances, some clusters failed to distinguish between the foreground and background, and would require additional human processing to sort the clusters. Therefore, we decided not to pursue this method.

### 3.3. Digital matting

We first tried to implement the Bayesian matting algorithm, as done in [3]. However, there were some points that the paper [1] does not address, in particular the initialization of the algorithm, how to choose the size of the Gaussian, and how is determined the neighborhood on which is applied the iterative solving to optimize the likelihood, in order to find the alpha value of a pixel.

We instead use an implementation of [2] to do the matting from the color image and the trimap we generated in the previous step.

## 4. Experiment

### 4.1. Data set

We used our own Kinect to collect data. This allowed us to have pictures that are relevant for digital matting, with basically an object or a subject that constitutes the foreground, and that you want to extract from the background.

We first encounter issues as we thought that the Kinect was providing us with clean data, with a depth map corresponding to the rgb picture, meaning that the depth of a pixel $(i, j)$ in the rgb picture was at the pixel $(i, j)$ in the depth map. We thus had to work on the calibration and implementation of the code to have the correspondence we were expected.

After this operation, there are zones where we do not know the depth. Those are generally at the border at the image, so we set the normalized depth to 1, as we did for unknown zones due to occlusion that the Kinect gives us.

As we mentioned before, our correspondent depth is off by about 40 pixels from the rgb image. We had to manually shift it to get results. Otherwise, the error is too big, and the trimap is useless.

We selected different scenarios, and show the results in the next section.

### 4.2. Evaluation metric

The innovative aspect of our project is the generation of a trimap using the depth map of the kinect, and use it do digital matting. To evaluate the quality of our depth-based trimaps, we generate 'ground truths' by manually drawing a trimap with a good precision. We then apply the same matting algorithm with this trimap. The results using this method are very good, and they extract the foreground with very little error.

We believe our approach is more robust than manually creating ground truths by hand, because first it is difficult to set an alpha between 0 and 1 (a human operator will likely set it to 0 or 1, but not to something else), and second, it does not rely too much on the performance of the matting algorithm. However, it obviously does, because if it can perform well with a bad trimap, then any bad trimap can be given.

We used this method because you compare since two different trimaps could give the same result, so there's no real absolute way to compare trimaps. Since the goal is to do matting, it makes sense to evaluate after this operation.

We first calculate the difference between the alpha mask computed by the matting algorithm from the generated trimap and the ground truth. We thought it was more relevant to express the error as a percentage, so we divide the difference by the sum of the values of the ground truth, which is about the area of the foreground. It is better than the actual area of foreground, because if the alpha value of a pixel is low, which means there's almost no foreground on it, then it is not very important if you have the good value.

### 4.3. Results

On figure 1, the chair is supposed to be easily separable from the background. And we can see, k-means correctly extracts the chair, and by superimposing it to the rgb picture, we can see it is almost an exact fit. The difference with the ground truth shows that most of the error arises from the legs of the chair. This can be explained by the fact that the legs are thin, and that on the generated trimap, the legs are almost entirely in the unknown zone. It is then up to the matting algorithm to determine that it is part of the chair, which it seems to be really able to (which understandable given the colors) .

So we see that one potential problem is in the way we generate the unknown zone in the trimap. Because we do not have any information on the probability that a pixel belongs to the foreground or to the background, we have to assume a uniform zone. The impact of this is that finer features easily fall into the unknown zone. Thus, we might want to use a small unknown zone. This is illustrated on figure 2.

However, because of the lack of precision of the Kinect (the depth map is not precise up to one pixel, but to 2 or 3 pixel on average), we have to set the unknown zone to be big enough, otherwise, the whole contour of the shape will be wrong. The figure 3 shows this.

The figure 4 shows a case where the foreground and the background are not easily separable. There's indeed a part of the wall on the left and on the right, part of the table behind that falls into the same cluster as the subject after applying k-means. Our method performs as expected, and removes the wall because it is not continuous with the subject, and removes the table because it's depth is much different from the arm. There are some holes in the result segment (in the head...), but this is due to the imprecision of the Kinect.

The table 1 shows some results we have, using the metric we defined earlier. We can see that the error can vary a lot depending on the image. This is mainly due to the actual size of the object. Indeed, for an image with a person that fills a lot of space, the error is relatively small, but for an image with a small object, like the chair (picture 4), the error is relatively much bigger, be-

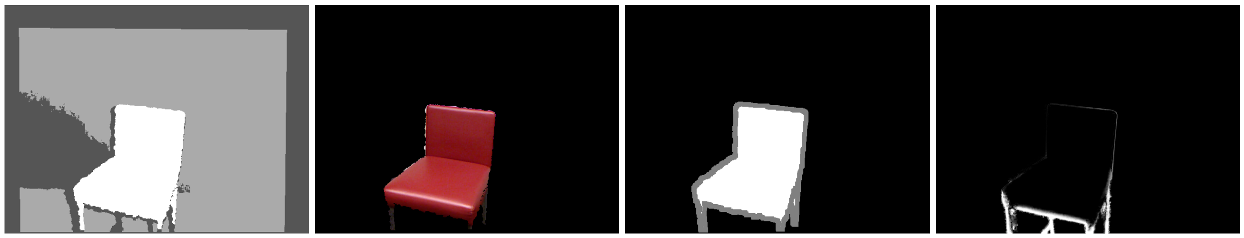

Figure 1. Foreground and background are easy to separate. From left to right: clusters from k-means, foreground segment superimposed to the rgb picture, generated trimap, difference with ground truth

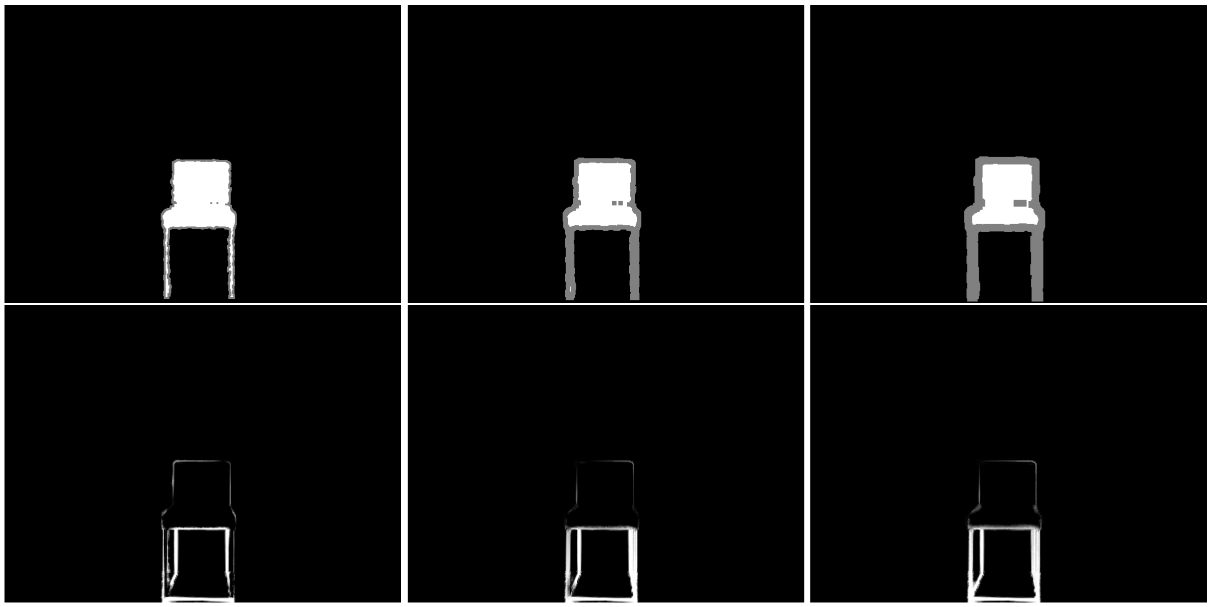

Figure 2. Impact on the size of the unknown zone, equal to, from left to right 1, 3 and 5. The above line shows the trimaps, and the bottom line the difference with the ground truth

cause it is defined as a percentage of the area of the object (or something close, as we mentioned before). This is also due to the fine details, that the generated trimap cannot correctly isolate.

If we ignore this picture for which the error is quite high, the error of the algorithm, compared to the ground truth, is about 10%, or even 5% for human subjects. Those results also show that a width of 4 pixels is a good compromise for the unknown zone. But for picture 4 for example, because of the skinny legs, the smaller the unknown zone is, the better is the result.

| image | error (1) | error (3) | error (4) | error (5) |
|---|---|---|---|---|
| (1) | 0.0551 | 0.0443 | 0.0341 | 0.0404 |
| (2) | 0.1534 | 0.1196 | 0.0746 | 0.0454 |
| (3) | 0.1163 | 0.1085 | 0.1067 | 0.1083 |
| (4) | 0.2021 | 0.2637 | 0.2815 | 0.2867 |
| (5) | 0.1051 | 0.0901 | 0.0794 | 0.0685 |
| (6) | 0.0728 | 0.0643 | 0.0636 | 0.0644 |

Table 1. Results corresponding to the images from figure 5. Number in parenthesis is the size of the unknown region in the generated trimap

## 5. Conclusion

In this paper we have shown that depth data could provide useful information to solve the RGB matting problem. We do so by calibrating a Kinect camera to obtain parameters that enable us to map the depth image to the RGB image. With the parameters, we then explored different segmentation techniques, namely K-means and N-cuts, to generate clusters with depth data. In im-

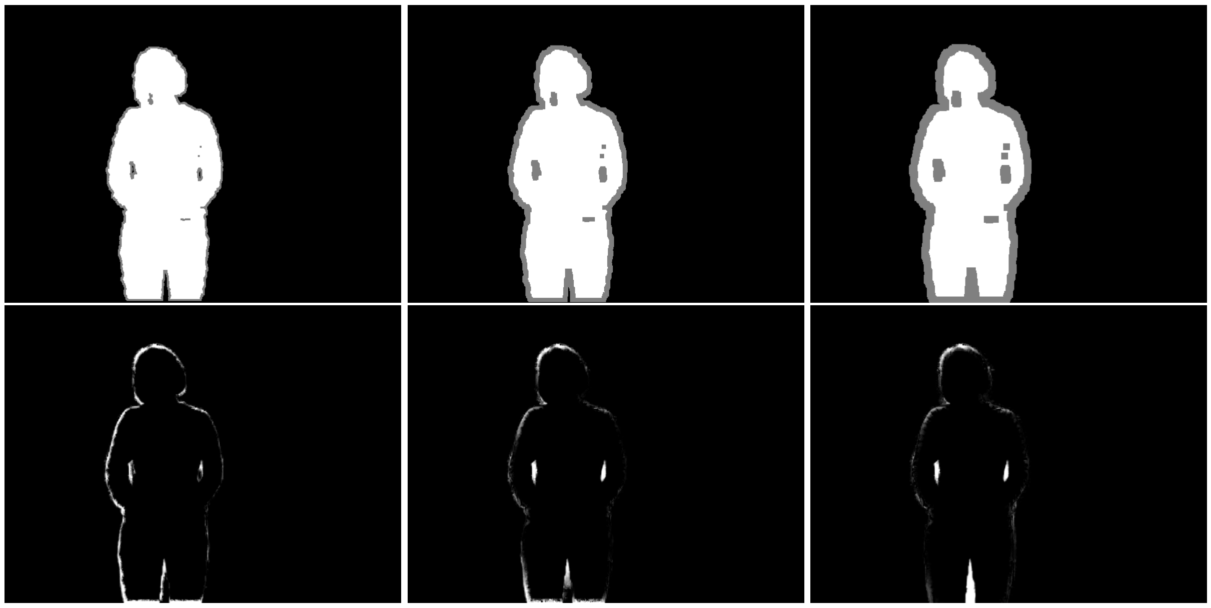

Figure 3. Impact on the size of the unknown zone. The pictures show the difference with the ground truth with a size of 1, 3 and 5

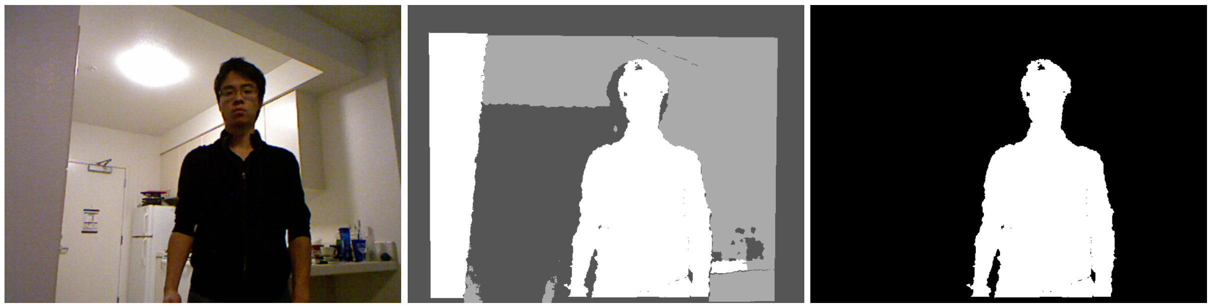

Figure 4. Foreground and background are difficult to separate. From left to right: rgb picture, clusters from k-means, segments to isolate the subject of interest

ages where the foreground object is located at the center of the image, we found that trimap generation with K-means worked fairly well when that assumption is applied. In most cases, the error was less than 10%. However, the generation of the unknown zone in the trimap is the source of undesirable results when fine details represent a significant part of the foreground. One of the limiting factors that undermines the performance of our algorithm lies in the hardware. The coarse resolution of the sensors, and the shadowing effect caused by the location of the sensor and projector increased the size of the unknown region in the trimap. This led to a high error in some of our test images. Another limiting factor is processing speed, which is partly due to our MATLAB implementation (we had to use some Java data structures, which slowed down the algorithm). We could improve the performance by implementing the code in another language. By overcoming the above limitations, we would hope to apply our approach to video matting on the xbox games, which would greatly enhance gamer experience.

## References

[1] A Bayesian approach to digital matting, B. Curless, D. Salesin, R. Szeliski

[2] Learning based digital matting, Y. Zheng, C. Kambhamettu

[3] Video matting from depth maps, J. Finger, O. Wang

[4] A closed-form solution to natural image matting, A. Levin, D. Lischinski, Y. Weiss

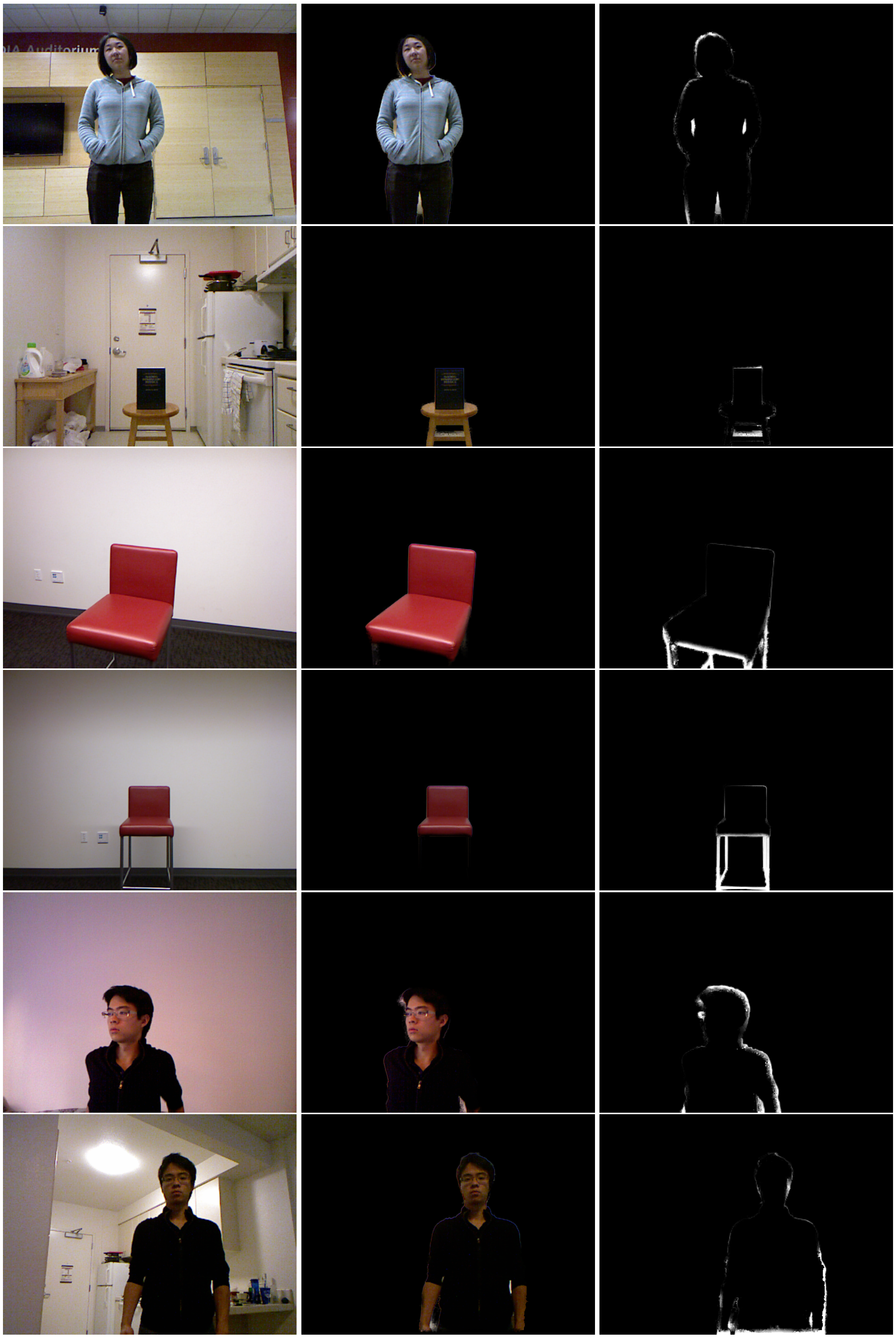
Figure 5. Results from a selection of pictures: rgb image, result using the automatically generated trimap and difference with ground truth

[5] Normalized Cuts and Image Segmentation, J. Shi, J. Malik

[6] A Normalized Cuts Based Image Segmentation Method, F. Sun and J. He

[7] J. Shi MATLAB Normalized Cuts Segmentation Code, `http://www.cis.upenn.edu/~jshi/software/`

[8] `http://nicolas.burrus.name/index.php/Research/KinectCalibration`

[9] `http://openkinect.org/wiki/Imaging_Information`

[10] `http://graphics.stanford.edu/~mdfisher/Kinect.html`