UNIVERSITY OF CALIFORNIA
RIVERSIDE

"MAGIC CAMERA"

A project report submitted in partial satisfaction of the requirements of the degree of

Master of Science

in

Computer Science

by

Adam Meadows

June 2006

Project Committee:
Dr. Eamonn Keogh
Dr. Doug Tolbert

The Project Report of Adam Meadows is approved:

Committee Chairperson

University of California, Riverside

ABSTRACT OF PROJECT REPORT

"Magic Camera"

BY

Adam Meadows

Master of Science, Graduate Program in Computer Science
University of California, Riverside, June 2006
Dr. Kim Possible, Chairperson

The magic camera is a piece of software that takes as its input a picture of a collection of objects in front of a common, solid background. The output is another picture, containing the same set of objects in front of the same solid background, organized so that similar objects are grouped close to one another and made to "face" the same way when applicable. The objects in the input picture are assumed to be separated by some noticeable margin, objects that are touching will be treated as one object. Pictures are assumed to have been taken with appropriate lighting to avoid shadowing effects. The perspective of camera which took the picture is assumed to be parallel to the surface being photographed, to minimize distortion.

Magic Camera

Adam Meadows

June 12, 2006

1 Introduction

This project is designed to create a piece of software which transforms an image into a more organized version of that image. The resulting image will be organized based on the similarities of the various objects contained in the original image. These similarities can be based on a number of different criterion. The three currently supported criteria are shape, color, and texture. Not only will similar objects appear close to one another in the resulting image, but all objects will "face" the same way. This orientation is accomplished by using the physical properties of each object, such as major axis and center of mass, to determine which way they should "face."

2 Motivation

The motivation behind this project is to be able to automatically organize collections of objects and display them visually. Ideally, this software would be incorporated into a digital camera, allowing users to take photographs of a collection of objects and instantly be able to see patterns in that collection. Possible uses for such a camera could be: categorizing archeological finds such as bones or arrowheads, visualizing similarities in the color of butterfly wings, organizing insect boards, etc. With a little additional work and in combination with a robotic arm, this software could even produce an automatic sorter to organize collections of objects.

3 Background

Here is a little background into the visualization of dissimilarities among objects. While both MDS and SOMs are both possible solutions, for this project I chose to use MDS because it seemed simpler to use and doesn't suffer from the local optima problem that SOMs face.

3.1 Multidimensional Scaling

"Multidimensional scaling (MDS) is a set of related statistical techniques often used in data visualisation for exploring similarities or dissimilarities in data." [4] A dissimilarity matrix, which contains the distances between each pair of objects, is converted into a set of coordinates in low dimensional (usually 2d or 3d) space. The euclidean distances between these coordinate points reflect the original distances given in the dissimilarity matrix. In classical MDS, which was used in this project, when computing the coordinates for each object, "an eigenvector problem is solved to find the locations that minimize distortions to the distance matrix" [4]

3.2 Self-Organizing Maps

Similar to MDS, self-organizing maps (SOMs) are a way to visualize similarities between objects in low dimensional space. Instead of using eigenvectors to determine where to place each object, SOMs use a number of iterations, during which similar objects are gradually moved toward one another, and dissimilar objects are repelled. After enough iterations, the movement will stabilize, yielding the resulting map. However, it is possible that a different initial arrangement could result in a better resulting map. This local optima problem is not present when using MDS.

4 Stepping Through Magic Camera

4.1 Identifying Objects

The first step in magic camera is to identify the individual objects in the input image. The image is first converted from color to grayscale. The grayscale image is then converted to a black and white image, where the background will be black, and all the objects will be white. The threshold for this conversion from grayscale to black and white can be computed automatically, using Matlab's *graythresh* [1] function, or specified by the user. The user need only specify a threshold if the background color is very similar to the color of one or more of the objects in the image. Since objects are assumed to be separated by some noticeable margin, each connected component of the black and white image is considered to be an object.

Once each object has been identified, the Matlab function *regionprops* [2] is used to find out some useful information about each object. Specifically, each object's bounding box, orientation, and centroid. Each objects is then cropped to it's bounding box (plus a five pixel border). Unfortunately, this is not always good enough. If the objects are shaped correctly and spaced close enough together, there is a possibility that the bounding box for one object may contain part of another object as well. To account for this, all pixels that do not correspond to the object being copped are filtered out and replaced with the solid background color. Figure 1 shows the results of this filtering.

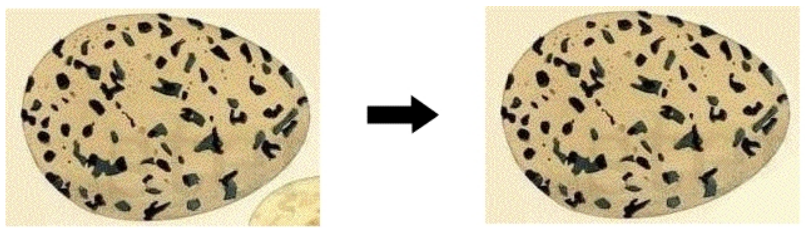

Figure 1: Filtering Cropped Image

Once the objects are separated into their own images, it's time to rotate them. The orientation is used to rotate the object so that its major axis is in line with the image's major axis. If the original image is wider than tall, each object is rotated so that its major axis is horizontal. Likewise, if the original object is taller than wide, each object will be rotated to be vertical. Once the major axis is aligned, one more check is performed. If the centroid of the object is not on the left/bottom half of the image (depending on the horizontal/vertical orientation), then the object is rotated so that it is. This should ensure that the same object, regardless of its original orientation will always be "facing" the same way in the resulting image.

4.2 Calculating Similarities

To calculate the similarities among the objects, each object is first converted to a numerical representation. The method for converting the object to a numerical representation is determined by the basis of comparison among objects. Currently three comparisons are supported: shape, color, and texture. Once each object has a numerical representation, a dissimilarity matrix is constructed. The distance between each pair of objects is computed (currently using Euclidean distance).

4.2.1 Shape

When comparing by shape, each object is converted into a time series. The time series represents a trace along the perimeter of the object, where each point in the time series is a distance between the center of the object, and the perimeter at that point. The code for this extraction was provided by Dr. Keogh. Figure 2 shows how a time series is extracted from an image.

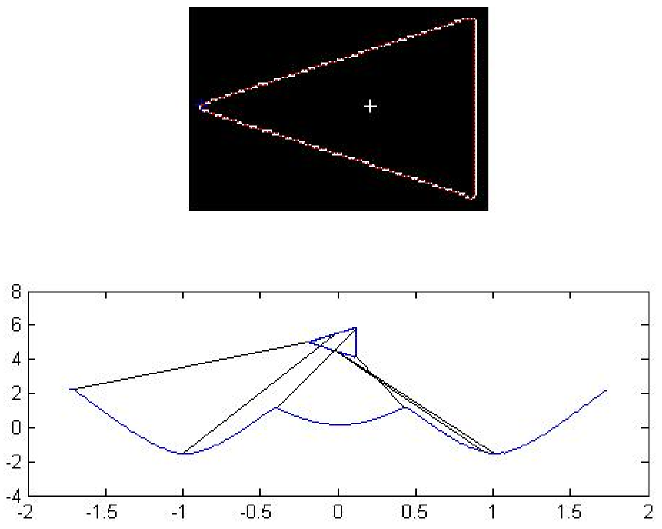

Figure 2: Converting an image to a time series

4.2.2 Color

When comparing by color, the numerical representation is simply the R G B values for the color of the object. To extract the color, the R, G, and B values are independently averaged over 1000 random pixels within the object. The pixels sampled are not unique, so that it does not matter if the objects contain less than 1000 pixels.

4.2.3 Texture

When comparing by texture, the numerical representation is a single numerical approximation of the texture of the object. This approximation is calculated by sampling 1000 random pixels within the object, computing the standard deviation of the nine pixel neighborhood surrounding that pixel, and averaging those values. Figure 3 shows a nine pixel neighborhood.

4.3 Creating New Image

Creating the new image consists of three main parts: extracting the background, finding the new positions for each object, and fixing overlaps that occur because of those new positions.

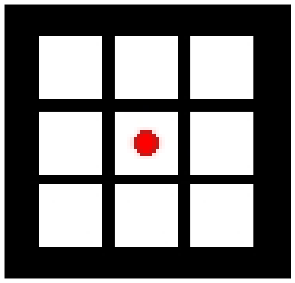

Figure 3: A nine pixel neighborhood

4.3.1 Extracting the background

Extracting the background of the image is accomplished in the same manner as determining the color of the individual objects. A random sample of 1000 pixels is taken, only this time the pixels are all chosen from the background of the image. The black and white image is used to identify the pixels which make up the background of the image. The RGB values of the background are then independently averaged to produce a solid color for the background. A new image, consisting only of this solid, average color, and having the same dimensions as the original input image is then created.

4.3.2 Finding new positions

The new positions for the objects are calculated by using classical multidimensional scaling. The dissimilarity matrix computed earlier is passed into the MDS algorithm [3], which returns a set of coordinates for each object in 2D. The y-values of each location are reversed due to the fact that images are indexed top-down (see Figure 4).

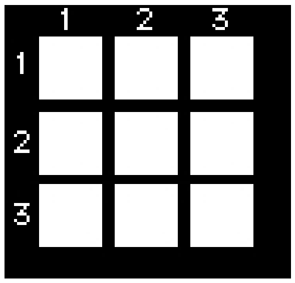

Figure 4: Top-down indexing of images

4.3.3 Fixing overlaps

Unfortunately, simply using the locations produced by the MDS algorithm is not sufficient. Due to the varying size of the objects being organized, simply placing them all at their appropriate positions can cause overlaps which obstruct the view of some of the objects. A simple approach is used to compensate for this. The objects are placed in the blank background image in a specific order. Since the only knowledge of the similarity between these objects is their respective positions in the resulting image, the order used to place them is that position. Of course, there are still many orders to choose from. For the purposes of this project, four possible orders are considered: left-right-top-down, left-right-bottom-up, right-left-top-down, and right-left-bottom-up. The user can choose on of these orderings, or specify that one should be chosen at random.

Once an ordering for placement of the objects has been decided, each object is placed, in order, into the image. Once an object is placed, it is not moved. Before placing each object, a check is made to see if placing the object at its specified location will create an overlap with previously placed objects. If no overlap is detected, the object is placed in its specified position. However, if an overlap is detected, the minimum safe distance to be moved is calculated. This minimum safe distance is defined to be the minimum distance (in one of the four cardinal directions) in which the object can be moved to avoid an overlap (see Figure 5). If no movement in any direction corrects the overlap, a random direction is chosen, and the process repeated. In such a manner, each object "walks" around the image, placing itself in the first free space it finds.

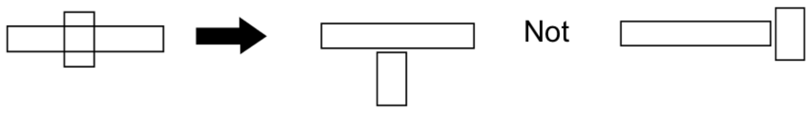

Figure 5: Fixing overlaps

5 Results

The following examples show input and output images of the magic camera to demonstrate its effectiveness in organizing collections of objects. Figures 6 and 7 show the input and output images of a very simple example of organizing objects by shape. Figures 8 - 11 show the varying output images using different orderings of placement of the objects into the blank background image. Figures 12 and 13 show the input and output images of a simple color example. In this case, the objects are not organized by their shape, but by their respective color. Figures 14 and 15 show the input and output images of an example of organizing a collection of eggs by texture. Figures 16 and 17 show the input and output images for an example of organizing a collection of skulls by shape. In this example, rotating the individual objects so that their center of mass was in the bottom/left of the object, did not increase clarity. In response to this result, an input parameter was added to magic camera allowing for the user to choose whether magic camera should try to rotate the objects or not. Figures 18 and 19 show the same example, with the no-rotation flag set. Figure 20 shows the same after image as Figure 19 with labels showing that the six species present in the original skull image are separated in the resulting image.

6 Conclusion

Magic camera takes an input image, consisting of a collection of objects in front of a solid background, separated by some noticeable difference, and creates an output image, consisting of the same set of objects, in front of the same solid background, where similar objects are grouped close to one another and rotated to "face" the same direction. Supported criteria for similarity measures are currently shape, color, and texture. Multidimensional scaling is used to determine the new locations of each object in the output image. Objects are placed in the output image in a specific order (user specified or randomly chosen). When overlaps occur, the object being placed is moved the minimum distance in one of the four cardinal directions to correct the overlap. If all movements create new overlaps, then a random direction is chosen, and the process is repeated. If the rotation performed by magic camera fails to increase clarity in the resulting image, it can be turned off by the user.

7 Future Work

Future work on magic camera will include further development of the color similarity measure, adding the ability to combine similarity measures, and an optional how-to option. Developing the color similarity measure should involve testing it on some real-world data (butterfly wings, etc.). Combining similarity measures will include finding some way to average the similarities calculated by different means (color and shape, texture and color, etc.) The how-to option might include displaying the original image, and upon a user selecting an object, displaying its new location in the output image.

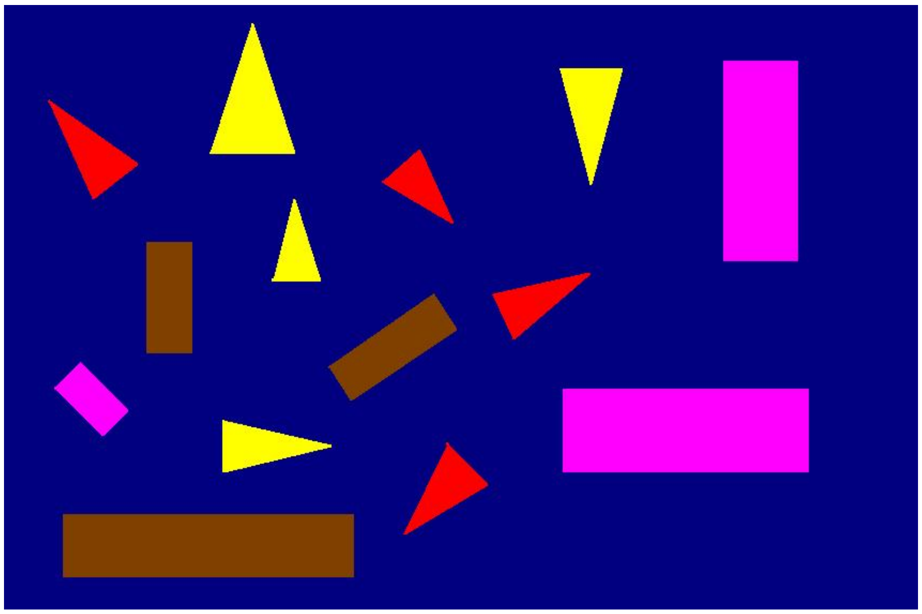
Figure 6: Simple Shape Example: Before

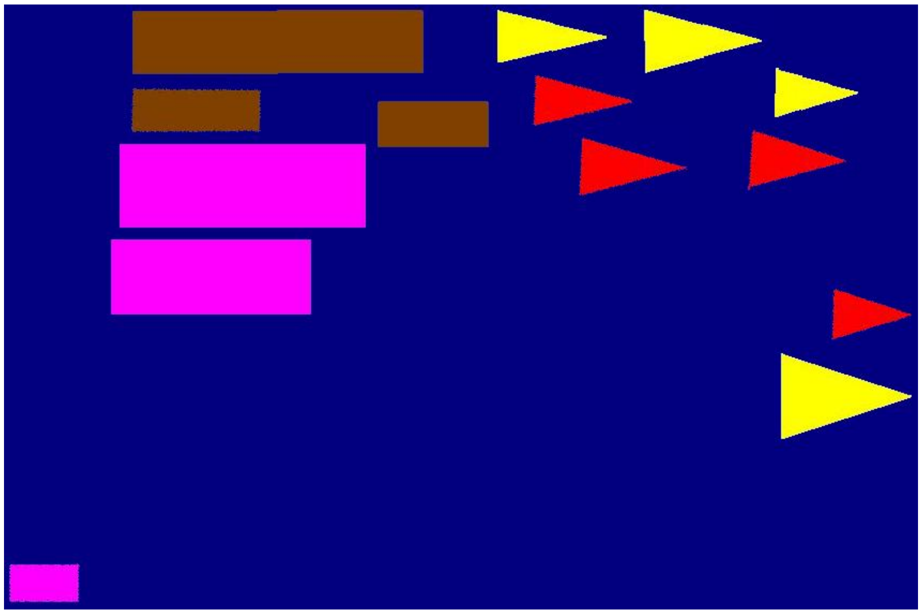
Figure 7: Simple Shape Example: After (R-L B-T)

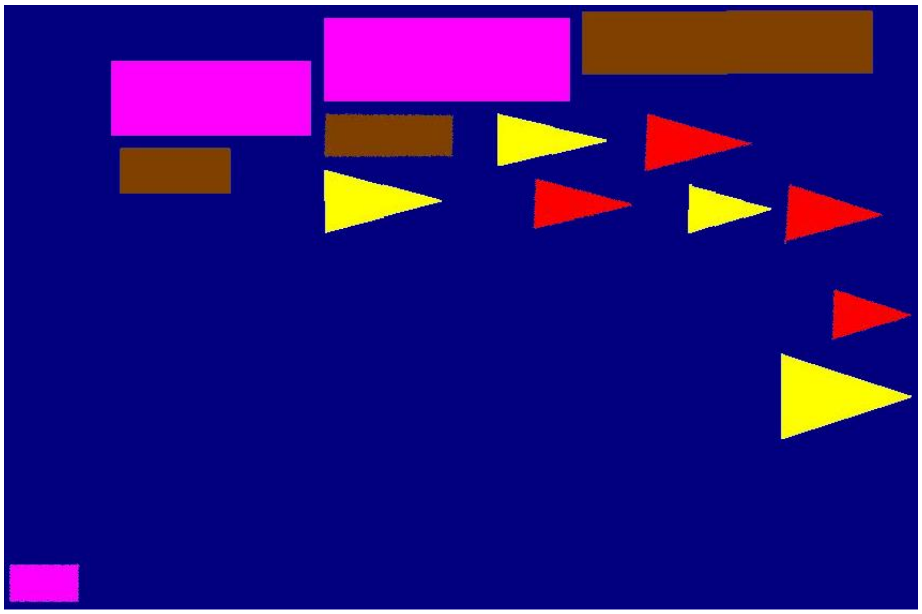

Figure 8: Simple Shape Example: After 1 (L-R T-B)

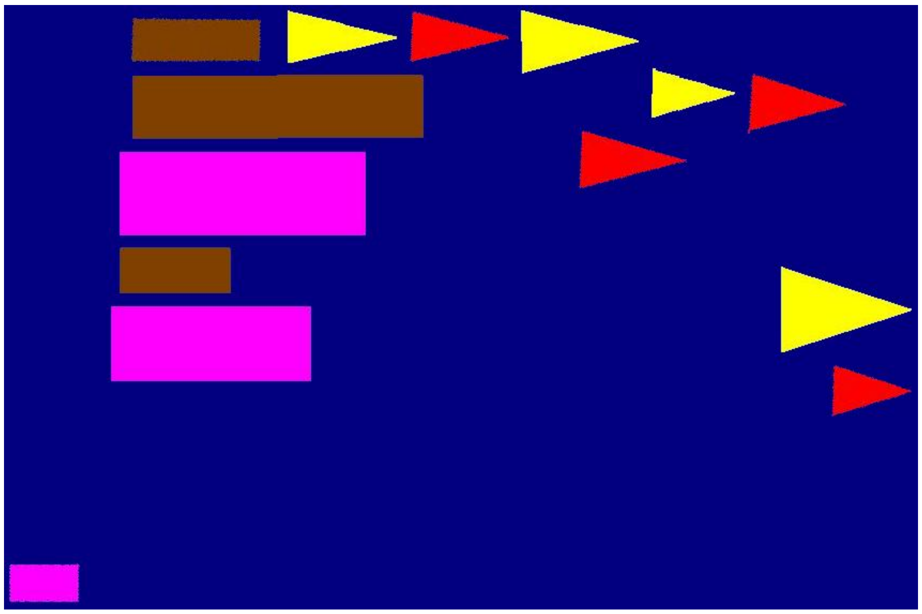

Figure 9: Simple Shape Example: After 2 (L-R B-T)

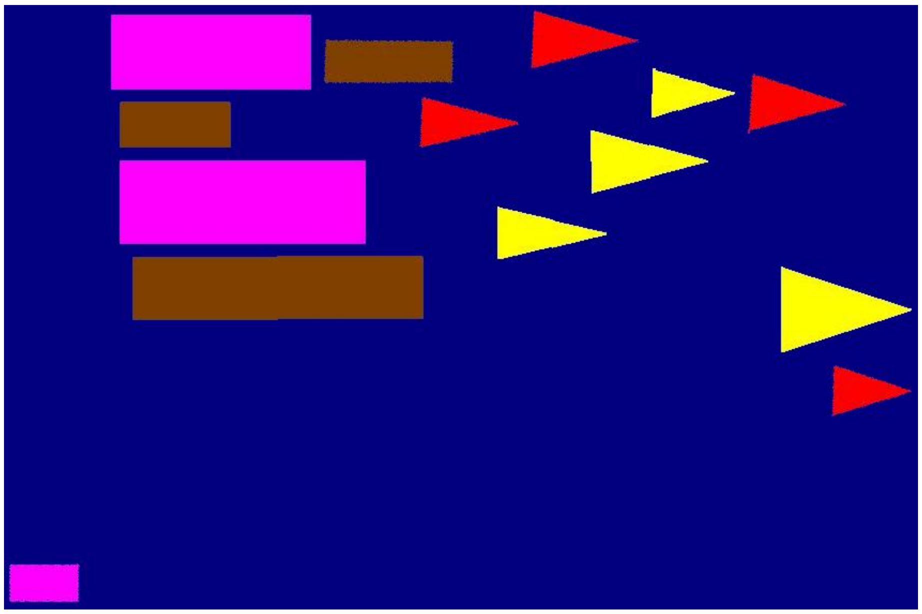

Figure 10: Simple Shape Example: After 3 (R-L T-B)

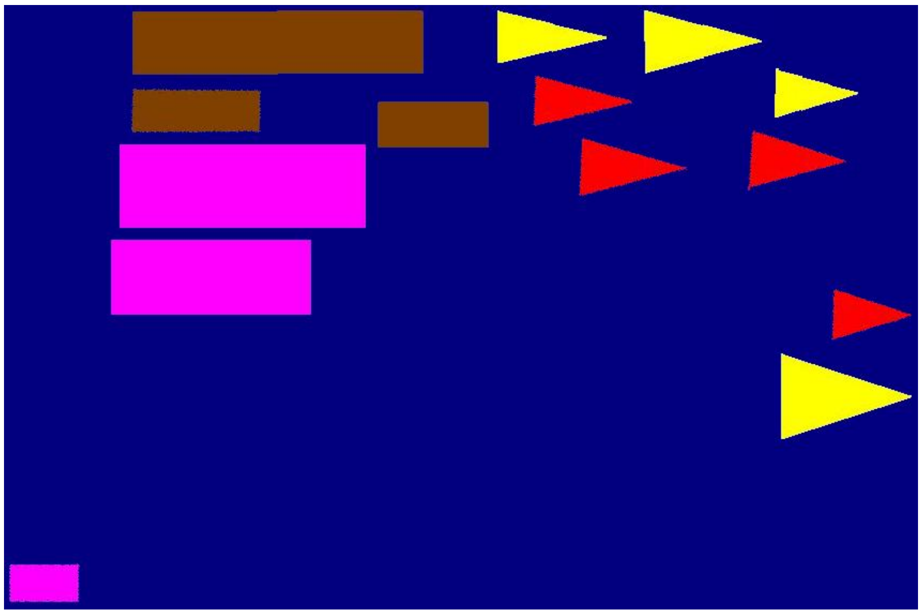

Figure 11: Simple Shape Example: After (R-L B-T)

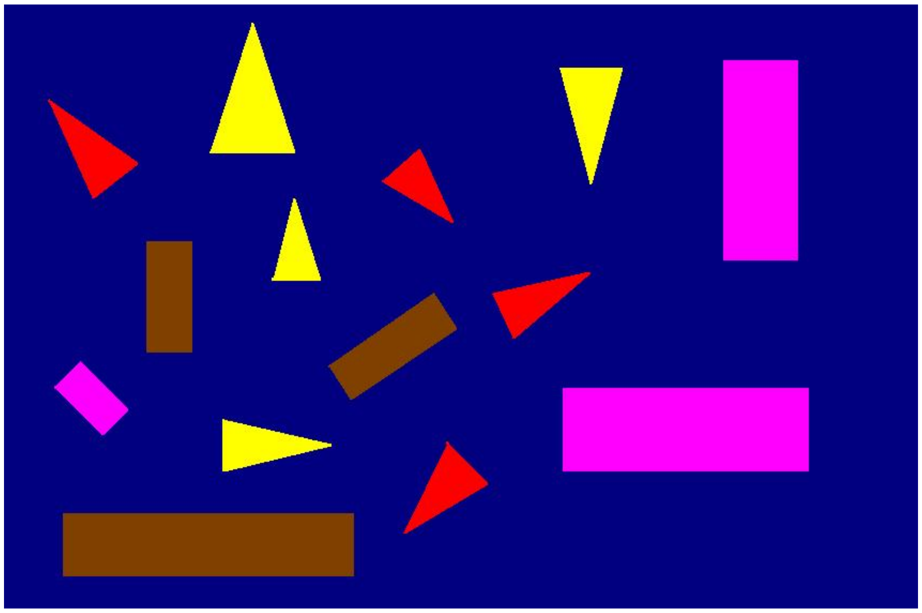

Figure 12: Simple Color Example: Before

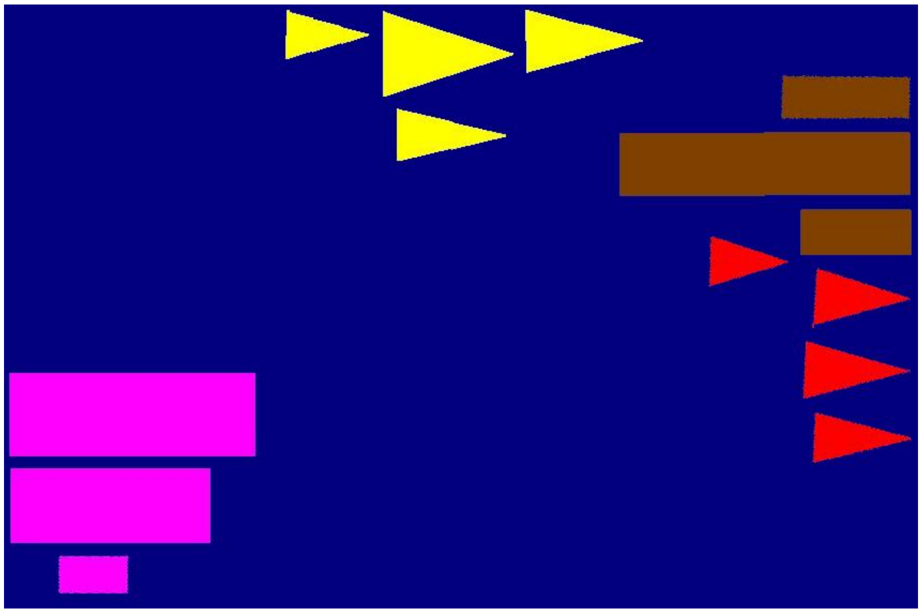

Figure 13: Simple Color Example: After

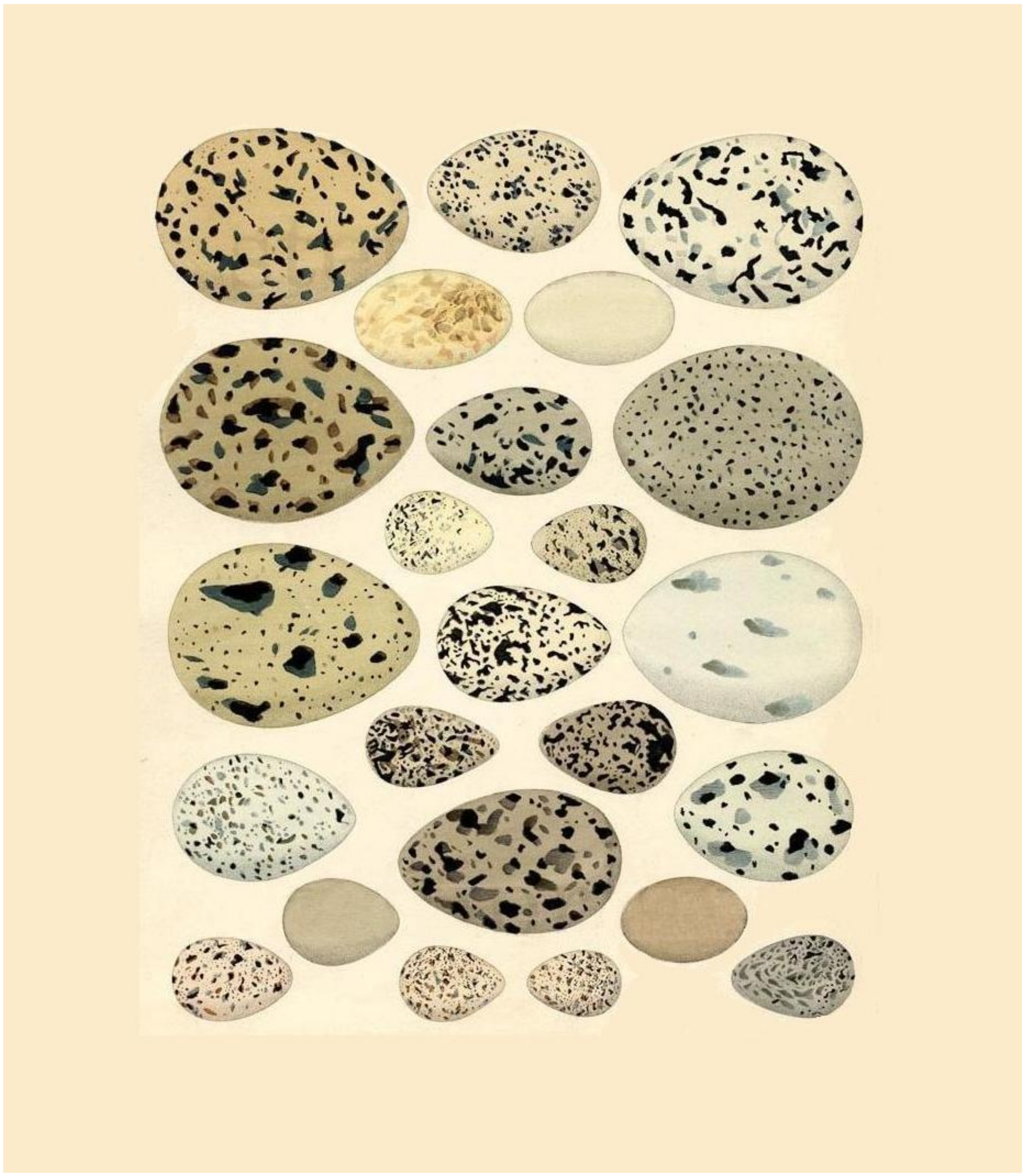

Figure 14: Eggs Texture Example: Before

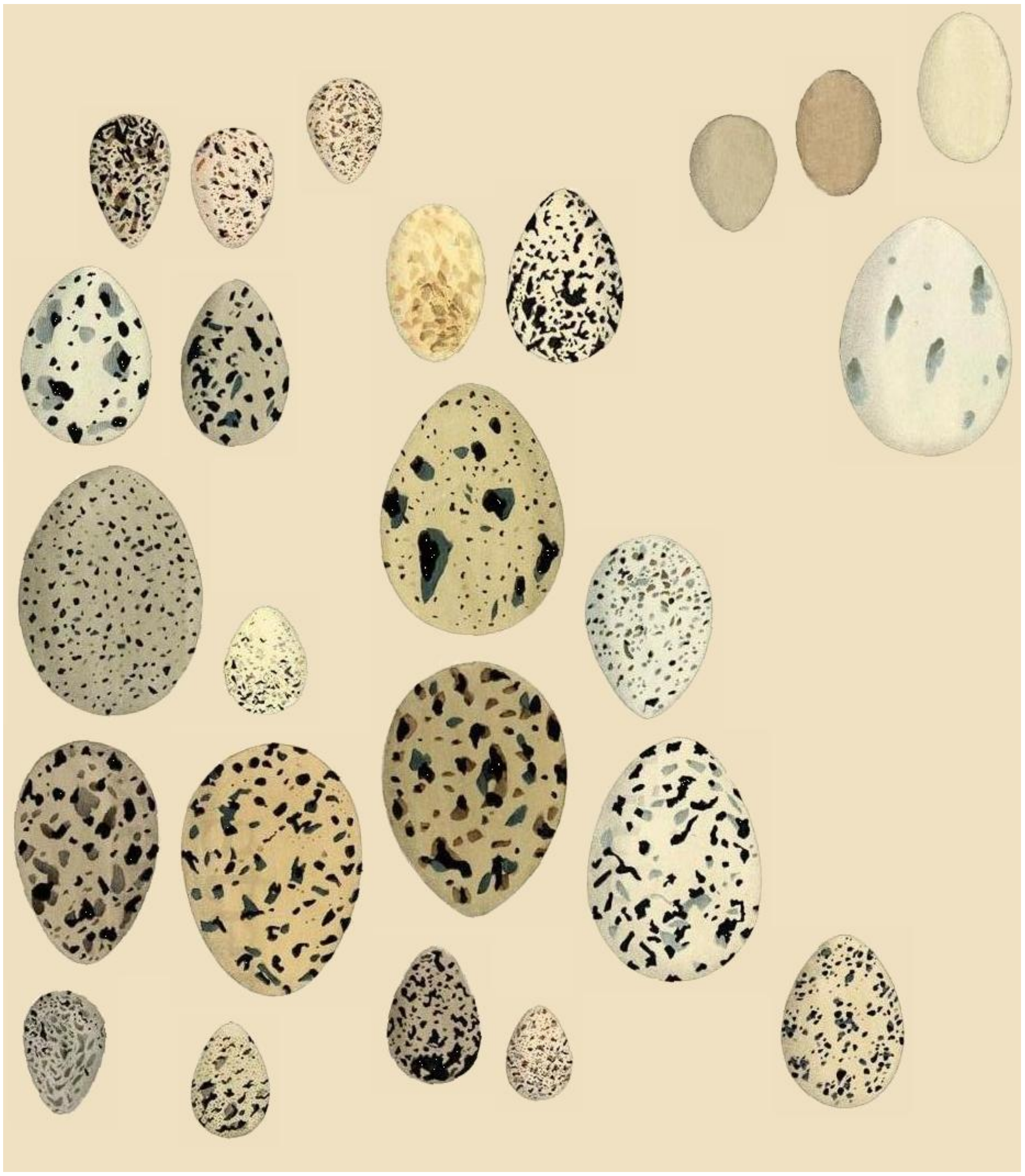

Figure 15: Eggs Texture Example: After

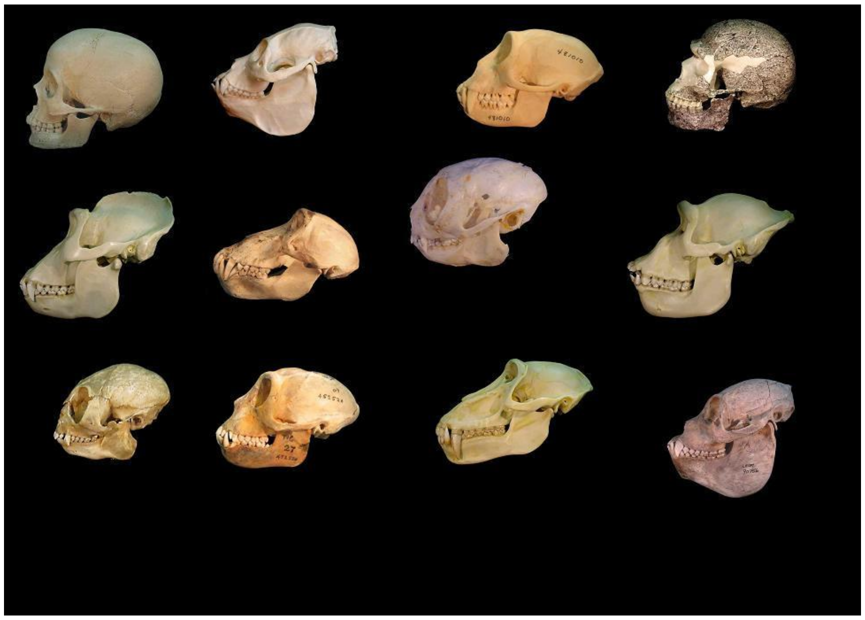

Figure 16: Skulls Shape Example: Before

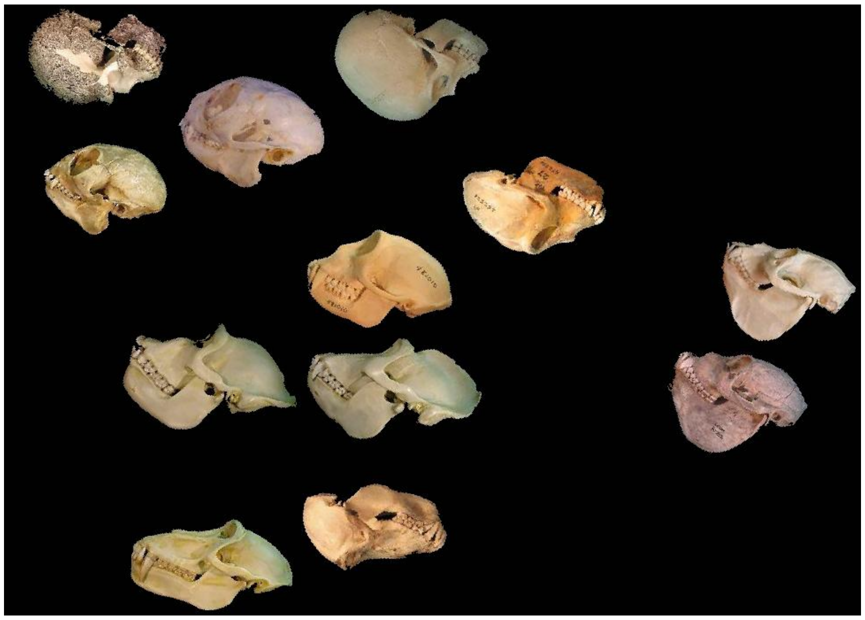

Figure 17: Skulls Shape Example: After

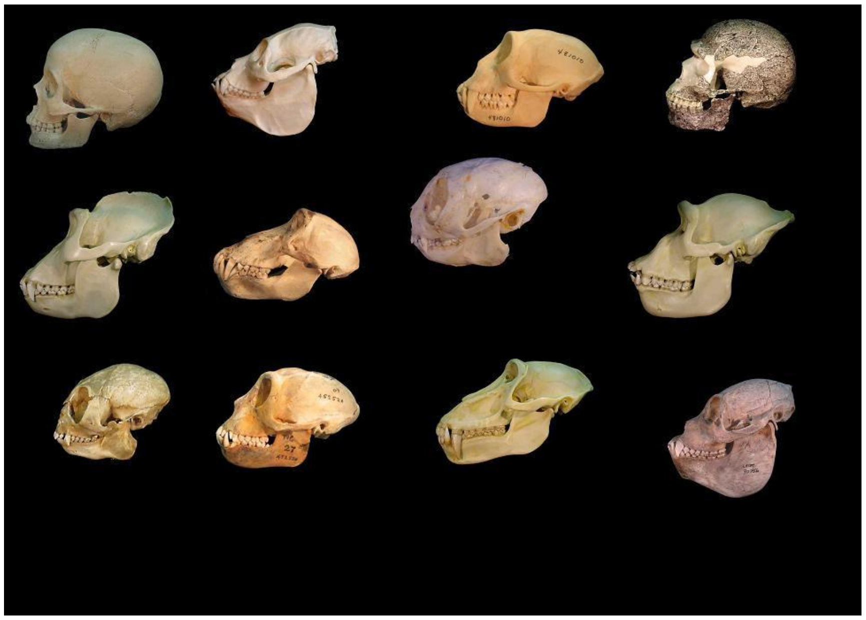

Figure 18: Skulls Shape Example: Before

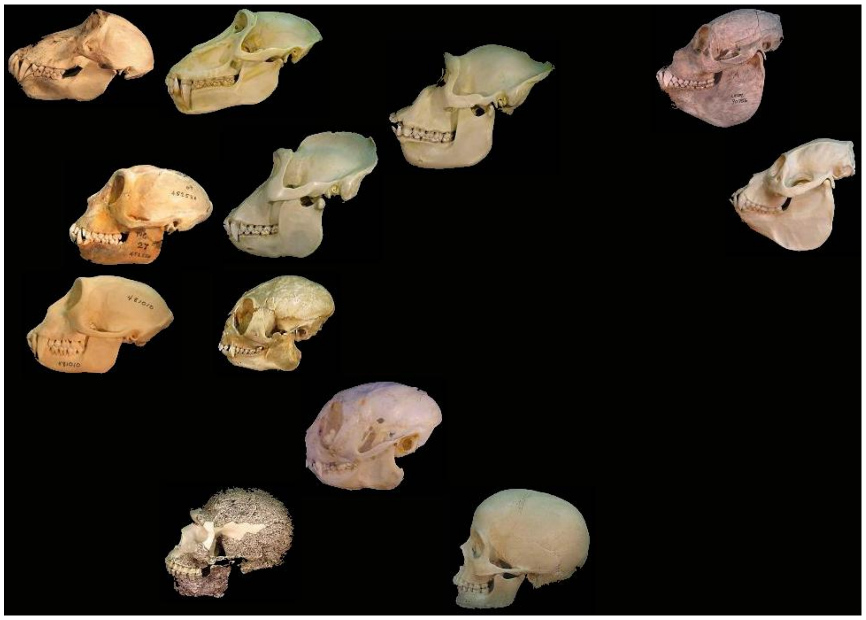

Figure 19: Skulls Shape Example: After No Rotation

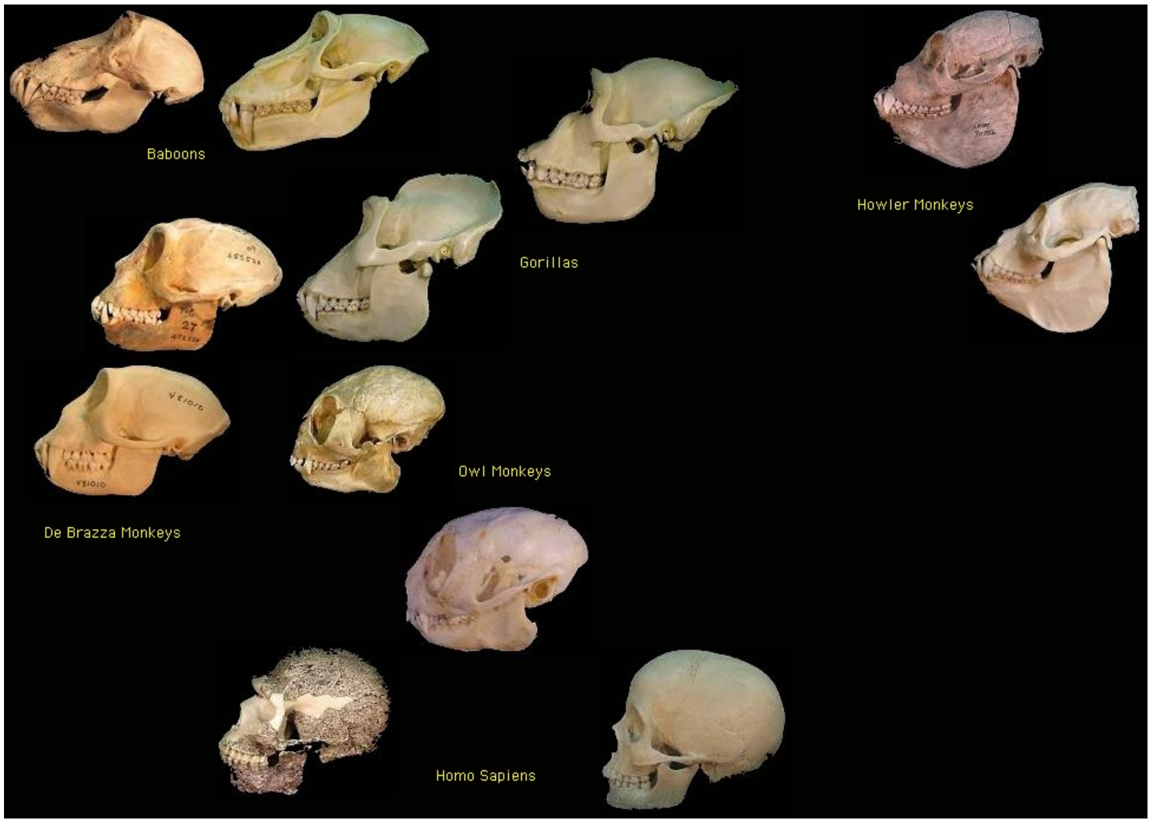

Figure 20: Skulls Shape Example: After No Rotation Labeled

References

[1] Mathworks Inc. Image Processing Toolbox: graythresh. `http://www.mathworks.com/access/helpdesk/help/toolbox/images/graythresh.html`; accessed June 7, 2006.

[2] Mathworks Inc. Image Processing Toolbox: regionprops. `http://www.mathworks.com/access/helpdesk/help/toolbox/images/regionprops.html`; accessed June 6, 2006.

[3] Michael D. Lee. `http://www.socsci.uci.edu/~mdlee/`; accessed June 6, 2006.

[4] Wikipedia. Multidimensional scaling. `http://en.wikipedia.org/wiki/Multidimensional_scaling`; accessed June 7, 2006.

[5] Wikipedia. Self-organizing map. `http://en.wikipedia.org/wiki/Self-organizing_map`; accessed June 7, 2006.