Shape matching

# Abstract

*Our project idea is to approach the CBIR (Content-Based Image Retrieval) problem with shape matching techniques, mainly using EMD (Earth Mover's Distance) as a similarity measure between representative weighted point sets extracted from the images. This paper explains which approaches were taken and presents an analysis of the obtained testing results.*

# Introduction

Nowadays there is the need to solve the problem of searching for digital images in large databases. CBIR is the application of computer vision techniques to this image retrieval problem. Although it is relatively expensive to compute, the results it offers are often more accurate (from the human point of view) than the results of image retrieval with concept based approaches (this is, image retrieval through image meta-data like associated descriptive keywords, size, format, etc.).

The CBIR problem can be translated to the problem of finding which are the most similar images in an image database to a query image. To answer this, it is natural to think about computing the similarity between the query image and all the images in the database, although there are some optimization methods to avoid this and improve the query efficiency (see section ???).

The similarity comparison algorithms normally don't take real images, as even with small databases the computation cost would be too heavy. Instead, they compare some reduced feature sets extracted from the original images. The difficulty here is how to extract from an image a representative and meaningful feature set while keeping it small.

Another important point to consider is which are actually the database images and query images to work with. Maybe some assumptions can be done over them, like which will be their size or format. Furthermore, it also matters if the query images are supposed to be compared fully with the database images, or only some subparts of them. This would involve the possibility of having to apply transformations like translation, scaling or rotation.

Recapping, when approaching a CBIR problem it has to be determined what strategy to follow in regard to these steps:

* Database images and query images preprocessing
* Feature extraction of the images
* Similarity comparison between the extracted feature sets

The next sections describe these steps in detail for the case of our project, which assumptions and simplifications were made, algorithms that were used and own implementations.

# Related work

Jon

# Earth Mover’s Distance

EMD is a well known similarity measure commonly used in computer vision and other areas.

# Project Description

Our project aims to solve the CBIR problem with shape matching techniques, mainly using EMD as a similarity measure between representative weighted point sets extracted from the images.

We will have a database with images clustered by object class (i.e. trees, birds, cars, etc.). Our system will be able to receive a query image and try to guess its objects class, by simply computing the similarity between the query image and all the images in the database, and by finding in which cluster the query image has more similar images. We will work with different datasets in order to compare the results, from images that represent simple shapes in black and white to real pictures (NOTE: object class is not currently implemented and we may use this idea of classifying images by category as a kind of evaluation metric.).

As said before, we don’t compute the EMD directly on images; first we will go over a feature extraction step to extract the more meaningful information of the images and store it as the so-called “signatures”, that are actually weighted point sets. Thus, we will have a data base of signatures, and when receiving a new query image we will generate its signature too. At this point, we will be able to calculate the EMD between the signature of the query image and all the signatures in the database, establishing a ranking ordered by these EMDs (the lower the EMD, the more similar the signatures, and supposedly the images). Then, we will be able to answer which are the top N most similar images to the query image.

We will assume the query images to be of a similar size of the database images. We will suppose too that the represented objects will be approximately centered and occupying most of the space of the images.

The next section describes what algorithms were used for the feature extraction step.

# Signature Generation

The generation of good quality signatures is the key point in order to obtain good results. In our shape matching approach, we will only consider the distinguishable shapes of the images to generate the signatures, instead of taking into account other parameters like color regions or textures. The resulting signatures will be weighted point sets, being the points positioned in the coordinates where the original image has something that could define a shape, like for example, edges, corners, borders, or contours. These will be usually detected through a sharp change of color.

We present two methods two obtain these weighted point sets, although each of the methods can produce a big variety of different signatures depending on the configuration of the parameters of the many algorithms used. In general, the second method can produce better quality signatures, but then the number of points in the generated weighted point sets is bigger and the EMD algorithm takes too long. This is the reason why most of the testing was done using the first method.

## Canny-MinEigenVal method

Here the signature generation process is inspired by the approach of logo matching in [1]. The process is outlined in Algorithm 1. The resulting weighted point set will be what we defined as a signature. To detect edges, corners and computation of the intensity gradient we will use a C++ library for computer vision called OpenCV (<http://opencv.org/>). This library also offers a function to calculate the EMD between signatures, but as there are several implementations on the web for computing it and we wanted to be able to learn, tweak and test the algorithm, we decided to use one of these implementations (<http://vision.stanford.edu/~rubner/emd/default.htm>).

**Algorithm 1** Generate image signature

*pointSet* ← ∅

Find *edges* in image using Canny edge detector (Figure 1)

Find *corners* in image using MinEigenVal corner detector (Figure 2)

Generate X partial derivative of intensity image using Sobel operator

Generate Y partial derivative of intensity image using Sobel operator

Combine X and Y partial derivatives to form intensity gradient magnitude image (Figure 3)

**for** *corner* in *corners* **do**

**if** *corner* is on an *edge* ∈ *edges* **then**

Add point of *corner* to *pointSet*, with weight equal to value of intensity of point on gradient image (Figure 4)

**end if**

**end for**

**return** *pointSet*

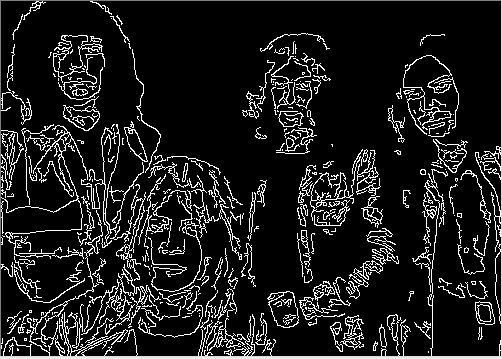
Figure 1: Output of Canny edge detector - edges are marked with a white outline

Figure 2: MinEigenVal Corner Detection - Detected corners are marked with blue

circles

Figure 3: Image of the magnitude of the intensity gradient, calculated using the Sobel operator

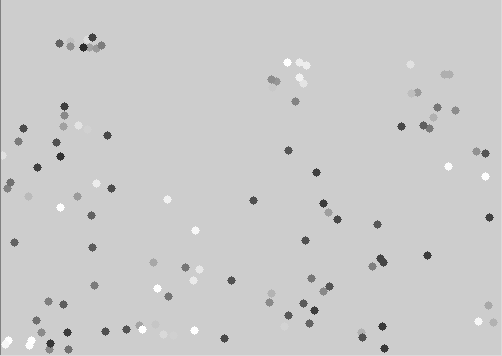
Figure 4: Each corner that is on a detected edge is given a weight, corresponding to the intensity gradient value at the same location, giving us a weighted point set. Low weights are black, high weights are white.

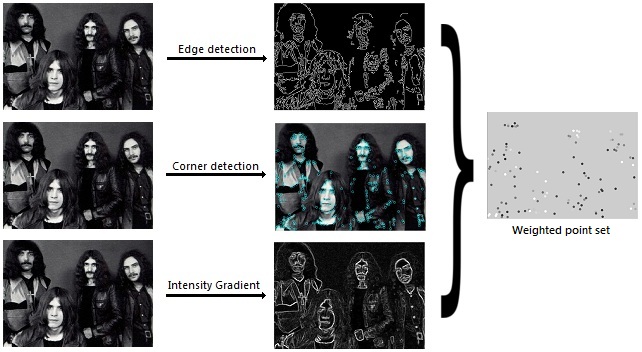
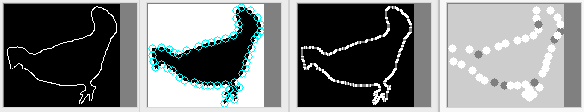
Figure 5: Scheme of the described signature creation method

Figure 6: Detected edges, detected corners, intensity gradient magnitude, and weighted point set for a simple shape (solid black shape on white background)

Could paste here the functions headers (or is it called prototypes?) and a short explanation of the important parameters... ?????

void Canny(InputArray image, OutputArray edges, double threshold1, double threshold2, int apertureSize=3, bool L2gradient=false )

void goodFeaturesToTrack(InputArray image, OutputArray corners, int maxCorners, double qualityLevel, double minDistance, InputArray mask=noArray(), int blockSize=3, bool useHarrisDetector=false, double k=0.04)

void Sobel(InputArray src, OutputArray dst, int ddepth, int dx, int dy, int ksize=3, double scale=1, double delta=0, int borderType=BORDER\_DEFAULT )

More detailed explanation of our adjustment parameters perhaps in Jon's sections??

// Canny parameters (edge detection)

double threshold\_low = 50.0;

double threshold\_high = 150.0;

// MinEigenVal parameters (corner detection)

int max\_corners = 500;

double quality\_level = 0.04;

double min\_distance = 5.0; <----- We could define this considering the image area

// Sobel parameters (intensity gradient)

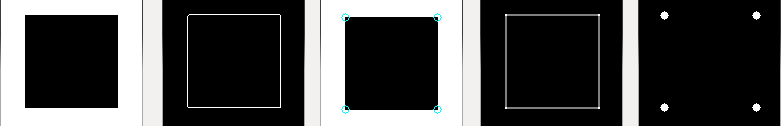
int scale = 1;

int delta = 0;

The Canny's edge detection algorithm can produce different outputs given an input image depending on many other input parameters used only for its adjustment, and the same applies for the other algorithms.

With the testing of this approach, we could adjust the parameters of Canny's edge detection and the Sobel operator enough so that they produce good outputs with almost any input image.

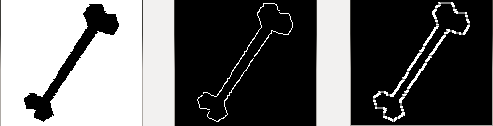
In the other hand, the corner detection algorithm can be inappropriate in some cases, even with the right adjustment parameters. In Figure 7 we can see an example of this: imagine our input image is a polygonal shape like a big square, then the corners detected will ideally correspond to the vertexes. This is correct, but then our final weighted point set will have only points over the vertexes, which is not enough representative of the original shape.

Figure 7: Example showing a weakness of this method. From left to right, original image (black square over white background), detected edges, detected corners, intensity gradient magnitude, and resulting weighted point set.

This observations lead us to the method described next.

## CannyOnly

Considering what was mentioned above, in this method we keep using the results of Canny's edge detection and the Sobel operator, to get the detected edges and the intensity gradient, respectively. We also keep using these algorithms from OpenCV. In Figure 8 a simple shape with its detected edges and intensity gradient can be seen, this would be the same in the previous method.

Figure 8: Original image, simple shape of a bone (left), its detected edges with Canny's edge detection (middle), intensity gradient image through the Sobel operator (right)

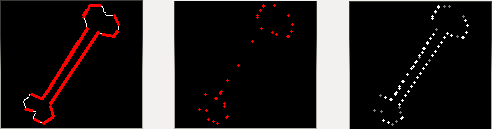
The novelty here is where actually the points come from, as we are not using any corner detection algorithm anymore. The approach now is to translate the output of Canny's edge detection to a point set. To do so, we use a line detection algorithm that OpenCV also provides, called hough lines probabilistic.

Could paste the header and explain here the parameters...

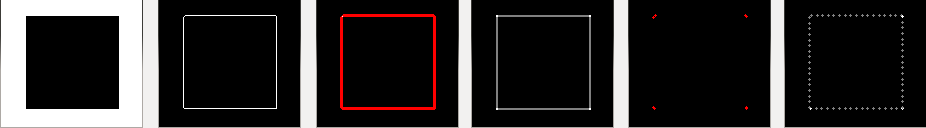
void HoughLinesP(InputArray image, OutputArray lines, double rho, double theta, int threshold, double minLineLength=0, double maxLineGap=0 )

This algorithm will detect straight line segments over the output of Canny's edge detection. In Figure 9 (in the left side) we can see the detected line segments marked in red over the outputted edges shown in Figure 8. The segments are actually defined through their source and target points; this reduced point set can be observed in Figure 9 as well (in the middle). Once this point set is obtained, some more points have to be added to it in order to represent well the segments. These extra points are added over the line segments using a “recursive middle point” algorithm. If a segment is long enough, its corresponding middle point is added to the point set, thus creating two new segments (from the source point to the middle point, and from the middle point to the target point). The same is applied to the newly created segments recursively until there are only small enough segments.

The resulting point set will be our final point set, and the weights will be assigned following the same strategy as in the first method: take the intensity gradient value in the corresponding coordinates of every point. In Figure 9 this final weighted point set can be observed (right side). Note the increase of points in respect to the reduced point set that only contained the source and target points of the segments (middle of Figure 9).

Figure 9: Detected straight line segments over the Canny's outputted edges (Figure 8 in the middle) using the hough lines probabilistic algorithm (left), source and target points of these lines (middle), resulting weighted point set (right)

Note that the problem described in Figure 7 with the previous method is solved now; in Figure 10 (right side) can be observed the resulting point set for the same simple square shape.

Figure 10: From left to right, original image (black square over white background), detected edges, detected line segments, intensity gradient magnitude, source and target points of the detected line segments, and the resulting weighted point set.

# Datasets

It is necessary to test how our system behaves with different datasets. Our project mainly uses shape matching techniques to solve the CBIR problem, rather than color or texture considerations. The first idea was to find an image data set with simple shape forms, and later extend the testing with real picture data sets. It was also of our interest to find data sets classified by category (to allow for the evaluation of our results using the correct guessing of object class as a kind of metric).

Responding to our needs, the following datasets were found and tested:

* Simple shapes (216 images): Different simple objects represented as black shapes over white backgrounds. In average, the images are about 100x100 pixels and weight over 2 KB. There are many images representing the same type of object class, so this dataset can be used with the purpose of category guessing.

<http://www.lems.brown.edu/vision/researchAreas/SIID/>

* Buffy data set s5e6 (52 images): Real pictures corresponding to frames of the TV show “Buffy the vampire slayer”. The size of the images is about 700x400 pixels, and the weight around 25KB. There are many pairs of two similar images. In most of the images there are persons.

<http://www.robots.ox.ac.uk/~vgg/data/buffy_pose_classes/>

* 101 object categories (9197 images): Huge dataset with real pictures representing a big variety of objects. The pictures are of many different sizes. Like the first database, the same object category has many occurrences, so this dataset can be also used with the purpose of category guessing.

<http://pascallin.ecs.soton.ac.uk/challenges/VOC/databases.html>

(I'm not adding info about the image format because I see now you converted some of them to JPG, is that done automatically now?)

# Results

Jon

# Evaluation? (this needs coding)

Jon

# Parameter Tuning? (this maybe needs coding)

Jon

# Performance? (this needs coding)

Jon

# Visualizing with Multidimensional Scaling

Jon

# Difficulties encountered

M&J

# Conclusions

M&J

(pasted from my email)

For "large" databases (>500 images) and not specially good signatures (point sets between 100 - 300 points) EMD is too slow. Not to talk about better quality signatures (500 - 1000 points), it gets really slow.

In general I don't think it matters that it needs some time to generate the signature database or the distance matrix, as long as later the queries are fast enough.

The way we generate the feature sets might be good enough for strictly the shape matching problem, but when it comes to real pictures probably it would be better to consider colors, textures or other properties, not only shapes. This way, maybe smaller feature sets, of 50-100 points, would be enough significant. And if we think of feature sets consisting of other properties, there might be other algorithms (perhaps less generic) that perform better than the EMD.

# Further Work

M&J

(Jon's last list with some added things:)

* Data collection and analysis from our shape matching algorithm, and tuning of parameters (DONE?)
* Performance analysis (DONE?)
* Optimize for larger data set (think about a better way for indexing the data base) (Efficient querying) (TRIED?)
* Build classified item database (DONE?)
* Consider transformations. At first we could start with rigid motions (translate, scale), implementing several approximation algorithms and heuristics, and later we could continue with scaling, maybe using SIFT (<http://www.aishack.in/2010/05/sift-scale-invariant-feature-transform/1/>). Right now we used no CGAL, but maybe if we consider transformations the CGAL arrangements would be useful.
* Compare with other implementations of EMD?
* Other methods of generating signatures? E.g. Different kinds of features (SEMIDONE, “onlyCanny” approach, which could maybe be improved using findContours)
* Deal with types of data other than images, i.e. 3D models

# References

[1] P. Giannopoulous. *Geometric matching of weighted point sets*. PhD thesis,

Universiteit Utrecht, Insitute of Information and Computer Science, 2005.

# TO DO (maybe)

* Prepare a nicer user interface (Qt?)

# Pano's suggested sections:

* The problem you worked on.
* Motivation.
* Algorithms that you've used to solve the problem, in particular the ones you've implemented.
* Which parts of CGAL you've used.
* Your solutions (experiments etc.): what worked well, what didn't and why.
* Problems that you've encountered with CGAL (if any).
* Citations of everything you've used of course.
* Future extensions: things you'd have liked to do but had no time, etc.