Algorithms to create Image Montages

Michael Troebs - m.troebs@gmx.de

May 30, 2000

This is a small collection of ideas - a draft.
Comments are welcome.

1 Introduction

Photomosaic^{TM}[1], photo montage[2] or photo-tiled mosaic are all used to describe the same kind of image: an image, that is composed of a series of smaller images.

2 Metric

To decide, what montage fits a source image best, one needs to measure the distance between the two. It is not obvious, what a good definiton of this distance is. A sensible one, which is described below, is used by the program juggle[3]. If the reader knows of a definition of distance, that matches human perception better than the one described here, the author would be happy to hear about it.

A monitor or TV screen uses the Red-Green-Blue (RGB) colour model, to represent colours. A triplet of integers (Red, Green, Blue) is attributed to every colour. With 24 bits resolution, the amount of each of the three components RGB is coded as an integer in the range from 0 to 255. (0,0,0) corresponds to black, (255,255,255) to white, (255,0,0) represents red and so on.

If there was only one number attributed to each colour as it is in greyscale images, one could use the difference between two numbers as the distance of the colours they represent.

Taking all three numbers into account, one can imagine every colour (R,G,B) to be a point in a three dimensional space. Black (0,0,0) would be found at the origin, white (255,255,255) at the upper right back end of the "colour cube". Pure red would be on the x-axis (or better to say "R-axis"), green on the y-axis and so on.

One possibility would be to use the Euklidean distance

$$\sqrt{(R_1 - R_2)^2 + (G_1 - G_2)^2 + (B_1 - B_2)^2}$$

to describe the distance between two colours (R_1, G_1, B_1) and (R_2, G_2, B_2). This is not the way that humans perceive colour differences though, as is described in [4] by Riemersma. The algorithm he suggests is used in juggle to measure the distance between two colours. For two colours $(R_1, G_1, B_1))$ and (R_2, G_2, B_2) the colour distance Δ is calculated as

$$\Delta = \sqrt{\left(2 + \frac{\overline{r}}{256}\right)(\Delta R)^2 + 4(\Delta G)^2 + \left(2 + \frac{255 - \overline{r}}{256}\right)(\Delta B)^2}$$

with

$$\begin{aligned} \overline{r} &= \frac{R_1 + R_2}{2} \\ \Delta R &= R_1 - R_2 \\ \Delta G &= G_1 - G_2 \\ \Delta B &= B_1 - B_2 \end{aligned}$$

Juggle calculates the distance of one tile to the appropriate spot of the source image by adding up the squares of the distances of every subtile to the spot it is representing.

The total distance of the montage to the source image is the sum of the tile-distances. Looking at the source code and being very accurate, one notes, that juggle actually does not compute distances, it sticks with squares of distances, since calculating the square root of numbers does not change their relative order, it "only" takes time.

3 The optimal montage

There be a database and a source image given. The database shall consist of t tiles, with some of them to be placed in m positions on the source image.

It is obvious that there exists at least one solution to this problem, that is equally good or better than all other solutions. There might be more than one such solution.

Without additional constrains finding the best mosaic is easy: For every spot on the montage, one chooses the tile with the smallest distance to this spot. That way one ends up with a montage with the smallest sum of differences between tiles and original image. The drawback of this method is though, that it is likely that a whole area of the montage is covered by just one tile.

The reason that the algorithm can find the best montage so easily is that the problem can be divided into a set of subproblems that can be solved individually!

The runtime performance scales linearly with both t and m.

4 Montages with a minimum distance constrain

To avoid this clustering of the same tile one can specify the additional constrain of a minimal distance between occurances of the same tile. If this minimum distance technique is used then care must be taken to ensure that there are enough tile images (of the right colour composition) in the database or that that the minimum tile distance is not set too high, otherwise the algorithm will need further rules to tell it what to do when your choice of minimum distance is so great that is leaves "holes" in the final image that cannot be filled.

4.1 Brute force runtime performance

One way to find the optimal montage of a given source image is to check every possible montage against that image and use the one that fits best in terms of the chosen metric.

It is easily seen that t^m different possible montages to build a montage of size m can be built from a database of t tiles:

Beginning with the trivial case of building a montage of size 1 from a database of t tiles there are obviously t possibilities. If in a montage with two tiles the first tile is chosen, there are t possibilities for the second one. Since the first tile can be one of t, there are t^2 possibilities.

One can reduce the number of montages to check by only considering those, that obey the mimimum distance restriction, instead of discarding them later.

4.2 Real-world example

For a database containing $t = 10$ tiles and a mosaic that consists of $m = 4$ tiles, there are $10^4 = 10000$ possible montages to be build, their colour distances to the original to be calculated and the best montage chosen.

For a more realistic example of a database containing $t = 1000$ tiles and a montage with $m = 400$ tiles this would be 1000^{400} possible montages to be build and compared. It is

$$1000^{400} = \left(10^3\right)^{400} = 10^{1200},$$

that is 1 followed by 1200 zeros in decimal notation.

Given a real-world montage problem with a minimum distance contrain, it is impossible to find the optimal solution (if it exits) using a brute force algorithm.

5 Approximations

Since it is impossible to find an optimal solution using the brute force algorithm described above, one tries to find approximations to an optimal solution. Two different algorithms are described in this section.

5.1 Position-based best fit

The task of finding a good fitting montage is divided into the sub-tasks of finding good fits for each part of the target image. For every spot on the target image, the distance of that part to every tile is calculated. For every spot in the map, the list of fitting value and tile number pairs is sorted in order of best fit. The resulting data structure is a vector (index runs from 0 to $m - 1$) containing vectors of number pairs (index runs from 0 to $t - 1$). It was named QoF which is short for Quality of Fit. An index vector with m entries, containing integers in the range 0 to $t - 1$ is created. Each entry points to the best fit in QoF that is not already placed on the map. This index vector is therefor initialised with 0 entries.

All the number pairs, the index vector points to, are sorted by distance to the original picture. The worst fit is chosen[1], the position in the map is stored and the minimum distance constrain is checked. The reader shall note, that this worst fit is the worst of all best fitting tiles. If the minimum distance constrain is obeyed, the tile is placed on the map. If occurances of the same tile are placed too closely, the index vector points to the next best fitting tile in the temporalily stored position.

This procedure is reapeated until the montage is complete or there is a position where there are no more tiles to point to. In the latter case, juggle reports an error message.

5.2 Linear best fit

Instead of sorting m lists with t entries each, as is done in the position-based best fit one can group these $m \cdot t$ tiles in one big list and sort this list. One could create a datastructure where each element in the list holds three entries:

- the quality of fit
- the position on the map
- the tilenumber

When placing tiles on the montage, one would start from the top of the sorted list and go down, looking for positions that are not occupied by tiles. If two occurances of the same tile were too close, the tile would be skipped.

References

[1] Robert Silvers; http://www.photomosaic.com

[2] Jordan Husney; Linux Image Montage Project (LIMP) http://linux.remotepoint.com

[3] juggle http://www.stud.uni-hannover.de/~michaelt/juggle/

[1]A variation is to choose the best fit.

[4] Thiadmer Riemersma, ITB CompuPhase, 1997-1999, The Netherlands http://www.compuphase.com/cmetric.htm

I thank Richard Forster for comments, discussions and help.