Square Packing as Algorithmic Art

Krishnendu Saha
11CS30047

Under Guidance of
Prof. Partha Bhowmick

Flow of Topics

- Introduction to Algorithmic Art
- Packing Algorithms
- Square Tiling
  - Clustering and segmentation
  - Distance Transform
  - Square Fitting
- Results
- Conclusion

Objectives :

- Design a square packing algorithm for irregular bounded region.

- Create tiled representation of any coloured image using the packing algorithm with monochromatic square tiles of different fixed denomination sets.

Introduction to Algorithmic Art

- Algorithmic Art: Aesthetically appealing design considered as algorithmic art, if the features of the design are only determined by an algorithm without or negligible human interference.
- Historically manual algorithms or mathematical techniques are used to create art. For example, *Islamic geometric patterns, Italian Renaissance paintings* uses linear perspective and proportion.
- First computer-generated algorithmic art were created by Georg Nees, Frieder Nake, A. Michael Noll, Manfred Mohr and Vera Molnár in the early 1960s using computer driven plotter.
- With increasing accessablity of computer more mathematically intensive complex form of art works are explored by algorithmic artist , also called algorist.
- At SIGGRAPH in 1995 ,a panel called titled "Art and Algorithms", established identity of such algorists.
- Some popular form of visual algorithmic art are fractal art, cellular automata based art ,genetic or evolutionary art.

Packing Problem

- Packing is covering a bounded region with non-overlapping elementary shapes .
- Packing problems are *optimization problems* ,where optimization criterion is fraction of area covered of the total bounded region, known as *packing density ratio*.
- Perfect Packing is *NP-complete problem*. Perfect packing algorithms are known some simple cases where container regions are regular .
- An example of perfect packing problem is tessalation, i.e perfect tiling of Euclidean plane .
- Some practical packing problem examples are stock cutting, VLSI chip design, advertisement placement, scheduling.

Square Tiling Algorithm

- Input: A coloured image, A set of denominations of square size
- Output: A set of squares each with size, position, axis alignment and colour information
- Objectives :
  - Packing Density Ratio should be high .
  - Number of square tiles used should be less.
  - The tiled image should capture significant features of the actual image
- Steps:
  - Clustering and Segmentation
  - Distance Transform
  - Square Fitting

Clustering and Segmentation

- Objective: Group the image pixels based on their colour and position similarity so that a single coloured tile may be fitted within cluster.
- Clustering is done by meanshift ,a discontunity preserving clustering algorithm.
- Clustering is applied on a d=p+2 dimensional space, p=3 for colour images, on a joint spatial-range domain.
- Segmentation is done by separating two clusters with a line in between.
- To remove noisy spurs morphological cleaning and skeletonization is applied on the segmentation line image.

Meanshift

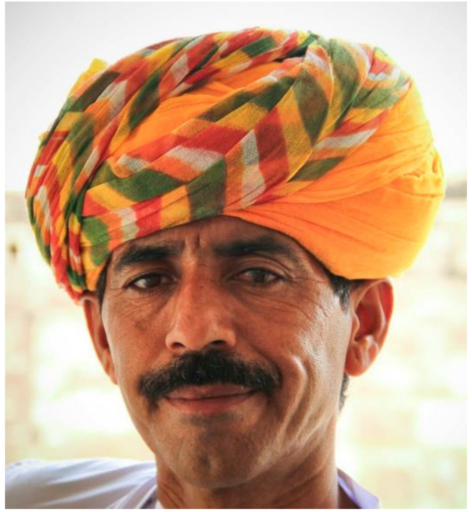
- This algorithm takes a radially symmetric kernel $K(x) = c_{k,d}k(\|x\|^2)$ where $c_{k,d}$ is a normalization function and $k(x)$ is monotonically decreasing function.
- Normal distribution is used as a kernel function.
- For each pixel kernel density is obtained as $f(x) = \frac{1}{nh^d}\sum_{i=1}^{n} K(\frac{x-x_i}{h})$
- The modes of the density function are located at the zeros of the gradient function $\nabla f(x) = 0$
- $\nabla f(x) = \frac{2c_{k,d}}{nh^{d+2}}\left[\sum_{i=1}^{n} g(\left\|\frac{x-x_i}{h}\right\|^2)\right]\left[\frac{\sum_{i=1}^{n} x_i g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)}{\sum_{i=1}^{n} g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)} - x\right]$,where $g(x) = -k'(x)$
- The second term $m_{h,g}(x) = \frac{\sum_{i=1}^{n} x_i g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)}{\sum_{i=1}^{n} g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)} - x$, is the mean shift
- The mean shift procedure, obtained by successive
  - computation of the mean shift vector $m_h(x_t)$,
  - translation of the window $x_{t+1} = x_t + m_h(x_t)$
- The converging is slow as the gradient decays. Hence a threshold parameter is kept.

Mean Shift : Results with different threshold

- Mean shift convergence rate decays as every pixel moves closer to its cluster mean . Hence total convergence takes too much time .
- Number of clusters increase as the threshold decreased .

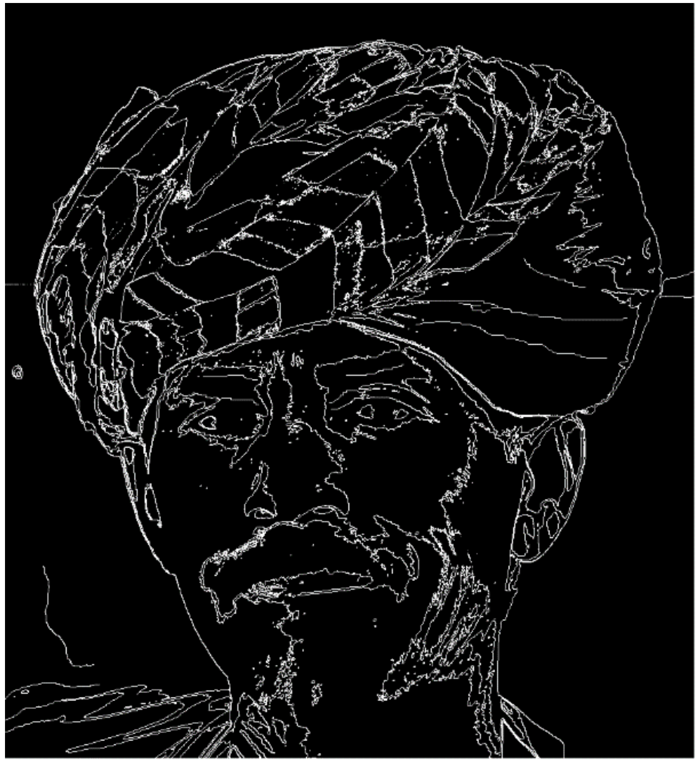
Thr =0.2
Clusters=20

Thr =0.1
Clusters=55

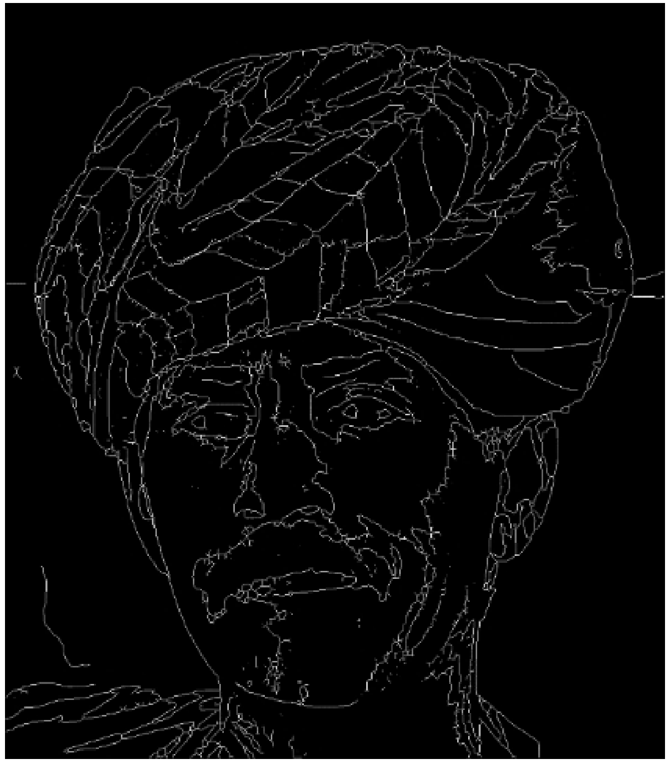


Segmentation and Morphological cleaning:

- The segmentation process gives a lot of noisy spurs , to reduce those morphological cleaning and skeletonization are done.



Segmentation Outline



Morphologically Cleaned and skeletonized Segmentation Outline

Distance Transform:

- The distance transform map labels each pixel of a image with the nearest distance to the nearest boundary pixel or in our case segmentation outline.
- Distance Transform Metrics:

  (a)*chessboard or 8 connected distance* , $d_8((i,j),(h,k))=max(|i-h|,|j-k|)$,

  (b)*cityblock or 4 connected distance*, $d_4((i,j),(h,k))=|i-h|+|j-k|$,

  (c)*euclidean distance,* calculated as $d_e((i,j),(h,k))=\sqrt[2]{(i-h)^2+(j-k)^2}$

- Distance Map contours:

$\boldsymbol{d_8}$ mapping

4	4	4	4	4	4	4	4	4
4	3	3	3	3	3	3	3	4
4	3	2	2	2	2	2	3	4
4	3	2	1	1	1	2	3	4
4	3	2	1	0	1	2	3	4
4	3	2	1	1	1	2	3	4
4	3	2	2	2	2	2	3	4
4	3	3	3	3	3	3	3	4
4	4	4	4	4	4	4	4	4

$\boldsymbol{d_4}$ mapping

8	7	6	5	4	5	6	7	8
7	6	5	4	3	4	5	6	7
6	5	4	3	2	3	4	5	6
5	4	3	2	1	2	3	4	5
4	3	2	1	0	1	2	3	4
5	4	3	2	1	2	3	4	5
6	5	4	3	2	3	4	5	6
7	6	5	4	3	4	5	6	7
8	7	6	5	4	5	6	7	8

$\boldsymbol{d_e}$ mapping

6	5	4	4	4	4	4	5	6
5	4	4	3	3	3	4	4	5
4	4	3	2	2	2	3	4	4
4	3	2	1	1	1	2	3	4
4	3	2	1	0	1	2	3	4
4	3	2	1	1	1	2	3	4
4	4	3	2	2	2	3	4	4
5	4	4	3	3	3	4	4	5
6	5	4	4	4	4	4	5	6

- This observation helps us hypothesize that d_8 metric distance calculation fits axes aligned squares better and likewise d_4 metric fits $45°$ rotated squares better and euclidean metric fits square of any alignment.

Distance Transform Methods:

- Distance Transform methods are of two basic types:
  - (a)Parallel and (b) Sequential
- In parallel algorithms the mapping is done in a single pass starting from boundary points

  S0 -> L'(S0) = S1-> L'(S1) = S2-> . . . -> L'(Sn-1) = Sn = L(S).

- In sequential algorithms the mapping is done in multiple passes. In the n[th] pass $L_n(i,j)$ is calculated from the previous mapping $L_{n-1}(i-1,j)$, $L_{n-1}(i,j-1)$ etc.
- Parallel algorithm is efficient in calculating *8 connected distance* and 4 connected distance and sequential distance mapping is suitable for euclidean distance calculation.
- Per Erik Daniellsson's 8 SED distance mapping is an efficient euclidean distance transform having error bound of 0.076.
- Parallel mapping have more memory requirement but takes one scan of the image compared to multiple scans in sequential mapping, hence faster.
- Both are O(n) algorithm n being number of pixels.

Algorithm 1: 8 Sequential Euclidean Distance Transform

The picture L is a two-dimensional array with the elements

$L(i,j) \quad 0 \le i \le M-1, 0 \le j \le N-1.$

Each element is a *two-element vector*

$\bar{L}(i,j) = (L_i, L_j)$, Li, Lj being positive integers,

$L(i,j) = (L_i(i,j), L_j(i,j)).$

The size of a vector *L(i,j)* is defined by

$$|L(i,j)| = \sqrt{L_i^2 + L_j^2}$$

8SED:

Initially:

$L(i,j) = (0, 0)$ for $(i,j) \in S$

$L(i,j) = (Z, Z)$ for $(i,j) \in S'$

S here denotes the background and S' the segmentation outline

First picture scan:

for j= 1,2 . . . . . N - 1,

for i = 0,1,2 . . . . . M- 1

$\bar{L}(i,j) = \min(\bar{L}(i,j), \bar{L}(i-1,j-1)+(1,1), \bar{L}(i,j-1)+(0,1), \bar{L}(i+1,j-1)+(1,1))$

for i = 1,2 . . . . . M- 1

$\bar{L}(i,j)=\min(\bar{L}(i,j), \bar{L}(i-1,j)+(1,0))$

for i = M-2,M-3 1,0

$\bar{L}(i,j)=\min(\bar{L}(i,j), \bar{L}(i+1,j)+(1,0))$

Second picture scan:

for j= N-2,N-3, ... 1,0

for i = 0,1,2 . . . . . M- 1

$\bar{L}(i,j) = \min(\bar{L}(i,j), \bar{L}(i-1,j+1)+(1,1), \bar{L}(i,j+1)+(0,1), \bar{L}(i+1,j+1)+(1,1))$

for i = 1,2 . . . . . M- 1

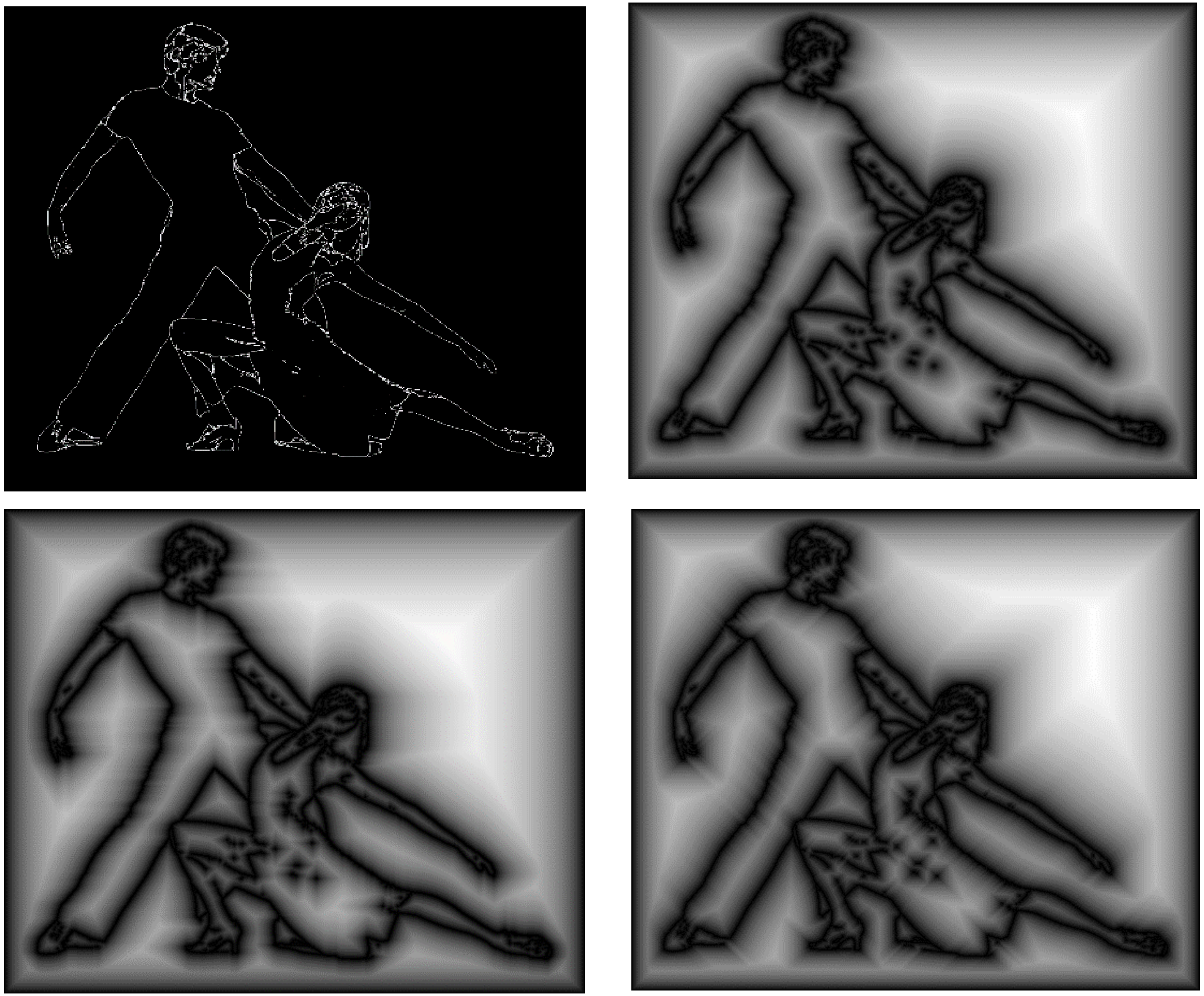
$\bar{L}(i,j)=\min(\bar{L}(i,j), \bar{L}(i-1,j)+(1,0))$

for i =M-2 ,. . . . 1,0

$\bar{L}(i,j)=\min(\bar{L}(i,j), \bar{L}(i+1,j)+(1,0))$

The euclidean distance of any point may be calculated as $\sqrt{L_i^2 + L_j^2}$

Distance Mapping Image:



top left- Segmented Image, top right- d_8 distance transform, bottom left-d_4 distance transform, bottom right- Euclidean distance transform

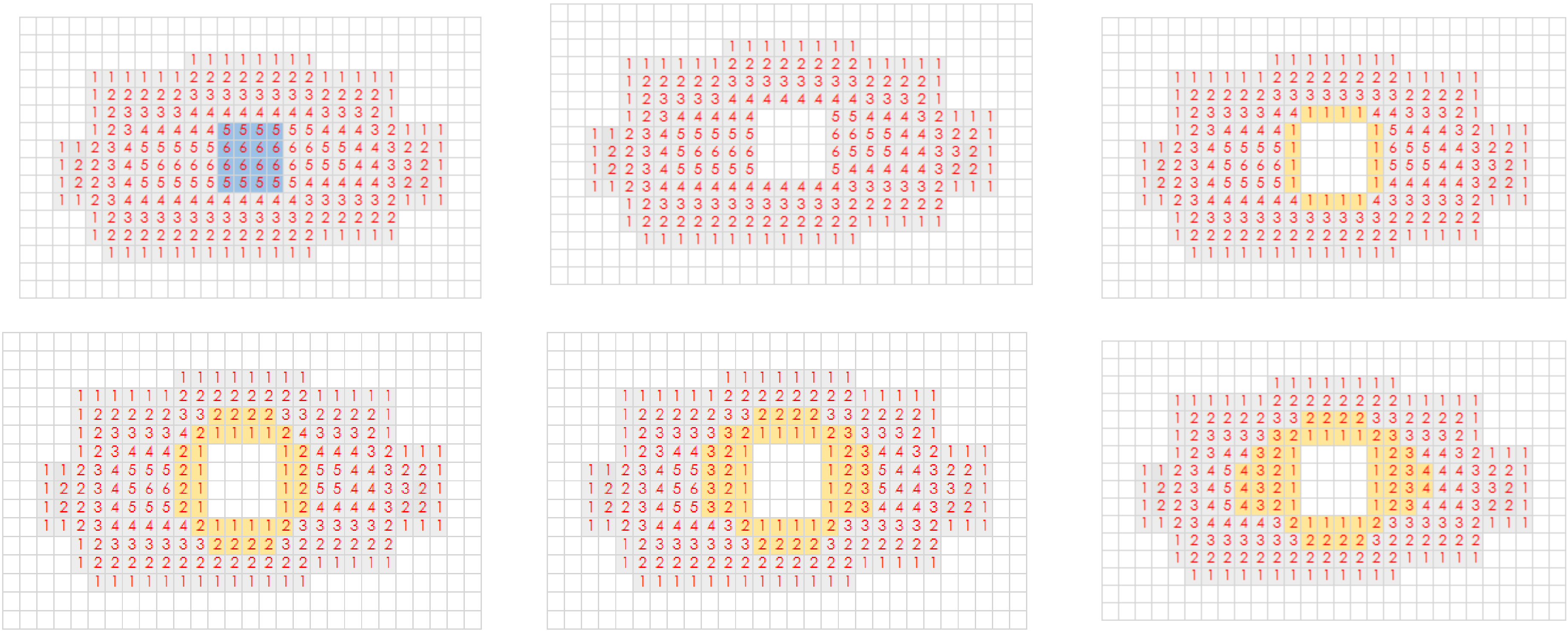
Square Fitting

- The distance map gives an estimate of biggest square fitting possible at any pixel.
- Distance mapping value at pixel ***(i,j)*** denoted by $\boldsymbol{d}$ and largest fitting square size $\boldsymbol{r_l}$ different relations in different metrics.

  $\boldsymbol{r_l(i,j)= d_4(i,j)}$ or $\boldsymbol{r_l=d_e(i,j)}$ but $\boldsymbol{r_l(i,j)= \sqrt{2} \times d_8\,(i,j)}$
- Based on distance mapping, different values to alignment angle $\boldsymbol{\alpha}$ is given. $0°$ in $\boldsymbol{d_4}$,$45°$ in $\boldsymbol{d_8}$ and it is chosen out of a set of angles $(0°, 15°, 30°, 45°, 60°, 75°)$ in case of $\boldsymbol{d_e}$.
- In case of $\boldsymbol{d_e}$ the alignment angle of square may be randomly assigned or based on *best fit angle calculation* heuristic.
- Overlapping square with segmentation outline or already drawn square are discarded.
- The distance recalculation after fitting a square is done locally .
- This distance recalculation may be easily done with parallel distance mapping starting with the new square boundary points.

Local Distance recalculation

- After each square fitting, the distance map changes.
- This may be easily handled by parallel distance mapping (***d_4*** or ***d_8***) taking the square boundary points as the initial set of points in effectively O(1) time complexity.
- The stopping criterion is newly calculated distances being equal to previous distances



Algorithm_2 : Square Packing

```
Input: Image I, Image M, Set D
Output: Set S
begin
    for each d in D
        α ← assign_angle();
        r ← k * d
        {sq_boundary, sq_internal} ← precalculate_square(r,α)
        while M(d) is not null
            P ← M(d).get_first();
            bool f=false;
            for p in sq_boundary
                if (M(P+p) is 0 )
                    f = true;
                end_if
            end_for
            if(f is true or variation_abv_threshold(I,sq_internal,P,threshold))
                continue;
            end_if
            bool f1=draw_square(P,sq_internal);
            if(f1)
                S ← S ∪ Sq(P,I,r,α);
                M({sq_internal})←0;
            distance_recalculation(P,sq_boundary,M);
            end_if
        end_while
    end_for
end
```

Time complexity : Square Fitting

- The algorithm is run sequentially for each of the denomination in the set D and for each denomination d∈D there is a O(1) computation involved in precalculate_square() function.
- The inside while loop draws square for a particular denomination d_i and maximum α_in$/d_i^2$ squares are drawn.

  n= number of pixels in image α_i = fraction of area covered by squares of denomination d_i.
- Drawing a square of denomination d_i takes O(d_i^2) and an additional O(1) for distance_recalculation().
- Hence overall complexity formula is given as,

$$O\ (|D|).O(1) + \sum_i \alpha_i \mathrm{n}/d_i^2 (\ O\left(d_i^2\right) + O(1))$$

$$= O(|D|) + \textstyle\sum_i \alpha_i \mathrm{n} \leq O(|D|) + O(n) \quad \text{, as } \textstyle\sum_i \alpha_i \leq 1$$

- In case of squares discarded due to failing variance check it adds an additional O($\sum_i k_i\, d_i^2$) time complexity where k_i number squares of size d_i are discarded.

Results: Comparison of three distance metric

Image Fig. No.	Size	DM	N_r	D	N_s	Thr	ρ	Cpu Time
Dog (a)	800x600	d_4	17	D1	4210	∞^*	0.8497	8 m16 s
Dog (b)	800x600	d_8	17	D1	3900	∞	0.9204	4 m07 s
Dog (c)	800x600	d_e	17	D1	6537	∞	0.9020	4 m19 s

(a)

(b)

(c)

Results: Comparison of algorithm with or without variance check:

Image Fig. No.	Size	DM	N_r	D	N_s	Thr	ρ	Cpu Time
Salsa (a)	997x1132	d_8	31	D1	5566	∞	0.9517	4 m 6 s
Salsa (b)	997x1132	d_8	31	D1	6176	50	0.9126	9 m 59s

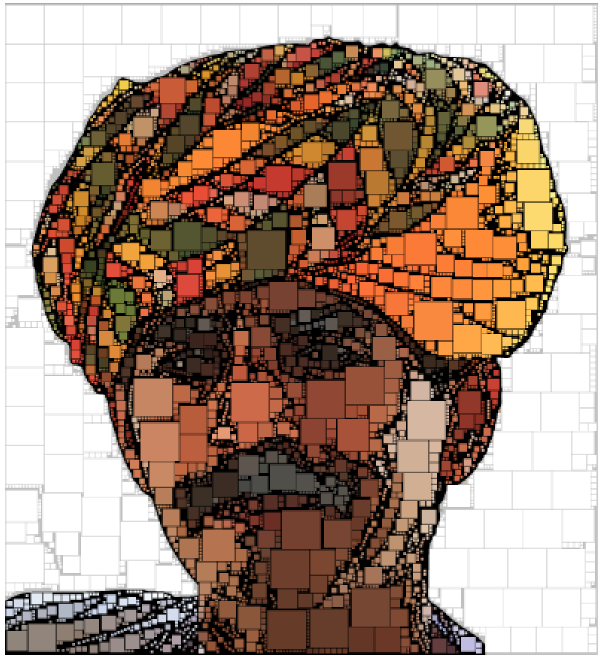
(a)

(b)

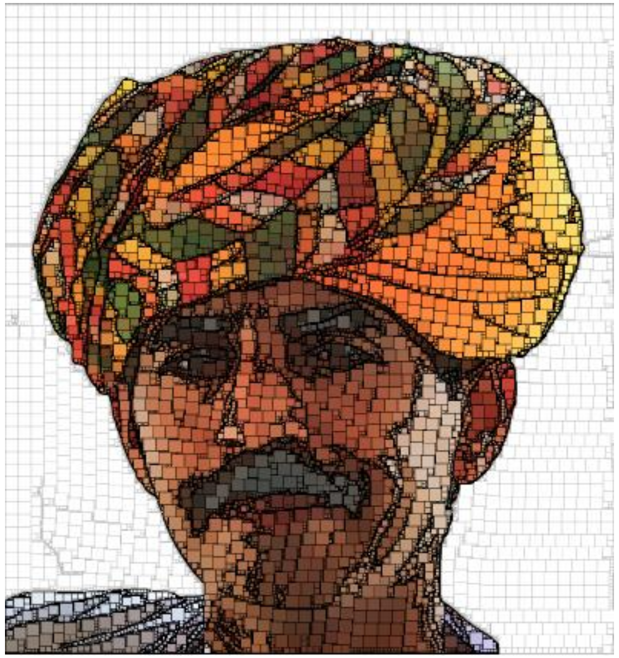
D1=(100,80,60,40,30,25,
20,15,12,10,8,5,3,2)

Results: Comparison of algorithm different sets denomination:

Image Fig. No.	Size	DM	N_r	D	N_s	Thr	ρ	Cpu Time
Turban (a)	896x982	d_8	20	D2	5413	100	0.8775	10m16s
Turban (b)	896x982	d_8	20	D3	12082	100	0.8523	17m45s



(a)



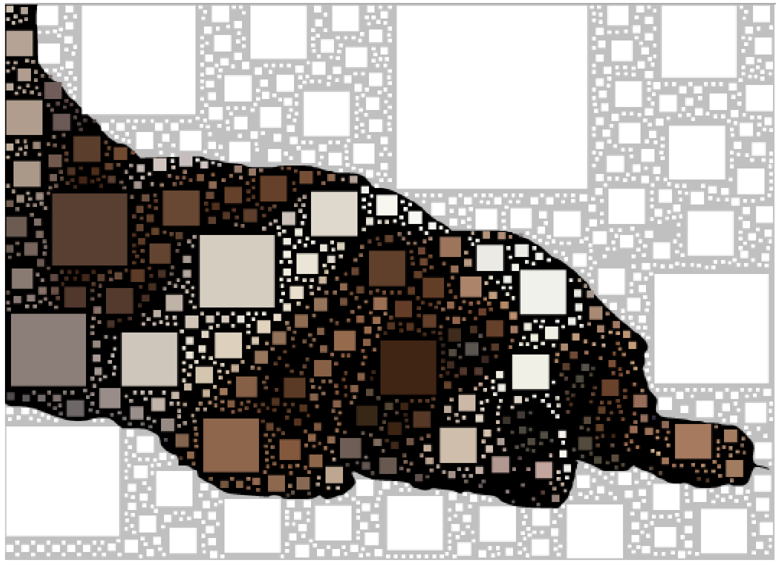
(b)

D2=(30,25,20,16,12,10,5,3,2),
D3=(10,8,7,5,3,2)

Results: Comparison of d_4 and d_8 as local distance recalculation metric:

Image Fig. No.	Size	DM	N_r	D	N_s	Thr	ρ	Cpu Time
Dog (a)	800x600	d_8 ($DR=d_4$)	17	D1	3900	∞	0.9204	4 m07 s
Dog (a)	800x600	d_8 ($DR=d_8$)	17	D1	2153	∞	0.7104	20m15s

(a)



(b)

Conclusion:

- From the results we see that d_8 distance mapping gives the best packing and d_4 as local distance recalculation metric.
- The packing algorithm gives as high as 95% packing in some case .The segmentation boundary points and their adjacent pixels are mostly left.
- If the cluster points are completely separated then square fitting may be done in parallel for each of the segment reducing time requirement .

Bibliography

[1] R. Verostko, “http://www.verostko.com/algorithm.html”.

[2] “https://en.wikipedia.org/wiki/Algorithmic_art”.

[3] “http://mathworld.wolfram.com/PenroseTiles.html”.

[4] “http://www2.stetson.edu/~efriedma/cirinsqu/”.

[5] R. Harren, “Approximating the orthogonal knapsack problem for hypercubes,” in International Colloquium on Automata Languages and Programming (ICALP 2006), 2006.

[6] K. Jansen and G. Zhang, “Maximizing the total profit of rectangles packed into a,” in Algorithmica, 2007, pp. 47, 323–342.

[7] D. Kleitman and M. Krieger, “An optimal bound for two dimensional bin packing,” in 16th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1975).

[8] K. Jansen and R. Solis-Oba, “13th International Conference, IPCO 2008 Bertinoro, Italy, May 26-28, 2008 Proceedings,” in Integer Programming and Combinatorial Optimization, 2008, pp. 184-198.

[9] M. M. Paulhus, “An Algorithm for Packing Squares,” Journal of Combinatorial Theory, , no. Series A 82,, 1997.

[10] S. De and P. Bhowmick, “Digital Circlism as Algorithmic Art,” in 9th International Symposium on Visual Computing, Rethymnon, Crete, Greece, 2013.

[11] D. Comaniciu and P. Meer, “Mean Shift: A Robust Approach Toward Feature Space Analysis,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 5, pp. Page 603-619, 2002.

[12] P.-E. Danielsson, “Euclidean distance mapping,” Computer Graphics and Image Processing, vol. 14, no. 3, pp. 227-248, 1980.

Thank You