Combinatorial Pyramids — An Implementation in Matlab

David Pfahler

Technical Report for
Structural Pattern Recognition 2016W

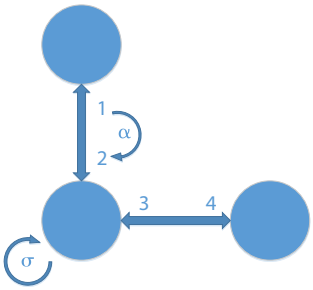


Figure 1: A graph with labeled darts. Table 1 shows the involution α of a dart and the permutation σ.

1 Introduction

1.1 Lecture

I wrote this work for the lecture *Structural Pattern Recognition* in the *WS 2016/17*. The content of the lecture was:

- Relations between patterns in space and time
- Representing structure: strings, arrays, trees, graphs, maps and grammars
- Shape and context, embedding in an n-dimensional space
- Distances on a given structure graph spectra and eccentricity

Table 1: Involution and permutation of the Combinatorial Map (CM) from Fig. 1

	1	**2**	**3**	**4**
α	2	1	4	3
σ	4	2	3	1

- Operations on structures and among structures: recognition, parsing
- Exact and inexact matching
- Modification of structure under motion and tracking
- Topology (nD holes, persistence) applications.

As my lab exercise in structural pattern recognition I chose the topic "*Combinatorial Pyramids*" from the lecture unit 5. To intensify my knowledge from the associated lecture by means of practical examples.

1.2 Problem Specification

To describe the content of an image it is necessary to extract the objects that an image shows. The positions of the identified objects to each other gives important information of the context of the image. An image is most commonly represented by an array of pixels. To understand the context of the image, it needs to be partitioned into regions. The context of this regions is called the Region Adjacency Graph (RAG).

To create the RAG the pixel array of the image needs to be transformed into a graph representation. This is a computational expensive process, because in contrast to the implicit representation in the system memory of a pixel array, a graph representation needs to save pointers for the darts to the nodes (which are the pixels) of the image.

The Combinatorial Map (CM) supports an efficient way to represent the darts in the system memory. Additionally it enables the application of the graph operations to *contract* and *remove* a dart. Table 1 shows the representation of the CM of Fig. 1 in the memory.

2 Development methodology

I separated the development process into four phases.

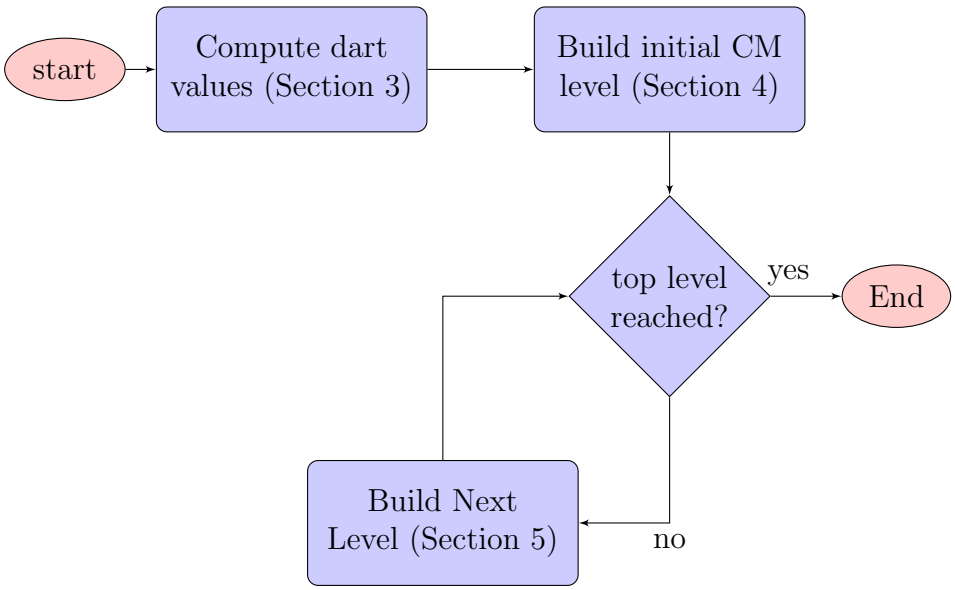


Figure 2: The CP creation process

Phase 1: Existing Work I did some literature research to get familiar with the topic. (Section 2.1)

Phase 2: Meet the Maps I created an `Python` application to explore the properties of an neighborhood graph of an image. (Section 2.2)

Phase 3: Implementation The actual `Matlab` implementation is based on the work of the first two phases. (Sections 3 to 5) The Fig. 2 shows a flowchart of the creation of the CP.

Phase 4: Results In the end I created some results. (Section 6)

2.1 Related work

The work from Torres and Kropatsch [4] presents a technical report about operations on a CP. The operations I used in my work are the removal of a dart and the contraction of a two darts. These operations are presented in detail later.

Fig. 3d shows the creation of a CP with these two operations on a small graph and also the unfolding of the pyramid with the inverse operations. Fig. 3b shows all darts that can be contracted or removed from the input image Fig. 3a. Fig. 3c shows the top of the CP, which is the RAG of the input image.

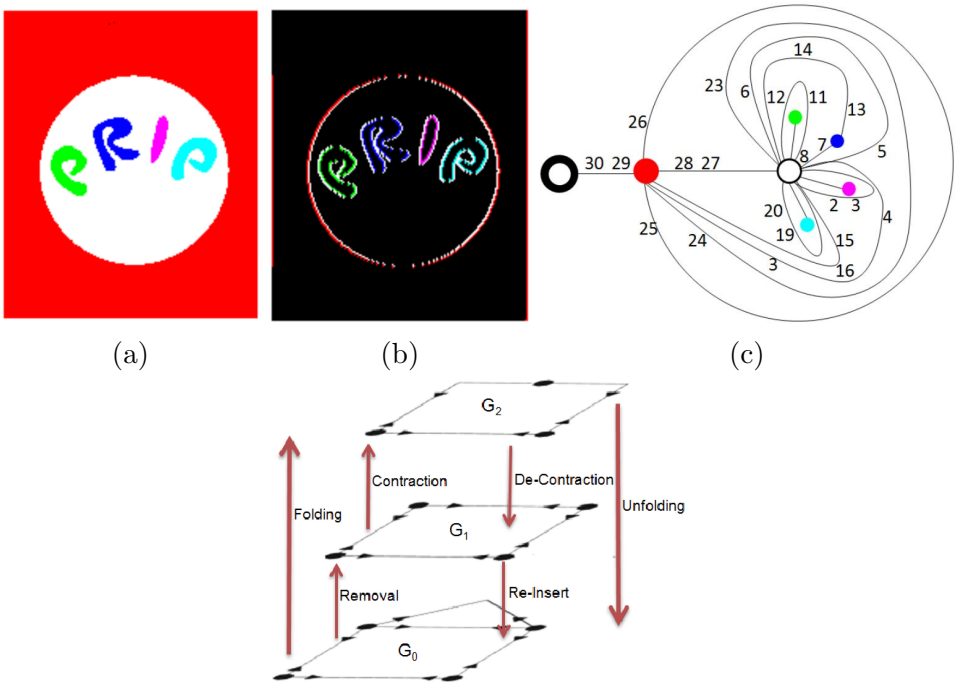


Figure 3: (a) an input image. (b) the segmented image at the first level. (c) the top level of the combinatorial pyramid. (d) Combine and remove operations on a graph

The work from Brun and Kropatsch [2] is an introduction to the CP. It presents background information to the contraction kernel of the CP. Fig. 4 shows the creation of a contraction kernel. From the neighborhood graph of the input image (Fig. 4a) survivor nodes are selected (Fig. 4b). A spanning forest is created, so that every node of the original graph can only be reached from one survivor node (Fig. 4c). Then every created tree is contracted into the surviving node and these nodes are connected again (Fig. 4d). Figs. 4e to 4h show the same steps again for this created new pyramid level.

The further work from Brun and Kropatsch [3] step into detail of the contraction of darts in parallel. Fig. 5 shows the conflict of the contraction in parallel. As one can see, if the two marked red darts are selected for contraction and contracted at the same time, it would be undefined were the dart marked with the red question mark would start and end.

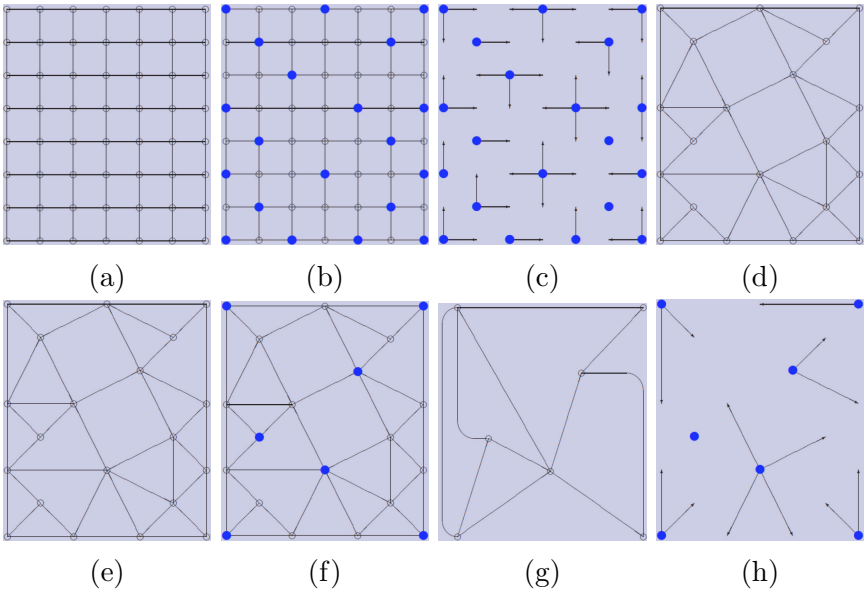


Figure 4: The creation of a contraction kernel

2.2 Python implementation

Listing 1: how to use the created `Python` class

```
1 import CombinatorialMap
2 map = CombinatorialMap()
3 map.setSize(3, 5)
4 map.printNodes()
```

To start working with CP I tried to explore the properties of a CM. This was not as easy as I thought and there are not many implementations of this matter available. So I decided ti implement a simple *Python* class for CM. This class was not for computations and operations on the CM but for the representation and especially the transformation of an image to a CM.

The created class is easy to use as the Listing 1 shows. As already mentioned the class is only a tool to get the representation of an image. In this example the dart indices of a 3×5 image are shown. The output of this function is shown in Listing 2.

Additionally I created functions to receive:

- All darts with a specific direction (N, E, S, W)

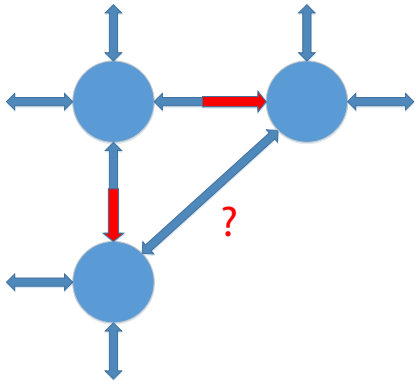


Figure 5: Contracting in parallel is not trivial

Listing 2: The output of the commands of Listing 1

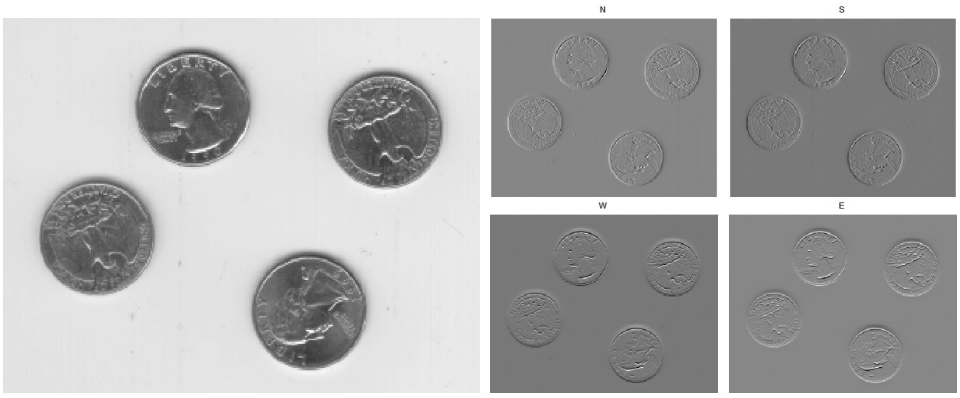
```
nodes:
    O -  5, 2 -   O -  8, 4 -   O - 11, 7 -   O - 13,10 -   O
    |             |             |             |             |
   1,14          3,17          6,21          9,25          12,29
    |             |             |             |             |
    O - 20,16 -   O - 24,19 -   O - 28,23 -   O - 31,27 -   O
    |             |             |             |             |
  15,32         18,34         22,37         26,40         30,43
    |             |             |             |             |
    O - 36,33 -   O - 39,35 -   O - 42,38 -   O - 44,41 -   O
```

- All involutions of specific darts
- Changing the neighborhood function
- The orbit of a node
- The permutation σ and the involution α of a set of darts
- ...

This helped me to understand the dependencies of the indexing of the darts.

3 Computation of the dart values

A dart in an image is a transition from one pixel to another. In my implementation I used the 4 neighborhood of the pixel for the creation of the image graph. The value of a transition should represent the change of the pixel value. So I computed from every pixel the change of the pixel value to



(a) (b)

Figure 6: (a) The gray scale input image (b) the transition values from every pixel to its North, South, West and East pixel neighbor. The displayed pixel value range of the differences is mapped from [-128,128] to [0,255]

every neighbor of their neighborhood (N,E,S,W).

Fig. 6 visualizes the resulting dart values for every cardinal direction.

4 Construction of the initial CM level

I used the `Python` implementation presented in Section 2.2 to understand the indexing of the darts. The following properties of the image need to get calculated to obtain an initial CM:

- The dart indices x of the dart values from the previous section.
- The permutation $\sigma(x)$
- The involution $\alpha(x)$
- The previous dart of the dart index $\rho(x) := \sigma^{-1}(x)$

4.1 Computation of the dart indices

The indexing of the dart values is motivated by the output from Listing 2. As one can see the next index with the same direction is gained by adding 4, if it is inside the image. On the border it is only added by 3, because one dart is missing. Additional similar exceptions occur on the corners.

Table 2: The permutation σ of a 3×3 image

x	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
σ	2	1	5	3	4	7	6	10	8	9	13	14	12	11	16	17	15	19	18	21	22	20	24	23
$x - \sigma$	1	-1	2	-1	-1	1	-1	2	-1	-1	2	2	-1	-3	1	1	-2	1	-1	1	1	-2	1	-1

4.2 Computation of the next index

By looking at the permutation of an image a pattern can be observed. Table 2 shows the permutation σ of a 3×3 image. One can see that for one row within the image the next dart has an image difference of:

```
next_darts_one_row = repmat([2; 2; -1; -3], width-2, 1)
```

Which results in the differences for the whole middle of the image:

```
repmat( ...
  [next_darts_one_row; 1; 1; -2; 2; -1; -1], ...
  height-2, 1)
```

Once again special cases occur for the borders of the image.

4.3 Computation of the other properties

The other properties of the image are easier to calculate:

- The involution α for the indices of the darts that point to the north are the indices that point to south (and vice versa).
- The previous dart ρ of the next dart σ is the dart x itself.

```
cm.involution(N) = S
cm.involution(S) = N
cm.involution(E) = W
cm.involution(W) = E

cm.prev(cm.next) = 1:num_darts
```

5 Construction of the pyramid

After the initial CM is created the map needs to be reduced. The iterative reduction of the CM with the loop described in Fig. 7 the CP is created.

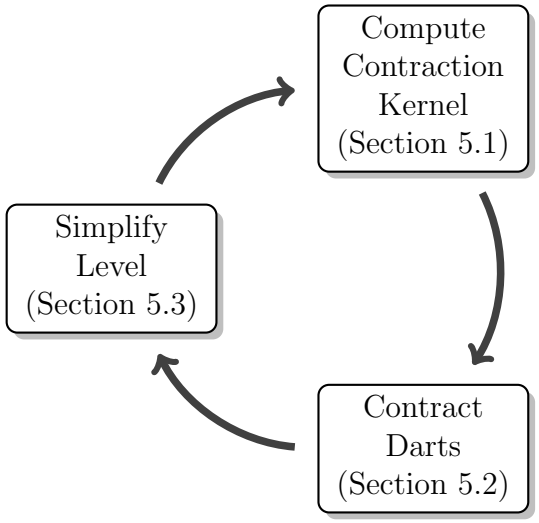


Figure 7: The creation loop of the pyramid

5.1 Computation of the contraction kernel

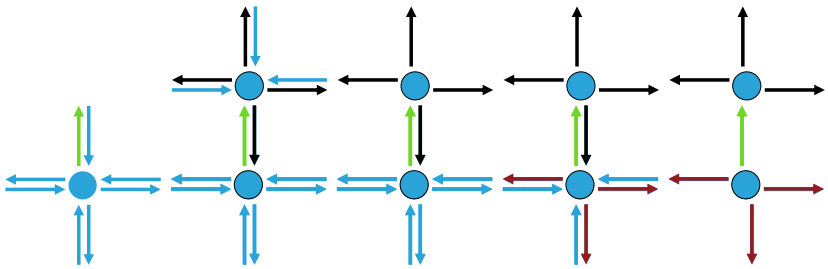
Initially the darts are sorted by its values. This sorting only needs to be done once for the initial CM. For the reduction a contraction kernel needs to be defined (see Section 2.1). I implemented a greedy algorithm to calculate the kernel.

Fig. 8 shows the creation of the kernel.

- Initially add all active darts of the CP-level to the set of valid darts.
- First assign the next best dart of all valid darts for the contraction kernel (Fig. 8a)
- Then get the involution of the dart and the orbit of the involution (Fig. 8b)
- Now remove the involution of the involution orbit from the set of valid darts (Fig. 8c)
- Now get the orbit of the dart (Fig. 8d)
- And remove its involution from the set of valid darts (Fig. 8e)
- Repeat until there are no valid darts left

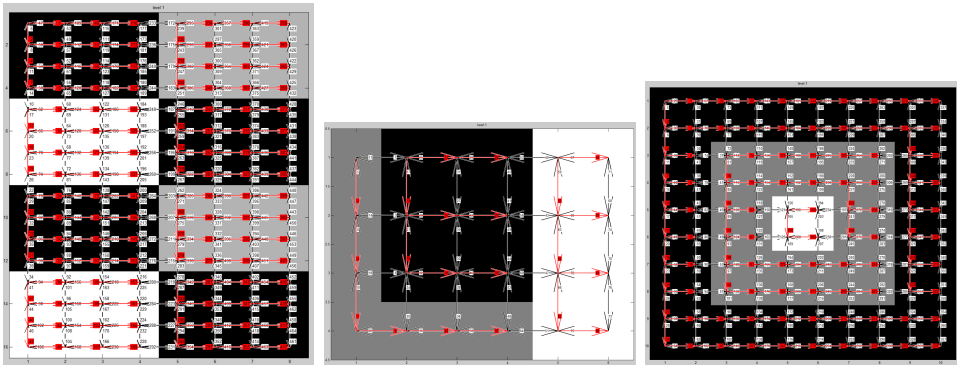
There are two exceptions for the adding of darts to the contraction kernel: *Self loops* and *pending edges*.

A self loop is detected by checking if the orbit of the dart is the same as the orbit of the involution of the dart. The orbit of a pending edge contains only the edge itself.



(a) (b) (c) (d) (e)

Figure 8: (a) Assign next best dart for the contraction kernel (b) Get the involution of the dart and the orbit of the involution (c) now remove the involution of the involution orbit from the set of valid darts (d) now get the orbit of the dart (e) And remove its involution



(a) (b) (c)

Figure 9: The red darts show the contraction kernels for the three images.

The red darts in Fig. 9 show the contraction kernels for three different images.

5.2 Contraction

The contraction of a dart in a CM is calculated with only 4 changes of the CM. The next $\sigma'(\rho(x))$ and previous $\rho'(\sigma(x))$ dart need to get updated for the dart x and its involution $\alpha(x)$. And x and $\alpha(x)$ are removed from the set of active darts.

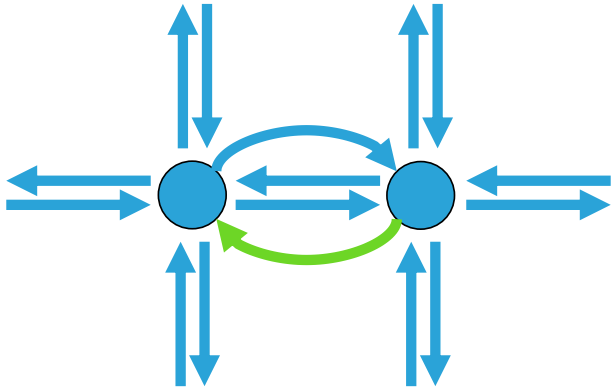


Figure 10: The number of darts of this face is less than 2 and it can be removed

$$\sigma'(\rho(x)) := \sigma(\alpha(x)) \tag{1}$$
$$\sigma'(\rho(\alpha(x))) := \sigma(x) \tag{2}$$
$$\rho'(\sigma(x)) := \rho(\alpha(x)) \tag{3}$$
$$\rho'(\sigma(\alpha(x))) := \rho(x) \tag{4}$$

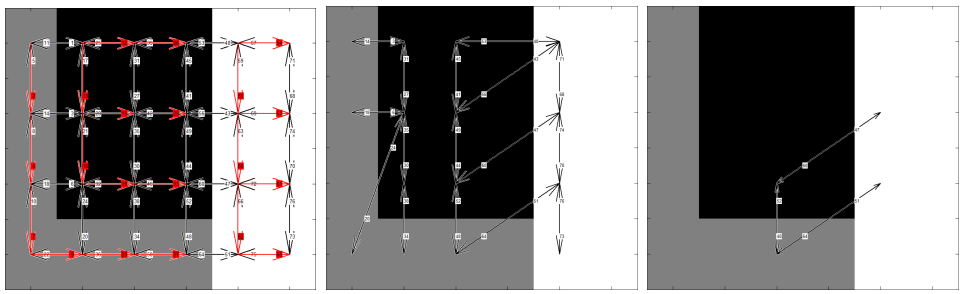
5.3 Simplification

When contracting darts double edges and self-direct-loops are created. By removing these darts the pyramid is easier to read.

For detecting them the face of a dart x is needed:

- Get the involution $\alpha(x)$
- Get the previous of the involution of the dart $\rho(\alpha(x))$
- add this to the face and repeat for this dart until the initial dart is reached

If the number of darts in the face is smaller than 3 the dart can be removed. Fig. 10 shows a configuration of darts that build up a double edge. The face of the green marked dart has two darts, which can be removed. The removal operation is similar to the contract operation:



(a) (b) (c)

Figure 11: (a) the contraction kernel of the input image. (b) the darts after the contraction step (c) the darts after simplification

$$\sigma'(\rho(x)) := \sigma(x) \tag{5}$$
$$\sigma'(\rho(\alpha(x))) := \sigma(\alpha(x)) \tag{6}$$
$$\rho'(\sigma(x)) := \rho(x) \tag{7}$$
$$\rho'(\sigma(\alpha(x))) := \rho(\alpha(x)) \tag{8}$$

If x is equal to $\alpha(x)$ a self-direct-loop is detected. The removal of this has to be treated differently:

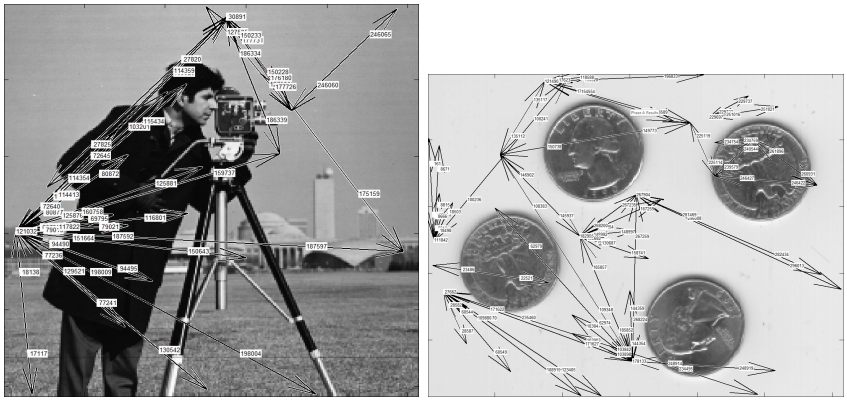
$$\sigma'(\rho(x)) := \sigma(\alpha(x)) \tag{9}$$
$$\tag{10}$$

Fig. 11 shows the simplification after the contraction of the contraction kernel. As one can see the number of darts is reduced significantly.

6 Results

6.1 Performance

The implementation was tested on a Processor Intel (R) Core (TM) i7–3770K CPU @ 3.50GHz, 3501 Mhz, 4 Core (s), 8 Logical Processor (s) with installed 16,0 GB physical memory (RAM).



(a) (b)

Figure 12: Two test images. The shown darts are the darts of the top pyramid level.

Table 3: The performance of the application for two input images

Image	**initial CM**	**Contraction level 1**	**Simplification level 1**	**Total**
Fig. 12a	$0.04s$	$4.73s$	$99.64s$	$114.20s$
Fig. 12b	$0.03s$	$3.38s$	$73.44s$	$89.12s$

The Table 3 shows the computation time of the proposed algorithm. It can be observed that the bottle neck of the implementation is the simplification process.

6.2 Publication

I published my work on the platform `Github`[1]. The repository is called: https://github.com/theShmoo/CombiPyr-ImSeg and it contains:

- The source code of the `Python` implementation (Section 2.2).
- The source code of the `Matlab` implemenation (Sections 3 to 5).
- My presentation of the work, held in the lecture.
- This documentation.

7 Discussion and Conclusion

The proposed algorithm creates a CP, which can be used to extract the RAG of an image. A simple use case would be to implement an image segmentation algorithm with the CP.

To create the segmentation you only need one pixel value at the top dart of the pyramid. The dart values of the contraction kernels represent the changes of this pixel value to the underlying pyramid levels.

References

[1] Github a web-based git or version control repository and internet hosting service. https://github.com/. Accessed: 2017-01-29.

[2] L. Brun and W. Kropatsch. Introduction to combinatorial pyramids. In *Digital and image geometry*, pages 108–128. Springer, 2001.

[3] L. Brun and W. Kropatsch. Contraction kernels and combinatorial maps. *Pattern Recognition Letters*, 24(8):1051–1057, 2003.

[4] F. Torres and W. G. Kropatsch. Canonical encoding of the combinatorial pyramid.