

Image Segmentation

Baptiste GENEST

Felix BARDY

Abstract

In this report we detail how we implemented both Region Growing and Split and Merge algorithms. We will also show how a reformulation of the problem allows for a more guided and richer approach. NOTE: project uses Qt 6.3.2 with measurements compiled with release mode. Open Segmentation.pro in qtcreator. Relative path to images is "../images/" so the executable must be run from the default build directory of qtcreator.

1	Pixel Regions as probability measures	1
2	Region Growing	1
2.1	Region Growing Algorithm	1
2.2	Distances between distributions	2
2.3	Optimal placement of germs	4
2.4	Empirical study of the effect of the threshold on the resulting variance	4
2.5	Results and efficiency	5
2.6	Conclusion	5
3	Split and Merge	5
3.1	Leaf indexing	5
3.2	Split	6
3.3	Merge	6
3.4	Conclusion	6

1 Pixel Regions as probability measures

The key insight we had was that instead of speaking of regions just as a group of pixels, we could rather consider them as probability measures in order to use the tools of probability theory and statistics to explore richer behaviors.

For instance, a region can be then be defined as

$$R = \sum_{c \in \mathcal{C}} p_c \delta_c$$

Where \mathcal{C} is the considered Color Space, which can be $[0, 1]$ for grayscale images, or $[0, 1]^3$ for RGB images, δ_c is the Dirac measure at c , and p_c being the proportion of the color c in the region.

That will to go beyond the reduction to the simple grayscale case will be the main focus of the rest of this part of the report.

2 Region Growing

2.1 Region Growing Algorithm

First of all, algorithmically speaking we use a BFS approach to expand regions efficiently (only considering the borders of each growing regions), combined with a UnionFind structure to allow for fast merging of regions.

In order to ensure full coverage of the picture, if a pixel is touched by a region but isn't merged with it, we place a new germ at that pixel.

2.2 Distances between distributions

The core on the algorithm relies on the two principles: for a given threshold value ε

- a pixel will be added to the region if $d(D, c) < \varepsilon$ where D is the region's distribution and c is the pixel color.
- two regions are merged if $d(D_1, D_2) < \varepsilon$ ¹, where D_1 and D_2 are the associated distributions.

The standard algorithm only uses the grayscale value of a pixel to determine if it should be merged with a region or not, hence the natural quest of trying to see how it would behave with full rgb information led to the following approaches in order to define such distances.

To define such metrics in the most general way we introduce the two following abstract classes:

- **ColorDistribution** which represents, as promised above, a probability measure on the color space, and stores by default the average color of a region. The method that allows for richer behaviors is **ComputeStatistics** which must indicate what statistical information to extract from the incoming pixel. On the other hand, **ColorMetric** asks to specify how to compute the distance between a distribution and a pixel, and between two distributions. Here are the Distribution/Metric couples that are made to work together:
 - **EuclideanColorMetric** is most natural way to compute the distance between two colors, and allows to run the standard algorithm: It requires two functions
 - * A **Projector** Π which is a map from a color, which we consider to be a complex physical object, to a vector in a finite vector space: the two main projectors are of course **grayscale** and **RGB**.
 - * A **Metric** (induced by a norm $||\cdot||$) is a function that computes one of the many standard distances between two vectors in \mathbb{R}^n , namely here:

$$\begin{aligned} ||x - y||_1 &= \sum_{i=1}^n |x_i - y_i| \\ ||x - y||_2 &= \sqrt{\sum_{i=1}^n (x_i - y_i)^2} \\ ||x - y||_\infty &= \max_i |x_i - y_i| \end{aligned}$$

Furthermore, in order to be able to compare them, since they will give different results in terms of magnitude², we normalize them by dividing by the average value of the metric on the unit cube, ie:

$$\frac{||x||_i}{\mathbb{E}(|X|_i)} \text{ where } X \sim \mathcal{U}([0, 1]^n)$$

¹we could use a different threshold for merging and adding pixels, but we consider a pixel as a region with one color.

²for instance for all x , $||x||_\infty \leq ||x||_2 \leq ||x||_1$

Hence, we then define the distances with such metric as:

$$\begin{aligned} \text{between a distribution } D \text{ and a color } c : d(D, c) &= \|\Pi(\mu_D) - \Pi(c)\| \\ \text{between two distributions } D_1 \text{ and } D_2 : d(D_1, D_2) &= \|\Pi(\mu_{D_1}) - \Pi(\mu_{D_2})\| \end{aligned}$$

Where Π is the chosen projector and $\|\cdot\|$ the chosen norm for the metric.

We recover the standard algorithm when $\Pi = \text{grayscale}$ and $\|\cdot\| = \|\cdot\|_1$ and the distribution is a simple **ColorDistribution**.

- **CovarianceDistribution** which iteratively updates the covariance matrix³ of the region with each new pixel. The formula to update the covariance matrix is, where c is the new pixel and n the number of pixels in the region, μ_n the mean of the region and Π a projector⁴:

$$\hat{\Sigma}_n = \frac{n-2}{n-1} \hat{\Sigma}_{n-1} + \frac{1}{n} (\Pi(c) - \mu_{n-1})(\Pi(c) - \mu_{n-1})^T$$

The metric **CovarianceMetric** is then defined as the Mahalanobis distance between the distribution and the pixel, and we use the Wasserstein-2 distance between two distributions, which are defined as:

$$\begin{aligned} d(D, c) &= \sqrt{(\Pi(c) - \mu)^T \hat{\Sigma}^{-1} (\Pi(c) - \mu)} \\ d(D_1, D_2) &= \sqrt{\|\mu_{D_1} - \mu_{D_2}\|_2^2 + \text{tr}(\hat{\Sigma}_{D_1} + \hat{\Sigma}_{D_2} - 2\sqrt{\sqrt{\hat{\Sigma}_{D_1}} \hat{\Sigma}_{D_2} \sqrt{\hat{\Sigma}_{D_1}}})} \end{aligned}$$

Which are famous distances from information geometry and statistics. Intuitively the Mahalanobis distance will consider a pixel to be further from the distribution if it isn't aligned with the current pixels' variation from the mean. It favors regions that are more homogeneous. Note that the covariance matrix is generally not invertible for small regions, so we add the identity matrix to it to ensure invertibility and remain close to the $\|\cdot\|_2$ metric. The square root of the covariance matrices, which are symmetric-positive-definite, are computed through diagonalization, which is not too heavy for 3x3 matrices.

- Finally, **HistogramDistribution** is a distribution that is defined by a histogram of the colors in the region, and is updated by adding the new pixel to the histogram. A very natural distance between histograms is the Wasserstein-1⁵ distance, which is defined between 1D histograms (hence limiting to grayscale) as:

$$\mathcal{W}_1(H_1, H_2) = \int_0^1 |\mathcal{C}_{H_1}(x) - \mathcal{C}_{H_2}(x)| dx, \text{ where } \mathcal{C}_{H_i} \text{ is the normalized cumulative histogram of } H_i \quad (1)$$

Used in the **WassersteinMetric** as:

$$\begin{aligned} d(D, c) &= \mathcal{W}_1(H_D, H\{c\}) \\ d(D_1, D_2) &= \mathcal{W}_1(H_{D_1}, H_{D_2}) \end{aligned}$$

The Wasserstein distance comes from optimal transport theory and is defined as the minimum cost to transport a given amount of mass from one distribution to another, and is a very natural distance to use in this context.

Of course, one can merge two distributions of the same type when two regions are merged. The merging of the two distributions is simple:

³simply the variance in the grayscale 1D case

⁴note that the covariance matrix boils down to the variance in the grayscale case

⁵we could use the general Wasserstein-p distance, but involves an inverse cumulative histogram which can be heavy to compute

- two **ColorDistributions** are merged by defining the new mean as:

$$\mu = \frac{n_1\mu_1 + n_2\mu_2}{n_1 + n_2}$$

- two **HistogramDistributions** are merged by summing the two histograms:

$$H = H_1 + H_2$$

- two **CovarianceDistributions** are merged by defining the new covariance matrix as:

$$\hat{\Sigma} = \frac{n_1\hat{\Sigma}_1 + n_2\hat{\Sigma}_2}{n_1 + n_2}$$

2.3 Optimal placement of germs

To be able to compare approaches between them, we tried to avoid randomness as much as possible. Furthermore, since we place a new germ at each free pixel encountered we had the intuition that placing multiple germs isn't very relevant to the efficiency of the algorithm and starting at the same point gives good results.

But, for the sake of scientific exploration, we also enable "optimal germ placement" by placing them at the pixels that minimizes:

$$\text{place } n \text{ germs at the } n \text{ pixels with smallest: } \|\nabla \Pi(\hat{I})\|_{i,j}^2 = (\Pi(\hat{I}_{i+1,j} - \hat{I}_{i,j}))^2 + (\Pi(\hat{I}_{i,j+1} - \hat{I}_{i,j}))^2$$

Where Π is the grayscale projector and \hat{I} is the image scaled down by a factor of F . We subsample the image in order to speed up the computation (since it's a $O(n^2)$ operations) and to avoid placing all the germs in the same region (germs are then placed at position (iF, jF)).

As we thought, the final result isn't much improved in the general case by that placement.

2.4 Empirical study of the effect of the threshold on the resulting variance

The most natural way to measure how well our algorithm did is through the average variance over all the regions:

$$\mathbb{V} = \frac{1}{N} \sum_{i=1}^N \mathbb{V}_i$$

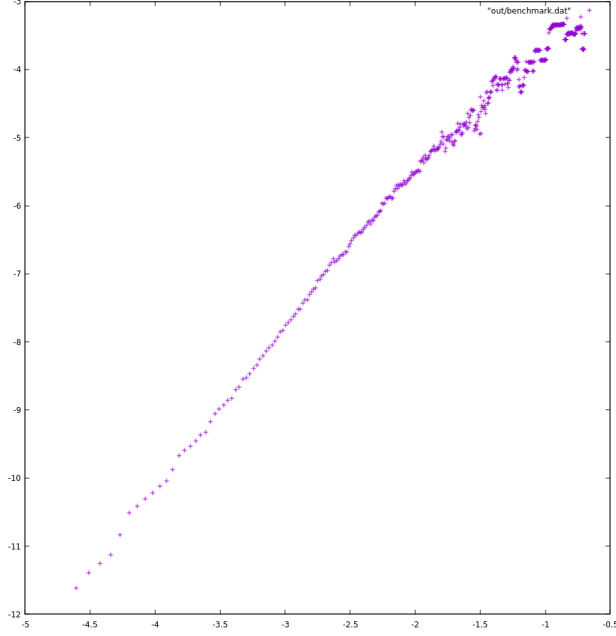


Figure 1: Variance of the resulting image as a function of the threshold ε in log/log scale

A log/log analysis combined with linear regression reveals that the variance decreases as $\mathbb{V} = \mathcal{O}(\varepsilon^2)$. Hence to counter the natural reduction of \mathbb{V} caused by the threshold, we then define a relevant score for the quality of the result as:

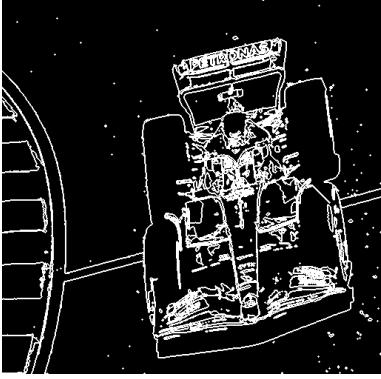
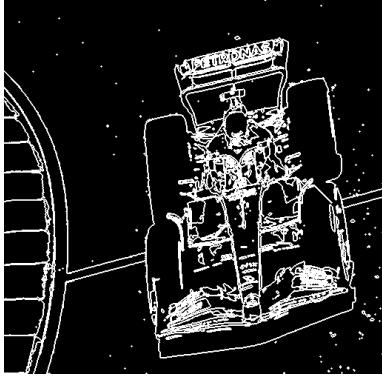
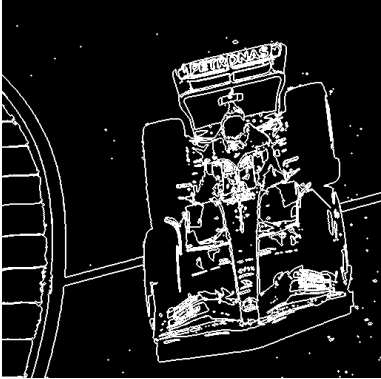
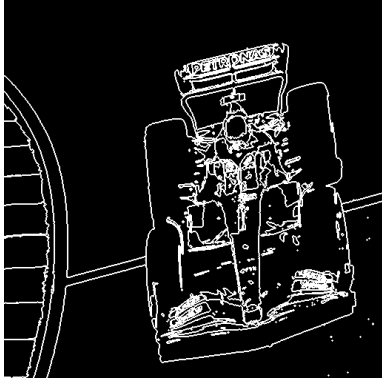
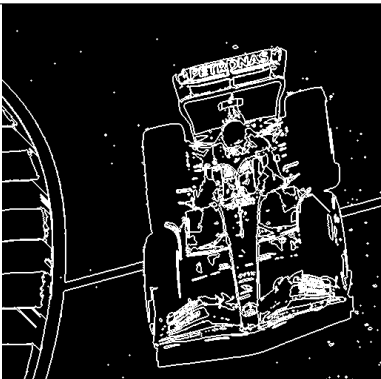
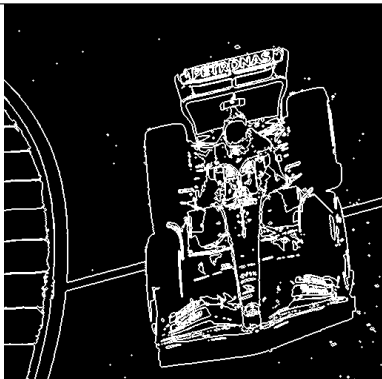
$$\text{score} = \frac{\mathbb{V}}{\varepsilon^2}$$

2.5 Results and efficiency

A clever use of polymorphism allows us to run the algorithm with any combination of distribution/metric, with very negligible cost, with the only constraint that the metric must be able to extract the information it needs from the distribution.

Also note that we could use **Histogram equalization** to improve the results, but we didn't want to use it to avoid biasing the results, and test the robustness of our algorithm.

For the same threshold of distance $\varepsilon = 0.33$ we obtain the following results, with 1 initial germ at the bottom left.

 <p>ColorDistribution Metric = EuclideanColorMetric(GRAY,L1) score = 0.0970204 time=78ms</p>	 <p>ColorDistribution Metric = EuclideanColorMetric(RGB,L1) score = 0.0938873 time=83ms</p>
 <p>ColorDistribution Metric = EuclideanColorMetric(RGB,L2) score = 0.112164 time=81ms</p>	 <p>ColorDistribution Metric = EuclideanColorMetric(RGB,Linf) score = 0.177714 time=74ms</p>
 <p>CovarianceDistribution Metric = CovarianceMetric(GRAY) score = 0.108666 time=112ms</p>	 <p>CovarianceDistribution Metric = CovarianceColorMetric(RGB) score = 0.109674 time=174ms</p>

Since Wassterstein distance scale is different, and relies a lot on the number of initial germs with obtain the following results with 100 initial germs and a $\varepsilon = 0.14$:

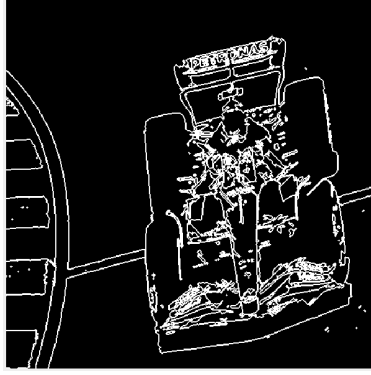


Figure 2: $\mathbb{V} = 0.0201$ time = 587ms

2.6 Conclusion

In conclusion, some approaches fairly improve the result, here the **WassersteinMetric** or the **EuclideanColorMetric** with RGB and $||.||_{\infty}$ make the road almost homogeneous! Overall we are very proud of everything we tried, it was a great scientific adventure.

3 Split and Merge

3.1 Leaf indexing

In order to split the quadtree until every leaf's variance is below the ε_{split} threshold efficiently while keeping relevant neighboring relationships between the leaves for the merge part, we define a clever way of indexing leaves:

A leaf can be uniquely localized with the two following informations:

- The depth of the leaf in the quadtree
- The succession of the directions to take to reach the leaf from the root of the quadtree encoded as bits in a long int.

Since we can define an order on such locations (the lexicographic order with the depth level first), **we can use them into maps as keys to access to any leaf information fast simply from its index without recursion.**

Note that since everything involves division by two and bit manipulations, we require the images to have a size that is a power of two.(not necessarily a square). Doing so ensure that all algorithms are exact and do not suffer from floating precision. We consider that it's a greater benefit next to the deformation caused by a rescaling to the nearest power of two.

3.2 Split

We keep all global information in a **QuadtreeAtlas** object, which contains:

- all the Quadtree that are leaves (since only them are relevant in the merge part)
- a map from a leaf to all its neighbors.

To decide if a leaf must be splitted, we reuse our work from the previous section and associate with every leaf a **CovarianceDistribution** object that computes the variance of the leaf, we split if it's smaller than ε_{split} .

Finally, to maintain the neighborhood relationships between leaves, at each split we apply the following algorithm:

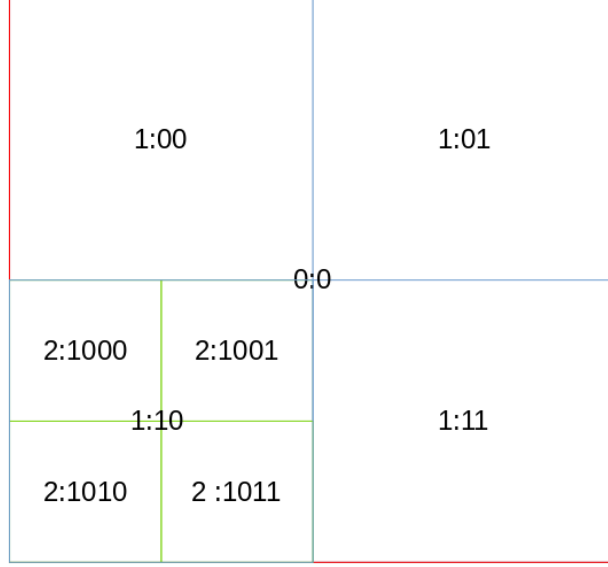


Figure 3: illustration of the leaf indexing system

- We create the four new leaves and add them to the leaf map.
- We remove the parent leaf from the neighborhood of its neighbors.
- We add the new leaves to the neighborhood of their neighbors if they are effectively neighbors (we use the simple test described below).
- we add to the new leaves neighborhood each of the new leaves that are effectively neighbors.

The test to know if two leaves are neighbors is geometric and very simple:

- we project the center of a leaf on the other leaf (we keep the center and the dimensions of each leaf at construction).
- if the projection is still inside the first leaf (at it's boundary but not its corner) then the two leaves are neighbors:

$$(|p.x - c.x| = \frac{\text{width}}{2} \text{ and } |p.y - c.y| < \frac{\text{height}}{2})$$

or

$$(|p.x - c.x| < \frac{\text{width}}{2} \text{ and } |p.y - c.y| = \frac{\text{height}}{2})$$

where c is the center of the leaf and p is the projection of it on the other leaf.

3.3 Merge

To keep the informations about the neighborhood relationships between leaves, we agregate leaves into LeafGroups using once again a UnionFind structure but this time with the quadtree index system with a map instead of an array. The representative for each group is one of the leaves of the group, when merging two groups we simply pick one of the two representatives as the new representative for the merged group.

We also define the neighbors of a group as the set of the leaves that are neighbors of at least one leaf of the group and when merging two groups we simply take the union of the two sets while removing the former neighbors that now belong to the group.

The merging part is then very simple, we simply iterate over all the groups and merge them if their variance is below the ε_{merge} threshold and repeat while two groups can be merged.

3.4 Results

We didn't have the time to develop the same methodology for the merge part as we did for the region growing part, but the results are a trade-off between ε_{split} , the smaller the finer the details, and ε_{merge} , the higher the usefull is the segmentation but the longer it takes to compute.:



Figure 4: CovarianceMetric grayscale $\varepsilon_{split} = 0.03$, $\varepsilon_{merge} = 0.3$, time = 1.8s

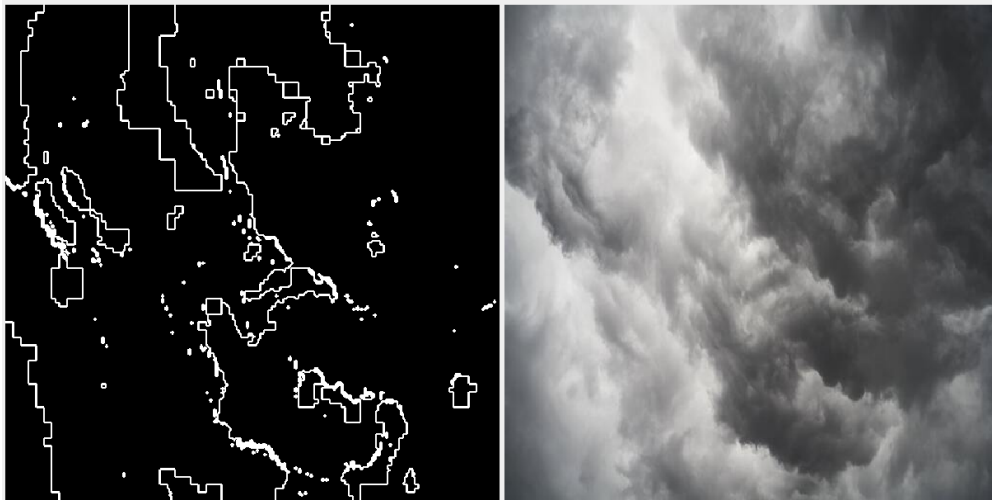


Figure 5: CovarianceMetric RGB $\varepsilon_{split} = 0.06$, $\varepsilon_{merge} = 0.3$, time = 1.1s

3.5 Conclusion

The indexing system allows for a very simple and elegant implementation of the algorithm (we don't have to use a graph structure, everything goes through maps and union find) and the neighborhood relationships are kept at all times.

The result is also satisfying, the main drawback next to the region growing approach is the speed for complex images and region frontiers are always very blocky.