**This will serve as a quick introduction to the project Set**

The purpose of the project is to create clustering algorithm that work in both RGB-space and XY-space as well as additional algorithm for more generic implementation.

The simplest clustering algorithm generate set recursively. That is, once a set has been initialized with a point, it is possible to find every other point in the image satisfying some conditions.

In the case of the XY-space, the conditions are proximity in XY-space and distance in the color space. For RGB-space, only the distance between points is considered.

The proximity condition is simply that of nearest- neighbor. The distance in the color space is defined as the Euclidean distance

Additional parameter can be used to changed the shape of the smallest volume, here a sphere to a spheroid

Lastly, considering the distribution of a cluster of color, we see it behaves as a gaussian. The problem arises when 2 clusters are too close together that they are no longer “well-behaved”. To circumvent this issue, an additional condition can be set up, that of the frequency of a color. By initializing a set with the color having the highest frequency and forcing it to add only point with lower frequency, we essentially move along the surface of the distribution toward local minimums.

To allow all this to happen in a simple fashion, the object Set is defined. It is made of an array of Point and a data type defined by its space. If the Set if defined in the XY-space, the second data type is the image. If the Set is defined in the RGB space, the second data type is a hashtable. Clustering operation simply transfer points from the second data type to the array of Point.

The Object Point is also defined dependently on the type of space of the set. If in the XY-space, the point is P( x , y , color ). If in the RGB-space, the point is P( r , g , b, int[] location). The array of integer location is filled with all the position in the XY-space of the color defined by the value ( r , g , b ).

This way, one can retrieve the image of a set from its array of point without needing to use the image.

This is more efficient. In the case of the XY-space, not so much, however, if one tries to see the image of a set in RGB, he would have to go through each pixel of the image, and then each point in the array of point. Thus, having a time complexity of O(W\*H\*N/2) where N is the length of the array. It is divided by 2 to account for the expected location of the point in the array. While having the location with each color point give a time complexity of O( N\*(1+) ), where is the expected frequency of each color, usually given by = N/16777256.

The same could be said about using a hashtable defined as a double array int[][]. An array whose first index is given by the hashcode of the color and referring to an array of location of the color in the image. That way, if we know a color, we know where it is in the image, instead of parse through the image and comparing with each color.

That is not the only reason for keeping track of each location. For more advanced computation, like the expected color of a set or the expected position of a set, of either type, the calculation can be done much quicker. For example, calculating the expected position of a RGB-set, without the array of location, the time complexity is again O( W\*H\*N/2), as opposed to O( N\*(1+) ).

The last problem is removing the set from the image in order to create another cluster. Keeping in memory either the image or the hashtable of the image, depending of the type of set allows for the user to simply call a getter to retrieve it since the array of point and the secondary data type are disjoint set. This saves computation time.