## Tracking of graph branches

* For each vertex in G1, find a corresponding vertex in G2.
* Let u and v in G1 and u’ and v’ are their correspondences in G2. If u and v are neighbors, then u’ and v’ should be neighbors as well.
* If we can establish good tracking, we can use the motion to improve grouping of vertices into separate objects.
* Motion should also be a part of the MST construction.
* If we can come up with **a good correspondence measure based on the graph structure**, then we can derive the motion vector and use it to derive the MST.
* The good measure should include: proximity,
* How to deal with edges whose motions are collinear.

## How to utilize the branches and their motions

* We can group branches based on their motions.
* How can we incorporate more complex motions? Articulated parts?
* How can we use the grouping to region partitioning?
* Once we can establish proper grouping of moving branches, we can probably draw partition via the space filling technique.

## Can we approach the problem as graph-cut?

* To compare b1 and b2 in the same frame, we should compare their ascendants back by a few frames.

## Unit testing motion detection, extraction, grouping.

* Need a simple test image with ground truth. => Use Processing?
* Merging is weird.. => In average, branches that are in parallel are more ‘similar’ than those that are in series.
* The rectangle clusters into a single entity – no cutting. The others (triangle, ellipse, trapezoid) break into multiple parts. The rectangle tends to pull the other parts although they are separated from the rectangle.

## Can we encode more complex motions (rotation, shear, etc.)?

* How to resolve motions when multiple points can be linked? How to let the correspondence evolve into consistency?
* I think it is time to make a big change to install consistency evolution:

1. We need a cluster of points to represent some non-local information.
2. When clusters overlap, we want to separate them based on their motions.
3. When a cluster split into two, can we simply generate/update clusters?
4. Initial clustering can be error-prone. We need to track each point and its correspondence to correct the clustering. [Is that it??? Do we really need to track each point? Maybe we do not have to keep track of every point. But representing a cluster by points may be an effective and simpler approach than more geometrical (such as lines) approaches. But then, we may not need to establish correspondence between the clusters in subsequent frames.]
5. Clusters can move into overlap. How can we resolve the ambiguity by incorporating the motion of individual clusters?

## Cluster based grouping (without point correspondence)

1. Initially, points are grouped into multiple clusters based on proximity.
2. As frames come in, clusters are tracked and their motions established.
3. As frames come in, clusters can overlap. Use motions and past clusters as features to distinguish the them.
4. As frames come in, a cluster can separate into multiple ones. It is easy to do this only when the separation becomes clear, but we want to make the separation early on based on the shape of the clusters. But can we do it in an efficient way?

What we need:

1. Core-cluster class: points, motions, pointer to the past instances
2. How to compare clusters – centroid proximity, some shape descriptors, etc. – For now, let’s use only the centroid proximity of the current and past instances.
3. How to split clusters – For now, let’s use only the centroid proximity.