# Main program and pipeline (Tiger)

## The 3D Reconstruction pipeline

1. Find 2D correspondences
2. Compute F (Gold standard)
3. Compute E from F and estimate R & t
4. Divide the observed points into new and old, depending on if they are seen by previous cameras or not.
5. Perform PnP on new camera and old points if enough old points are seen by the new camera, use previously estimated R & t as initial values.
6. Triangulate new 3D points
7. Remove new points that are too close to old points
8. Update the visibility
9. Perform Bundle Adjustment

## Data structures & Visibility

The main thought when designing the data structures was to enable easy access to all needed data for each stage in the pipeline each with different requirements and needs. One early thought was to render the image plane and draw lines from each pixel with a correspondence to its 3D-point, as a visualization and debugging tool. This was never implemented but also affected the way the data structures look like. They evolved from the different requirements of constant look-up of various sets of information. A unique 3D-point exists only as one data object but is shared by various data structures:

* The camera knows which image point corresponds to which 3D-point
* Each camera pair knows which image point-pair that are corresponding to which 3D-point
* Each 3D-point knows which camera pairs, with which corresponding image point-pair, it corresponds to.

All this data is managed by a single class and accessible by a single simple interface.

Why not a visibility-table?:  
It just happened to be this way. It seemed inefficient with a growing reallocating vector-in-vector approach and too static with a huge statically allocated matrix with a fixed limit on the number of points. The different ways of access thought up at the beginning also meant that certain lookups would require repeated searches.  
However, we probably should have used some Mat-data-types which might have speeded up BA slightly (not as much as other improvements though).

## Problems (and solutions)

The dinosaur data set was a bit harder than it seemed at first. We had some huge problems with accumulating errors making it seem as if our pipeline had some major flaw. However, when we tested with the cleaned up data points the reconstruction worked very well!

A huge problem came from outliers that weren’t removed properly. Using PnP to improve the pose estimate of the new camera ONLY according to the currently best estimates (old 3D-points), triangulating the new 3D-points and then removing those with too high re-projection-error made a huge difference! It now managed to run on our own correspondences from the dinosaur data set with very good results coming out.

## Another general problem is that the program takes a very long time to run (without additional improvements in the nonlinear algorithms) and it was therefore of great importance that we made a file format and could save/load states of the program. We also built a stand-alone Viewer that could load the files and display the progress as it was made by the main program.

## Improvements

One possible improvement would be to extract fewer but much better correspondences to estimate good camera poses, and then to mostly triangulate the rest of the image points that are intrinsic-2-dimensional. Such better correspondences might be those that link to other correspondences in sequent image-pairs and thereby are part of as long chains as possible. With various verifications, both of the quality of the points in the chain and in the later triangulation, this should probably produce good results in much lesser time. Additionally, building on this idea, clusters could possibly be constructed using cameras which look at similar points and then to optimize over such clusters rather than over all camera-parameters. None of this was unfortunately tested, but there exists a lot of exciting things to try out!