FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Diplomarbeitspräsentation

A Composable and Reusable Photogrammetric Reconstruction Library

Attila Szabo

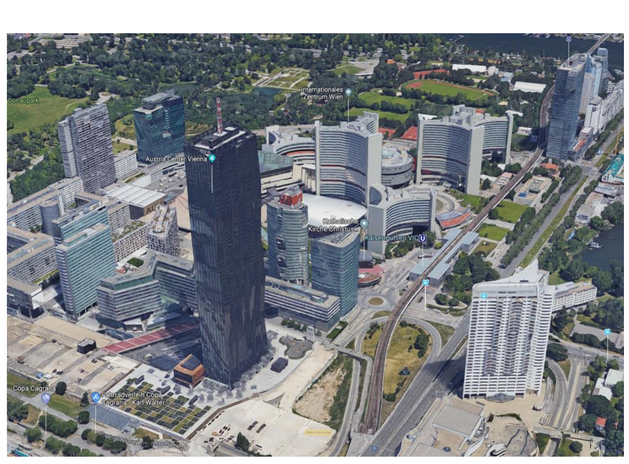
Masterstudium:
Visual Computing

Technische Universität Wien
Institute of Visual Computing & Human-Centered Technology
Computer Graphics Research Division
Betreuer: Prof. Dr.Dr.h.c. Werner Purgathofer
Dr. Dipl. Ing. Maierhofer Stefan

Photogrammetry

Photogrammetry means taking measurements in photographs. Enough images from different viewpoints allow recovery of 3D structure. Thanks to digital photography and the internet, photos are abundantly and cheaply available. Accordingly, photogrammetry applications can exhibit enormous variety.

Existing software is often either closed source or tailored toward specific use cases, making it difficult to reason about algorithmic details or make structural modifications. In this diploma thesis, we present a library for creating photogrammetry pipelines through function composition, emphasising modularity and clean design to support rapid scientific experimentation.



Photogrammetry in Google Maps

Image Features

Features describe objects in an image. Different features work well for different things: Points, lines, rectangles, etc.

```
type Feature<'a> =
{
    spatial : 'a
    descriptor : byte[]
}

type PointFeature = Feature<Vec2>
```

Interest Points

Feature Matching

Matching Features

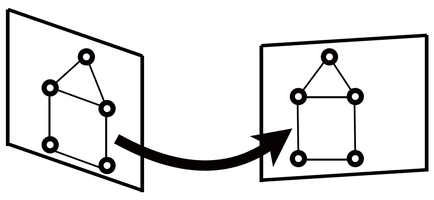
We identify the same feature in two images.

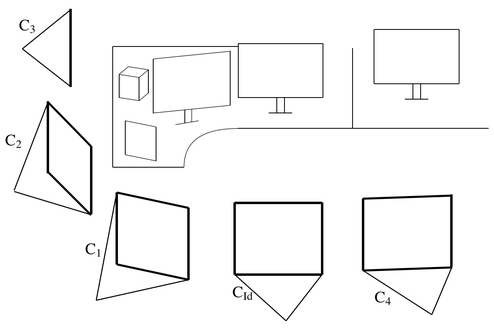
Features must **look** and **move** similarly across viewpoints.

```
val match : list<Feature> -> list<Feature> -> list<Feature * Feature>
```

Pose Recovery

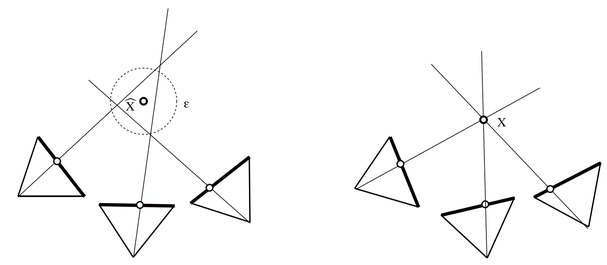
Obtain the **camera motion** between two photos through matching features.





Repeat iteratively to create a **Photo Network**.

Global optimisation rectifies cumulative measurement errors.



Functions ...

Every component is implemented as pure function.

```
val motion : list<Feature * Feature> -> Trafo
```

```
let createPhotoNetwork =
    List.fold ( fun left network ->
        let right = network.Head
        let newCam =
            left.transformed motion match left right
        newCam :: network
    ) []
```

```
val bundleAdjust : PhotoNetwork -> PhotoNetwork
```

λ

... and Composition

Sequential computation through function composition.

```
let extractFeaturesFast files =
    Array.Parallel.map extractFeatures

let myPhotogrammetryPipeline files =
    files
        |> Array.map readImages
        |> extractFeaturesFast
        |> createPhotoNetwork
        |> bundleAdjust
        |> render
```

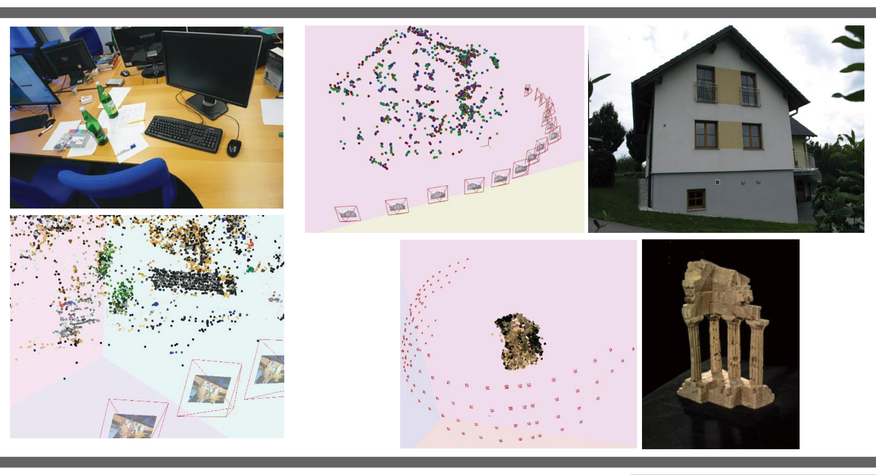
»

Conclusions

A pure functional implementation facilitates **reasoning about** behaviour.

Immutable data structures promote **structural changes**, such as recursion or parallelisation.

Statically enforced conventions ease changing implementations, increasing **code reusability**.
Code at https://github.com/aardvark-platform/aardvark.mondo



Kontakt: szabo@vrvis.at

vrvis