

# Entwurf & Implementation einer Refotografie-Applikation für iOS

**Autor**

Rasmus Diederichsen

**Betreuung**

Prof. Dr. Oliver Vornberger

Ann-Katrin Häuser, MSc.

**AG Medieninformatik**

# Inhalt

1. Einleitung & Motivation
2. Ziele der Arbeit
3. Versatzschätzung
  - 3.1. Probleme der Versatzschätzung
4. Korrespondenzfindung
5. Demo
6. Stand der Arbeit & Ausblick

# Refotografie



## Rekonstruktion von Aufnahmen

- Position, Orientierung
- Brennweite, Hauptpunkt, etc.

## Erstellung von Vorher-Nachher-Vergleichen

- Historische Entwicklung von Gebäuden
- Baufortschritt
- Jahreszeiten





# Problem: Wie Pose finden?

## Bisher

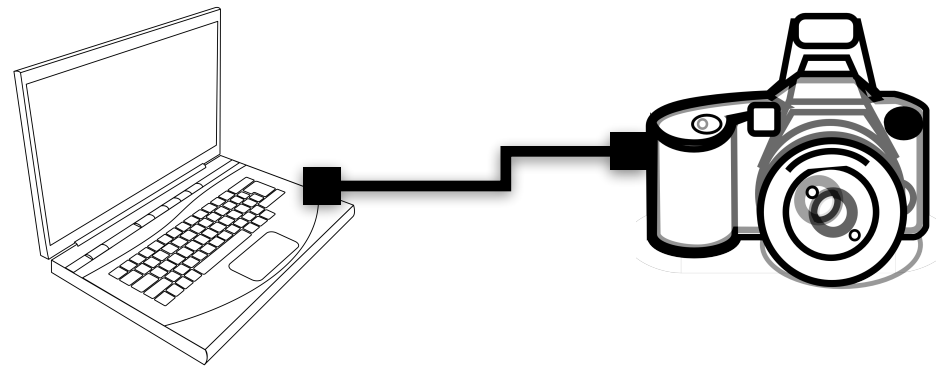
- Bisher: Augenmaß, Trial & Error, Annäherung & Nachbearbeitung
- *Timera, rePhoto*: Hilfestellung durch Overlay
  - Rekonstruktion erfolgt manuell

## Neu

- Bae et al., 2010



# *Computational Re-Photography (Bae, Agarwala, Durand, 2010)*



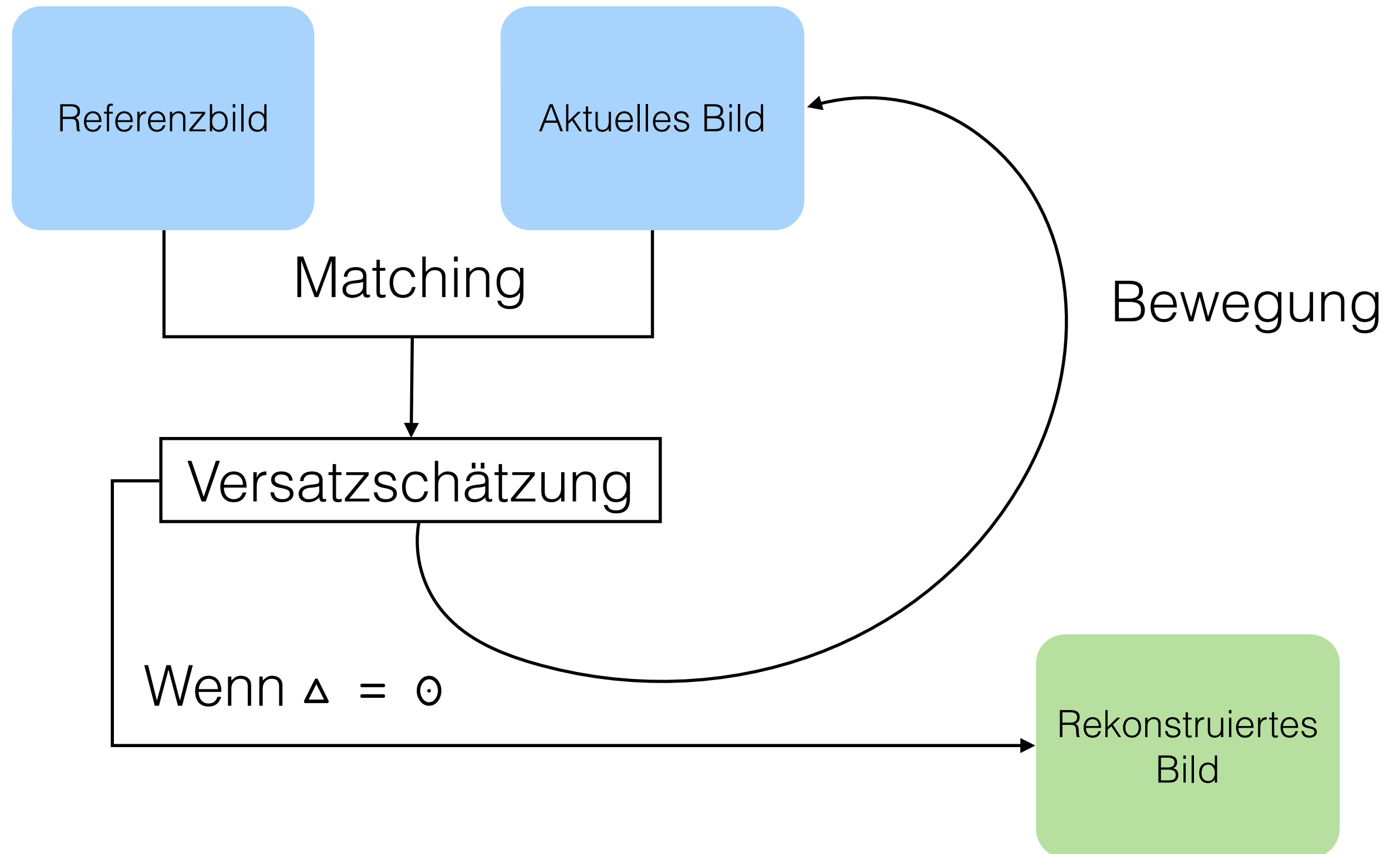
## Rekonstruktion von Brennweite, Hauptpunkt und Pose historischer Fotos

- Komplette Rekonstruktion aller Parameter
- Pfeile zur Veranschaulichung der 3D-Translation
- Weder mobil noch verfügbar

# Ziele

- Vereinfachung des Ansatzes von Bae et al.
- Automatisierte Refotografie
  1. (möglichst) Automatische Rekonstruktion eines Aufnahmestandortes eines *ähnlichen* Bildes
  2. Führung des Nutzers zur richtigen Pose (Translation, *ggf. Rotation*)
  3. Anfertigung & Anzeigen der ggf. nachoptimierten Refotografie

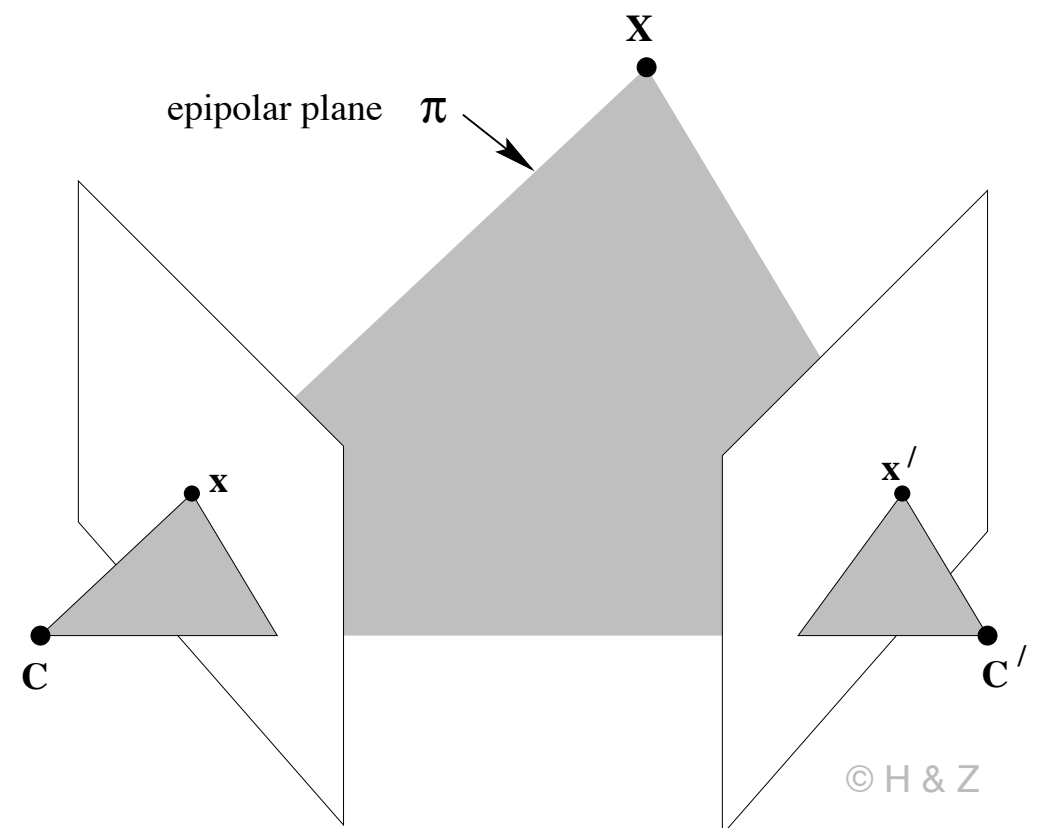
# Ablauf (naiv)



# Versatzschätzung – Epipolargeometrie

Gegeben 2 kalibrierte Aufnahmen einer Szene, berechne  
Posedifferenz  $(R, T)$

$$\begin{aligned} \lambda_2 \mathbf{x}' &= R \lambda_1 \mathbf{x} + T & | \cdot \hat{T} \\ \lambda_2 \hat{T} \mathbf{x}' &= \lambda_1 \hat{T} R \mathbf{x} + \underbrace{\hat{T} T}_{=0} & | \cdot \mathbf{x}'^T \\ \lambda_2 \underbrace{\mathbf{x}'^T (T \times \mathbf{x}')}_{=0} &= \lambda_1 \mathbf{x}'^T \hat{T} R \mathbf{x} \\ 0 &= \mathbf{x}'^T \underbrace{\hat{T} R}_E \mathbf{x} \end{aligned}$$



© H & Z



# Versatzschätzung – Epipolargeometrie

Gegeben 2 kalibrierte Aufnahmen einer Szene, berechne  
Posedifferenz  $(R, T)$

$$\mathbf{x}'^T E \mathbf{x} = 0$$

Verschiedene Algorithmen zur  
Schätzung

- 8-Punkt-Algorithmus  
(*Longuet-Higgins, 1981*)
- 5-Punkt-Algorithmus (*Níster, 2003*)

4 Lösungen für  $(R, T)$

- Richtige Lösung führt zu rekonstruierten Punkten mit positiver Tiefe in beiden Frames
- Eindeutige Lösung (bis auf Skalierung für  $T$  😊)

# Probleme der Versatzschätzung

1. Naive Idee: Referenzbild immer mit aktuellem Bild vergleichen
  - ▶ Schätzung wird für  $\Delta = 0$  instabil
  - ▶ je näher an der Wahrheit, desto weniger weiß man weiter
2. Unbekannte Skalierung erschwert Vergleich über Iterationen

# Problem 1: Schätzungsdegeneration

## Lösung (Bae et al.)

1. 2 Bilder der Szene aufnehmen
  - ▶ *First frame*, etwa  $20^\circ$  vom Original rotiert
  - ▶ *Second frame*, beste Annäherung an Original nach Augenmaß
2. Versatz  $T_{first,ref}$  zwischen Referenzbild und *First frame* berechnen
3. Anstatt  $T_{current,ref} = 0$  lieber  $T_{first,current} = T_{first,ref}$  anpeilen
  - ✓ Instabilität der Schätzung vermeiden

# Problem 2: Unbekannte Skalierung

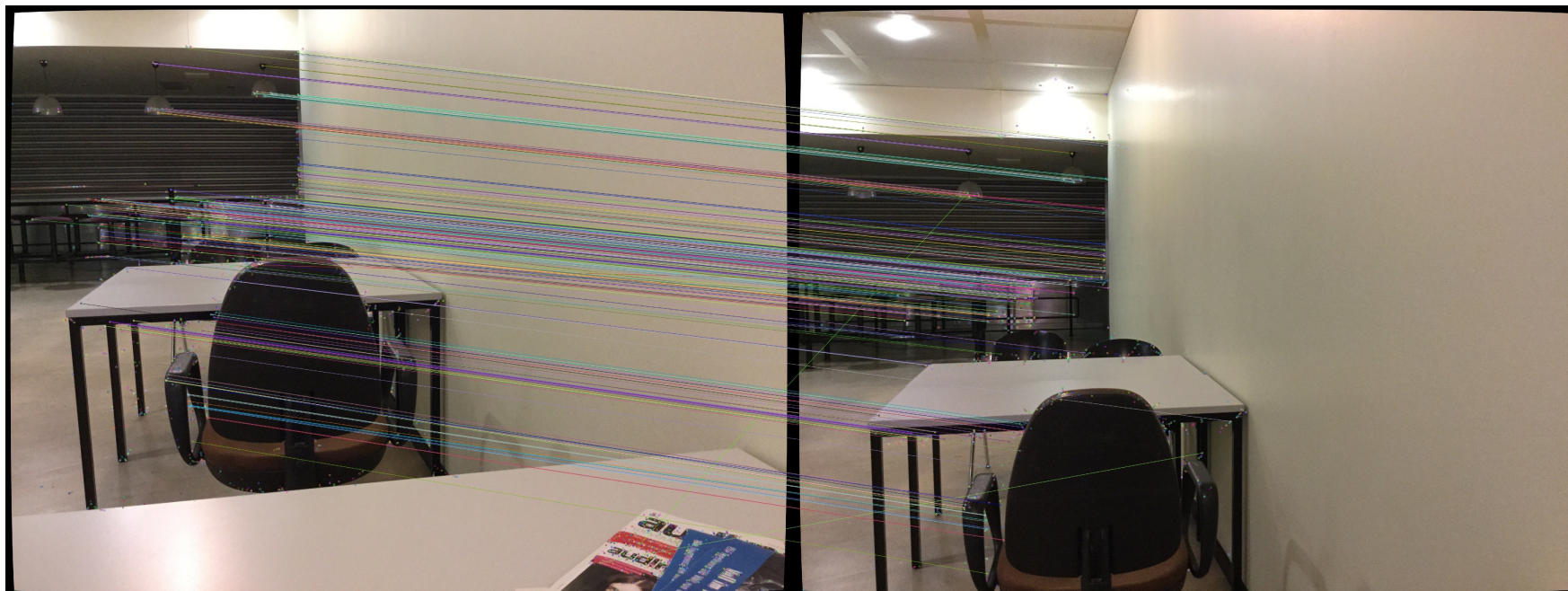
## Lösung (Bae et al.)

1. 2 Bilder der Szene aufnehmen
  - ▶ *First frame*, etwa  $20^\circ$  vom Original rotiert
  - ▶ *Second frame*, beste Annäherung an Original nach Augenmaß
2. 3D-Rekonstruktion der Szene (Punktwolke)
3. Mittlere Distanz der Punkte zur Kamera als Skala verwenden
4. Jeweils mit aktuellem Frame neu Rekonstruieren, Distanz vergleichen,  $T$  skalieren
  - ✓ Vergleichbarkeit der Iterationen
  - ✓ Maß für Nähe (nicht nur Richtung) des Ziels

# Korrespondenzfindung

## Feature-Detektion

- SIFT (patentiert, effektiv, langsam, reelle Deskriptoren)
  - SURF (patentiert, effektiv, schnell, reelle Deskriptoren)
  - AKAZE (frei, schnell, binäre Deskriptoren, schneller matchbar, *Alcantarilla 2013*)
- ▶ AKAZE-Features auf beiden Bildern finden
  - ▶ (*Brute-force*) Matching der Deskriptoren
  - ▶ Als Korrespondenzen verwenden
  - ▶ **Essentielle Matrix + Posedifferenz berechnen**

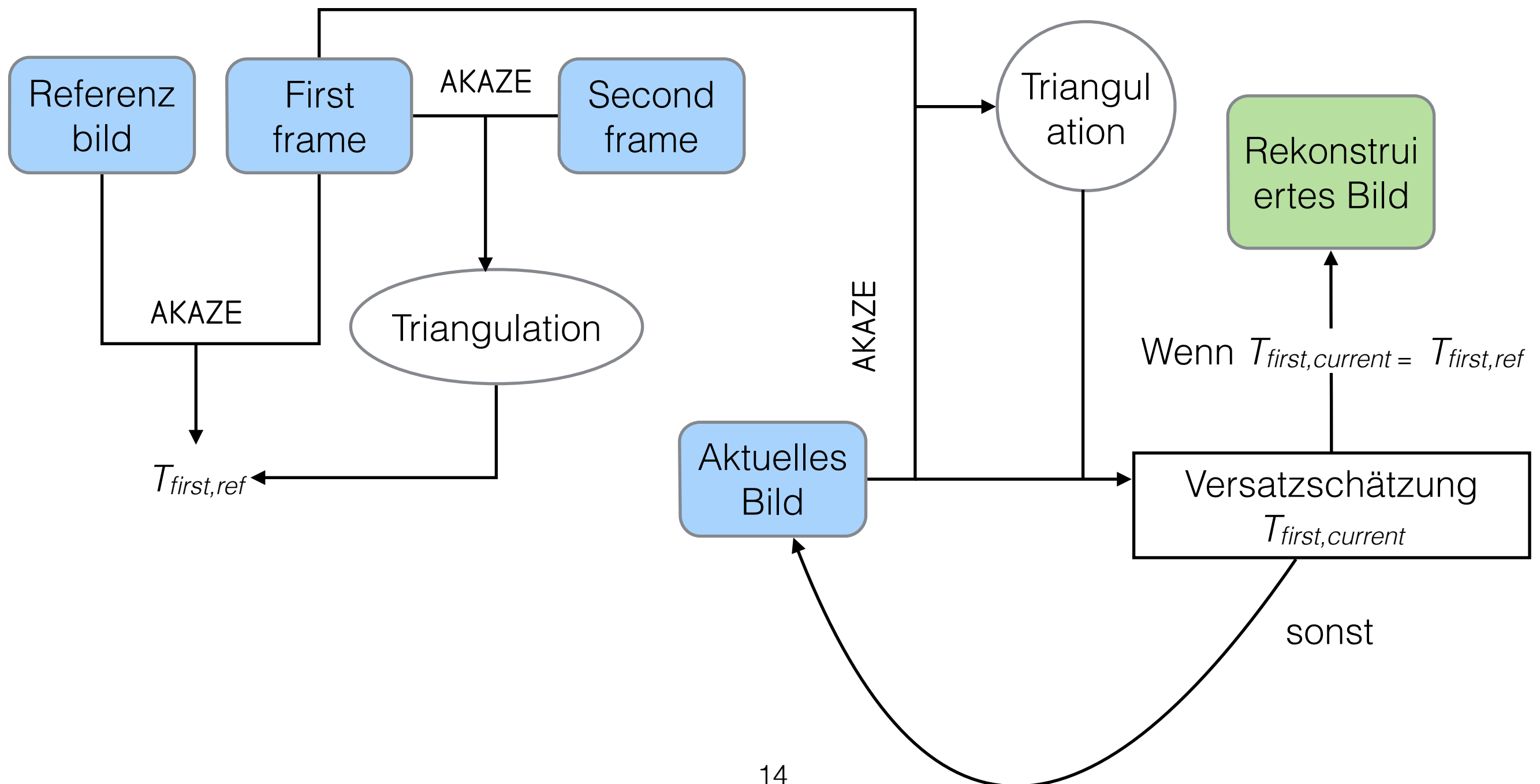




# Ablauf

Preprocessing

Iterativ



# Demo

# Stand & Geplante Features

## Stand

- 1. Version des UI
  - ▶ Auswahl
  - ▶ Kantenoverlay, Bild machen
  - ▶ anzeigen
- 1. Implementierung Bildversatzschätzung (noch nicht integriert)
- Ansatz zum Upload
- Theoretische Überlegungen

## Fehlend, Geplant

- Galerie (anzeigen aller Refotos)
- Assistenzsystem
- Optimierung d. Ergebnisses
- *Code-Qualität, Portabilität, Ästhetik*

Danke für die Aufmerksamkeit

# Referenzen

- Bae, S. (2009) *Analysis and Transfer of Photographic Viewpoint and Appearance* (Doctoral dissertation) Retrieved from DSpace MIT (<http://hdl.handle.net/1721.1/55085>)
- Bae, S., Agarwala, A., & Durand, F. (2010). *Computational Re-Photography*. ACM Transactions on Graphics, 29(3).
- Hartley, R., & Zisserman, A. (2003). *Multiple view geometry in computer vision* (2nd ed.). Cambridge, UK: Cambridge University Press.
- Longuet-Higgins, H. (1981). *A computer algorithm for reconstructing a scene from two projections*. Nature, 293(5828)
- Nister, D. (2004). *An efficient solution to the five-point relative pose problem*. The IEEE Transactions on Pattern Analysis and Machine Intelligence, 26(6).
- Vornberger, O. <http://www.vorher-nachher.photo/>



# Extras

# Kameraparameter

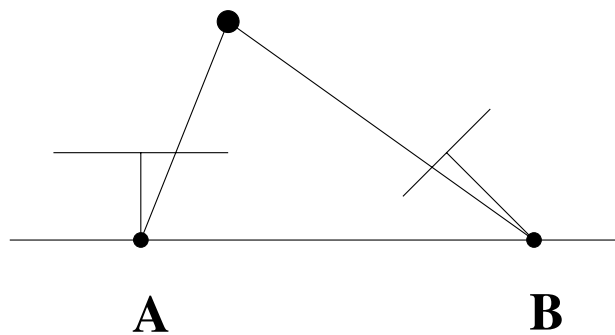
Abbildung von Welt- in Pixelkoordinaten:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{K}[\mathbf{R} \mid \mathbf{T}] \begin{bmatrix} x \\ y \\ w \\ 1 \end{bmatrix}$$
$$= \begin{bmatrix} f_x & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} [\mathbf{R} \mid \mathbf{T}] \begin{bmatrix} x \\ y \\ w \\ 1 \end{bmatrix}$$

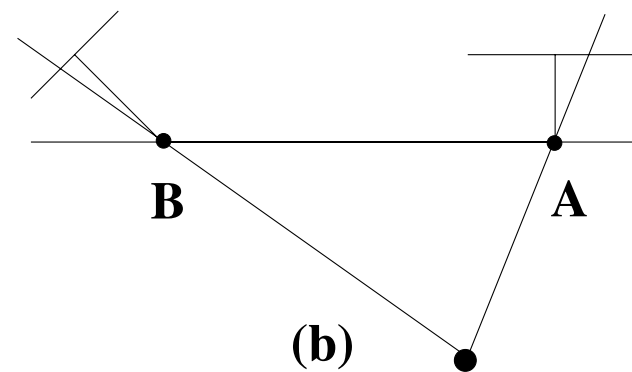
- Kameramatrix als bekannt vorausgesetzt
- Annahme: Kameramatrizen für alle Bilder identisch

# Versatzschätzung – Epipolareometrie

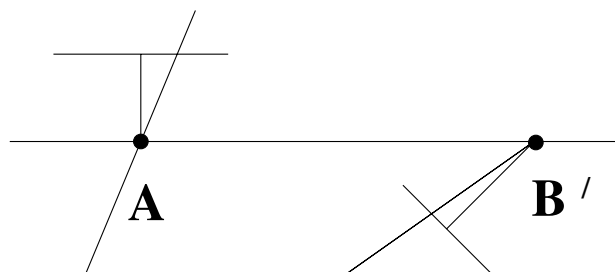
Gegeben 2 kalibrierte Aufnahmen einer Szene, berechne  
Posedifferenz  $(R, T)$



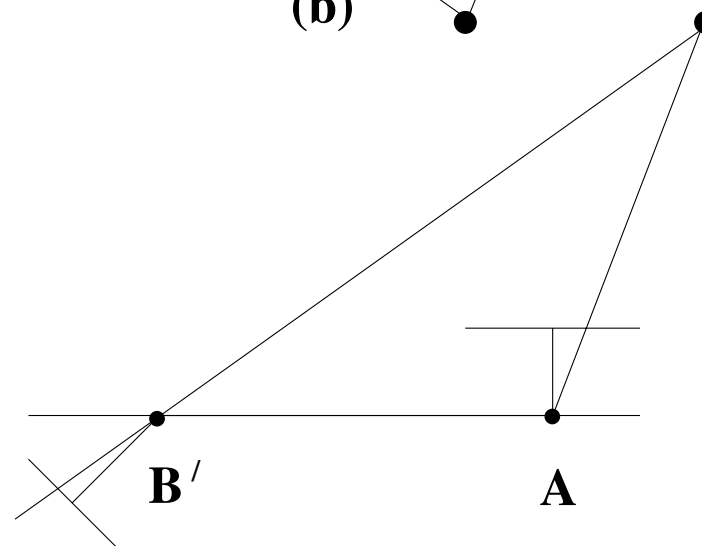
(a)



(b)



(c)



(d)

H&Z