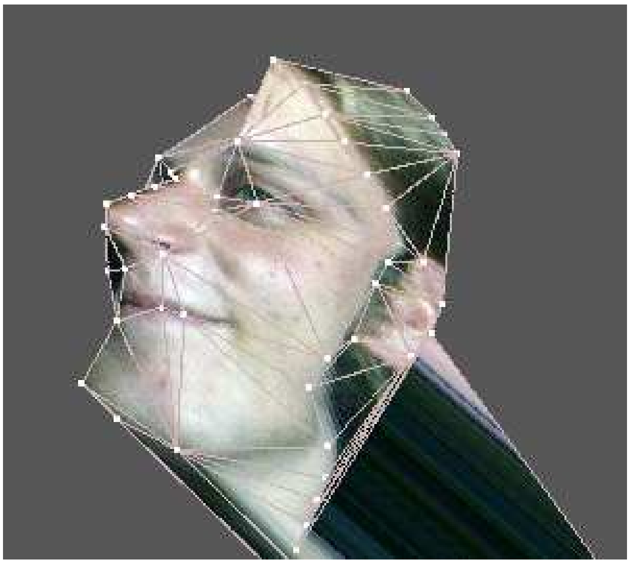
# 3D-Rekonstruktion und Visualisierung natürlicher Szenen auf der Basis von Polygonmodellen



Belegarbeit von Norman Feske

## *Gliederung*

1. Problem - Überblick
2. Segmentierung
2.1. Vorverarbeitung
2.2. Netzwerkmodelle
2.3. Bewertung der Netze
2.4. Triangulation
3. Tiefenschätzung
3.1. Stereogeometrie
3.2. lokale Korrespondenzanalyse
3.3. Verbesserungen
3.4. Bewertung
4. Visualisierung
5. Implementation
6. Zusammenfassung und Ausblick

## *(1-1)* ***Problem***

---

**Problem:** 3D-Rekonstruktion natürlicher Objekte

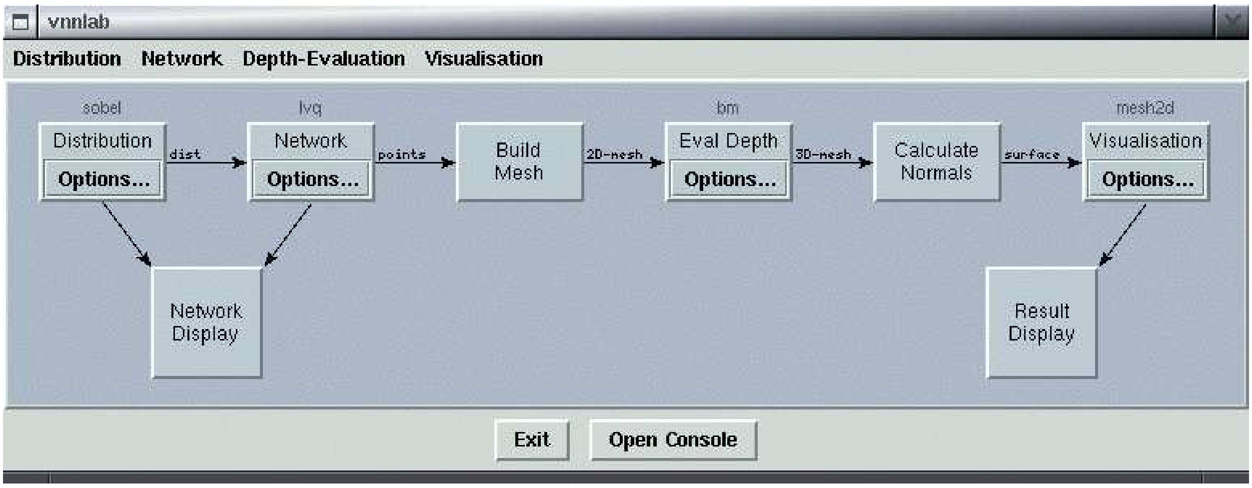
**Ansatz:**

- Bildsegmentierung zur Generierung eines Polygonnetzes

- Tiefenschätzung durch lokale Korrespondenzanalyse

**Ziel:** Lehrbeispiel für die Verwendung eines Stereokopfes

# *(1-2)* ***Überblick***



## *(2) Segmentierung*

---

**Vektorbasiertes Neuronales Netz** besteht aus:

- Menge von Einheiten (Referenzvektoren) und (eventuell)

- Menge von Verbindungen

**Zusätzlich:** Wichtigkeit einer Einheit
$w := w + s * (1.0 - w)$, $s \in (0, 1)$

**Prinzip:** die Einheiten versuchen, sich an eine gegebene Verteilung anzupassen

**Einschränkungen:** endlicher 2-dimensionaler Datenraum

## *(2.1-1)* ***Vorverarbeitung***

---

**Ziel:** Generieren einer diskreten Verteilung

- diskrete Verteilung soll aus 'interessanten' Punkten bestehen

**Vorgehensweise:**

- 'interessante' Punkte $\rightarrow$ große Kontraste

- Sobel-Filter

- Schwellwert

## (2.1-2) ***Sobel-Filter***

$$SM_{hor} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad SM_{ver} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Gradienten:

$$grad_{hor}(x,y) =$$

$$\sum_{i=-1}^{1} \sum_{j=-1}^{1} SM_{hor}(i+1, j+1) * B(x+i, y+j)$$

$$grad_{ver}(x,y) =$$

$$\sum_{i=-1}^{1} \sum_{j=-1}^{1} SM_{ver}(i+1, j+1) * B(x+i, y+j)$$

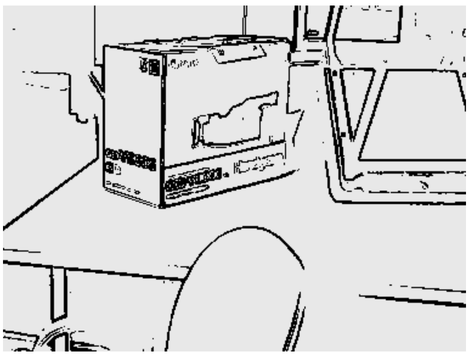
Sobel-Filter-Antwort:

$$sobel(x,y) = \sqrt{grad_{hor}(x,y)^2 + grad_{ver}(x,y)^2}$$

# (2.1-3) **Vorverarbeitung - Beispiel**


Ausgangsbild

Sobel Filter



diskrete Verteilung

## (2.2-1) ***Learning Vector Quantisation***

- Wahl eines zufälligen Datenvektors $\xi$ aus der gegebenen Verteilung

- Ermitteln der Einheit $c$ mit dem geringsten (euklid.) Abstand zu $\xi \to c$ ist 'Gewinner'

- Adaption des Gewinners in Richtung $\xi$ :
$\triangle w_c := \epsilon * (\xi - w_c)$

- konstante Lernrate $\epsilon$

- es wird nur der Gewinner adaptiert $\to$ 'hartes Wettbewerbslernen'

## *(2.2-2)* ***k-means***

---

- jede Einheit $c$ hat eine eigene Lernrate: $\epsilon(t_c) = \frac{1}{t_c}$, $t_c$ : Anzahl der Iterationen, für die $c$ bisher der 'Gewinner' war

- Mittelwert aller Datenvektoren, für die eine Einheit Gewinner war

**Iteration:**

1. Ermitteln des 'Gewinners' $c$ für einen zufälligen Datenvektor $\xi$ der gegebenen Verteilung

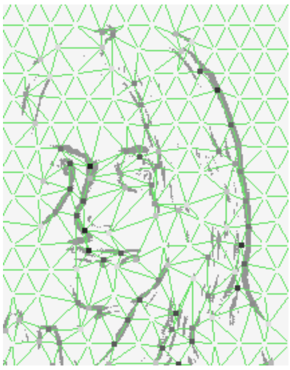
2. Adaption des 'Gewinners':
   $\triangle w_c := \epsilon(t_c) * (\xi - w_c)$

3. Inkrementierung von $t_c$ des 'Gewinners'

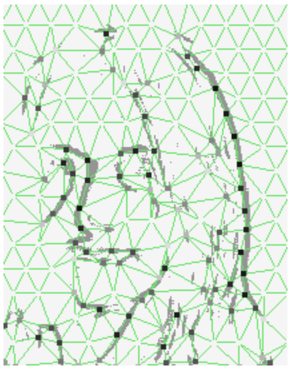
# (2.2-3) *k-means* ***Beispiel***

Ausgangsbild

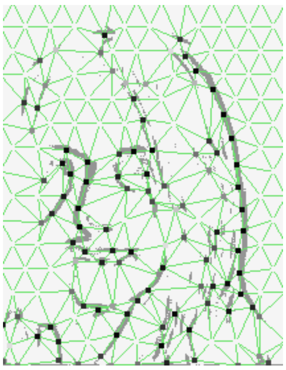
diskrete Verteilung



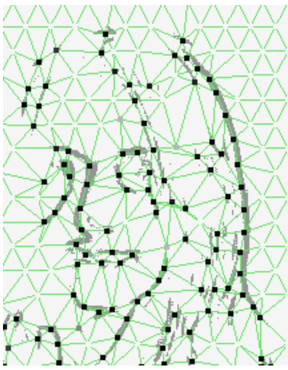
nach 500 Iterationen



nach 1000 Iterationen



nach 2000 Iterationen



nach 10000 Iterationen

## (2.2-4) ***Simulated Annealing***

- Lernrate $\epsilon(t) = \epsilon_i * (\frac{\epsilon_f}{\epsilon_i})^{\frac{t}{t_{max}}}$

- $\epsilon_i$ und $\epsilon_f$ sind die Start- und Endwerte der Lernrate

- Verfahren konvergiert

## *(2.2-5)* ***Neuronales Gas***

---

- Einheiten werden nach der Entfernung zum Datenvektor $\xi$ sortiert

- jede Einheit hat einen Ranglistenplatz $k$

- Adaption einer Einheit:
  $\triangle w_{s_k} := \epsilon(t) * h(t,k) * (\xi - w_{s_k})$

- Einfluß des Ranglistenplatzes auf die Adaption:
  $h(t,k) := exp(\frac{1-k}{\lambda(t)})$

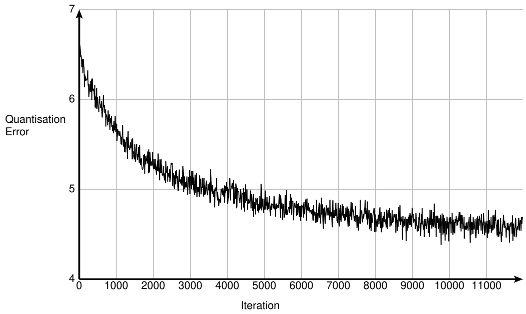
- 'Zähigkeit': $\lambda(t) = \lambda_i * (\frac{\lambda_f}{\lambda_i})^{\frac{t}{t_{max}}}$

# *(2.3-1)* ***Bewertung der Netzwerkmodelle***

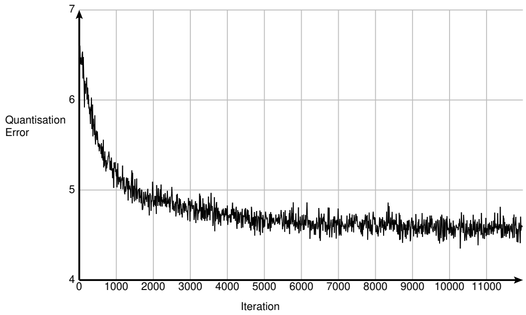
Bewertungskriterien:

- mittlerer Quantisierungsfehler
- Konvergenz
- Berechnungsgeschwindigkeit
- liegen die Einheiten auf Punkten der Verteilung?
- Parameterraum
- qualitative Beurteilung

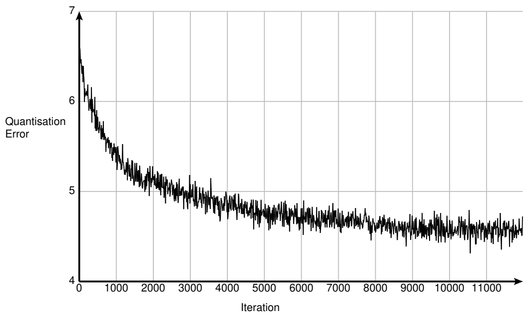
# *(2.3-2) **mittlerer Quantisierungsfehler***




Learning Vector Quantisation




k-means



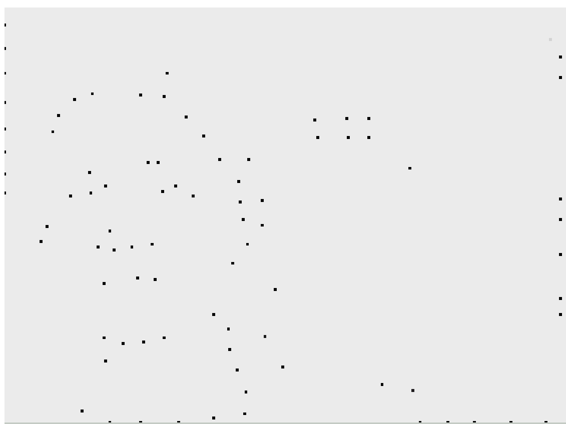

Simulated Annealing

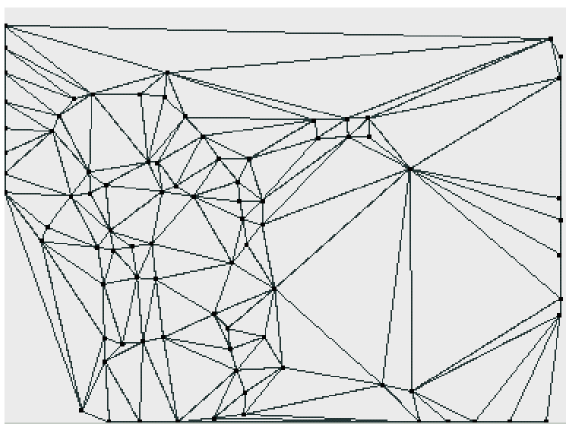
# (2.4) ***Triangulation***

Verbinden der Netzwerk-Einheiten durch Polygone

**Ziel:** geschlossenes 'schönes' Polygonnetz

**mögliches Kriterium:** Delaunay-Triangulation





## *(3) Tiefenschätzung*

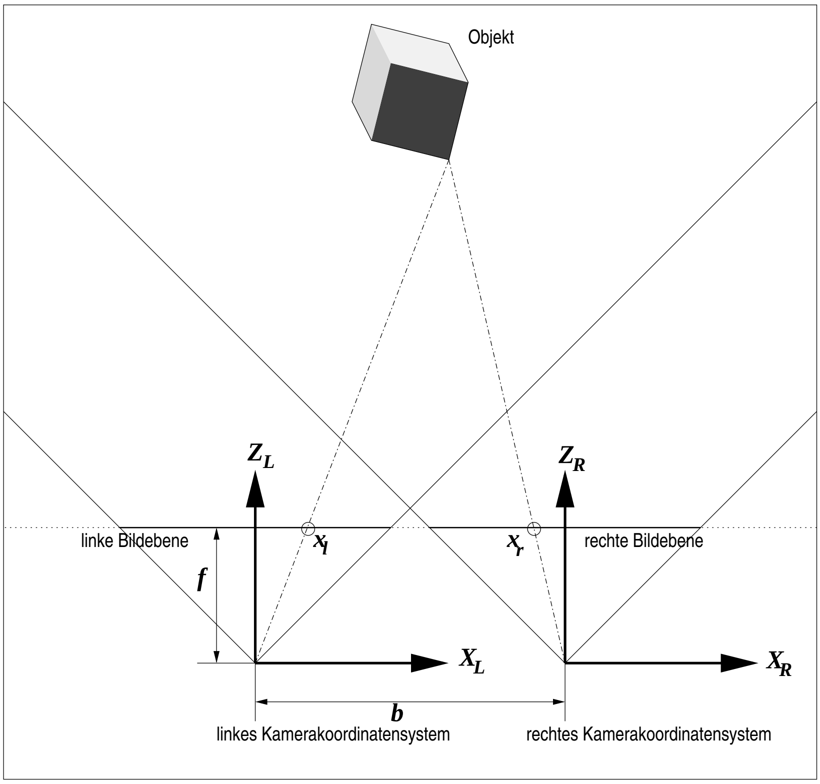
**gegeben:**

- Stereobildpaar
- Menge von Punkten im 2-dimensionalen Raum, gegeben im Koordinatensystem des rechten Bildes

**gesucht:**

- Menge der entsprechenden Punkte im 3-dimensionalen Raum

# (3.1) **Standardstereogeometrie**



## (3.2) **lokale Korrespondenzanalyse**

- Disparität bei Verwendung der Standardstereogeometrie: $x_l - x_r$

- Zusammenhang zwischen Disparität eines Punktes und dessen Entfernung vom Ursprung des Kamerakoordinatensystems:

$$Z = \frac{f * b}{x_l - x_r}$$

**Problem:** das Finden korrespondierender Punkte im linken und rechten Bild

**Einschränkungen:** 1-dimensionaler Suchbereich, Disparitätslimit

## *(3.2)* ***Blockmatching***

- gegeben ist ein Pixelblock (an Position $(x_r, y)$) im rechten Bild $E_r$

- Welcher Pixelblock im linken Bild $E_l$ (in Zeile $y$) ist diesem am ähnlichsten?

- (Un-)Ähnlichkeitsmaß zwischen zwei Pixelblöcken:

$$dissim_{SE}(E_l, E_r, x_l, x_r, y) =$$

$$\sum_{i=-k}^{k} \sum_{j=-k}^{k} (E_l(x_l + i, y + j) - E_r(x_r + i, y + j))^2$$

## *(3.3-1)* ***Verbesserungen***

- Glätten der Ausgangsbilder $\rightarrow$ erhöht Invarianz gegenüber Rauschen und vert. Verschiebungen

- Hochpaß-Filtern der Ausgangsbilder $\rightarrow$ erhöht Invarianz gegenüber Shading

- es gehen Informationen verloren!

- andere (Un-)Ähnlichkeitsmaße (Kreuzkorrelation, Shirai, absoluter Fehler)

## (3.3-2) ***Farbe als Merkmal?***

RGB-Farbraum:

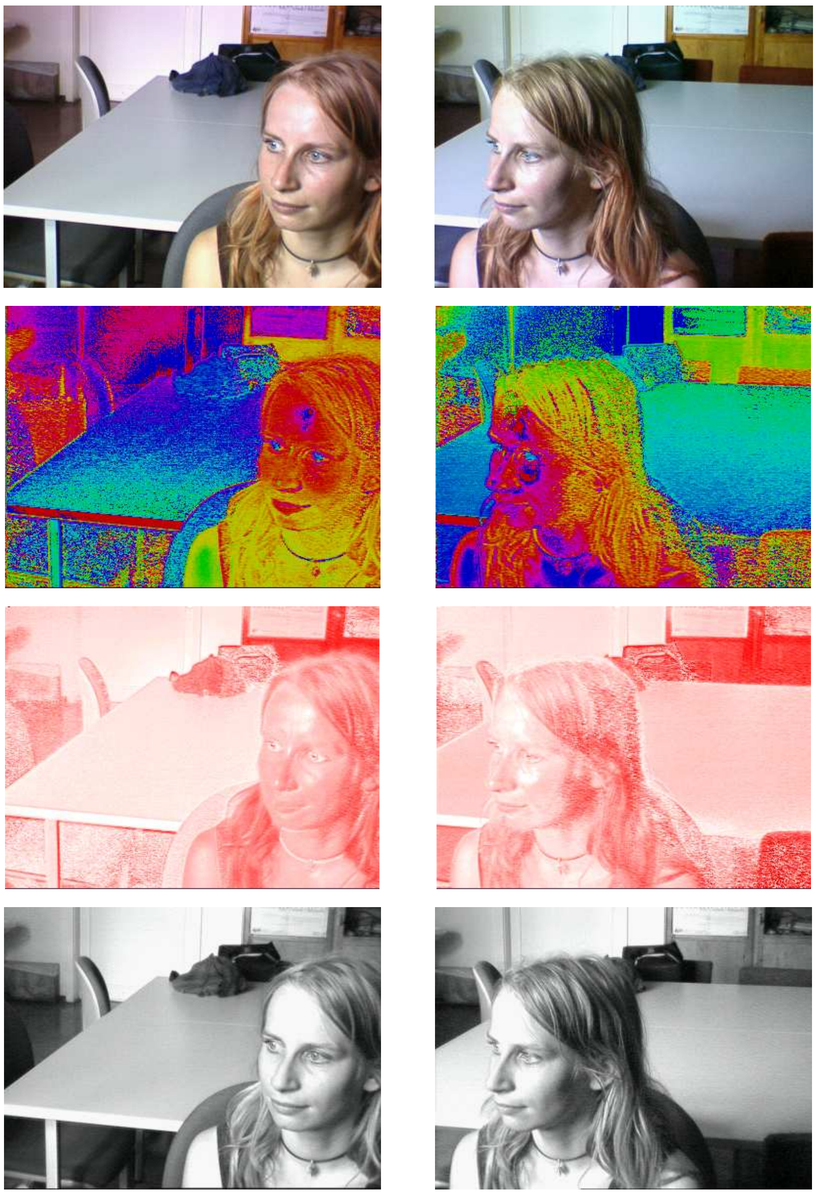
- Kodierung der Farbe unanschaulich

- mögliche Ähnlichkeitsmaße (aus Klette: "Computer Vision") für RGB-Raum:

$r = \frac{R}{R+G+B}$, $g = \frac{G}{R+G+B}$, $b = \frac{B}{R+G+B}$

$$D_1(F_1, F_2) = \sqrt{(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2}$$
$$D_2(F_1, F_2) = \mid r_1 - r_2 \mid + \mid g_1 - g_2 \mid + \mid b_1 - b_2 \mid$$
$$D_3(F_1, F_2) = \mid r_1 - r_2 \mid^2 + \mid g_1 - g_2 \mid^2 + \mid b_1 - b_2 \mid^2$$
$$D_4(F_1, F_2) = \max\{\mid r_1 - r_2 \mid, \mid g_1 - g_2 \mid, \mid b_1 - b_2 \mid\}$$

# (3.3-3) ***HSV-Komponenten***



Stereobildpaar

Farbwert

Sättigung

Helligkeit

## *(3.3-4)* ***weitere Ideen***

- Betrachten von 'Unähnlichkeitskurven' (lokale Minima)
- FFT-Blockmatching
- Ausblenden unmöglicher Suchbereiche (Nutzung von Farbinformationen)

## *(3.4) Bewertung der Tiefenschätzung*

**Problem:** prinzipbedingte Einschränkungen lokaler Korrespondenzanalyse

- intensitätsbasierende Verfahren sehr störanfällig (Shading, Rauschen)

- Wahl des richtigen Disparitätslimits sehr wichtig!

- großer Einfluß der Blockgröße

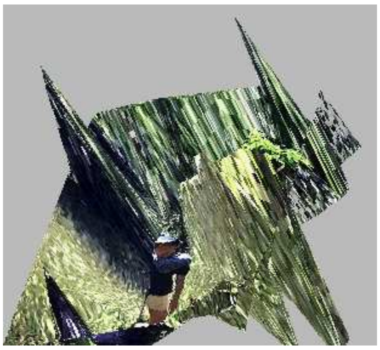
- Farbe als Merkmal nicht überzeugend

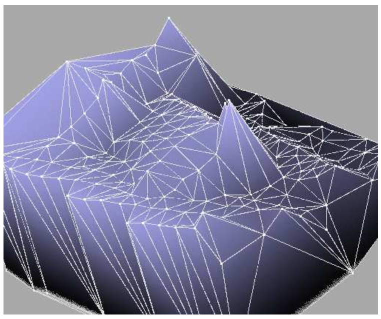
- stabilstes Verfahren: FFT-Blockmatching

## (4) ***Visualisierung***

vielfältige Visualisierungsmöglichkeiten der

- Funktionsweise der Netze und
- des Polygonmodelles

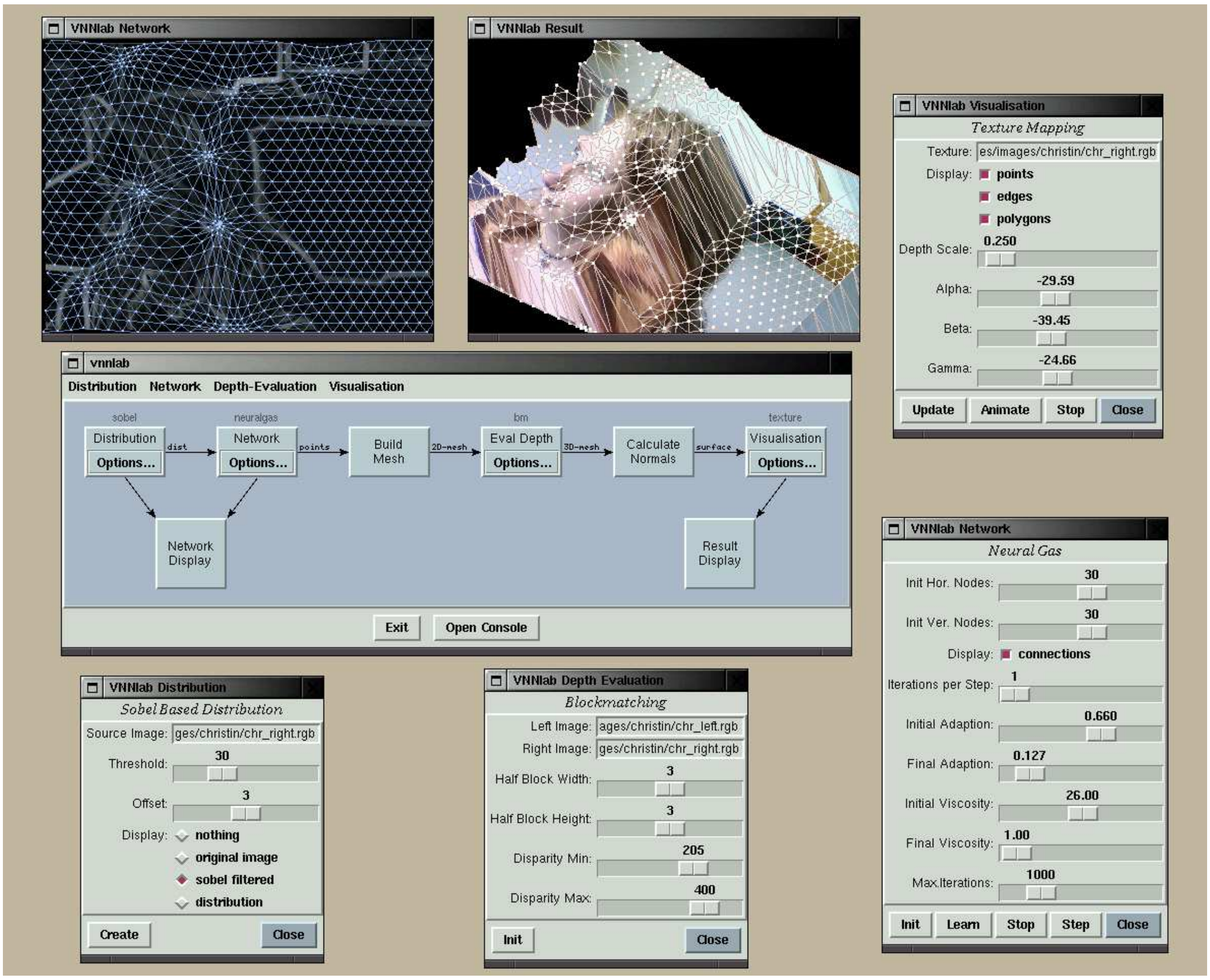




## Standardschnittstellen

- OpenGL
- Tcl/Tk
- Posix-Threads (Thread-Synchronisation mit Posix-Semaphoren)
- modulares Konzept unter Nutzung von Shared Objects

# *(5-2)* ***Screenshot***



## (6) *Zusammenfassung und Ausblick*

- Grundidee zeigt Erfolge
- Schwachstelle ist die Tiefenschätzung
- Programmsystem als Grundlage von Experimenten

**Ausblick**

- inkrementelle Netzwerke
- Netzwerktopologie als Triangulation?
- hybride Korrespondenzanalyseverfahren
- Subpixelgenauigkeit durch FFT-Blockmatching?