Visualisierung 2

Gruppe: Face3d
Pinetz Thomas, Scheidl Harald
SS2016

Zusammenfassung der Publikation

Ziel der in der Publikation [Weon] vorgestellten Implementierung ist es, aus zwei orthogonalen Bildern eines Gesichts ein 3D Modell des Gesichts zu berechnen, auf welches schließlich die Gesichtstextur aufgebracht wird.

Wir gehen hierbei von [Weon] aus, wobei nicht immer alle Einzelheiten klar aus dieser Publikation hervorgehen (z.B. Definition von Face Feature Points). Da [Weon] allerdings großteils nach [Akimoto] vorgeht, bietet eben diese zweitgenannte Publikation gute Anhaltspunkte bei Unklarheiten, da deren Methode auf stets nachvollziehbare Weise erklärt wird.

In der folgenden Graphik ist die Implementierungspipeline nach [Weon] gezeigt, welche einen groben Überblick über die notwendigen Schritte gibt.

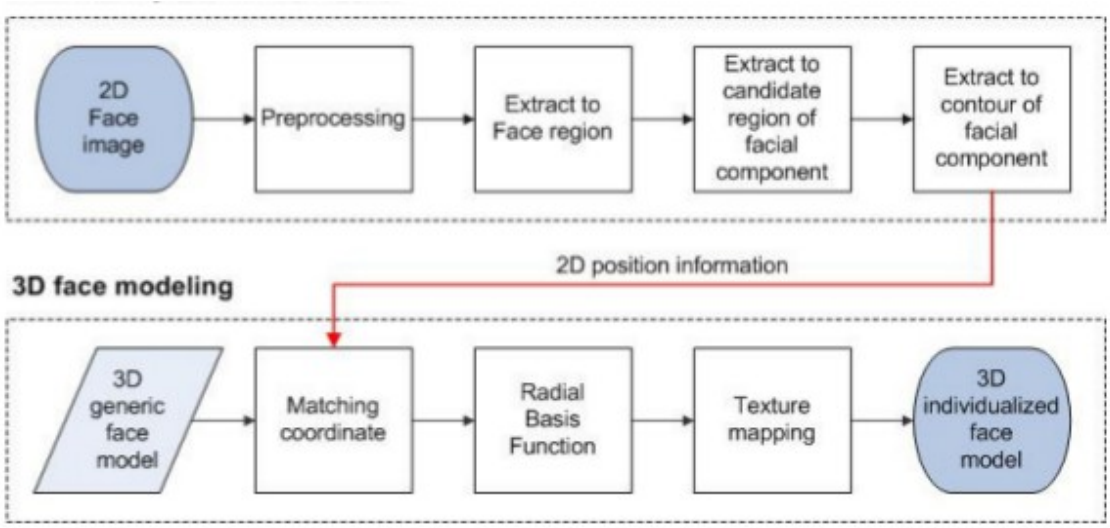

Abb.: Implementierungspipeline [Weon]

Preprocessing: dieser Schritt ist im Artikel nicht näher erklärt, es ist aber davon auszugehen, dass damit Dinge wie Kontrastanpassung oder Rauschunterdrückung gemeint sind.

Extract face regions: hierbei wird im ersten Schritt mit einem Active Contour Algorithmus („Snake“) die Gesichtskontur gesucht. Innerhalb dieses gefundenen Bereichs wird anschließend eine Segmentierung im YCbCr Farbmodell durchgeführt, wobei das Bild in Haut und Nicht-Haut segmentiert wird. Die letztgenannte Kategorie beinhaltet die gesuchten Gesichtskomponenten (Augen, Mund, …).

Extract canditates of facial components: die segmentierten Nicht-Haut Komponenten werden wiederum dem Active Contour Algorithmus unterworfen, wodurch die genaue Lage der Gesichtskompoenten ermittelt wird. Mit einem Satz von einfachen Regeln [Akimoto] erfolgt schließlich die Zuordnung zu den Komponenten, also z.B. Augen oder Mund.

Matching coordinate: nach obigem Schritt ergeben sich somit die 2D Positionen in beiden

orthogonalen Bildern, welche zu einer 3D Position kombiniert werden können. Während die Horizontalkomponente jeweils nur in einem der beiden Bilder zu finden ist (einmal x, das andere mal y Koordinate), ist die Vertikalkomponente z aus beiden Bildern feststellbar. Daher wird für die zusammengesetzt 3D Position wie folgt vorgegangen Pos3D=(x, y, (z_1+z_2)/2)

(Einschub) Ein paar Begriffe um die weiteren Schritte erklären zu können: es ist ein 3D Gesichtsmodell (**3D Mesh**) zu verwenden, welches ein generisches menschliches Gesicht darstellt. Dieses Gesichts besteht aus Punkten (Vertices). Dabei werden eine gewisse Teilmenge dieser Punkte als **Feature Points** definiert. Diese sind einerseits im generischen Gesichtsmodell definiert (z.B. als Label zusätzlich zur Positionsangabe), andererseits über die zuvor beschriebene Bilderkennung in den 2D Bildern auffindbar. Es ist somit möglich, das generische 3D Gesichtsmodell an die jeweilige Person anzupassen.
Bei diesem Teil sei darauf hingewiesen, dass [Weon] einen großen Interpretationsspielraum für dessen Beschreibung lässt, weswegen wir uns hier hauptsächlich an [Akimoto] halten bzw. noch nach weiterer Literatur suchen.

Radial Basis Function: die Anpassung der als Feature Points definierten Vertices im 3D Gesichtsmodell erfolgt anhand der aus den Bildern berechneten 3D Position. Um ein „weiches“ (natürliches) Gesicht als Ergebnis zu bekommen, ist für alle anderen Vertices eine Interpolation durchzuführen. Einfach gesagt bedeutet das, dass die Verschiebung jedes (Nicht-Feature-Point) Vertex hauptsächlich von der Verschiebung der nächsten Feature-Point-Vertices beeinflusst wird. Es gibt verschiedene Methoden um dies zu bewerkstelligen, Radial Basis Functions sind eine davon und in [Weon] vorgeschlagen. Das Ergebnis ist ein personalisiertes 3D Gesichtsmodell.

Texture Mapping: mittels Cylinder-Mapping wird die Textur (2D Bilder) auf dem nun personalisierten 3D Modell aufgebracht.

Implementierung

Wir werden uns bei der Implementierung an die oben vorgestellte Pipeline halten, weswegen wir hier nicht die einzelnen Schritte nochmal aufführen werden, sondern uns auf die verwendeten Frameworks, Bibliotheken und Formate beschränken.

Die Implementierung zerfällt in zwei Komponenten, einerseits in die Bilderkennungskomponente und andererseits in die 3D Modellierungskomponente.

Als Programmiersprache verwenden wir **C++**.
Für die Bilderkennungskomponente verwenden wir **OpenCV** als Bildverarbeitungsbibliothek.
Für die 3D Modellierungskomponente verwenden wir **OpenGL** zur Darstellung.
Das 3D Gesichtsmodell werden wir mittels **Blender** erstellen. Das Labeling der Feature Points ist im besten Fall bereits mit einem Standard 3D – Format möglich, andernfalls werden wir ein eigenes, einfaches Format definieren (z.B. als Zeilen einer Textdatei im CSV-Format: x;y;z;Label)

OpenCV bietet bereits eine rudimentäre GUI Unterstützung, falls wir mehr Möglichkeiten brauchen werden wir eine der bekannten C++ **GUI Frameworks** verwenden (MFC, wxWidgets o.ä.).

Literatur

[Weon] Weon, SunHee, SungIl Joo, and HyungIl Choi. "Individualized 3D Face Model Reconstruction using Two Orthogonal Face Images." *Proceedings of the World Congress on Engineering and Computer Science*. Vol. 1. 2012.

[Akimoto] Akimoto, Takaaki, Yasuhito Suenaga, and Richard S. Wallace. "Automatic creation of 3D facial models." *Computer Graphics and Applications, IEEE* 13.5 (1993): 16-22.