

# Marching Cubes - Erstellung von Polygonmodellen aus Voxelgittern

Matthias Kirschner

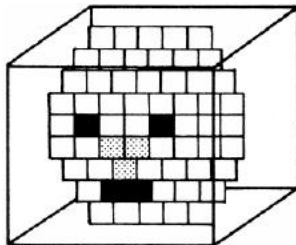
Advanced Topics in Computer Graphics - WS06/07

# Gliederung

- 1 Einführung
  - Problemstellung
  - Marching Cubes im Kontext
- 2 Marching Cubes Algorithmus
  - Überblick
  - Lokalisieren der Oberfläche
  - Berechnung der Oberflächennormalen
- 3 Mehrdeutigkeitsproblem
- 4 Nachteile von Marching Cubes
- 5 Zusammenfassung/Quellen

# Motivation

CT-Scan eines Patienten liefert Voxelgitter aus Dichtewerten



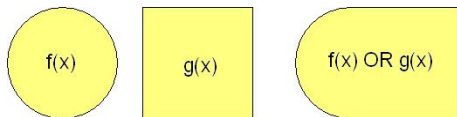
Problemstellung:

- Visualisierung bestimmter im Voxelgitter enthaltener Objekte
- Darstellung in Echtzeit wünschenswert, beispielsweise interaktive Wahl des Blickwinkels

# Allgemeinere Sicht

Objekt durch implizite Funktion  $f(x, y, z) = c$  modelliert.  
Gesucht ist die Oberfläche des Objektes.

- Sensordaten, zum Beispiel CT-Scan (Funktion ist *gesampled*)
- CSG: Verknüpfung einfacher Primitive, die durch implizite Funktionen gegeben sind



# Verfahren zum Darstellen von Volumendaten

- 1 Daten aufschneiden, Schnitt darstellen  
⇒ Zweidimensionale Darstellung
- 2 **Extraktion eines Polygonmodells eines eingebetteten Objektes**
- 3 Volumenrendering
  - Jedes Voxel hat Farb- und Transparenzwert
  - Werte der Voxel auf einem Strahl aggregieren

# Warum Marching Cubes statt Volumenrendering?

Großer Vorteil: Modellierung als Polygonmodell

- Wiederverwendung bekannter Darstellungstechniken
- Ausnutzen effizienter Hardwareunterstützung für Darstellung von Polygonmodellen

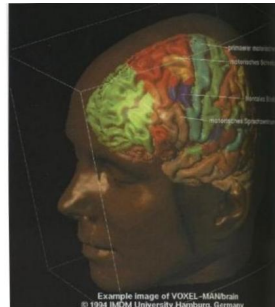
Vorgehensweise:

- Berechnen des Polygonmodells „offline“
- Interaktive Darstellung des vorberechneten Modells mit Standardhardware/-techniken

# Kombinierte Verfahren

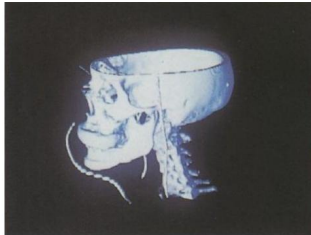


Example image of VOXEL-MAN/brain  
© 1994 ACM University Hamburg, Germany



Example image of VOXEL-MAN/brain  
© 1994 ACM University Hamburg, Germany

# Vergleich von MC und Volume Rendering



# Ein- und Ausgabe

- Eingabe
  - Voxelgitter:  $D(i, j, k)$  = Dichtewerte des Voxels  $(i, j, k)$
  - Schwellwert  $t$ , der Oberfläche identifiziert
    - $D(i, j, k) \geq t$  - Voxel liegt im Objekt
    - $D(i, j, k) < t$  - Voxel liegt außerhalb des Objektes
- Ausgabe
  - Dreiecke eines Polygonmeshs
  - Oberflächennormale für jeden Meshknoten

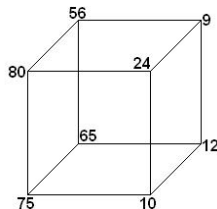
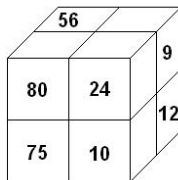
# Prinzip: Divide & Conquer

- Lösen des Problems in Voxel-Würfeln aus acht Voxeln
  - 1 Oberfläche des Objektes innerhalb des Würfels berechnen
    - 1 Feststellen der Oberflächenstruktur
    - 2 Berechnung der Meshknoten
  - 2 Berechnung der Oberflächennormalen der Meshknoten
- Zusammensetzen der Oberflächenstücke ergibt Polygonmesh



# Marching Cube

- „logischer“ Würfel wird in das Voxelgitter gelegt
- Jeder Knoten des Würfels liegt auf einem Voxel
- Dichtewert eines Würfelknotens = Dichtewert des entsprechenden Voxels
- Würfel wandert durch das Voxelgitter (*Marching*)



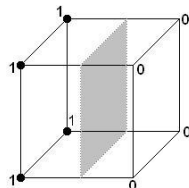
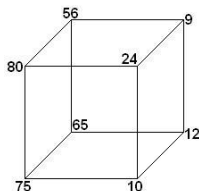
# Zuweisen von Werten

Jedem Knoten  $v$  des Würfels wird ein Wert  $\in \{0, 1\}$  zugewiesen:

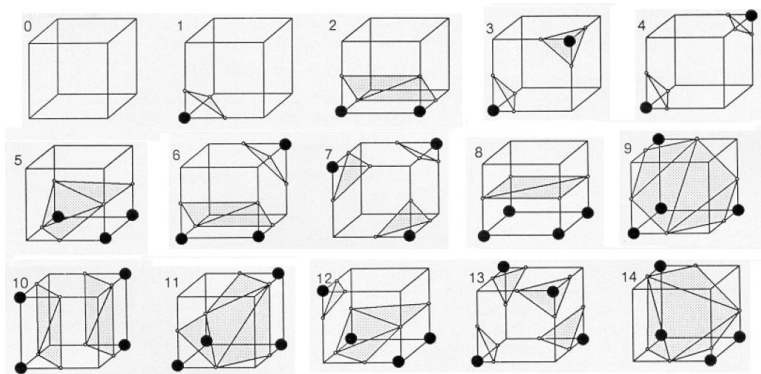
- 0 falls  $D(v) < T$  (Knoten außerhalb des Objektes)
- 1 falls  $D(v) \geq T$  (Knoten innerhalb des Objektes)

Dies führt zu  $2^8 = 256$  verschiedenen Fällen (*Konfigurationen*).

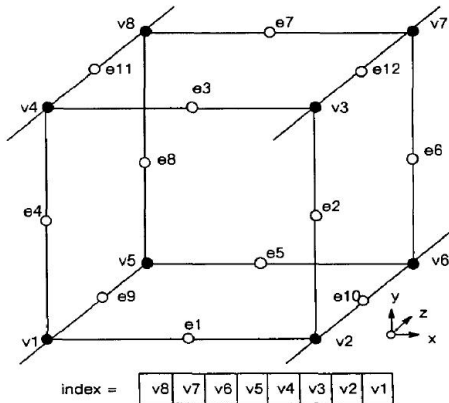
Beispiel:  $t = 50$



# 15 unterschiedliche Würfelklassen

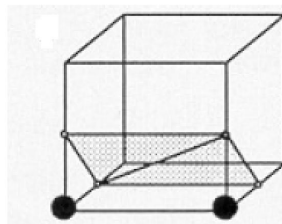
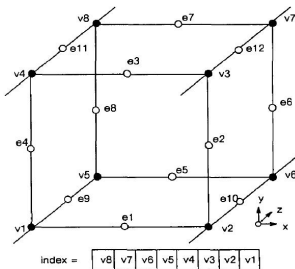


# Berechnung des Würfelindex



# Verwendung des Würfelindex

- Würfelindex dient als Zeiger in eine Kantentabelle
- Schnittpunkte werden für alle Kanten des Tabelleneintrages berechnet

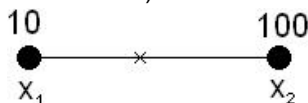


Konfiguration 3

$$\text{Edges}[3] = \{2, 4, 9, 10\}$$

# Berechnung des Meshknoten

- Bekannt: Von Oberfläche geschnittene Kante
- Gesucht: Schnittpunkt = Knoten des Polygonmeshes
- Schnittpunkt wird anhand Dichtewerte der Würfelknoten linear interpoliert
- Beispiel (Schwellwert  $t = 50$ )



$$10\alpha + 100(1 - \alpha) = 50$$

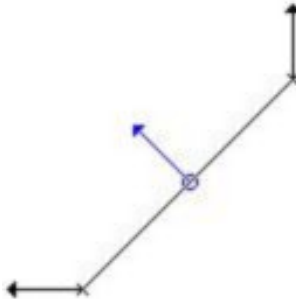
$$\Rightarrow \alpha = \frac{5}{9}$$

Gesuchter Schnittpunkt:

$$x^* = \alpha x_1 + (1 - \alpha)x_2$$

# Berechnen der Normalen der Mesh-Knoten

- Schätzen des Gradienten an jedem der acht Würfelknoten
- Lineare Interpolation des Gradienten an den Mesh-Knoten
- Normierung liefert Normale



# Schätzen des Gradienten an einem Würfelknoten

- Daten sind gesampelt: Nur Schätzung möglich
- Betrachten Würfelknoten auf Dichtewert  $(i, j, k)$  mit Dichtewert  $D(i, j, k)$   
 $\Delta x, \Delta y, \Delta z$  sei Würfellänge in entsprechender Richtung

$$G_x(i, j, k) = \frac{D(i+1, j, k) - D(i-1, j, k)}{\Delta x}$$

$$G_y(i, j, k) = \frac{D(i, j+1, k) - D(i, j-1, k)}{\Delta y}$$

$$G_z(i, j, k) = \frac{D(i, j, k+1) - D(i, j, k-1)}{\Delta z}$$

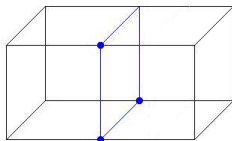
# Marching Cubes im Überblick

Für jeden logischen Würfel

- (1) Berechne Index des Würfels anhand Dichtewerte
- (2) Erfrage mittels Index Liste der geschnittenen Würfelkanten
- (3) Berechne Meshknoten mittels linearer Interpolation
- (4) Approximiere die Normalen an Würfelknoten
- (5) Interpoliere die Normalen der Meshknoten
- (6) Ausgabe der gefundenen Knoten und der Normalen

# Kohärenz

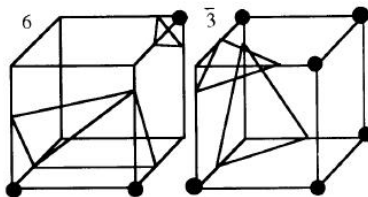
Zwei aufeinanderfolgende Würfel teilen sich gemeinsame Seite



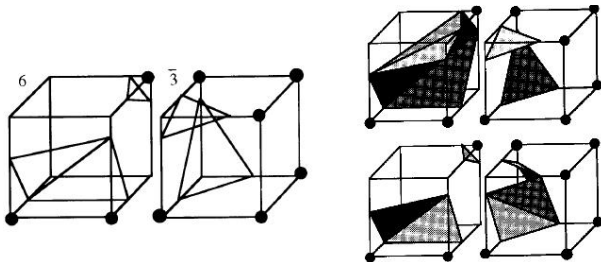
- Bereits berechnete Schnittpunkte können wiederverwendet werden!
- Analog: Kohärenz in  $y$ - und  $z$ -Richtung
  - Theorie: Nur drei neue Berechnungen bei inneren Würfeln
  - Praxis: Speicheraufwand bei Kohärenzausnutzung berücksichtigen

# Löcher in der Oberfläche

Es können Löcher bei der Verbindung einzelner Oberflächenstücke entstehen



# Mehrdeutigkeitsproblem

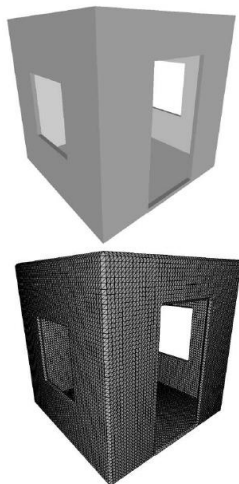


# Auflösen von Mehrdeutigkeiten

- Löcher mit Dreiecken „stopfen“
  - ⇒ Kann dazu führen, dass drei Dreiecke eine Kante teilen!
- Andere Algorithmusvariante verwenden (z.B. *Tetraeder* statt *Würfel*)
- Verfahren zur Mehrdeutigkeitsauflösung einbauen

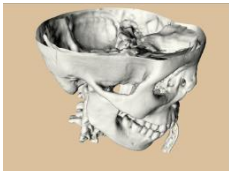
# Hohe Anzahl von Polygonen (1)

- MC arbeitet lokal und ohne Wissen über gesuchte Struktur
- ⇒ Viele Polygone auch für einfach darstellbare Objekte
- Anzahl der Polygone abhängig von
  - Größe des Voxelgitter
  - „Wüfelbreite“
- Realistisches Ergebnis: 2000k Dreiecke (bei 100 Scheiben à  $1024 \times 1024$ )

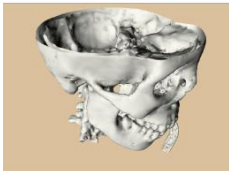


# Hohe Anzahl von Polygonen (2)

- Effizienzvorteil von MC geht bei hoher Polygonzahl verloren
- Mögliche Lösungen
  - Algorithmus zur Vereinfachung des Polygonmodells nachschalten
  - Auflösung des Datensatzes verkleinern (Averaging)



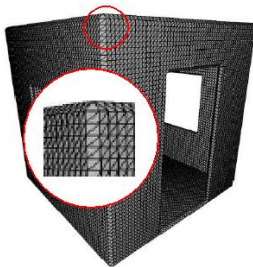
Full Resolution  
(569K Gouraud shaded triangles)



75% decimated  
(142K Gouraud shaded triangles)

# Probleme bei spitzen Details

- Original MC geht von glatten Oberflächen aus
- Artefakte des Polygonmeshes insbesondere an spitzen Kanten



# Begrenzte Interaktivität

- Schnelle Darstellung aufgrund Polygonmodells möglich
- MC berechnet Modell **vor** der Darstellung
  - ⇒ Angezeigt werden kann nur, was vorberechnet wurde

# Original MC ungeeignet für adaptive Gitter

- Voxel nicht immer in uniformen Gittern gegeben
- Adaptive Strukturen wie Octrees sind ökonomischer
- Neuere Techniken übertragen MC auf adaptive Gitter, zum Beispiel *Dual Marching Cubes*

# Zusammenfassung

- MC als Alternative oder Ergänzung zum direkten Volumenrendering
  - Vorberechnung eines Polygonmeshes
  - Darstellung des Meshes mit Standardverfahren
- Divide and Conquer-Verfahren
- Mehrdeutigkeiten müssen gelöst werden!
- Nachteile des Verfahrens
  - Hohe Polygonzahl
  - Probleme bei spitzen Details
  - Begrenzte Interaktivität
  - Beschränkt auf Gitter

# Quellen

- **W. Lorensen, H. Cline: *Marching Cubes: A high resolution 3D surface construction algorithm*, SIGGRAPH 1987.**
- A. Watt: *3D Computergraphics*, Addison Wesley, 3rd ed., 2000.
- G. Nielson, B. Hamann: *The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes*, IEEE Visualization 1991. pages 83-91, 1991.
- W. Schroeder, J. Zarge, W.Lorensen: *Decimation of Triangle Meshes*, SIGGRAPH 1992.
- S. Schaefer, J. Warren: *Dual Marching Cubes: Primal Contouring of Dual Grids*, Computer Graphics Forum (24), 2005.
- Tutorial mit Java-Applet unter:  
<http://www.essi.fr/~lingrand/MarchingCubes/accueil.html>