Nichtrealistische Darstellung von Gebirgen mit OpenGL

Großer Beleg

Torsten Keil

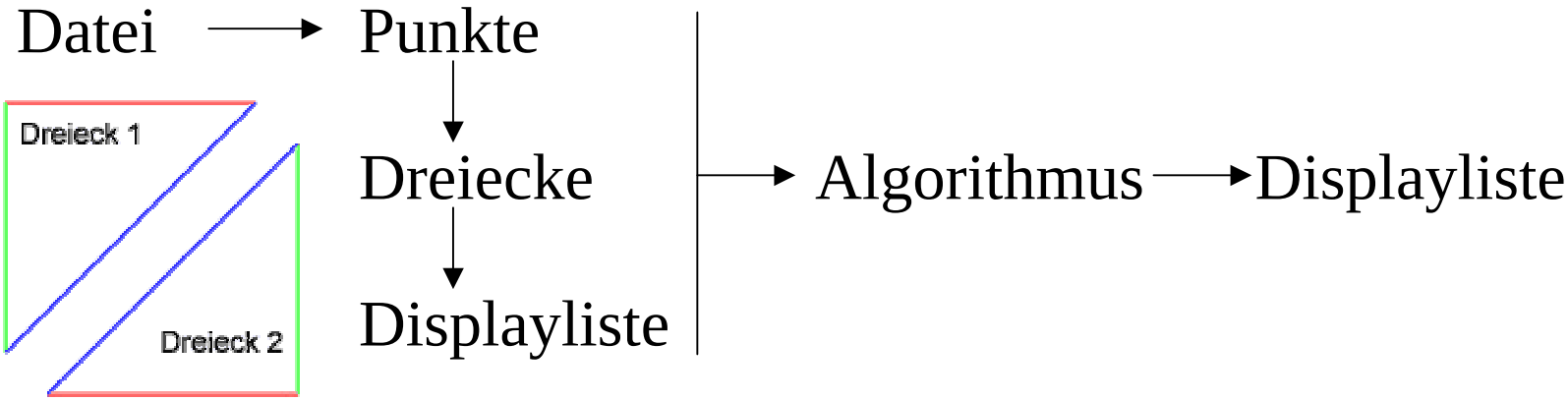
Betreuer: Prof. Deussen

Zielstellung

- Entwicklung eines Algorithmus, der die 3D-Daten einer Geometrie in eine nichtrealistische Darstellung überführt

Vorgehensweise

- Eingabe: Höhendaten im *.pgm-Format
- Ausgabe: Grafikdarstellung mit OpenGL



Algorithmus (1)

- Der Algorithmus greift auf verschiedene Daten zu:
  - Die Daten der Struktur (Punkte, Kanten und Dreiecke)
  - Den Z-Buffer des Ausgabefensters
  - Das Bild des Ausgabefensters

Algorithmus (2)

- Es werden folgende Informationen ermittelt:
  - Pixel und Tiefenwert: Kanten
  - Pixel und Farbwert: Index des Dreiecks
  - Pixel und Farbwert: Kante des Dreiecks
- Ergebnis ist eine Liste aller Kanten, die gezeichnet werden müssen

Z-Buffer (1)

- Zu Beginn des Algorithmus wird der Z-Buffer ausgelesen
- Er repräsentiert für jedes Pixel des Ausgabefensters den Tiefenwert
- Er dient zum Ermitteln der Ränder der Geometrie
- Er dient zum Ermitteln von Kanten innerhalb der Geometrie

Z-Buffer (2)

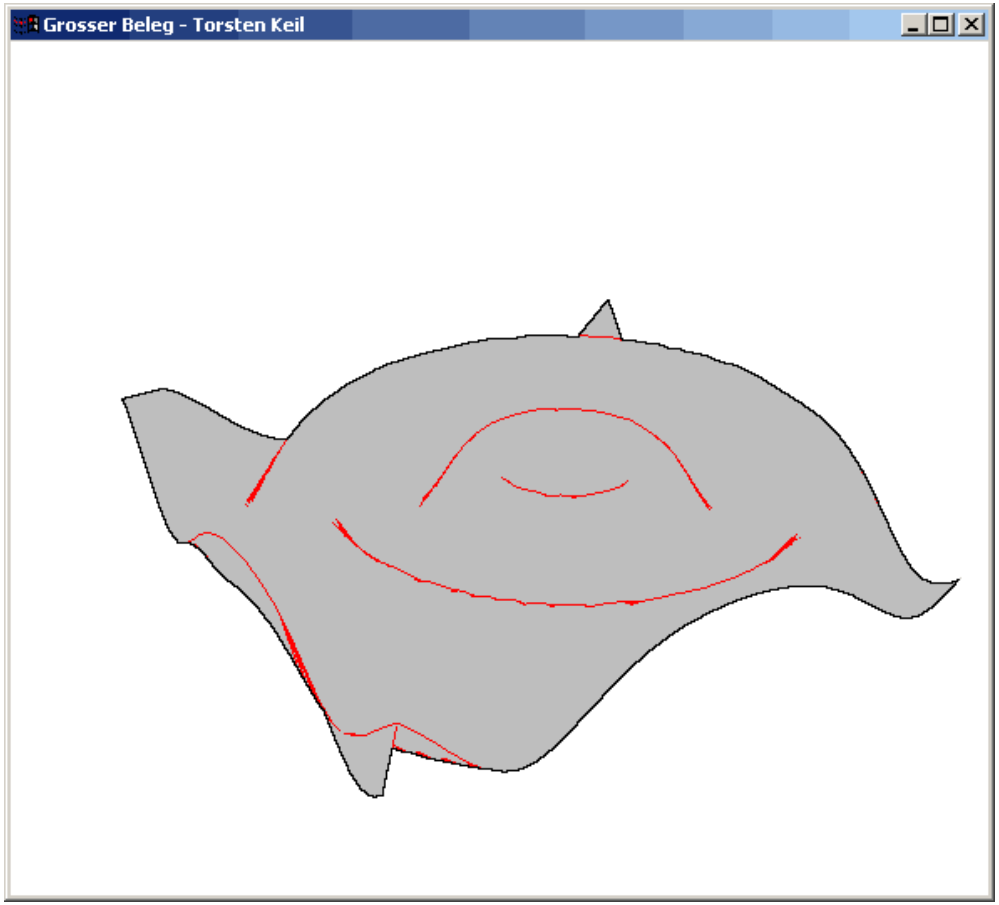
- Ränder der Geometrie
  - Tiefenwerte kleiner 1.0
    →Pixel gehört zur Geometrie

						0	0	0	
			0	0	0	0	1	2	
	0	0	0	1	2	3	4	5	
0	0	1	2	4	5	6	7	8	
0	1	3	5	7	8	9	9	9	
0	2	5	8	9	9	9			
0	3	6	9	9					

- Kanten innerhalb der Geometrie
  - Vergleiche Tiefenwerte benachbarter Pixel
  - Differenz größer als Schwellwert und Tiefenwert ist Minimum
    →Pixel ist Kante innerhalb der Geometrie

Z-Buffer (3)

- Weiße Pixel: kein Objekt
- Schwarze Pixel: Randpixel der Geometrie
- Graue Pixel: Pixel inmitten der Geometrie
- Rote Pixel: Kanten innerhalb der Geometrie

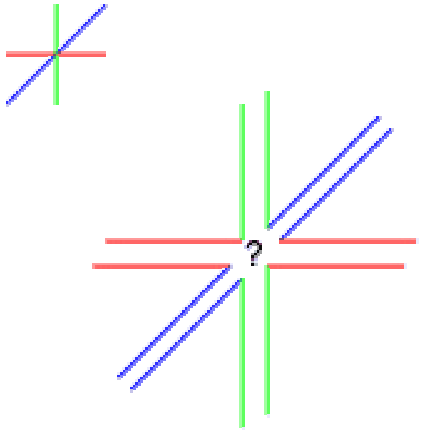


Dreiecke

- Jedem Dreieck wird beim Erstellen der Displayliste eine eindeutige Farbe zugeordnet
- Von jedem Pixel der Geometrie kann so der Dreiecks-Index ermittelt werden
- Welche der 3 Kanten des Dreiecks ?

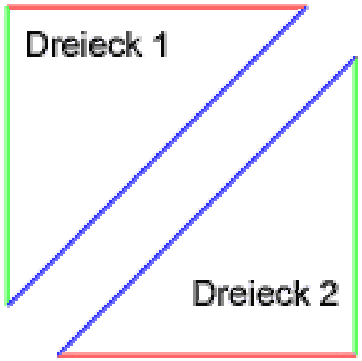
Kanten (1)

- Probleme beim Erkennen der Kanten:
  - Jeder Eckpunkt gehört zu 6 Dreiecken
  - Jede Kante gehört zu 2 Dreiecken



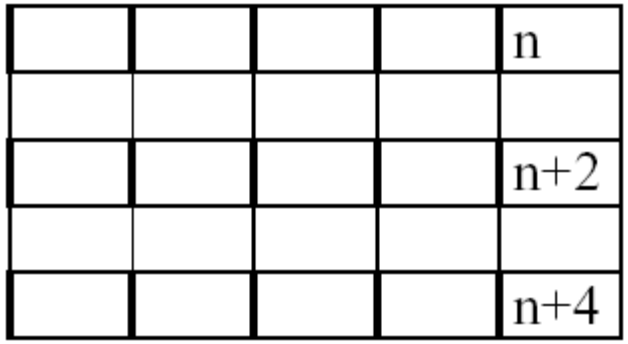
Kanten (2)

- Der Algorithmus
  - behandelt die Dreiecke „1“ und „2“ getrennt
  - behandelt die Kanten (horizontal, vertikal und diagonal) jeweils getrennt
  - zerlegt jeden Kantentyp in 2 Teilschritte, so dass benachbarte Kanten nicht in einem Schritt gezeichnet werden
- Der Algorithmus setzt sich so aus 12 Teilschritten zusammen

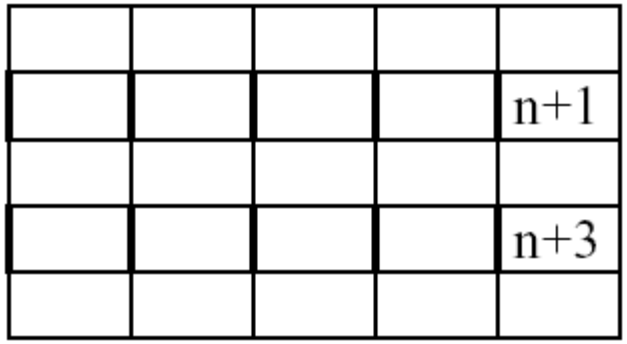


Kanten (3)

- In jedem Teilschritt werden die Kanten gezeichnet und als Farbinformationen gespeichert
- Alle potentiellen Kantenpixel (rot und schwarz) werden getestet, ob sie auf einer Kante liegen



Vertikale Kanten – erster Teilschritt



Vertikale Kanten – zweiter Teilschritt

Kanten (4)

- Das Ergebnis der bisherigen Schritte ist noch unbefriedigend, da
  - einzelne Kanten nicht erfasst werden bzw.
  - Kanten zuviel erfasst werden
- Es werden an die eigentliche Kantenerfassung zwei zusätzliche Schritte angeschlossen:
  - Feinarbeit hinsichtlich Kantenerfassung
  - Intelligente Nachbereitung der ermittelten Kanten

Kanten – Feinarbeit

- Durch numerische Ungenauigkeiten kommt es zu Abweichungen der Position (Kante eines Dreiecks, einzelne Linie) von einem Pixel
- Diese Abweichungen müssen erkannt und korrigiert werden
  - Zu jedem Pixel (rot oder schwarz) wird erfasst, ob ihm eine Kante zugeordnet werden konnte
  - Ist dies nicht der Fall, wird eine zusätzliche Untersuchung der benachbarten Pixel durchgeführt
  - Wird eine Kante ermittelt, so wird sie mit in die Kantenliste aufgenommen

Intelligente Nachbereitung

- Als erstes werden die Kanten „normiert“, d.h. alle Kanten des „Dreiecks 2“ werden auf Kanten des „Dreiecks 1“ abgebildet
  - dies geschieht durch Addition der Kantenzähler
- Es kann nun nach Lücken innerhalb eines Kantenzuges und vereinzelten Kanten gesucht werden
- Diese „Fehler“ werden behoben, indem
  - alle einfachen Lücken entfernt werden
  - alle einzelnen Kanten entfernt werden

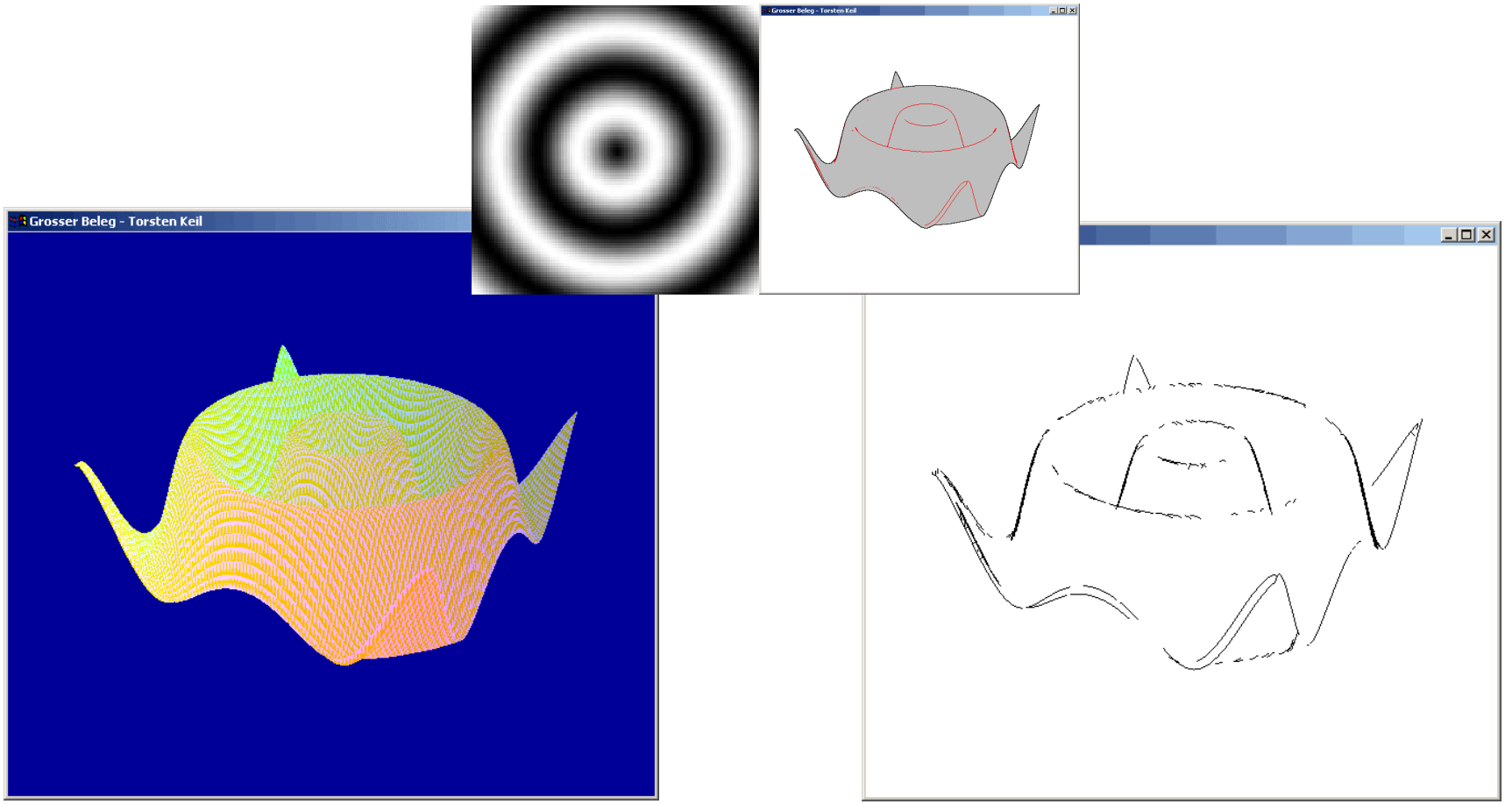
Zusammenfassung

- Schwächen
  - Ungenügende Ergebnisse bei geringer Genauigkeit
  - Sehr viele Dreiecke verschlechtern das Laufzeitverhalten
  - Die Methode der Farbindizes begrenzt das Verfahren
- Stärken
  - ✓ Eine Erhöhung der Bildschirmauflösung führt zu Qualitätssteigerung
  - ✓ Eine flexiblere Unterteilung der Quadrate kann zu einem besseren Ergebnis führen

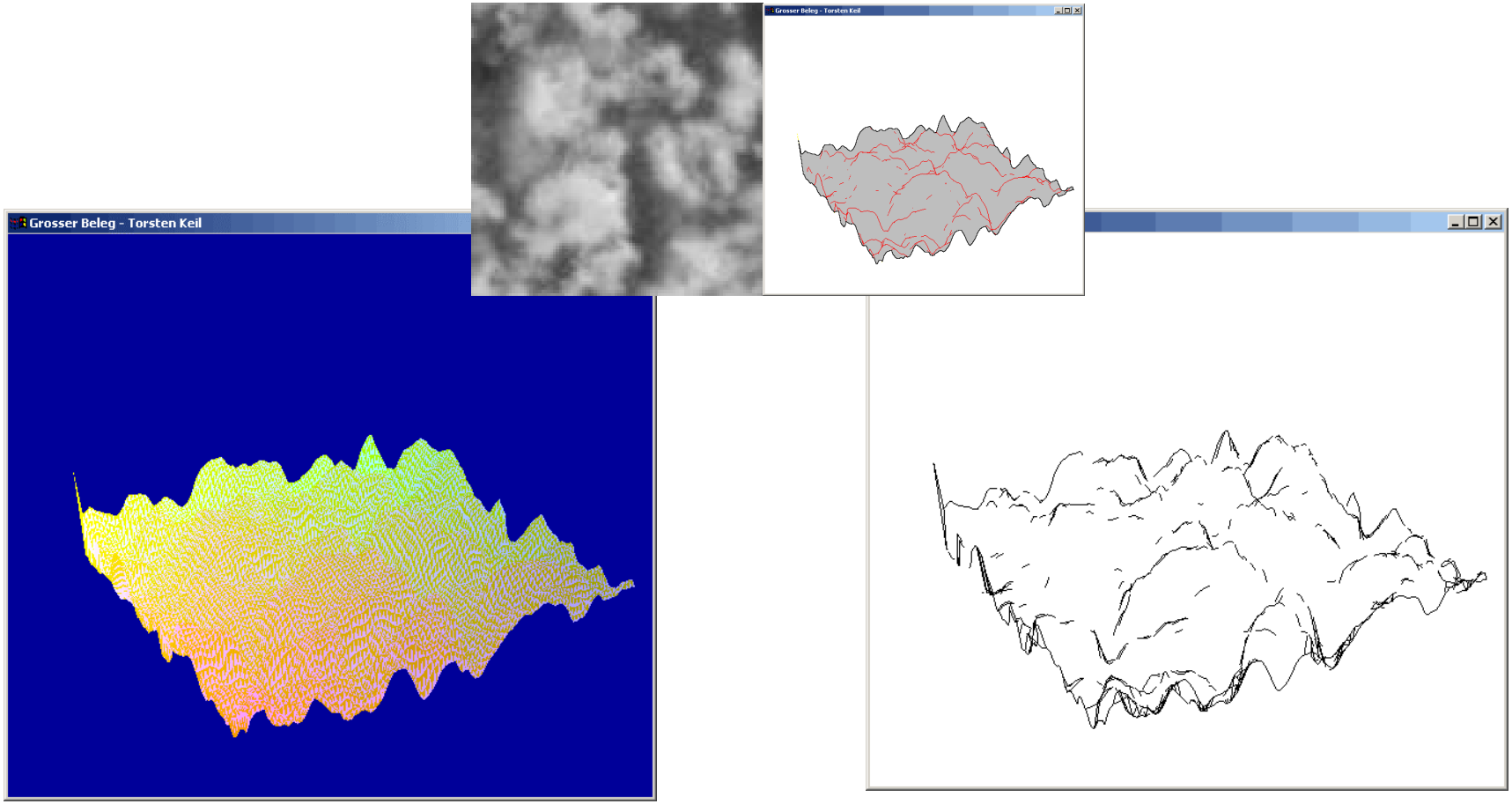
Ausblick

- Eine Implementierung für ein beliebiges Dreiecksnetz
  - sollte die Leistung erhöhen
  - sollte die Geometrie besser wiedergeben können
- Eine Optimierung des Laufzeitverhaltens
- Arbeiten des Algorithmus im Hintergrund

Beispiele (1)



Beispiele (2)



Informationen

- Torsten Keil:
  Großer Beleg „Nichtrealistische Darstellung von Gebirgen mit OpenGL“
  http://www.inf.tu-dresden.de/~tk19/studium/stuff/GroßerBeleg_TorstenKeil.pdf