

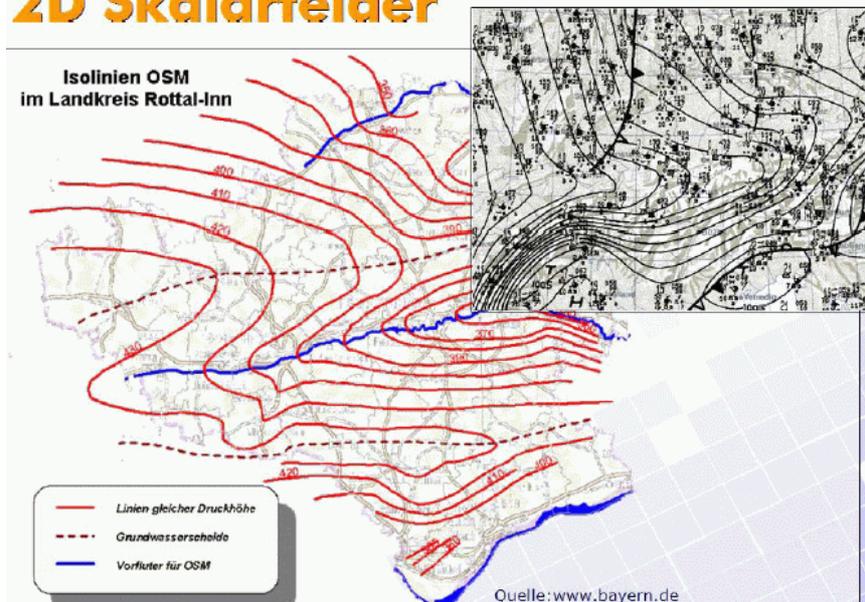
## 7. 2D-Skalarfelder

(nach Rezk-Salama, o.J.)

Definitionsbereich: 2-dimensional

(Beobachtungs-) Werte: skalar (1 reelle Zahl je Punkt)

### 2D Skalarfelder



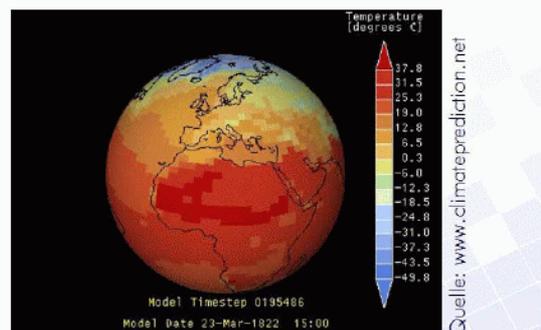
### 2D Skalarfelder:

- selten analytisch gegeben:  $s = f(x, y)$
- üblicherweise als diskrete Daten:  
 $(x_i, y_i) \mapsto f_i \quad i \in [0, \dots, N]$

### Visualisierungsmöglichkeiten:

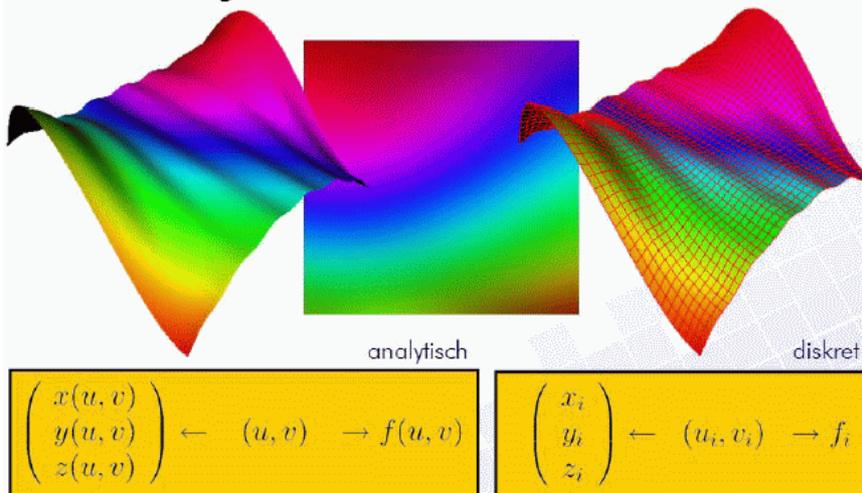
- Color Plots (Farbcodierung)
- Surface Plots (Oberflächendarstellung)
- Isolinien

- beachte:
- „2D“ heißt nicht zwangsläufig „kartesische Koordinaten“!
  - 2D Skalarfelder müssen nicht **eben** sein!



Beispiel: Temperaturverteilung über dem Globus

auf beliebigen Oberflächen in 3D



## Farbcodierung (Colorplots)

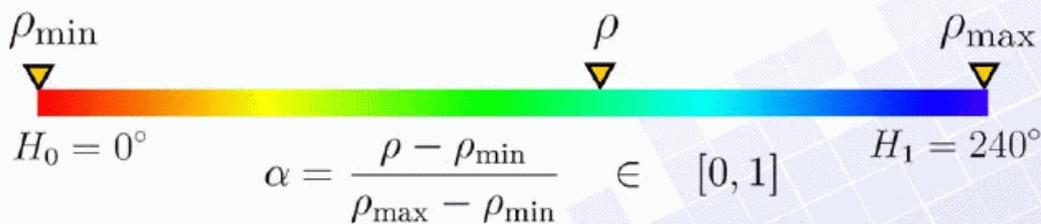
- Spezifiziere eine Farbtabelle

d.h. eine Abbildung: Skalarwert  $s \rightarrow$  Farbwert

Beispiel: Gegeben 2D Druckverteilung

$$\rho_{\min} \leq \rho \leq \rho_{\max}$$

Lineare Abbildung im HSV-Farbmodell auf Hue



$$H(\rho) = \alpha H_1 + (1 - \alpha) H_0$$

bei uniformen, rektlinearen 2D-Gittern:

- trivial bei feinen Gittern:

interpretiere jeden Abtastpunkt als 1 Pixel  
Skalarwert dort bestimmt Farbe des Pixels  
per Farbtabelle

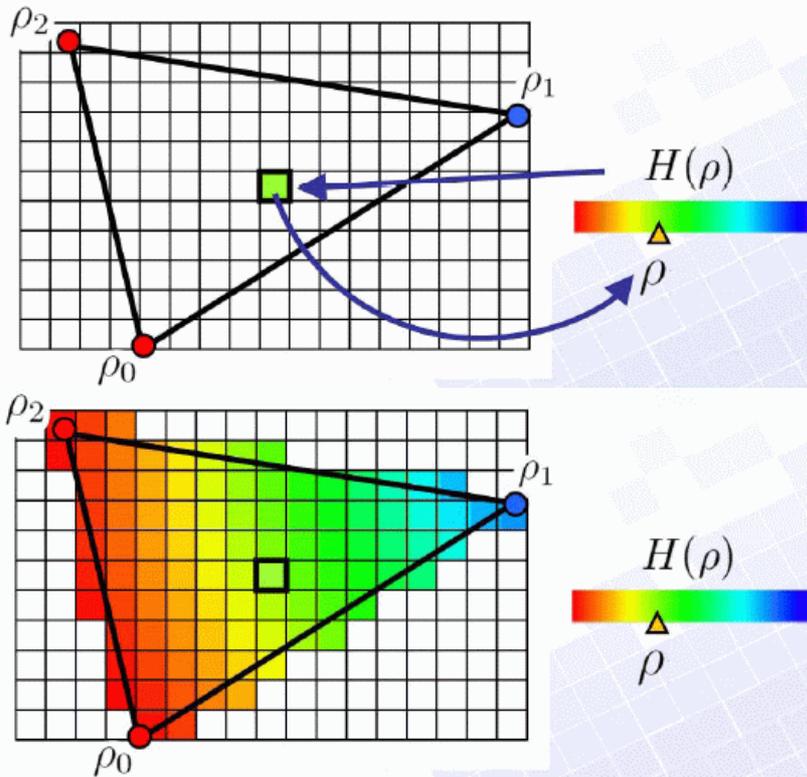
- im allgemeinen Fall: Rasterisierung erforderlich

bei curvilinearen und unstrukturierten 2D-Gittern:

- Rasterisierung erforderlich

## Rasterisierung – Grundidee:

- Zerlege das Dreieck in einzelne Pixel
- Interpoliere für jeden Pixel den Skalarwert
- Bestimme die Farbe des Pixels anhand des Skalarwerts



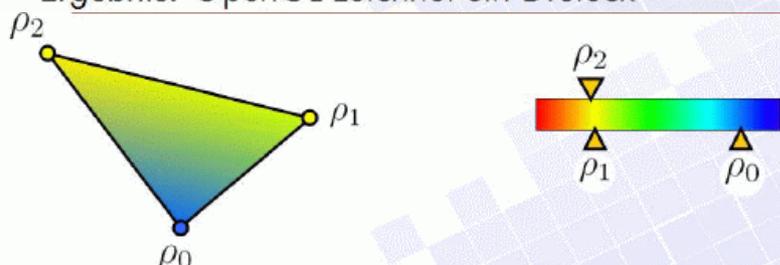
## Umsetzung: einfache Implementierung

- *Nutzung von Graphik-Hardware:*  
Hardware übernimmt die Rasterisierung der Dreiecke

Beispiel OpenGL:

```
00  glBegin(GL_TRIANGLES);  
01      glColor3d(R[i], G[i], B[i]);  
02      glVertex2d(x[i], y[i]);  
03      glColor3d(R[j], G[j], B[j]);  
04      glVertex2d(x[j], y[j]);  
05      glColor3d(R[k], G[k], B[k]);  
06      glVertex2d(x[k], y[k]);  
07  glEnd();
```

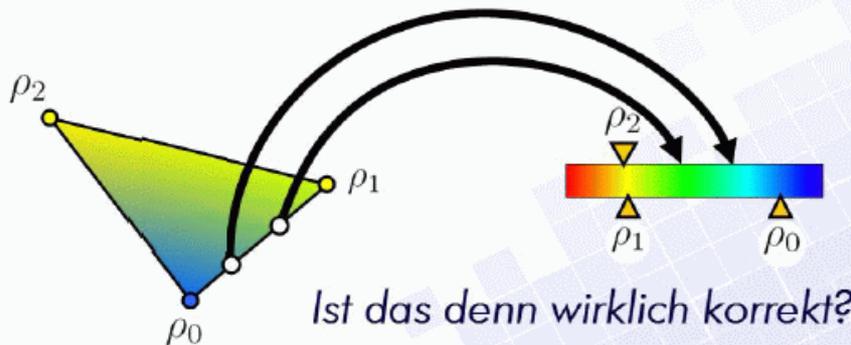
Ergebnis: OpenGL zeichnet ein Dreieck



Problem: OpenGL bzw. die Grafikkarte interpoliert hier automatisch einfach die Farbwerte zwischen den Ecken.

- Betrachte z.B. Punkte entlang der Kanten:

Farbinterpolation ist nicht korrekt!  
Es muss der Skalarwert interpoliert werden!

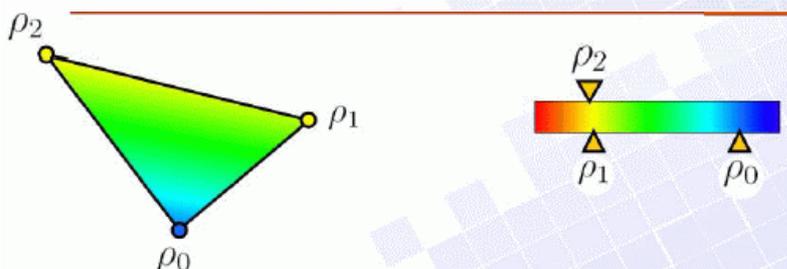


Verbesserte Implementierung:  
Skalarwerte interpolieren, zugehörige Farben pixelweise aus Lookup-Table entnehmen.

- Verwendung von 1D Texturen:  
Speichere die Farbtabelle in einer 1D Textur:

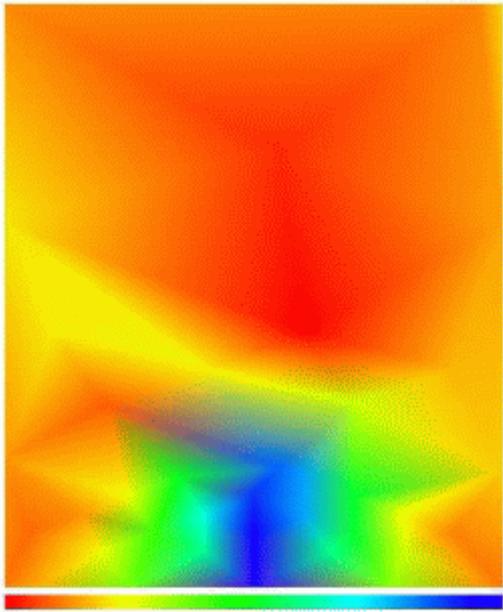
Beispiel OpenGL:

```
00 glBegin(GL_TRIANGLES);  
01 glTexCoord1d(Rho[i]);  
02 glVertex2d(x[i], y[i]);  
03 glTexCoord1d(Rho[j]);  
04 glVertex2d(x[j], y[j]);  
05 glTexCoord1d(Rho[k]);  
06 glVertex2d(x[k], y[k]);  
07 glEnd();
```

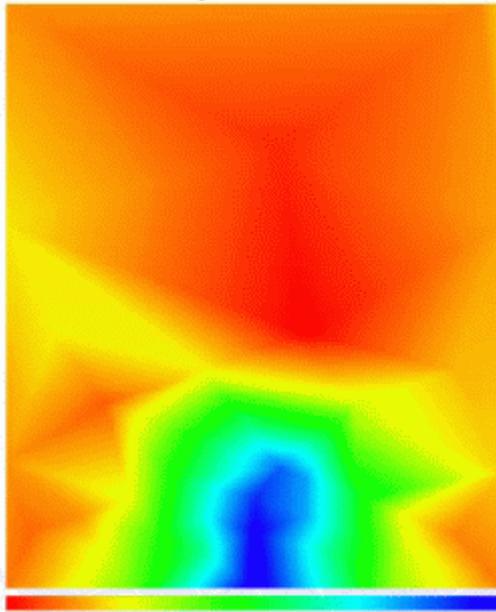


Vergleich:

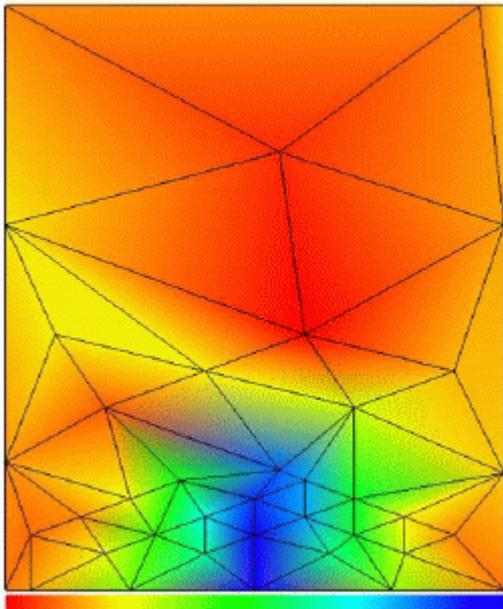
Farbinterpolation



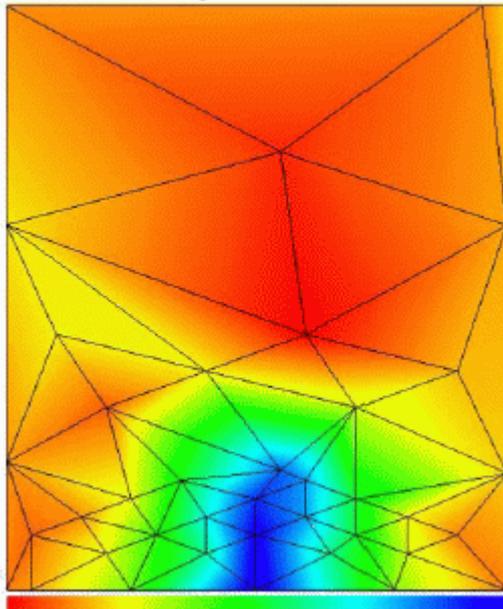
Texturinterpolation



Farbinterpolation



Texturinterpolation



## Fazit:

### Farbkodierung von Skalarfeldern:

#### ● Einfache Implementierung:

- Bestimme Farbwerte für die Eckpunkte der Zellen (mittels Farbtabelle)
- Interpoliere die Farbwerte für jeden Pixel im Inneren der Zelle

Per-Vertex  
Operations

„Gouraud-Shading“

#### ● Aufwändigere Implementierung:

- Interpoliere den Skalarwert für jeden Pixel im Inneren einer Zelle
- Bestimme Farbwerte für jeden interpolierten Skalarwert (Farbtabelle)

Per-Pixel  
Operations

„Phong-Shading“

## Zusammenfassung

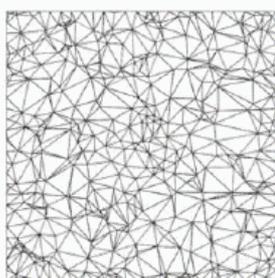
### Color Plots / Farbcodierung

- Spezifiziere eine Farbtabelle, die den Skalarwert auf einen Farbwert abbildet (= *Transferfunktion*)
- Unstrukturierte und Curvilineare Gitter müssen rasterisiert werden
- *Rasterisierung* mit Grafik-Hardware
  - Einfach aber inkorrekt: Farbinterpolation (*Gouraud*)
  - Korrekte Interpolation des Skalarwerts mit 1D Texturen

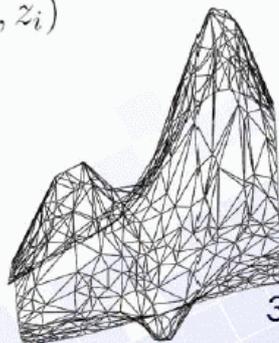
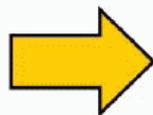
### Surface Plots

#### ● Oberflächendarstellung:

Interpretiere das Skalarfeld als Höhenfeld und zeichne eine 3D Oberfläche  $(x_i, y_i) \mapsto (x_i, y_i, z_i)$



2D

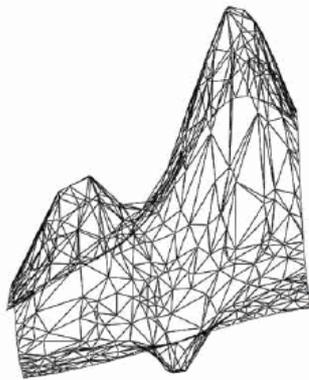


3D

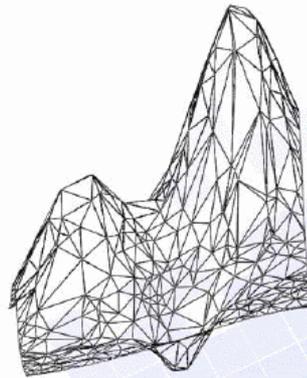
$$\alpha = \frac{\rho - \rho_{\min}}{\rho_{\max} - \rho_{\min}}$$

$$z_i = \alpha z_{\max} + (1 - \alpha) z_{\min}$$

## ● Liniendarstellungen



„wireframe“



„hidden line removal“

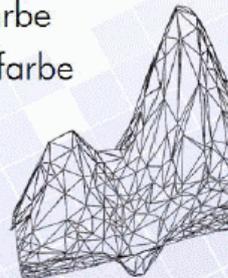
Hidden Line Removal (vgl. Skript zum Projektstudium Computergrafik und Softwaretechnik, Wintersem.):

### HLR Algorithmen für Vektorgrafik:

- z.B. Robert's Algorithm, Appel's Algorithm  
(siehe Foley, et al: *Computer Graphics, Principles and Practise*)

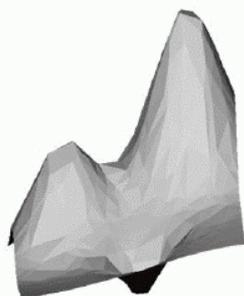
### HLR für Rastergrafik:

- OpenGL-Tricks:
  - Zeichne die Linien in Vordergrundfarbe
  - Zeichne die Flächen in Hintergrundfarbe
  - Der Z-Buffer besorgt den Rest.  
(siehe auch `glPolygonOffset`)

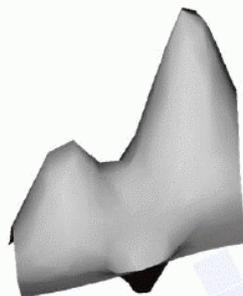


Intuitiverer Tiefeneindruck durch Rendering der Fläche:

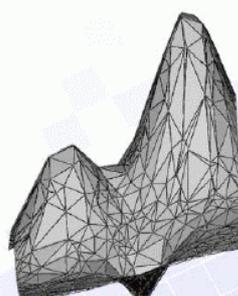
- Flächendarstellungen mit *Shading*  
(Lokale Beleuchtung)



flat shading



smooth shading  
Gouraud-Shading



Kombination mit  
wireframe

# Lokale Beleuchtungsrechnung

## Phong-Beleuchtungsmodell: 3 Summanden

### 1. *Ambiente* Beleuchtung



Konstanter Term

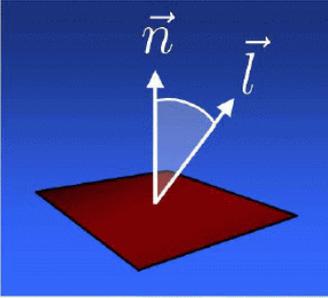
- Approximation indirekter Beleuchtung (Mehrfachreflexion)
- Aufhellen von Schatten

### 2. *Diffuse* Reflexion



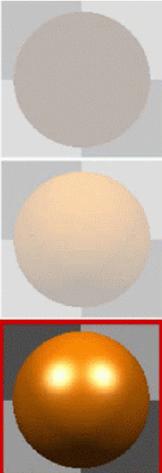
Konstanter Term

Abhängig vom Einfallswinkel des Lichts



- gleichmäßige Reflexion in alle Richtungen
- matte Oberflächen
- „Lambert’sche“ Reflexion

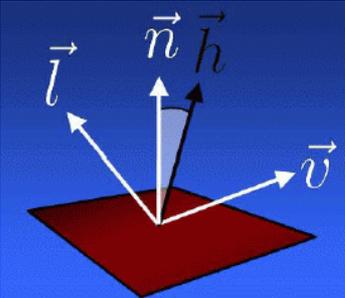
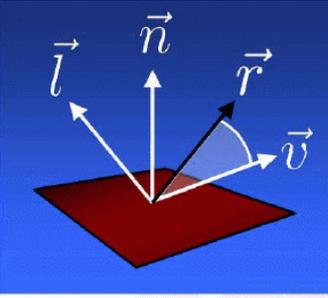
### 3. *Spekulare* (spiegelnde) Reflexion



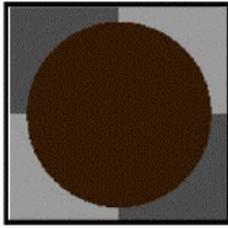
Konstanter Term

Abhängig vom Einfallswinkel des Lichts

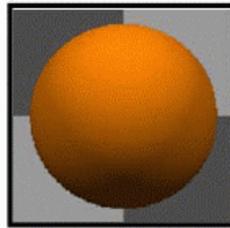
Abhängig vom Lichtrichtung und Blickrichtung



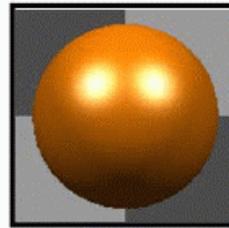
## Blinn-Phong Beleuchtungsmodell:



ambient



diffuse  
(Lambert)



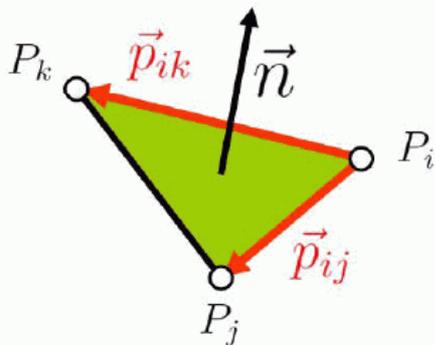
spekular  
(spiegelnd)

$$I = I_a + I_d (\vec{n} \circ \vec{l}) + I_s (\vec{n} \circ \vec{h})^r$$

(hier  $\circ$  = Skalarprodukt)

Wie bestimmt man den Oberflächen-Normalenvektor?

● Die Normale für ein einzelnes Dreieck:



$$\vec{p}_{ij} = P_j - P_i$$

$$\vec{p}_{ik} = P_k - P_i$$

Das normalisierte  
Kreuzprodukt ergibt  
den Normalenvektor

$$\vec{n} = \frac{\vec{p}_{ij} \times \vec{p}_{ik}}{\|\vec{p}_{ij} \times \vec{p}_{ik}\|}$$

einfachste Art des Shading: jedes Polygon einheitlich färben, entsprechend den Beleuchtungsverhältnissen an einem repräsentativen Punkt dieses Polygons. D.h.:

*flat shading:*

Bestimme einen  
Normalenvektor  
für jedes Dreieck



OpenGL Beispiel:

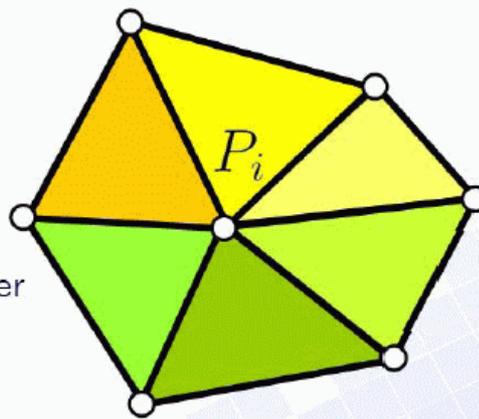
```
00 glBegin(GL_TRIANGLES);  
01     glNormal3d(nx, ny, nz);  
02     glVertex3d(x[i], y[i], z[i]);  
03     glVertex3d(x[j], y[j], z[j]);  
04     glVertex3d(x[k], y[k], z[k]);  
05 glEnd();
```

Nachteil: Kanten deutlich sichtbar.

*Interpolierende Shading-Verfahren* (Gouraud, Phong)  
glätten die Kanten zwischen den Polygonen  
(smoothing).

*smooth shading:*

Bestimme einen  
Normalenvektor  
für jeden Vertex



- Betrachte den Dreiecksfächer um den Punkt  $P_i$
- Bestimme die Normalenvektoren für alle Dreiecke
- Der Normalenvektor der Punktes  $P_i$  ergibt sich als Mittelwert der Normalen der angrenzenden Dreiecke

- Gouraud-Shading: wie beschrieben
- Phong-Shading: die Normalenvektoren werden interpoliert, für jedes Pixel wird die lokale Beleuchtungsrechnung durchgeführt (s. VL Grundzüge der Computergrafik).

Zusammenfassung:

## Surface Plots / Oberflächendarstellung

- Interpretiere den Skalarwert als Höhenwert und zeichne eine 3D Fläche
- Liniendarstellung:  
wireframe und hidden lines
- Flächendarstellung mit Beleuchtung:
  - benötigt Normalenvektoren
  - Normalenvektor pro Dreieck (*flat shading*)
  - Normalenvektor pro Vertex (*smooth, Gouraud shading*)

*Isolinien* (vgl. Kapitel 4.2)

Definition:

- Für ein gegebenes Skalarfeld  $f : \mathbb{R}^2 \mapsto \mathbb{R}$  und einen Skalarwert  $c \in \mathbb{R}$  entspricht die Menge
$$\{(x, y) \mid f(x, y) = c\}$$
einer Isolinie.
- Falls  $f$  differenzierbar ist und  $\text{grad}(f) \neq 0$ , dann sind die Isolinien *Kurven*.
- Falls  $\text{grad}(f) = 0$ , können die Isolinien auch *Flächen* und *Punkte* sein.

Konstruktion von Isolinien:

bereits kennengelernt:

1. Pixel by pixel contouring (für alle Pixel überprüfen, ob Isolinie hindurchgeht)
2. *Marching squares* (Werte nur in Knoten eines Rechteckgitters verwenden, Nullstellen auf den Kanten approximieren, diese verbinden).

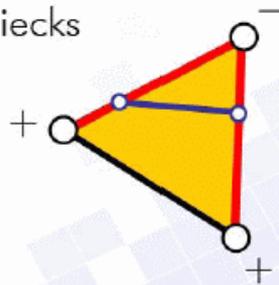
hier Verallgemeinerung und Vertiefung des 2. Ansatzes.

#### 4 Ergänzungen:

- Verallgemeinerung auf Dreieckszellen
- Strategien der Auflösung der Mehrdeutigkeit bei Viereckszellen
- Reihenfolge der Abarbeitung der Zellen
- Glättung der Isolinien

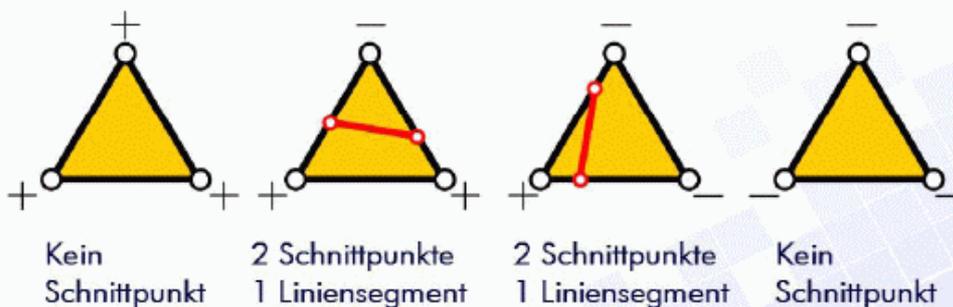
#### Bei dreieckigen Zellen:

- Betrachte jede Zelle separat
- Markiere alle Vertices des Dreiecks
  - mit  $+$  wenn  $f_i > c$
  - mit  $-$  wenn  $f_i \leq c$
- Untersuche alle Kanten mit Vorzeichenwechsel
- Interpoliere Schnittpunkte entlang der Kanten
- Verbinde die Schnittpunkte durch Liniensegmente



#### Für dreieckige Zellen:

- entweder keine oder zwei Kanten mit Vorzeichenwechsel



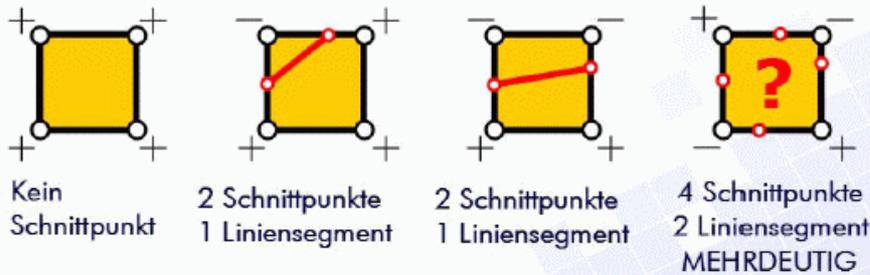
(symmetrische Fälle wurden weggelassen)

- keine Mehrdeutigkeiten

# Isolinien für Rechteckgitter (Marching squares):

Für rechteckige Zellen:

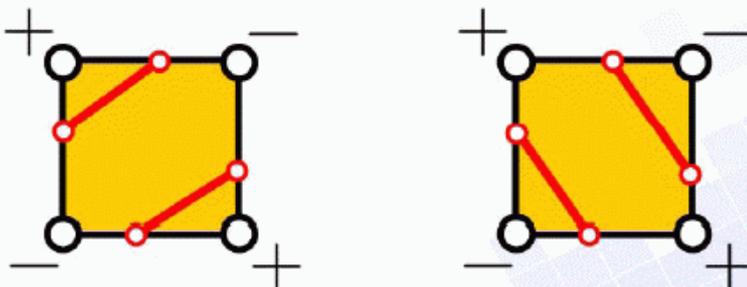
- entweder keine, zwei oder vier Kanten mit Vorzeichenwechsel



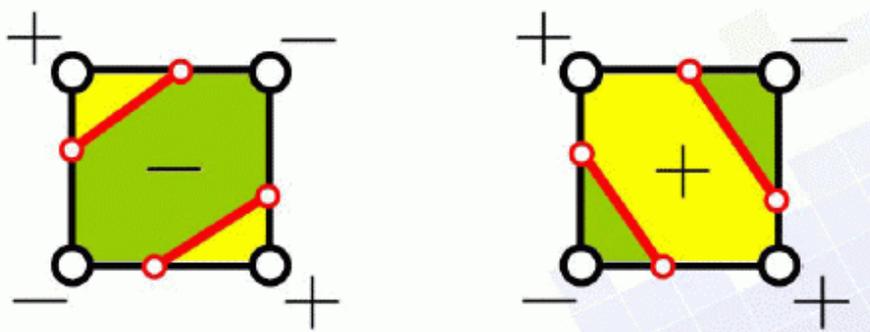
(symmetrische und invertierte Fälle wurden weggelassen)

Mehrdeutigkeit:

- vier Kanten mit Vorzeichenwechsel
- 2 mögliche Konfigurationen:

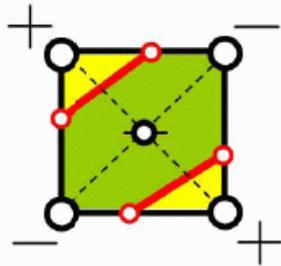


Verwende einen „Decider“ um zu entscheiden welche Konfiguration vorliegt.

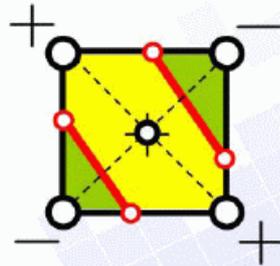


## ● Midpoint Decider

Interpoliere den Skalarwert im Mittelpunkt des Rechtecks:



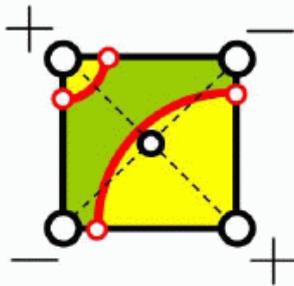
$$f_M \leq c$$



$$f_M > c$$

Schwachpunkt des Midpoint Deciders:

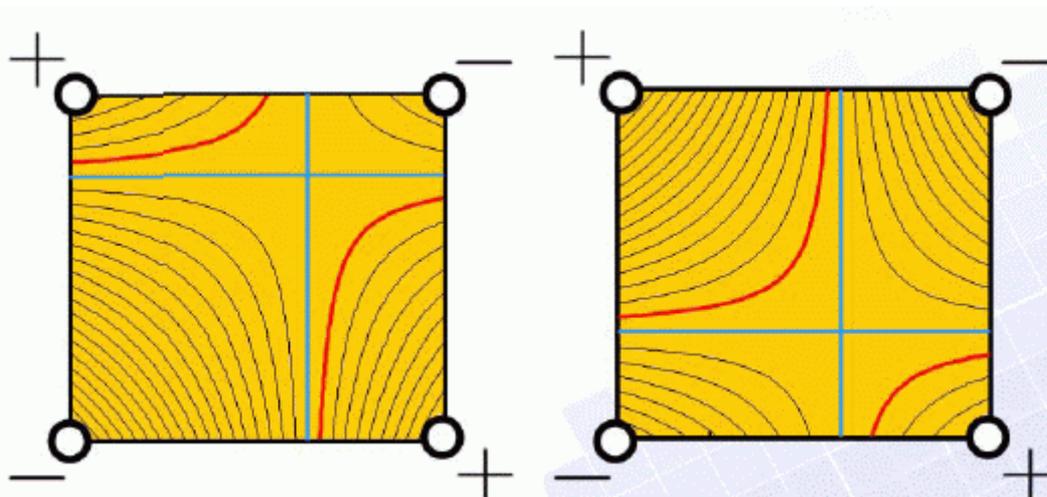
Isolinien sind keine Geraden, sondern Kurven



● Hier ist  
 $f_M > c$ ,  
obwohl der  
Mittelbereich  
negativ ist

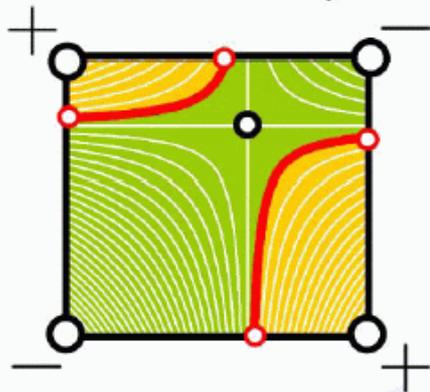
daher genauere Betrachtung des Verlaufes von typischen Isolinien in dieser Situation:

bei bilinearer Interpolation innerhalb der Zelle sind die echten Isolinien Hyperbelbögen.



## ● Asymptotic Decider

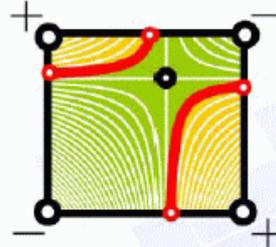
Bestimme den Skalarwert nicht am Mittelpunkt, sondern am *Schnittpunkt der Asymptoten*



- Der Schnittpunkt der Asymptoten liegt immer innerhalb des Mittelbereichs

Stelle die Interpolationsfunktion auf:

$$\begin{aligned} f(x, y) = & f_{00}(1-x)(1-y) + \\ & + f_{10}x(1-y) + \\ & + f_{01}(1-x)y + \\ & + f_{11}xy \end{aligned}$$



Bringe diese Gleichung in die Form

$$f(x, y) = \alpha(x - x_0)(y - y_0) + \gamma.$$

Dann ist  $\gamma$  der Skalarwert am Schnittpunkt der Asymptoten

zur Reihenfolge der Zellen-Abarbeitung:

sequenzieller Ansatz: alle Zellen in fester Folge abfahren

Nachteile des *sequentiellen Ansatzes*:

- Jeder Vertex der Isolinie und jede Kante des Gitters wird zweimal bearbeitet
- Das Ergebnis ist eine ungeordnete Folge von Liniensegmenten. Diese müssen erst sortiert werden um Polygonzüge zu erhalten.

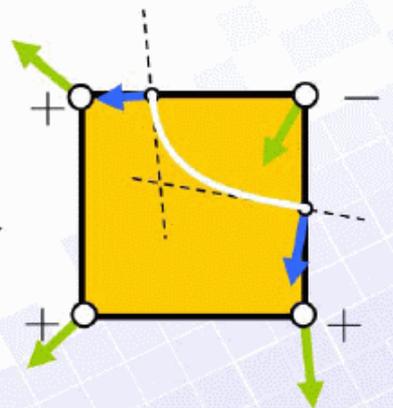
➔ **Contour Tracing**

## Contour tracing, Algorithmus:

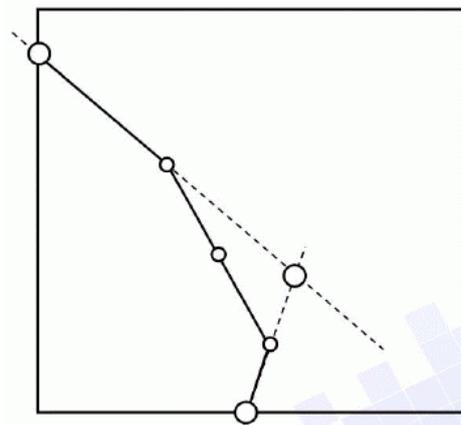
- Markiere alle Zellen mit Vorzeichenwechsel
- Solange es noch markierte Zellen gibt:
  - finde die Isolinie(n) innerhalb dieser Zelle
  - für alle (1 oder 2) Isolinien in dieser Zelle:
    - solange die Isolinie nicht geschlossen ist und der Rand nicht erreicht ist:
      - folge der Isolinie in die Nachbarzelle
      - entferne die Markierung (außer wenn die 2. Isolinie noch nicht abgearbeitet ist)

## Glättung der Isolinien:

- Bestimme die Gradientenvektoren an den Vertices
- Interpoliere die Gradientenvektoren an den Schnittpunkten
- Bestimme Tangenten an den Schnittpunkten (senkrecht auf Grad.)
- Zeichne einen Bogen, der an den Schnittpunkten die Tangenten interpoliert

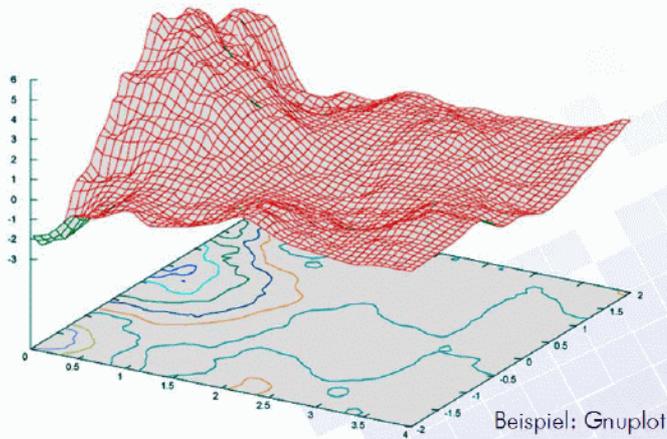


für den Bogen:  
verwende z.B. Bézierkurven  
(siehe VL "Grundzüge der  
Computergrafik")

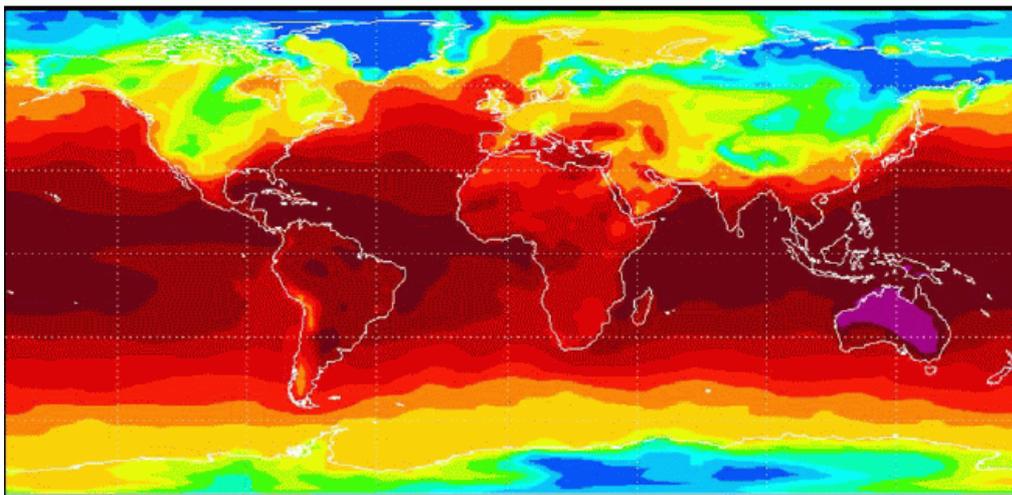


# Kombination

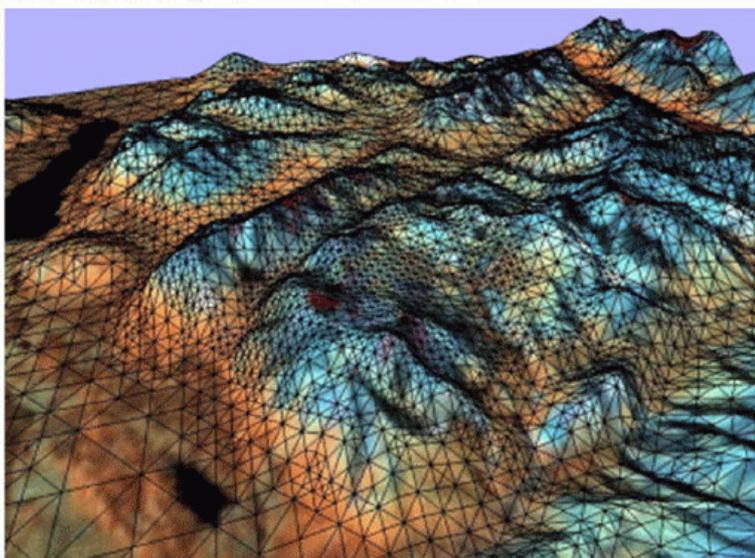
## ● Isolinen + Surface Plot



## ● Farbkodierung + Isolinen



## ● Farbkodierung + Surface Plot



(Rezk-Salama, o.J.)