Visualisierung

Datenstrukturen

1. Die 100^3 Punkte werden als Volumen von $100 \times 100 \times 100$ Punkten angenommen. Für `int` und `float` werden 4 Byte angesetzt.

   kartesisches Gitter: Es werden 3 `float` Werte für den Ursprung und 3 `float` für die Schrittweiten in die Koordinatenrichtungen benötigt, plus 3 `int` Werte für die Anzahl der Punkte in jede Richtung. Dann werden für den Datensatz 36 *Byte* benötigt.

   rechtwinkliges Gitter: Für den Ursprung 12 Byte, für die Anzahl der Punkte in jede Richtung 12 Byte und für jede Koordinatenrichtung 99 Schrittweiten, die wir auch als `float` annehmen. Insgesamt ergibt sich ein Speicherbedarf von rund 1.2 *MB*.

   strukturiertes Gitter: Die Punktkoordinaten müssen explizit gespeichert werden plus 12 Byte für die Anzahl der Punkte in jeder Richtung. Dies ergibt einen Speicherbedarf von rund 4 *MB*.

   unstrukturiertes Gitter: Wieder müssen die Punktkoordinaten explizit gespeichert werden, dafür werden 4 MB benötigt. Für jede Zelle (Hexaeder) werden 8 `int` Werte benötigt, um die Topologie abzuspeichern. Gehen wir von 1 000 Zellen aus, dann ergibt sich ein Speicherbedarf von rund 7, 2 *MB*. Allgemein ist der Speicherbedarf gegeben durch 4 *MB* + *NoZ* ·32 *Byte*; wenn NoZ die Anzahl der Zellen bezeichnet.

2. Für `int` und `float` wird von einem Speicherbedarf von 4 Byte ausgegangen; bei `double` von 8 Byte. Für die $100^3 = 1\ 000\ 000$ Punkte entstehen dann je nach Attributwert folgender Speicherbedarf für die Attribute:

   - `unsigned int`: 4 MB;
   - `float`: 4 MB;
   - `float[3]`: 12 MB;
   - `double[3][3]`: 72 MB.

3. Für die Berechnung wird einmal davon ausgegangen, dass die Punkte im Datensatz als strukturiertes Gitter vorliegen. Dann benötigt man für den Datensatz 4 MB. Sind nur die Attribute zeitabhängig, dann entsteht pro Zeitschritt ein Speicherbedarf von 12 MB und ingesamt für die 100 Zeitschritte 4 *MB* + $100 \cdot 12$ *MB* $\approx 1, 2$ *GB*.

   Ist auch die Geometrie zeitabhängig, dann entsteht ein Speicherbedarf von $100 \cdot (4 + 12)$ *MB* $\approx 1, 6$ *GB*.

4. Ein kartesisches Gitter in Polarkoordinaten hat gleichmässige Schrittweiten im Radius und im Winkel. Dadurch entstehen konzentrische Kreise wie in Abbildung 22. Für ein rechtwinkliges Gitter wird ein Vektor mit Radiuswerten und ein Vektor mit Winkelwerten angegeben; mit gleichmässiger Topologie. Wenn wir immer von Vierecken ausgehen entsteht ein Bild wie in Abbildung 24. Für diese Abbildung werden die Radien $(0.5, 1.0, 2.0, 2.5, 2.75)$ und die Winkel $(0°, 45°, 60°, 90°, 135°, 360°)$ angenommen.

In Abbildung 23 sehen Sie ein kleines Beispiel eines strukturierten Gitters in Polarkoordinaten. Dabei sollen die Punkte $V_1 = (0.5, 0°)$, $V_2 = (0.7, 30°)$, $V_3 = (0.4, 90°)$, $V_4 = (1.0, 10°)$, $V_5 = (0.9, 45°)$ und $V_6 = (1.5, 75°)$ enthalten sein. Als Zellen sind die „Vierecke“ mit den Ecken $1, 4, 5, 2$ und $2, 5, 6, 3$ enthalten.

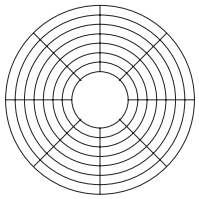

Abbildung 22: Ein kartesisches Gitter in Polarkoordinaten

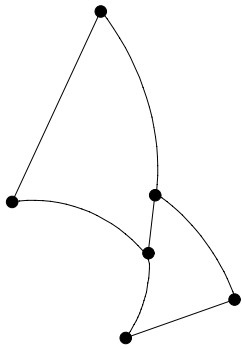

Abbildung 23: Ein strukturiertes Gitter in Polarkoordinaten

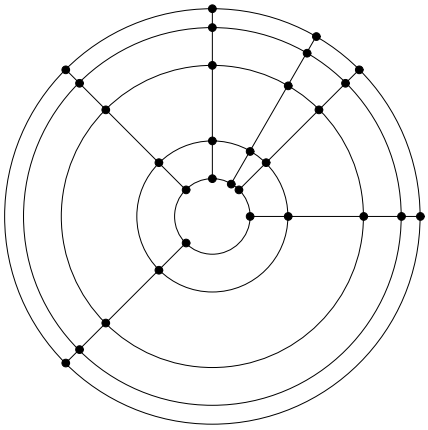

Abbildung 24: Ein „rechtwinkliges“ Gitter in Polarkoordinaten

Algorithmen für skalare Attribute

1. Schwarz, Weiß, Grün, Rot, Gelb, Cyan. Die minimale Differenz in der Luminanz ist dann 0.11.
2. Für die beiden Auflösungen finden sie in Tabelle 4 die entsprechenden Skalare; in Abbildung 25 die grafische Darstellung.
3. Man kommt durch Symmetrie und Rotation auf 4 Fälle:
   (a) $0, 15$;
   (b) $1, 2, 4, 7, 8, 11, 13, 14$;
   (c) $3, 6, 9, 12$;
   (d) $5, 10$.

Direkte Volumen-Visualisierung

1. Den Filter nach Mitchell und Netravali sehen Sie in Abbildung 26. An der Stelle $|x| = 2$ gilt immer $H_t(B, C) = 0$; für $|x| = 1$ ist der Funktionswert $H_t(B, c) = B$ und an der Stelle $x = 0$

Tabelle 4: Die Daten für Aufgabe 2

4.5	2.8125	2.25	2.8125	4.5
2.8125	1.125	0.5625	1.125	2.8125
2.25	0.5625	0	0.5625	2.25
2.8125	1.125	0.5625	1.125	2.8125
4.5	2.8125	2.25	2.8125	4.5

4.5	3.61	2.94	2.5	2.27	2.27	2.5	2.94	3.61	4.5
3.61	2.72	2.055	1.61	1.388	1.388	1.61	2.055	2.72	3.61
2.944	2.055	1.388	0.944	0.722	0.722	0.944	1.388	2.055	2.944
2.5	1.61	0.944	0.5	0.277	0.277	0.5	0.944	1.61	2.5
2.277	1.388	0.722	0.277	0.055	0.055	0.277	0.72	1.388	2.277
2.27	1.388	0.722	0.277	0.055	0.055	0.277	0.722	1.388	2.277
2.5	1.61	0.944	0.5	0.277	0.277	0.5	0.944	1.61	2.5
2.944	2.055	1.388	0.944	0.722	0.722	0.944	1.388	2.055	2.944
3.61	2.722	2.055	1.61	1.388	1.388	1.61	2.055	2.722	3.61
4.5	3.61	2.944	2.5	2.277	2.277	2.5	2.944	3.61	4.5

Abbildung 25: Die Berechnung von Konturlinien durch lineare Interpolation für Aufgabe 2; links Auflösung 5×5, rechts 10×10

gilt $H_t(B, C) = 6 - 2B$.

2. Eine scharfe Trennung ist nicht möglich bei der Lage; eine Möglichkeit wäre die Kurve in Abbildung 27. Möglich wäre auch eine Realisierung mit einer überlappenden, „fuzzy“ Transferfunktion.

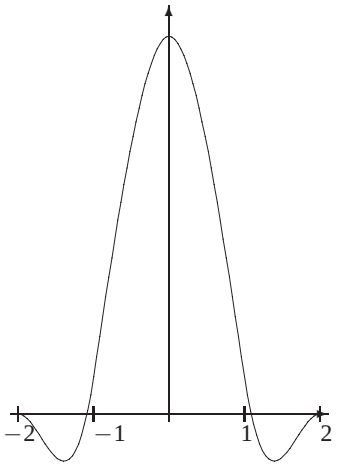

Abbildung 26: Der Filter nach Mitchell und Nevanlina für $B = 0.5, C = 0.85$

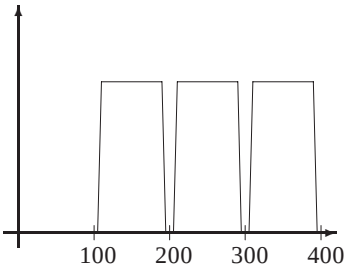

Abbildung 27: Ein Vorschlag einer Transferfunktion für Aufgabe 2

Visualisierung von Vektorfeldern

1. Die Skizzen finden Sie in den Abbildungen 28 bis 31.

2. Die Gradientenvektorfelder finden Sie in den Abbildungen 32 und 33.

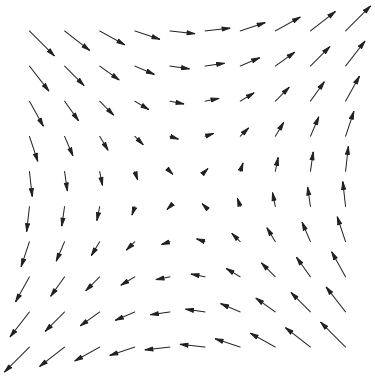

Abbildung 28: $f(x,y) = (y,x)$

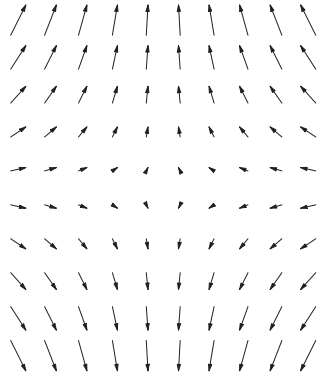

Abbildung 29: $f(x,y) = (-x,2y)$

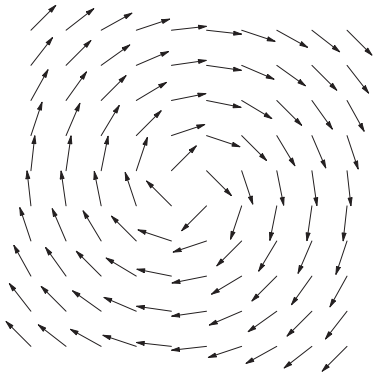

Abbildung 30: $f(x,y) = \frac{1}{\sqrt{x^2+y^2}}(y,-x)$

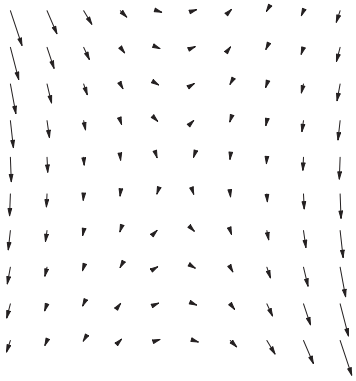

Abbildung 31: $f(x,y) = (y^2 - 2xy, 3xy - 6x^2)$

3. Durch Ableiten des Potenzials bestimmt sich das Vektorfeld als

$$f(x,y,z) = v_0 \left[\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + \frac{R^3}{2\sqrt{x^2+y^2+z^2}^5} \begin{pmatrix} -3xz \\ -3yz \\ x^2+y^2-2z^2 \end{pmatrix} \right]$$

Auf den Koordinatenachsen erhält man damit

$$f(0,0,\pm R) = v_0 \left[\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + \frac{R^3}{2R^5} \begin{pmatrix} 0 \\ 0 \\ -2R^2 \end{pmatrix} \right] = \mathbf{0},$$

$$f(\pm R,0,0) = v_0 \left[\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + \frac{R^3}{2R^5} \begin{pmatrix} 0 \\ 0 \\ R^2 \end{pmatrix} \right] = \frac{3}{2} v_0 \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix},$$

$$f(0,\pm R,0) = v_0 \left[\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + \frac{R^3}{2R^5} \begin{pmatrix} 0 \\ 0 \\ R^2 \end{pmatrix} \right] = \frac{3}{2} v_0 \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

Für den Punkt $P = (x,y,z)$ und $R = \|P\|$ gilt

$$\langle f(x,y,z), P \rangle = v_0 (z + \frac{1}{2R^2}(-2z)R^2) = 0.$$

Das Vektorfeld steht also immer senkrecht auf dem Punkt; also tangential zur umströmten Kugel! Die Abbildungen 6.43, 6.45 und 6.48 zeigen dieses Vektorfeld.

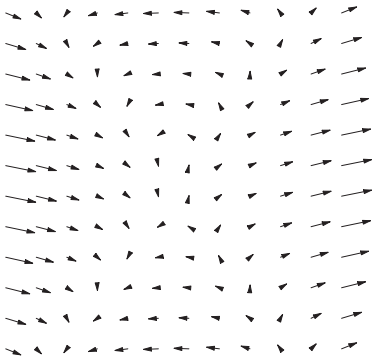

Abbildung 32: $F(x,y) = x^2 - \frac{1}{2}y^2$

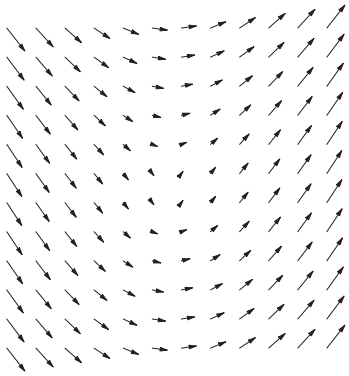

Abbildung 33: $F(x,y) = \ln(\sqrt{x^2+y^2})$

4. Die Feldlinien dieses Vektorfelds sind die Hyperbeln $y = C/x$. In Abbildung 34 sehen Sie die Trajektorien zum Zeitpunkt $t = 0$.

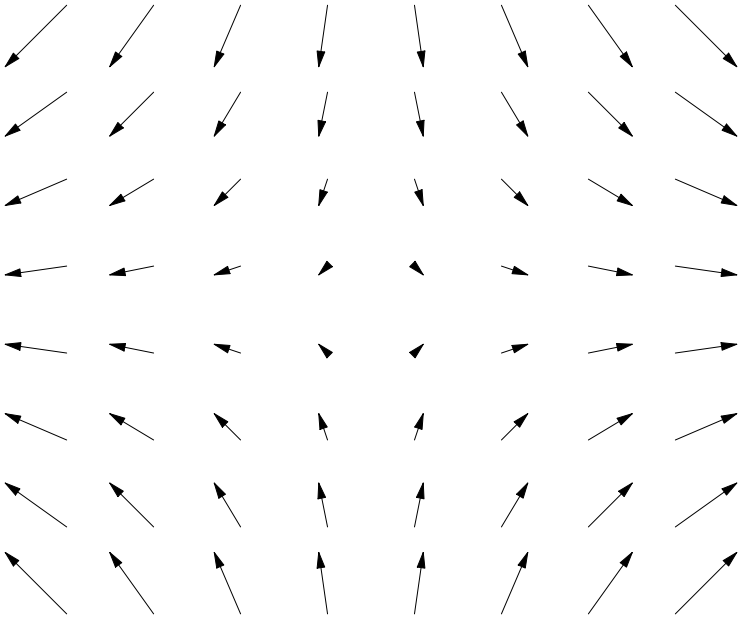

Abbildung 34: Die Trajektorien des Felds in Aufgabe 4 zum Zeitpunkt $t = 0$

Die Visualisierungs-Pipeline in VTK

1. Finden Sie bei den Quelltexten dieses Kapitels

2. Für alle drei Formate müssen auf jeden Fall die Attribute, also die Farben, gespeichert werden. Möglich wäre natürlich eine Zahl und eine Farbtabelle; oder direkt die RGB-Werte. Wenn wir 8 Bit für die Farbe pro Kanal ansetzen entsteht dadurch ein Bedarf von 30*KB*.

   Für `vtkUnstructuredPoints` ergibt sich ein Speicherbedarf von 36 *Byte*; der Ursprung, die Anzahl der Punkte und das Spacing.

   Für `vtkStructuredGrid` müssen die Punktkoordinaten explizit gespeichert werden; dadurch ergibt sich ein Bedarf von 40 *KByte*.

Für `vtkPolyData` ergibt sich ein Bedarf von 40 *KByte* für die Punktkoordinaten, plus 5 `int`-Werte pro Zelle für die Topologie; also insgesamt $40 + 200 = 240$ *KByte*.

3. Finden Sie bei den Quelltexten dieses Kapitels

Volumen-Visualisierung mit der VTK

1. In den Abbildungen bis finden Sie das Ergebnis eines Ray-Castings mit einer Pipeline wie für Bild 6.67 und Farbtafel 17; jeweils mit und ohne Schattierung und mit tri-linearer und mit nächster Nachbar- Interpolation. Die Bilder sind mit einem Datensatz mit Viertel-Auflösung („headsq / quarter“) berechnet; was man deutlich am Aliasing erkennt. Probieren Sie einmal den Datensatz mit voller Auflösung; Sie finden Ihn unter den Quellen zu Kapitel 6; genauso wie die nötige Pipeline. Die Bilder sind alle mit einer Post-Shading Pipeline (in VTK InterpolateFirst) gerechnet; der Default in VTK.

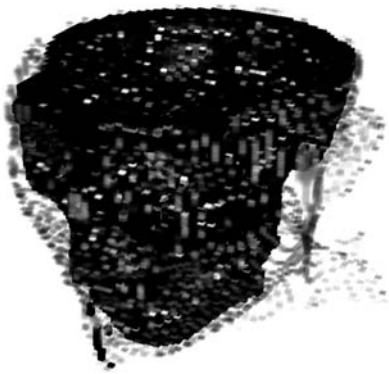

Abbildung 35: Ray-Casting mit Nearest-Neighbour und Schattierung

Abbildung 36: Ray-Casting mit Nearest-Neighbour ohne Schattierung

Abbildung 37: Ray-Casting mit tri-linearer Interpolation und Schattierung

Abbildung 38: Ray-Casting mit tri-linearer Interpolation ohne Schattierung

2. Finden Sie bei den Quelltexten dieses Kapitels
3. In Abbildung 39 sehen Sie das Ergebnis der Pipeline für Abbildung 37 und Farbtafel 17; allerdings mit einer Pre-Shading-Pipeline. Beachten Sie unbedingt, dass Sie in VTK zwischen den

beiden Pipelines nur umschalten können, wenn Sie tri-lineare Interpolation verwenden!

Abbildung 39: Die Pipeline aus Abbildung 37 mit einer Pre-Shading Pipeline

4. Die Klasse `vtkLODProp3D` akzeptiert sowohl Ray-Casting-Mapper oder Texture-Mapping als auch von `vtkPolyDataMapper`. So ist es beispielsweise möglich, mit `vtkOutlineFilter` die Bounding-Box des Datensatzes zu extrahieren und als niedrigsten Level-Of-Detail einzustellen. In den Quellen zu diesem Kapitel finden Sie eine Pipeline. Für eine Instanz `volume` der Klasse `vtkLODProp3D` werden drei verschiedene Levels hinzugefügt. Der letzte Parameter bedeutet, dass VTK selbst versucht die benötigte Renderzeit zu schätzen.

```
// Bounding - Box mit eigenem Material
outline->SetInput(reader->GetOutput());
outlineMapper->SetInput(outline->GetOutput());
outlineProperty->SetColor(0.0, 0.0, 0.0);

// Texture2D Instanz als Level 2
lowresMapper->SetInput(reader->GetOutput());

// Ray-Casting als level 3
hiresMapper->SetInput(reader->GetOutput());
hiresMapper->SetVolumeRayCastFunction(composite);
hiresMapper->SetSampleDistance(distance);

int level1 = volume->AddLOD(outlineMapper, outlineProperty, 0.0);
int level2  = volume->AddLOD(lowresMapper, prop, 0.0);
int level3 = volume->AddLOD(hiresMapper, prop, 0.0)
```

Visualisierung von Vektorfeldern mit der VTK

1. Finden Sie bei den Quelltexten dieses Kapitels
2. Finden Sie bei den Quelltexten dieses Kapitels