Digitales Sommersemester 2021:

# Datenvisualisierung und GPU-Computing

## Übung 4: Stromlinien

02.06.2021

**Andreas Beckert**

andreas.beckert@uni-hamburg.de

***Regionales Rechenzentrum, Visual Data Analysis Group,***

***FB Informatik Arbeitsgebiet „Scientific Visualization and Parallel Processing“***

# Diese Woche

- Besprechung von Übung 3: Contouring

- Übung 4:
  - FlowVis: Stromlinien

# Übung 3: Marching Squares

Als nächstes Ziel fügen wir dem farbkodierten 2D-Schnitt aus Übung 2 Konturlinien hinzu, die mit dem Marching Squares Algorithmus berechnet werden.

- Füge dem Projekt, analog zu den Klassen `HorizontalSliceToImageMapper` und `HorizontalSliceRenderer` aus Übung 2, zwei neue Klassen `HorizontalSliceToContourLineMapper` und `HorizontalContourLinesRenderer` hinzu. Der Mapper implementiert den Marching Squares Algorithmus, der Renderer zeigt die berechneten Konturlinien in unserer Szene an (es bietet sich zunächst eine noch nicht in der Szene verwendete Farbe für die Konturen an, z.B. weiß).

# Übung 3: `HorizontalSliceToContourLineMapper`

- Statt der Methode `mapSliceToImage(int iz)` aus Übung 2 bietet sich nun eine Methode `mapSliceToContourLineSegments(int iz)` an, die den Marching Squares Algorithmus implementiert und eine Liste von Liniensegmenten zurückgibt, z.B. als `QVector<QVector3D>`.
  - Verwende vorerst nur die Komponente „0“ (also „x-Wind“) der Daten.
  - Zunächst empfiehlt es sich, nur ein beliebiges Liniensegment zurückzugeben und den Renderer zu implementieren, um sicherzustellen, dass die Pipeline richtig funktioniert.
  - Tipp: `QVector` stellt eine Methode `data()` bereit, die einen Zeiger auf das Datenfeld liefert und genutzt werden kann, um die Liniensegmente in den Vertex Buffer zu übertragen (vgl. Übungen 1 und 2).

# Übung 3: Datenexploration

- Wenn die Konturlinien sichtbar sind, füge eine analog zu Übung 2 eine Methode „`moveSlice(int steps)`“ für den Renderer hinzu, die die z-Position des Schnittes verändert. Über die Cursor-Tasten (up/down) wird nun also der komplette Schnitt mit Colour-Mapping (Übung 2) und Konturlinien gesteuert.

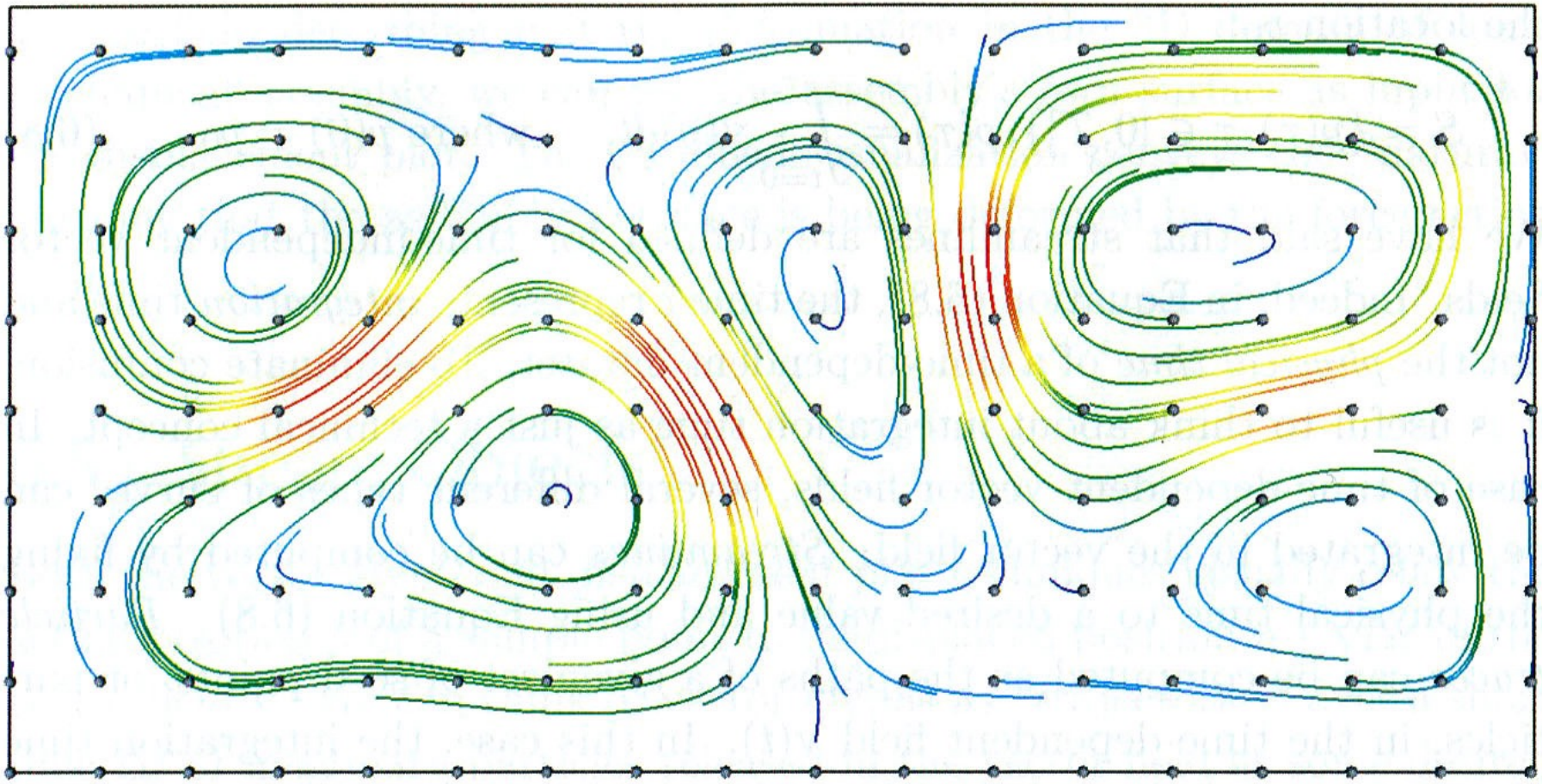
- Welche Isowerte für die Konturen helfen bei der Interpretation des Datenfeldes (wir konzentrieren uns weiterhin erst nur auf die Komponente „0“ (also „x-Wind“) der Daten)?

# Übung 3: Leitfragen und Tipps

Der grundlegende Marching Squares Algorithmus kann recht knapp und übersichtlich implementiert werden.

- Wie kann, ohne viele `if-then-else`-Abfragen, der Marching Squares-Fall effizient bestimmt werden? (Tipp: Wie können die Zustände der Gitterpunkte als Bitfeld interpretiert werden?)
- Warum genügt es, 7 Fälle im Code zu behandeln?
- Tipp: Eine zusätzliche Methode `isoCrossingBetweenTwoVertices(…)`, die den interpolierten Schnittpunkt einer Konturlinien zwischen zwei Vertices berechnet, ist hilfreich.
- Wie können die Mehrdeutigkeiten behandelt werden?
- Es kann passieren, dass die Konturlinien nicht komplett angezeigt werden, sondern „zufällige Löcher“ aufweisen. Woran kann das liegen?

# Stromlinien: Was benötigen wir?



Startpunkte (seed points)

Interpolation

Integrationsverfahren

Rendering der Liniengeometrie

# Übung 4: Stromlinien

Mit dem derzeitigen Stand unseres Visualisierungsprogramms können wir zwar die einzelnen Komponenten der erzeugten „Wind“-Vektoren als Skalarfelder betrachten, das 3D-Strömungsfeld des „synthetischen Tornados“ an sich lässt sich damit jedoch schwer analysieren. Als nächstes wollen wir daher dieses Strömungsfeld visualisieren. In der Vorlesung wurden u.a. Stromlinien (Streamlines) als eine Möglichkeit besprochen, die Windrichtung an verschiedenen Stellen des 3D Volumens darzustellen.

Aufgabe 4:

- Implementiere Stromlinien in das Visualisierungsprogramm und erkunde damit das Strömungsfeld des synthetischen Tornados.
- Stelle die Stromlinien (zunächst) als einfache Liniengeometrie dar, analog zu den Konturlinien aus Übung 3 (also ohne Beleuchtung).
- Vergleiche Integration mit dem Euler- und dem Runge-Kutta-Verfahren.

# Übung 4: Leitfragen und Tipps

- Als „Template“ für den `StreamLinesMapper` und `StreamLinesRenderer` für die Stromlinien bieten sich die beiden Klassen aus Übung 3 an.
  - Wer möchte: Der Renderer unterscheidet sich kaum (es macht sicher Sinn, eine andere Farbe zu verwenden), daher wäre an dieser Stelle eine Möglichkeit über Vererbung die Struktur des Programmes zu vereinfachen.
- Es bietet sich wieder an, zunächst im Mapper ein beliebiges Liniensegment zu erzeugen und damit zu prüfen, dass der Renderer funktioniert.
- Bei der Entwicklung bietet es sich an, den Horizontalschnitt aus Übungen 2 und 3 als Hilfe zu verwenden: Stimmt die Richtung der Stromlinien mit dem Vorzeichen der betrachteten Windkomponente überein?
- Wie sollte das Programm den Fall behandeln, dass eine Stromlinie bei der Berechnung das Datenvolumen verlässt?

# Übung 4: `StreamLinesMapper`

- In dieser Klasse benötigen wir wieder eine Methode `setDataSource(...)`, diese kann aus Übung 2 bzw. 3 übernommen werden.
- Hier bietet es sich an, für die unterschiedlichen Integrationsverfahren jeweils eine Methode zu implementieren z.B. `EulerIntegration(QVector3D position, float stepSize)`. Diese Funktion soll dann jeweils die Endposition eines Liniensegmentes einer Stromlinie zurückgeben.
- In der Methode `computeStreamLines(QVector<QVector3D> seedPoints)` werden dann für jeden „Seed Point“ die Stromlinien berechnet. Diese wird dann von der Methode `updateStreamLines()` der Klasse `StreamLineRenderer` aufgerufen und gibt die Liniensegmente zurück.
- Zur Berechnung der Stromlinien benötigen wir noch eine weitere Methode, `interpolateWindVector(QVector position)` die den interpolierten Wert des Windvektors zurückgibt.

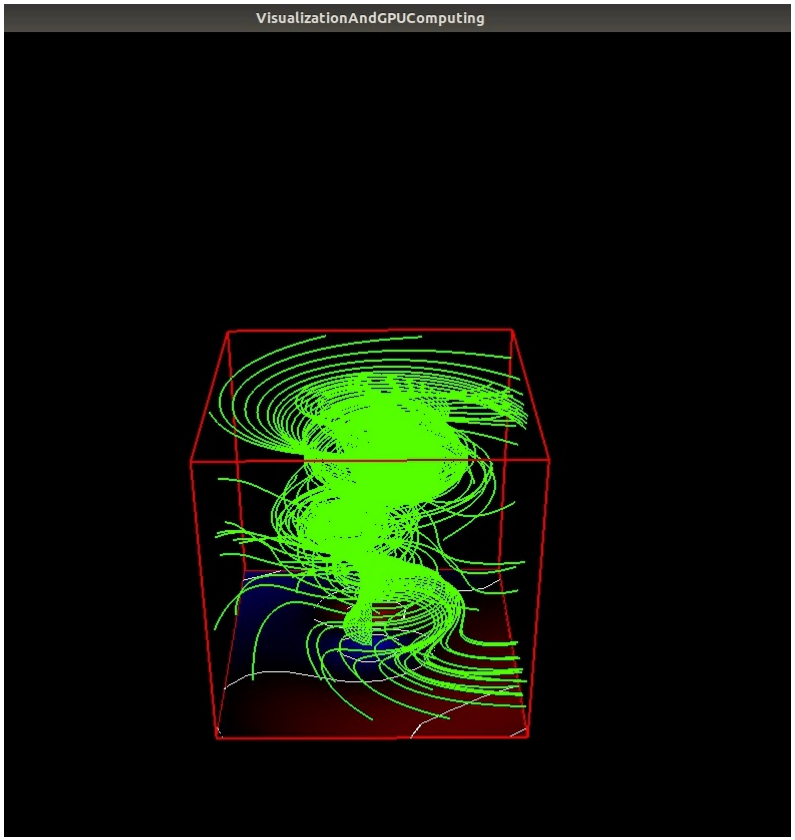
# Übung 4: `StreamLinesMapper` / `StreamLinesRenderer`

- Tipp: Es ist sinnvoll für die Integration den “World Space“ (0...1) zu verwenden. Lediglich für die Interpolation ist es notwendig, die Koordinaten des “World Space“ in die Koordinaten unseres Gitters (0...nGridPoints-1) zu transformieren.
- Es bietet sich an in der Methode `updateStreamLines()` der Klasse `StreamLinesRenderer` die Positionen der “Seed Points“ der Stromlinien zu erstellen und anschließend die “Seed Points“ an die Methode `computeStreamLines(...)` vom `StreamLinesMapper` zu übergeben.

# Übung 4: Zielfragen

- Sobald die Berechnung funktioniert, betrachte Stromlinien, die am Punkt (x,y) = (0.2, 0.2) sowie in regelmäßigen Abständen von 0.1 entlang der z-Achse gestartet werden. Was kann über das Strömungsfeld ausgesagt werden?
- Für z = 0.8 vergleiche Euler- und RK-Integration für eine Schrittweite von 0.2 und eine maximale Schrittzahl von 100. Wie unterscheiden sich die berechneten Stromlinien? Passt die Euler-Integration zur x-Komponente des Windfelds (betrachte mit dem Horizontalschnitt)?
- Starte nun Stromlinien auf einem regelmäßigen Gitter von 0.1 in allen drei Raumrichtungen. Ist die Visualisierung noch lesbar?
- Verringere den Abstand der Seed Points auf 0.01 (eine Dimension nach der anderen…). Dauert die Berechnung schon länger?

# Übung 4: Beispiel-Visualisierung Stromlinien




Beispiel-Visualisierung von Stromlinien für (x,y) = (0.5, 0.5) sowie z in regelmäßigen Abständen von 0.01 entlang der z-Achse gestartet.