Digitales Sommersemester 2021:

Datenvisualisierung und GPU-Computing

Übung 3: Contouring

19.05.2021

Andreas Beckert

andreas.beckert@uni-hamburg.de

Regionales Rechenzentrum, Visual Data Analysis Group,

FB Informatik Arbeitsgebiet „Scientific Visualization and Parallel Processing“

Diese Übung

- Besprechung von Übung 2: Colourmapping

- Übung 3:
  - 2D Contouring mit Marching Squares auf unseren farbcodierten 2D-Schnitten.

Besprechung Übung 2: Colour Mapping

- Verwende das Qt-OpenGL-Template.
- Füge die Datenquelle aus Übung 1 hinzu.
  - Wir fügen alle Elemente der Pipeline zum `OpenGLDisplayWidget` hinzu, also auch eine Instanz der Datenquelle.
  - Die Datenquelle kann in der Methode `initVisualizationPipeline()` des Widgets initialisiert werden. Zum Testen kann hier die Textausgabe aus Übung 1 aufgerufen werden.
- Füge dem Projekt zwei neue Klassen `HorizontalSliceToImageMapper` und `HorizontalSliceRenderer` hinzu.
  - Auch von diesen Klassen werden Instanzen in `OpenGLDisplayWidget` angelegt, sie werden in `initVisualizationPipeline()` erzeugt und im Destruktor freigegeben. Der Rendering-Aufruf erfolgt analog zum Aufruf der Bounding-Box-Methode.

Besprechung Übung 2 : `HorizontalSliceToImageMapper`

In einer klassischen Pipeline-Architektur werden die einzelnen Module „hintereinandergeschaltet“.

- Unser Mapper bekommt dafür einfacherweise eine Methode `setDataSource()`, die als Argument einen Zeiger auf eine Datenquelle erhält, hinter die sich das Modul in der Pipeline „hängt“.
- Als Ausgabe soll der Mapper ein Bild eines angefragten Gitterlevels ausgeben: Eine Methode „`QImage mapSliceToImage(int iz)`“ soll, analog zu der Textausgabe in Übung 1, ein Bild erstellen und als Typ „`QImage`“ zurückgeben.
  - Verwende vorerst nur die Komponente „0“ (also „x-Wind“) der Daten.
  - Wir verwenden eine einfache Transferfunktion: Werte über 0 werden auf den roten Farbkanal gemappt, Werte unter 0 auf den blauen. Eine brauchbare Skalierung auf Rotwerte/Blauwerte im Bereich 0..255 erhält man, wenn die Fließkommawerte mit dem Faktor „3*255“ skaliert.
  - Tipp: Verwende den Typ `QColor` sowie die Methode `setPixelColor()` von QImage.

Besprechung Übung 2: `HorizontalSliceRenderer`

Der Renderer für das im Mapper erstellte Bild wird am besten analog zum Renderer für die Bounding Box erstellt.

- Für den Aufbau der Pipeline bietet sich eine Methode „`setMapper()`“ an, die als Argument einen Zeiger auf einen Mapper bekommt.
- Eine Methode „`drawImage()`“ zeichnet, analog zur Bounding Box, das Bild in den OpenGL Viewport.
  - Als ersten Schritt zeichne nur den Rahmen für das Bild. Dies benötigt nur minimale Änderungen vom Bounding-Box-Code (copy&paste…).
  - Welcher Code (C++ sowie GLSL im Shader) ist nötig, um das Bild vom Mapper in eine Textur zu überführen und darzustellen (zunächst nur für den Schnitt bei iz = 0)?
  - Tipp: Die Vertex-Koordinaten können in diesem einfachen Fall als Texturkoordinaten „recycelt“ werden, so dass keine zusätzlichen Texturkoordinaten an den Vertex-Shader übergeben werden müssen.

Besprechung Übung 2: Datenexploration

- Wenn das Bild sichtbar ist, füge eine Methode „`moveSlice(int steps)`“ hinzu, die die z-Position des Schnittes verändert. Diese Methode muss also die Geometrie des Slices aktualisieren sowie ein neues Bild vom Mapper anfordern. Die Methode soll über die Cursor-Tasten (up/down) gesteuert werden (siehe `OpenGLDisplayWidget::keyPressEvent()`).
  - Tipps: Die Daten eines Vertex Buffers können einfach mit „`allocate()`“ überschrieben werden. Bei Verwendung einer Textur mit `QImage` muss die Textur hingegen erst „zerstört“ und neu erstellt werden, bevor „`setData()`“ wieder angewendet werden kann.

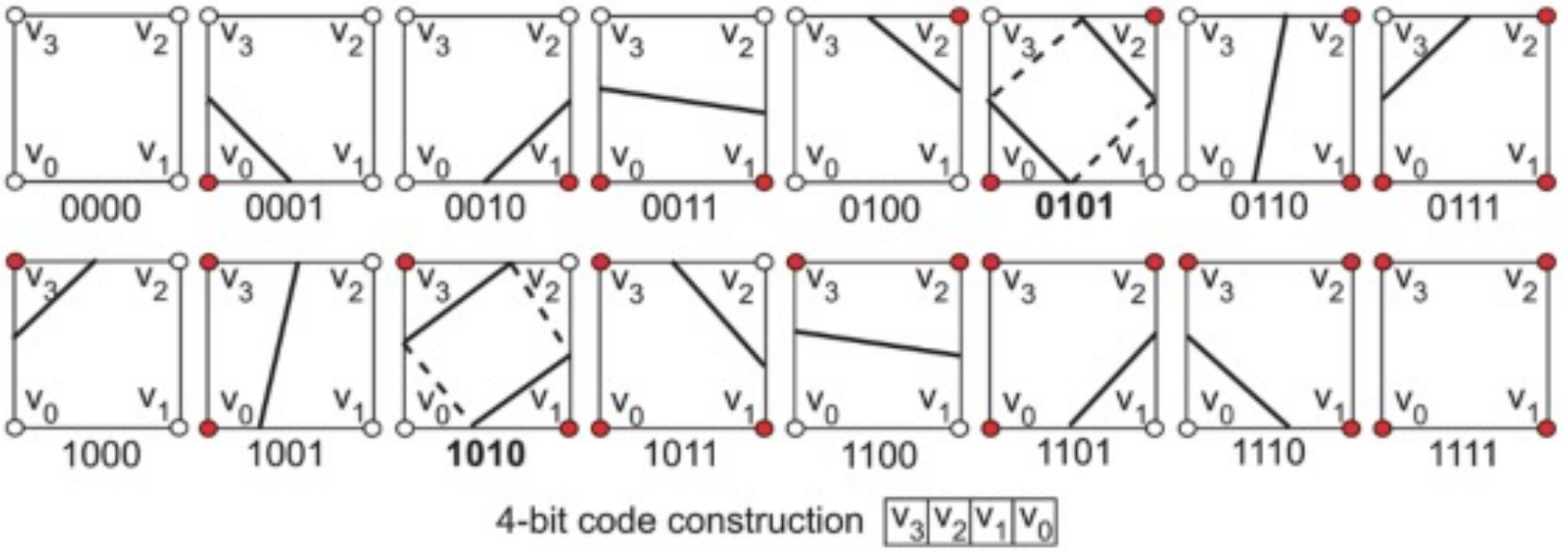
Sobald der Schnitt sichtbar und verschiebbar ist, erkunde das Datenfeld:

- Was verrät die Visualisierung über die Windgeschwindigkeiten (in x-Richtung) im Feld?
- Wie ändert sich die Geschwindigkeit mit der Höhe?

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.

Aus der Vorlesung: Marching Squares: Algorithm



For each grid cell:

1. Determine the cell configuration
2. Depending on configuration, find intersection points of isoline with edges
3. If case is ambiguous, decide
4. Output line segment(s)

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?

Beispiellösung Übung 3: Contouring

Iso-Werte: (-0.1, 0, 0.1)