WiMa-Praktikum 2
Prof. Dr. Karsten Urban
Kristina Steih, Klaus Stolle

Universität Ulm
Institut für Numerische Mathematik
WS 2013/2014, 9.12.2013

Projekt 5: Adaptive Gitter-Verfeinerung

1 Adaptive Netz-Verfeinerung

Um ein Netz aus Dreiecken adaptiv verfeinern zu können, werden Fehler-Indikatoren benötigt. Diese geben für jedes Element eine Abschätzung für den Fehler an. Grob gesprochen werden dann diejenigen Elemente verfeinert, auf denen der Fehler am größten ist. Wir wollen in diesem Projekt verschiedene Strategien zur Markierung der zu verfeinernden Elemente betrachten: Zu einer Triangulierung $\mathcal{T}$ sei der Vektor $\eta = (\eta_T)$, der für das Element $T \in \mathcal{T}$ den Fehlerindikator η_T enthält, gegeben.

- Strategie 1 (Absolutes Kriterium):
  Verfeinere alle Dreiecke $T \in \mathcal{T}$ für die gilt: $\eta_T \geq \theta$, $\theta \in \mathbb{R}$.
- Strategie 2 (Relatives Kriterium):
  Verfeinere alle Dreiecke $T \in \mathcal{T}$ für die gilt: $\eta_T \geq \theta \max\limits_{T' \in \mathcal{T}} \eta_{T'}$, $\theta \in [0,1]$.
- Strategie 3 (Dörfler-Marking/Bulk-Chasing):
  Suche eine minimale Menge $\mathcal{M} \subset \mathcal{T}$ für die gilt: $\theta \sum_{T \in \mathcal{T}} \eta_T^2 < \sum_{T \in \mathcal{M}} \eta_T^2$, $\theta \in [0,1]$. Verfeinere dann alle $T \in \mathcal{M}$.

Es gibt nun drei Möglichkeiten, ein Element zu verfeinern: die sogenannte Rot-, Grün- oder Blau-Verfeinerung, siehe Abbildung 1.

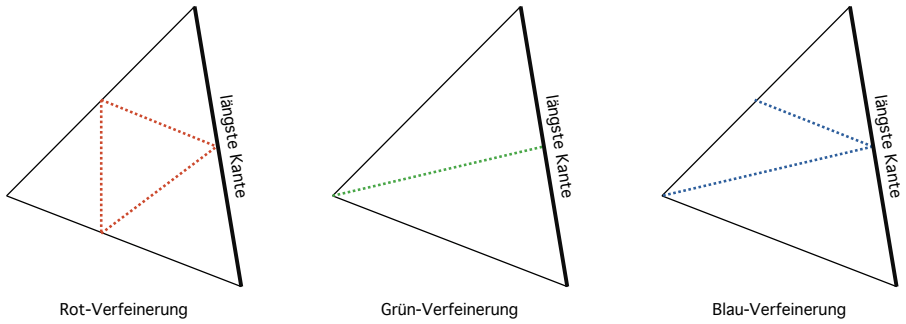


Abbildung 1: Rot-, Grün- und Blau-Verfeinerung

Um eine verfeinertes Netz zu erhalten, das immer noch zulässig ist (d.h. keine hängenden Knoten usw.), geht man bei der Verfeinerung wie folgt vor:

1. Alle markierten Elemente werden rot-verfeinert.
2. Die Elemente, die zwei markierte Nachbarn haben, deren Nachbar entlang der längsten Kante aber nicht markiert ist, werden zusätzlich rot-verfeinert.
3. Die Elemente, die zwei markierte Nachbarn haben, wobei einer der Nachbarn an der längsten Kante liegt, werden blau-verfeinert.
4. Die Elemente, die einen markierten Nachbarn haben, welcher nicht an der längsten Kante liegt, werden zusätzlich blau-verfeinert.

5. Die Elemente, die einen markierten Nachbarn haben, welcher an der längsten Kante liegt, werden grün-verfeinert.

2 Aufgabenstellung

In diesem Projekt möchten wir mithilfe eines adaptiven Algorithmus die Gitterverfeinerung implementieren. Üblicherweise folgt man dabei dem Schema

SOLVE → **ESTIMATE** → **MARK** → **REFINE**,

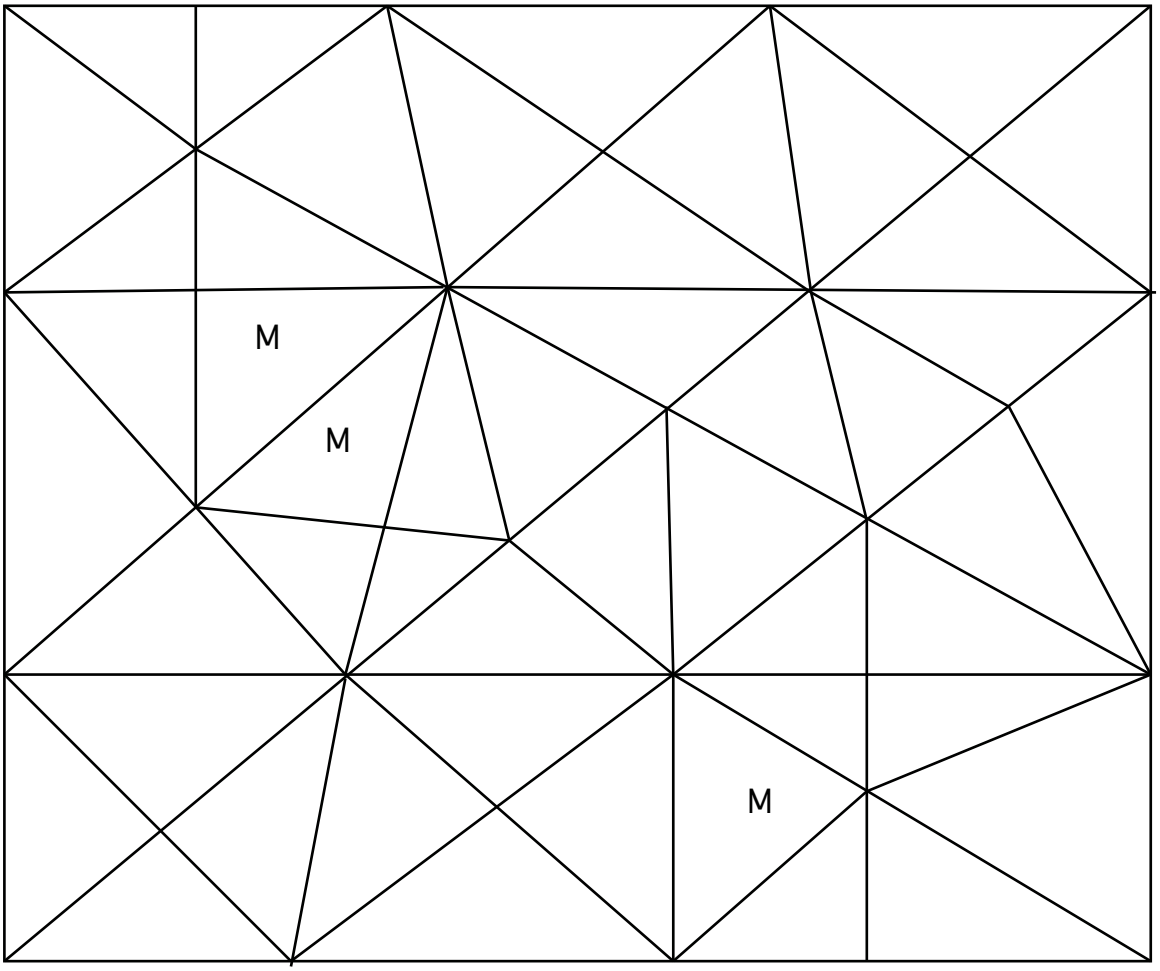
was so lange wiederholt wird bis der Fehlerschätzer im **ESTIMATE**-Schritt klein genug ist. Wir wollen hier aber nun keine partielle Differentialgleichung, sondern einfache Bilddaten betrachten, so dass der **SOLVE**-Schritt wegfällt.

Die Aufgabe ist nun, die adaptive Netzverfeinerung zu programmieren.

Laden Sie dazu das Material von der Homepage herunter und führen Sie folgende Arbeitsschritte durch.

2.1 Theorie verstehen

Zeichnen Sie im folgenden Bild eine mögliche Verfeinerung für die markierten Elemente (enthalten ein "M") ein.



Schreiben Sie stichpunktartig auf, wie man bei der Verfeinerung vorgehen könnte.

2.2 ESTIMATE: Fehler berechnen

Eine Funktion `computeEtaR`, welche für alle Elemente die maximale Differenz zwischen den Farbwerten (in Graustufen) berechnet, ist bereits gegeben.

2.3 MARK: Elemente markieren

Schreiben Sie Funktionen `markElements_absolute`, `markElements_max` und `markElements_Doerfler`, welche die obigen Strategien umsetzen, um zu einem gegebenen Fehlerindikator `etaR` die zu verfeinernden Elemente zu markieren.

2.4 Einige Hilfs-Matrizen aufbauen

Wir gehen davon aus, dass wie üblich alle Elemente, Kantenstücke und Knoten durchnummeriert sind (die sogenannte *globale* Nummerierung). Zum Verfeinern einzelner Elemente ist es jetzt hilfreich, z.B. zu wissen, welche Kanten gerade das i-te Element bilden. Dafür brauchen wir die folgenden Hilfs-Datenstrukturen, welche aus den Strukturen `coordinates`, `elements` und `boundary` erzeugt werden (wir unterscheiden hier diesmal nicht zwischen Dirichlet- und Neumann-Rand, weil wir ja eh Bilder betrachten):

- `element2edges`: Gibt die globalen Kanten-Indizes der lokalen 3 Kanten eines Elements an. Dabei soll `element2edges(i,l)` die Nummer der Kante zwischen den Knoten `elements(i,l)` und `elements(i,(l mod 3)+1)` angeben.
- `edge2nodes`: Gibt die Indizes der zu einer Kante gehörenden Knoten an. Die Zeile `edge2nodes(l,:)` enthält also gerade die Nummern j, k der Knoten an der l-ten Kante.
- `boundary2edges`: Gibt die globalen Kanten-Indizes der Rand-Kanten an, `boundary2edges(l)` ist also die Nummer der l-ten Kante des Randes.

Schreiben Sie dazu eine Funktion

```
[edge2nodes,element2edges,boundary2edges] = provideGeometricData(elements,boundary).
```

Hinweis: Die Funktion kann als 6-Zeiler und völlig ohne Schleifen programmiert werden.

2.5 REFINE: Verfeinerung durchführen

Mithilfe der obigen Matrizen kann nun die eigentliche Verfeinerung durchgeführt werden. Schreiben Sie dazu eine Funktion

```
[coordinates,newElements,newBoundary] = refineRGB(coordinates,elements,boundary).
```

2.6 Hauptprogramm

Das Hauptprogramm ist im Wesentlichen vorgegeben. Vollziehen Sie nach, was dort passiert, und probieren Sie die verschiedenen Verfeinerungs-Strategien (und evtl Bilder) aus.