Zentrum für Technomathematik

Prof. Dr. A. Schmidt
M.Sc. A. Narimanyan

Praktikum Numerik partieller Differentialgleichungen

SS 2001 — 31.05.2001
Abgabe: Mittwoch, 20.06.2001

Programmieraufgabe 4 (6 Punkte)

Zum Einarbeiten in das Programm–Paket ALBERT sind folgende Unterprogramme für zwei- und dreidimensionale Triangulierungen zu implementieren:

a) Schreiben Sie eine Funktion `random_refine(MESH *mesh, int k)`, die mit Hilfe der Funktion `drand48()` eine „zufällige“ Auswahl von Elementen des Gitters `mesh` zum Verfeinern markiert. Verfeiner Sie dann das Gitter durch einen Aufruf

```
refine(mesh);
```

Dieses „zufällige“ Markieren und Verfeinern soll `k` mal durchgeführt werden.

Implementieren Sie auch eine Funktion `random_coarsen(MESH *mesh, int k)`, welche eine „zufällige“ Auswahl von Elementen des Gitters `mesh` zum Vegröbern markiert und dann das Gitter mit

```
coarsen(mesh);
```

versucht zu vergröbern. Dies soll auch `k` mal durchgeführt werden.

Schreiben Sie schließlich eine Funktion `coarse_to_macro(MESH *mesh)`, die ein Gitter zurück bis zur Makrotriangulierung vergröbert. Eine Triangulierung ist genau dann Makrotriangulierung, falls für alle Makroelemente `macro_el` der Makrotriangulierung

```
macro_el->el->child[0] == nil
```

gilt. Die Makroelemente sind als verkettete Liste mit erstem Element

```
mesh->first_macro_el
```

gespeichert. Führen Sie nun mehrmaliges zufälliges Verfeinern, gefolgt von zufälligem Vergröbern und schließlich Vergröbern bis zur Makrotriangulierung durch. Geben Sie nach jedem Schritt die Anzahl der Elemente, Kanten und Knoten aus.

b) Implementieren Sie eine Funktion `refine_at_origin(MEHS *mesh, REAL dist)`, welche alle Dreiecke verfeinert, deren Schwerpunkt höchstens den Abstand `dist` zum Ursprung haben.

Tip: Bedenken Sie dabei, daß Koordinateninformation auf den Dreiecken vorhanden sein muß, um den Schwerpunkt zu berechnen. Daher muß das `FILL_FLAG`, welches der Traverse–Routine übergeben wird, `CALL_LEAF_EL|FILL_COORDS` sein.

Rufen Sie diese Funktion mehrfach mit kleiner werdendem Abstand `dist` auf (halbiere z.B. `dist` nach jedem Aufruf).

c) Schreiben Sie eine Funktion `measure_omega(MESH *mesh)`, welches auf jedem Dreieck die Elementfläche berechnet. Diese Werte sollen auf eine statische globale Variable aufaddiert werden. Nach dem Traverse enthält diese Variable dann den Flächeninhalt des zugrundeliegenden Rechengebiets.

In

```
http://www.math.uni-bremen.de/~schmidt/SS01/prakt4.tgz
```

befindet sich eine gezipte Tar-Datei, die Sie in ein eigenes ALBERT-Verzeichnis kopieren können und dort mit

```
gtar xovzf prakt4.tgz
```

entpacken. Damit werden im aktuellen Verzeichnis folgende Daten erzeugt:

```
Makefile

2d:
INIT/      Macro/     Makefile   albert.c@

3d:
INIT/      Macro/     Makefile   albert.c@

Common:
albert.c
```

Die Datei `albert.c` ist dimensionsunabhängig und daher *nur* in dem Unterverzeichnis `Common` gespeichert. Auf diese Datei ist jeweils in den `2d` und `3d` Verzeichnissen ein *Verweis* auf die Datei `../Common/albert.c` eingetragen. Alle anderen Daten in diesen Unterverzeichnissen sind dimensionsabhängig und separat gespeichert.
Für jeweils `2d` und `3d` enthält das Unterverzeichnis `INIT` eine Datei `init.dat`, in welcher Parameter initialisiert werden. Das Verzeichnis `Macro` enthält Daten zu verschiedenen Makrotriangulierungen.
Die Datei `albert.c` enthält den Quellcode für ein Demoprogramm und `Makefile` Information zum Übersetzen und Linken von ALBERT Programmen. Wichtigste Variable dabei ist `ALBERT_LIB_PATH`, welche den Pfad der ALBERT–Lib enthält. Ferner sind in dem Makefile die wesentlichen Konstanten, die die Triangulierung beschreiben, definiert: `DIM`, `DIM_OF_WORLD`, `NEIGH_IN_EL` und `EL_INDEX`. Im Makefile wird eine weitere Make–Datei eingefügt, in dem rechnerabhängige und ALBERT–spezifische Variable gesetzt werden.
Das Demoprogramm kann dann jeweils in `2d` oder `3d` mit dem Kommando

```
make
```

zum ausführbaren Programm `albert` übersetzt werden.

Bei Interesse ist auch eine Linux-Installation von ALBERT verfügbar.