Grafik-Ausgabe vom Parallelrechner für 3D-Gebiete

Magdalene Meyer

1 Einleitung

In dieser Dokumentation werden die an der TU Chemnitz entwickelten Grafikroutinen vorgestellt, mit deren Hilfe 3D-Netze und Ergebnisse visualisiert werden können. Die Netze und Ergebnisse entstehen dabei auf dem Parallelrechner, die Visualisierung geschieht auf einer Workstation, die mit dem Parallelrechner vernetzt ist (Ether-Net). Der Datentransfer (Knoten, Elemente, Lösungen, Steuervariable) läuft über Sockets. Grundlage des interaktiv arbeitenden Grafikprogrammes auf der Workstation bildet die objektorientierte Visualisierungs- und Numerikplattform GRAPE ([1]), von der im wesentlichen die Bibliothek `libgr.a` und das Headerfile `grape.h` benutzt wurden. Entsprechend der Verfügbarkeit von GRAPE liegt das auf der Workstation auszuführende Grafikprogramm für verschiedene Hardware vor:

- `f3_sun` für SUN-Workstations unter der Verwendung der X11-Grafik
- `f3_sgi` für SiliconGraphics-Workstations unter Verwendung der dort vorhandenen Hochleistungsgrafik.

Auf Parallelrechnerseite wurde die für die Grafik verantwortliche Funktion `gebgrape` so konzipiert, daß die Parameterliste ganz analog der der 2D-Routine `gebraf` ([4]) aufgebaut ist und der Aufruf auch in analoger Weise erfolgen kann. Sie ist die einzige Funktion, die der Nutzer in seinem Anwendungsprogramm auf dem Parallelrechner aufrufen muß, um die Grafik-Ausgabe zu realisieren.

2 Programmschnittstellen

Bevor im Anwendungsprogramm auf dem Parallelrechner das erste mal `gebgrape` aufgerufen wird, muß auf der Workstation das ausführbare Programm `f3_sun`

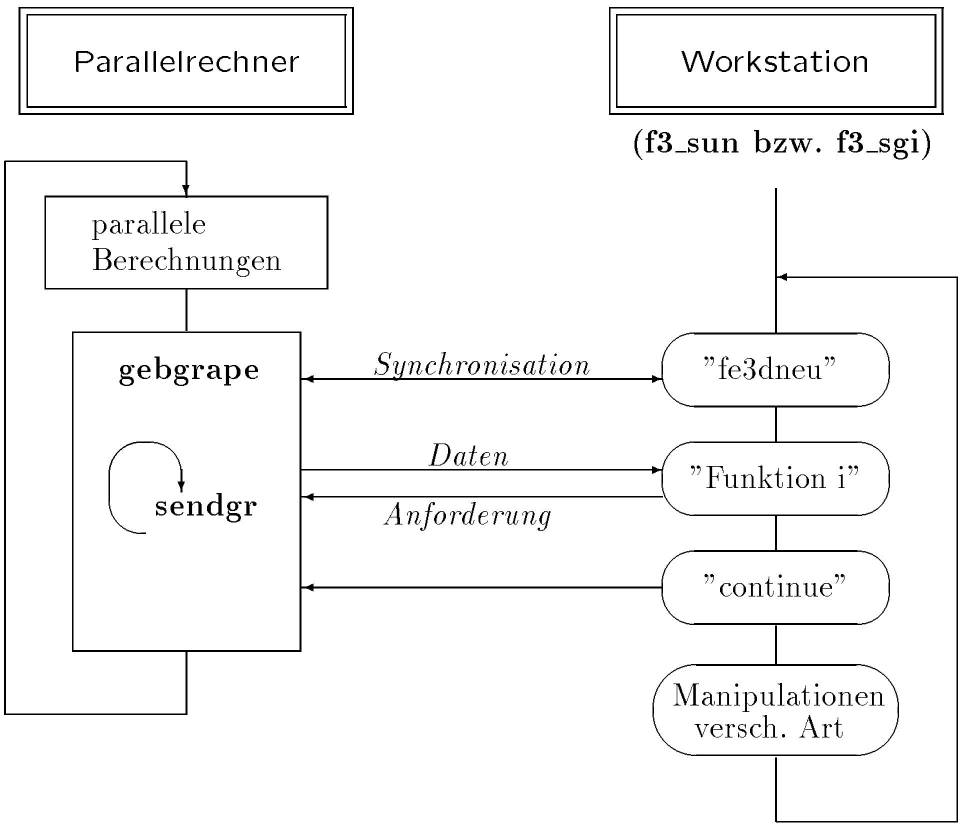

Abbildung 1: Zusammenwirken von Parallelrechner und Workstation

(bzw. `f3_sgi`) gestartet worden sein, welches den einen Teil der Socketverbindung öffnet. Nach Eingabe des im Netz bekannten Namens der Workstation über die Standardeingabe am Parallelrechner wird die Socketverbindung vollständig hergestellt. Die im Anwendungsprogramm als Parameter in der Rufzeile von `gebgrape` übergebenen Daten werden automatisch zur Workstation gesendet, wo ein Grafik- und ein Steuerfenster initialisiert werden, und es erfolgt eine Kommunikation zwischen beiden Rechnern, die durch den Nutzer über das Steuerfenster interaktiv gestaltet wird. Von nun an kann innerhalb des Programms am Parallelrechner die Funktion `gebgrape` jederzeit gerufen werden, und zwar notwendigerweise zugleich auf allen Prozessoren. Sie selbst ruft die Funktionen `sendgr` und `handlef` auf und beim ersten Durchlauf auch noch eine Funktion `getdofs` (siehe nächster Punkt). Intern wird geregelt, daß die Kommunikation mit der Workstation (Senden von Grafikdaten, Empfangen von Steuergrößen) nur über einen Prozessor (Prozessor 0) läuft, d. h., die lokalen Daten der einzelnen Prozessoren werden innerhalb von `gebgrape` vor dem Senden an Prozessor 0 gegeben. Zur Kommunikation zwischen den Prozessoren werden die in ([2]) dokumentierten gängigen Routinen verwendet.

An dieser Stelle muß noch folgendes bemerkt werden: Die Routinen `gebgrape`, `getdofs`, `sendgr` und `handlef` sind in der Programmiersprache C geschrieben. Um eine Verwendung auch in einer FORTRAN-Umgebung zu gewährleisten, wurden die Parameter generell als Pointer definiert. In einer C-Umgebung sind output-Parameter ohnehin als Pointer zu vereinbaren. Wird bei der Beschreibung der obengenannten Routinen in der Parameterliste ein Wert zum Beispiel als `int` ausgewiesen, so handelt es sich tatsächlich um einen Pointer auf `int`. Bei Feldern wird die Adresse des ersten Feldelementes übergeben.

3 Aufbau der Socketverbindung zwischen Parallelrechner und Workstation

In diesem Abschnitt wird beschrieben, wie die Verbindung zwischen Parallelrechner und Workstation für den Datentransfer hergestellt wird ([3]). Für jeden der beiden Rechner ist eine Funktion entwickelt worden, wobei beide funktionell zusammengehören. Sie liefern jeweils als Rückgabewert einen Filedeskriptor, der beim Lesen und Schreiben der Daten Verwendung findet. Diese Funktionen bleiben dem Nutzer der Grafikroutinen verborgen. Da sie aber auch unabhängig davon verwendbar sind, werden sie hier mit aufgeführt. Am Parallelrechner lautet der entsprechende Funktionsaufruf
`fdp = c_connect(port_nr, name)`.

`Parameter` (die Parameter sind input):

`port_nr` (int)	:	Portnummer, die als Konstante selbst definiert werden muß (z. B. 6667). Sie wird nur temporär während des Verbindungsaufbaus verwendet und muß mit der Portnummer der Funktion für die Workstation identisch sein.
`name` (char.)	:	Zeichenkette, die den Namen der Workstation enthält; `name(1000)`.

`fdp` ist der zum Lesen und Schreiben auf Parallelrechnerseite zu verwendende Filedeskriptor vom Typ `int`. Kernstück der Funktion `c_connect` ist der Aufruf der Systemfunktion `socket()`, weshalb die beiden Headerfiles `types.h` und `socket.h` eingebunden werden.

Auf der Workstation liefert der Aufruf
`fdw = net_init(port_nr, addr)` den benötigten Filedeskriptor `fdw`.

`Parameter:`

`port_nr` (int)	:	Portnummer, die als Konstante selbst definiert werden muß (z. B. 6667). Sie wird nur temporär während des Verbindungsaufbaus verwendet und muß mit der Portnummer der Funktion für den Parallelrechner identisch sein (input).
`addr` (int)	:	Hilfsgröße (output).

`fdw` ist der zum Lesen und Schreiben auf der Workstation zu verwendende Filedeskriptor vom Typ `int`. Im Gegensatz zur Funktion `c_connect`, wo der Name des "Partnerrechners" benötigt wird, spielt es für die Funktion `net_init` keine Rolle, mit welchem konkreten Rechner in Verbindung getreten wird. Es wird also nur eine Art "Briefkasten" aufgemacht. In der zeitlichen Reihenfolge muß das zuerst geschehen. Wenn der andere "Partner" dann aktiviert ist, steht die Verbindung. Kernstück der Funktion `net_init` ist der Aufruf der Systemfunktion `accept()`, weshalb auch hier die beiden Headerfiles `types.h` und `socket.h` eingebunden werden.

4 Beschreibung der am Parallelrechner für die Grafik zum Einsatz kommenden Funktionen

Wie schon oben erwähnt, ist der Aufruf der Funktion `gebgrape` die unmittelbare Schnittstelle für die Grafikanwendung. Die Funktionen `sendgr` und `handlef` können aber auch anderweitig als Bausteine benutzt werden, und werden in diesem Punkt deshalb ebenfalls kurz beschrieben.

```
call gebgrape (Iart, Nel, Ne, EL, Nkn, Nodes, U, H)
```

`Parameter` (alle Parameter sind input):

`Iart` (int)	:	Auswahl einer Darstellungsart; =0: nur Netzdarstellung (keine Lösungsdaten) >0: Nummer des zu Beginn der Grafik gewünschten darzustellenden Freiheitsgrades
`Nel` (int)	:	Anzahl der FEM-Elemente in `EL`
`Ne` (int)	:	Anzahl der Knoten pro Element
`EL` (int)	:	Elementliste `EL(Ne,Nel)`; enthält für jedes Element die Knotennummern
`Nkn` (int)	:	Anzahl der Knoten in `Nodes`
`Nodes` (real*4)	:	Knotenliste mit (x,y,z)-Koordinaten; `Nodes(3,Nkn)`
`U` (real*4)	:	Feld, das die Lösungen beinhaltet; `U(Nkn, . )` enthält jeweils für die einzelnen Freiheitsgrade einen Wert pro Knoten, d. h. zum Beispiel für eine vektorielle Lösung zunächst alle x-Werte, dahinter die y- und z-Werte.
`H` (int)	:	Hilfsfeld hinreichender Länge.

Die ersten drei Teile (jeweils der Länge `Nkn`) des Feldes `U` werden bei mehreren (≥ 3) Freiheitsgraden als dreidimensionales Feld interpretiert. Nach diesen drei Teilen können sich noch weitere Lösungen in `U` befinden. Beim ersten Aufruf der Funktion `gebgrape` wird die Socketverbindung zur Workstation hergestellt, nachdem deren Name über die Tastatur eingegeben wurde. Die Gesamtanzahl bzw. Bezeichnungen der Freiheitsgrade und ob eine vektorielle Lösung vorhanden ist, sowie deren Bezeichnung erfährt das Grafikprogramm durch den Aufruf einer Funktion `getdofs` innerhalb von `gebgrape`. Da die Anzahl und die Bedeutung der Freiheitsgrade vom Anwendungsbeispiel abhängen, ist es zweckmässig, wenn der Nutzer diese Funktion selbst schreibt:

```
call getdofs (nDoFs,DoFs)
```

`Parameter` (alle Parameter sind output):

`nDoFs` (int)	:	Anzahl der darstellbaren Freiheitsgrade (≤20)
`DoFs` (char.)	:	Feld von Zeichen, das hintereinander die Kurznamen (jeweils 15 Zeichen) für die Freiheitsgrade enthält, zuzüglich den Namen einer Vektorfunktion (siehe unten); `character*15 DoFs(nDoFs+1)`. Diese Namen dienen zugleich als Buttonbeschriftung im Grafiksteuerfenster.

Das Feld `DoFs` muß so belegt werden, daß es hintereinander die `nDoFs` Zeichenketten mit den Namen der Freiheitsgrade enthält. Daran schließt sich der Name einer Vektorfunktion an. Ist keine Vektorfunktion vorhanden, wird eine leere Zeichenkette angefügt. Es kann auch eine Standardfunktion `getdofs` genutzt werden, wobei dann festgelegt ist, daß `nDoFs` = 20 ist, d. h. daß im Lösungsfeld `U` zwanzig Freiheitsgrade hintereinander abgespeichert sind, und die Buttonbezeichnungen innerhalb der Grafikanwendung lauten "`Funktion1`", "`Funktion2`", ..., "`Funktion20`". Bei Betätigung des Buttons "`Vektor`" werden die ersten drei Freiheitsgrade als vektorielle Lösung dargestellt. Auch wenn die tatsächliche Anzahl der Freiheitsgrade kleiner als 20 ist, kann die vorhandene Funktion `getdofs` verwendet werden, im Steuerfenster dürfen dann die "überzähligen" Funktionsbutton einfach nicht genutzt werden. Letzteres gilt auch für den Button "`Vektor`", falls `nDoFs`≤2 ist. In Abbildung 2 ist ersichtlich, daß unter der voreingestelten Option 0 ("`opt 0`") nur vier Funktionsbutton zugänglich sind. Gibt es noch weitere, sind sie unter Option 1 zugänglich.
`gebgrape` ruft außerdem noch die Funktionen `sendgr` und `handlef`.

Das eigentliche Senden der Daten vom Parallelrechner an die Workstation leistet der Funktionsaufruf
`call sendgr (pkenn, panz, pfeld, fdd)`.

`Parameter` (alle Parameter sind input):

`pkenn` (int)	:	Kennziffer für Art der zu sendenden Daten
`panz` (int)	:	Anzahl der zu sendenden Daten
`pfeld` (int)	:	Datenfeld
`fdd` (int)	:	Filedeskriptor.

Wie `panz` konkret zu belegen ist, hängt von `pkenn` ab und wird weiter unten noch beschrieben. Je nach Belegung von `pkenn` werden entweder

- Informationen über das gesamte Netz,
- die Knotenliste (bzw. ein Teil davon),
- die Elementliste (bzw. ein Teil davon),

- oder die Funktionswerteliste (bzw. ein Teil davon)

an die Workstation geschickt.
Werden nur Teile der einzelnen Datenlisten geschickt (z. B. weil sie von verschiedenen Prozessoren stammen), wird `sendgr` entsprechend mehrere Male aufgerufen. Dies geschieht auf Prozessor 0, der wie schon oben erwähnt, vorher die Daten der anderen Prozessoren erhält. Dabei ist der Ablauf so, daß die Abarbeitungsreihenfolge beim Senden der Daten in der Knotenliste und in der Liste der Lösungen diesselbe ist. Es genügt also, die Knotenkoordinaten nur einmal zu senden.

`Beim ersten Aufruf` werden Informationen, die sich auf das gesamte Netz beziehen, gesendet. Dabei wird die folgende Belegung der Parameter realisiert:

`pkenn`	=	0,
`panz`	=	4,
`pfeld(1)`	=	Gesamtknotenanzahl,
`pfeld(2)`	=	Gesamtelementanzahl,
`pfeld(3)`	=	Knotenanzahl pro Element,
`pfeld(4)`	=	Nummer des als erstes zu übertragenden Freiheitsgrades; (`pfeld(4)` = 0: ohne Lösungen).

`Bei jedem weiteren Aufruf` haben die Parameter folgende Bedeutung:
`pkenn` = 5: Senden eines Knotenlistenteils,
`pkenn` = 6: Senden eines Elementlistenteils,
`pkenn` = 1: Senden eines Funktionswertelistenteils (Lösungen).
Entsprechend enthält dann `anz` die Anzahl der Knoten, Elemente, bzw. die Anzahl der Knoten, deren Funktionswerte durch diesen einen Funktionsaufruf gesendet werden sollen.
`pfeld` beinhaltet das jeweilige Datenfeld, das entsprechend aus `real`- oder `int`-Werten besteht.

Zur Steuerung des Sendens einzelner Lösungen dient der Funktionsaufruf
`call handlef(LOES, HILF, np, knot, fd).`

`Parameter` (alle Parameter sind input):

`LOES` (float)	:	Feld, das hintereinander alle Lösungen enthält
`HILF` (float)	:	Hilfsfeld
`np` (int)	:	Knotenanzahl des jeweiligen Prozessors
`knot` (int)	:	Gesamtknotenzahl (nur bei Prozessor 0 belegt)
`fd` (int)	:	Filedeskriptor (nur bei Prozessor 0 belegt).

Die Funktion `handlef` wird von `gebgrape` nur dann aufgerufen, wenn überhaupt Lösungen zur Verfügung stehen. In `handlef` werden solange auf Anforderung durch die Workstation Lösungsdaten an diese gesendet, bis ein Endesignal von der Workstation eintrifft, was ein Verlassen der Funktion bewirkt (vgl. Abb. 1). `handlef` selbst ruft noch die Funktion `sendgr`.

5 Beispiel für den Ablauf einer Sitzung und einige Abbildungen aus der Grafikanwendung

1. Start des Programmes `f3_sun` bzw. `f3_sgi` auf der Workstation. Es erscheint die Ausschrift "Warte auf Verbindungsaufnahme.".

2. Start des Anwendungsprogrammes (Netzgenerator, Lösungsalgorithmus) auf dem Parallelrechner.
   Dieses Programm muß irgendwo die Rufzeile `call gebgrape(...)` beinhalten. Nach der Aufforderung "Host=" erfolgt die Eingabe des Namens der Workstation. Im Beispiel sei `Iart`=1, d.h., zusammen mit den Netzdaten wird der erste Freiheitsgrad der Lösung an die Workstation automatisch übertragen.

3. Es erscheinen ein Steuerfenster (vgl. Abb. 2) und ein Grafikfenster (vgl. Abb. 3). Über das Steuerfenster kann unter anderem geregelt werden,

   - welche weiteren Freiheitsgrade dargestellt werden sollen (z. B. Button "`Funktion 2`"),

   - ob ein neues Beispiel behandelt werden soll (zuerst Button "`continue`", der ein Verlassen von `gebgrape` bewirkt, danach Button "`FE3D neu`", was einen neuen `gebgrape`-Ruf erwartet) oder

   - ob die Sitzung beendet werden soll (Button "`Exit`"). Das geschieht unabhängig vom weiteren Ablauf auf dem Parallelrechner, hat aber zur Folge, daß die Funktion `gebgrape` dort nicht mehr genutzt werden kann.

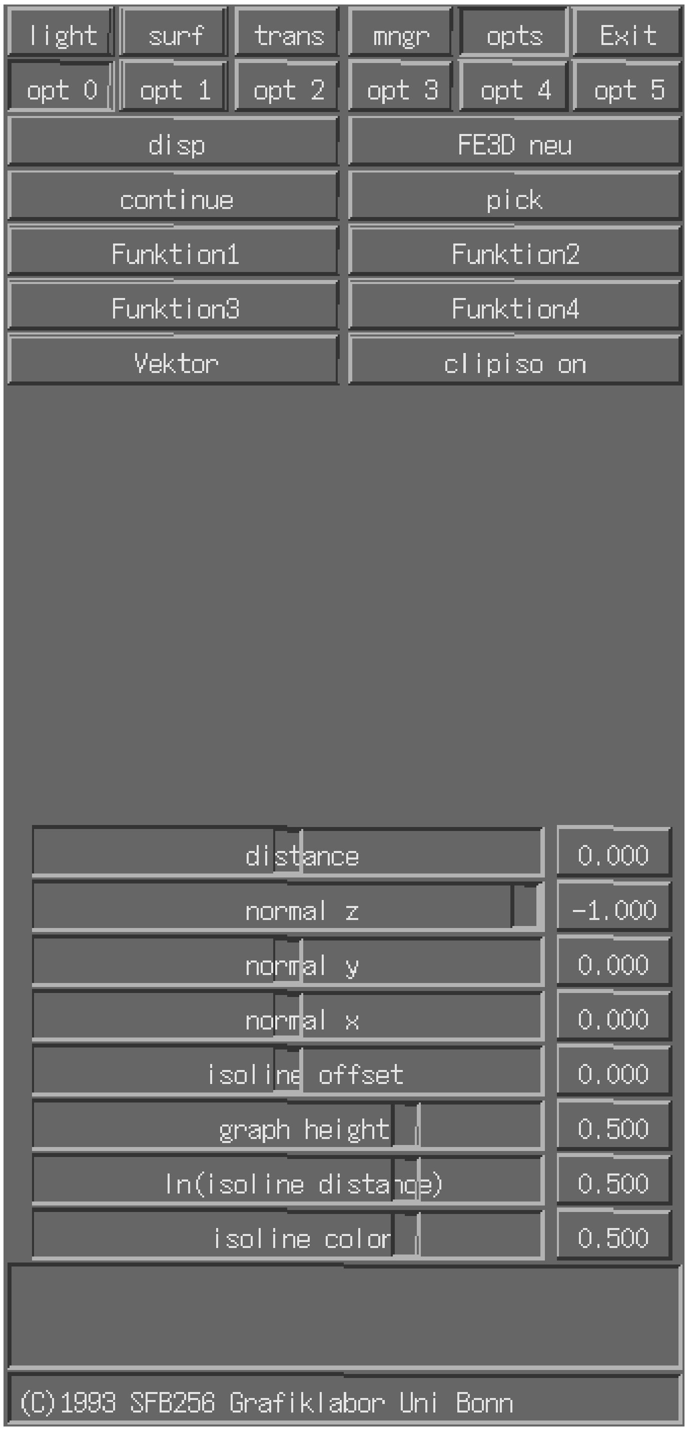

Abbildung 2: Fenster mit Steuermöglichkeiten für grafische Darstellungen

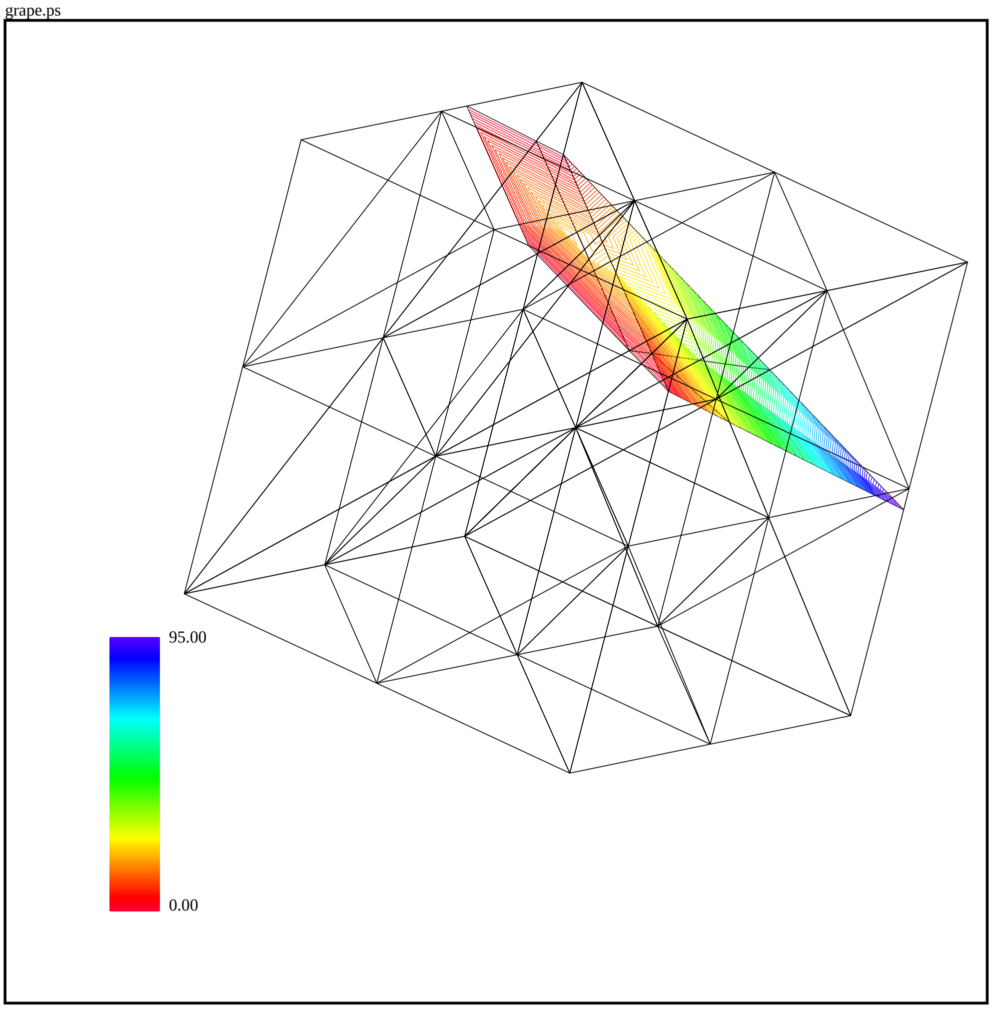

Abbildung 3: Schnittfläche, auf der eine Lösung dargestellt ist (Isolinien)

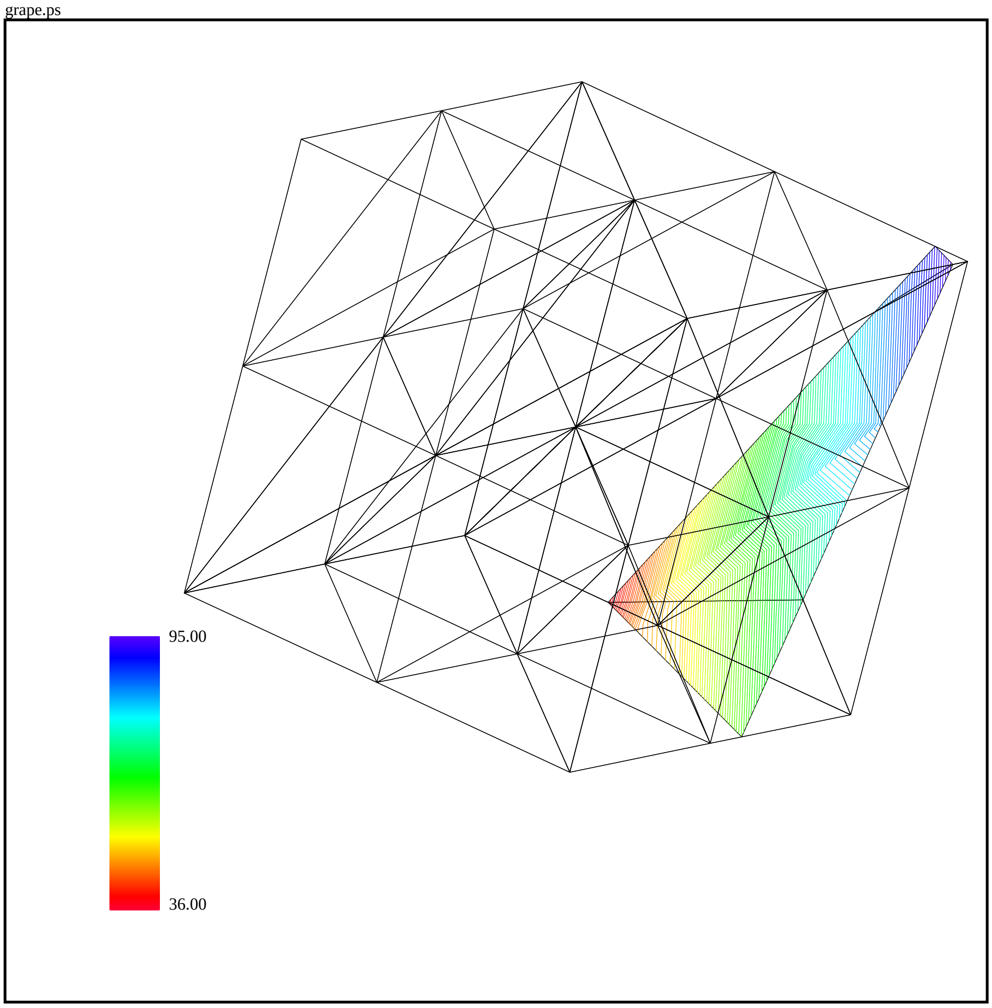

Abbildung 4: weitere Schnittfläche mit Lösung

Literatur

[1] Wierse, A. and Rumpf, M. (1992). GRAPE Eine objektorientierte Visualisierungs- und Numerikplattform. *Informatik Forsch. Entw.*, **7** 145-151.

[2] G.Haase, T.Hommel, A.Meyer, M.Pester. Bibliotheken zur Entwicklung paralleler Algorithmen. *Preprint SPC 94_4*, TU Chemnitz-Zwickau, 1994.

[3] Rieken, B. and Weimann L. (1992). *Adventures in UNIX Network Applications Programming.* John Wiley & Sons, Inc., New York - Chinchester - Brisbane - Toronto - Singapore.

[4] M.Pester. Grafik-Ausgabe vom Parallelrechner für 2D-Gebiete. *Preprint SPC 94_24*, TU Chemnitz-Zwickau, 1994.