Diplomarbeit
Fachhochschule Wedel

Tesselierung und Komprimierung von dreidimensionalen Geometriedaten

ANHANG

Martin Wendt
Fachrichtung: Technische Informatik
Referent: Dipl.-Ing. P. Pook-Haffmans

August 1992

[Konvertiert im November 2002]

INHALTSVERZEICHNIS ANHANG

Programmlistings

I.Bedienungshinweise

A.Hard- und Softwarevoraussetzungen

Comp3D benötigt zur Ausführung:

- Einen IBM-PC oder kompatiblen Rechner mit 80386 Prozessor oder höher.
- DOS 3.0 oder höher.
- Windows 3.0 oder höher.

Wünschenswert sind außerdem:

- Ein mathematischer Coprozessor.
- Mindestens 4MB Speicher (je mehr desto besser!)
- Farbgrafikkarte
- Festplatte > 100MB

B.Programmstart

Comp3D wird vom DOS Prompt mit

WIN COMP3D↵

gestartet.
Von Windows aus genügt ein Doppelklick mit der linken Maustaste auf das entsprechende Icon.

C.Menüstruktur

Nach dem Programmstart erscheint das Comp3D-Fenster. Zusätzlich wird das Fenster mit dem Hauptmenü geöffnet (1).

In diesem Fenster lassen sich sämtliche Optionen und Parameter einstellen. Von hier aus wird außerdem der Reduktionslauf gestartet[1].

Über die **[Input...]**-Taste gelangt man in eine Dialogbox, wo die Eingabedatei gewählt wird[2]. Je nach der Namenserweiterung (DXF oder ZOF) wird der entsprechende Dateityp angenommen.

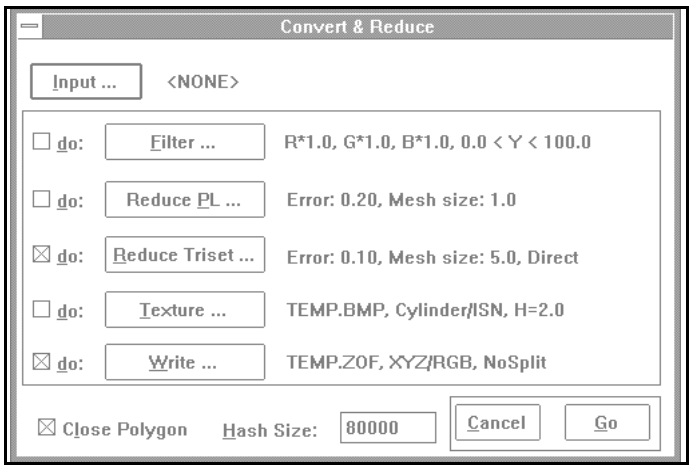

Abb. I.1 Comp3D Hauptmenü

Am linken Fensterrand können über Checkboxes einzelne Optionen an- oder abgeschaltet werden. Über die daneben angeordneten Tasten gelangt man in Untermenüs, wo man die Parameter der einzelnen Funktionsgruppen angeben kann (nähers dazu im folgenden Abschnitt). Rechts daneben sind die wichtigsten aktuellen Einstellungen für diese Gruppe angezeigt.

Am unteren Fensterrand sind zwei weitere Einstellungen möglich:

1. Wenn **Close Polygon** gesetzt ist, werden der erste und der letzte gelesene Polygonzug tesseliert, so daß ein rundherum geschlossenes Triset entseht.
2. **Hash Size** definiert die Anzahl von Einträgen, die in der Hash-Tabelle reserviert werden sollen. Dieser Wert sollte mindestens doppelt so groß sein wie die Zahl der eingelesenen Vertizes. Pro Eintrag werden vier Bytes belegt. Falls es während des Einlesens zu Clusterbildung kommt (weil die Tabelle zu klein gewählt wurde), wird auf Hashing verzichtet. Durch Hashing kann der Lesevorgang stark beschleunigt werden[3].

Über die Taste **[Go]** wird der Reduktionslauf mit den vorgegebenen Parametern gestartet. **[Cancel]** schließt die Dialogbox.

Alle eingestellten Parameter werden in einer Datei gespeichert und bleiben auch nach dem Programmende erhalten.

---

[1] Alle Funktionen des Hauptmenüs lassen sich auch über die Menüleiste am oberen Bildschirmrand erreichen.

[2] Bei Eingabe bestimmter Schlüsselworte werden die Daten nicht eingelesen, sondern künstlich generiert. Diese Schlüsselwörter sind: '**Sphere**', '**Box**', '**Cylinder**' und '**Plane**'.

[3] Allgemein gilt: je größer die Tabelle, desto schneller das Programm. Andererseits darf **Hash Size** aber auch nicht zu groß gewählt werden, weil die Tabelle bei kleiner Speicherausstattung des Rechners sonst von Windows ausgelagert wird ('Paging'). In diesem Fall würde die Rechengeschwindigkeit stark abfallen.

D.Hinweise zur Wahl der Parameter

1.Filterfunktionen

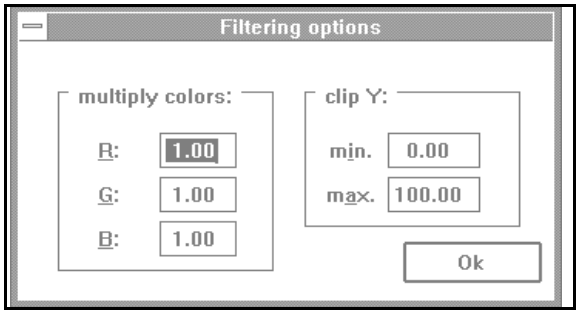

Abb. I.2 Filteroptionen

Diese Option erlaubt es, die Farben der Eingabedaten getrennt nach Rot-, Grün- und Blauanteil mit einem Faktor zu Multiplizieren. Auf diese Weise kann z.B. das Objekt aufgehellt (R=G=B=2) oder rötlich eingefärbt werden (R=5, G=B=1).

Über die Clipping-Option können Punkte mit zu großer oder zu kleiner Y-Koordinate herausgefiltert werden. Diese Möglichkeit wurde eingeführt, um ausgefranste Ränder, wie sie beim Digitalisieren entstehen können, zu begradigen.

2.Reduktion der Polygonzüge

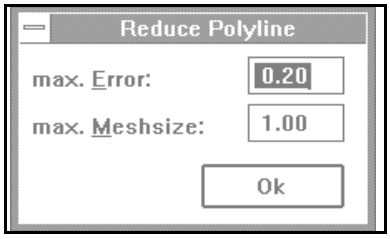

Abb. I.3 Reduktion 1

Hier werden die Restriktionen für die erste Stufe der Reduktion definiert: der größte zulässige Fehler ε_{max} und der maximale Abstand zwischen zwei Punkten l_{max}.

Die Einheit entspricht der Einheit der Eingabedaten. Beim Digitizer ist dies 1 cm. Um beispielsweise zu erreichen, daß Original und reduziertes Triset um nicht mehr als 1mm voneinander abweichen, muß $\varepsilon_{max} = 0{,}1$ angegeben werden.

3.Reduktion des Triset, Annealing

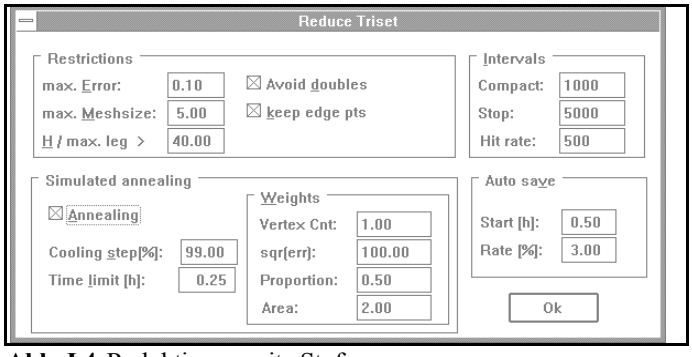

Abb. I.4 Reduktion, zweite Stufe

Diese Dialogbox ist die umfangreichste, weil hier die vier Gruppen von Optionen eingestellt werden können.

1.**Restrictions**: die hier angegebenen Restriktionen gelten sowohl für die direkte Reduktion als auch für das Simulated Annealing. Neben ε_{max} und l_{max} wird hier die 'maximal zulässige Schmalheit' eines Face definiert (H/max leg: Verhältnis von der längsten Seite zur Höhe darüber).

Keep Edge Points unterbindet das Löschen von Randpunkten, um ein 'Ausfransen' der Ränder zu verhindern.

Avoid Doubles prüft vor jeder Änderung des Triset, ob doppelte Edges entstehen würden. Auf diese Weise wird die Entstehung von entarteten Faces verhindert.

2.**Simulated Annealing**: Zunächst wird hier zwischen **Annealing** und direkter Reduktion gewählt. Außerdem kann der Benutzer die gewünschte Laufzeit wählen (bzw. einen konstanten Abkühlfaktor, wenn **Time Limit** zu 0 gesetzt wird).

Im Feld **Weights** wird der Anteil definiert, mit dem die einzelnen Faktoren in die Berechnung der Energie eingehen sollen.

3.**Intervals**: **Compact** gibt an, wie häufig eine Garbage Collection für das Triset durchgeführt werden soll. Nachdem <**Stop**> Vertizes nicht gelöscht werden konnten, wird der Reduktionslauf beendet.

HitRate definiert ein Intervall, über das die dynamischen Mittelwerte gebildet werden.

4.**Auto Save**: Hier wird die Zeit angegeben, die vergehen muß bevor das erste Backup angelegt wird. **Rate** gibt an, um wieviel Prozent die Energie jeweils reduziert werden muß, bevor eine weitere Sicherheitskopie erstellt wird.

4.Texture Mapping

Diese Dialogbox enthält die Parameter für das Two-Part Mapping. Dazu gehören der Name der Ausgabedatei, Mappingmethode und Zwischenfläche.

U und **V** definieren die Größe der Texturbitmap (U x V Pixel). **H** wird benötigt, wenn als Zwischenfläche ein Zylindermantel gewählt wurde und gibt dessen Höhe an. Die Einheit ist abhängig von der Einheit der Eingabedaten.

Wenn **Interpolate** gesetzt ist, werden während des Texture Mappings die Farben zwischen den einzelnen Punkten in der Textur interpoliert.

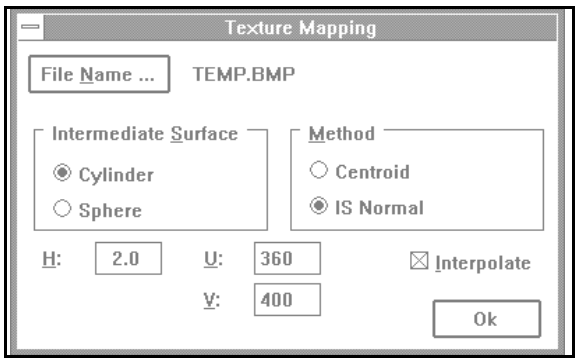

Abb. I.5 Mappingoptionen

5.Ausgabe

Hier werden der Dateiname und -Format (ZOF, DXF) der Ausgabedaten definiert.

Außerdem muß hier angegeben werden, welche Informationen ausgegeben werden sollen (nur die Koordinaten, Koordinaten+Farbe oder Koordinaten+Texturparameter).

Die Option **Small** sorgt dafür, daß auf eine spaltenweise Formatierung zugunsten einer kompakten Datei verzichtet wird.

Im Feld rechts unten kann ggf. erzwungen werden, daß das Triset in mehren kleinen Blöcken geschrieben wird.

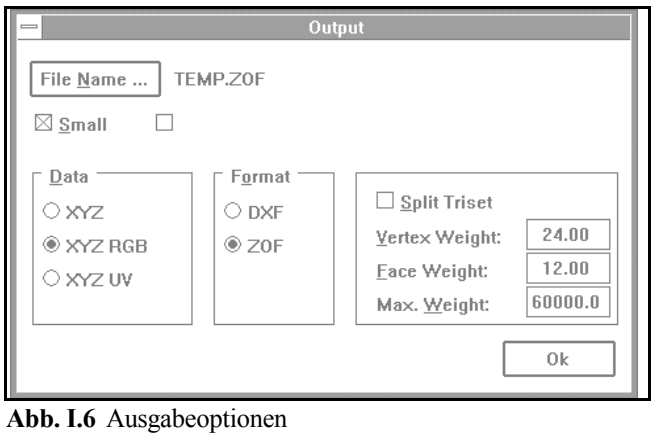

Abb. I.6 Ausgabeoptionen

Für jeden Block gilt:

VertexWeight·N_V + **FaceWeight**·N_F < **MaxWeight**

E.Statusinformationen, Logfiles

Da ich während der Entwicklung von Comp3D viel mit verschiedenen Parametern und Algorithmen experimentiert habe, brauchte ich umfangreiche Informationen zur Auswertung der Testläufe.
Während des Programmlaufs werden deshalb in mehreren Fenstern und Dateien Informationen protokolliert.

1.Statusfenster

Das Statusfenster öffnet sich, sobald die die Reduktion gestartet wird. Es ist in mehrere Bereiche unterteilt:

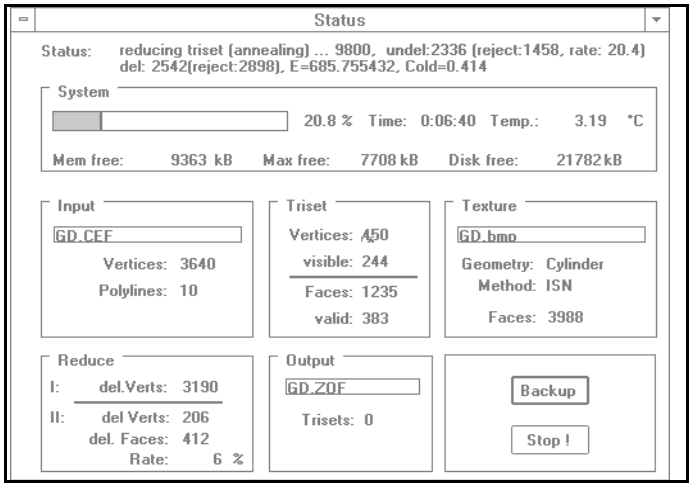

Abb. I.7 Comp3D Statusfenster

1.**Statuszeile** mit Angaben über Anzahl der untersuchten Punkte und mittlere Lösch- und Entlöschraten.
2.**System**: Angaben über Effizienz (Balken), Laufzeit, Temperatur und freien Speicherplatz.
3.**Input**: Name der Eingabedatei, Anzahl der gelesenen Punkte und Polygonzüge.
4.**Triset**: Anzahl der Vertizes im Triset (insgesamt und sichtbar), sowie Anzahl der Faces. Dabei ist zu beachten, daß das Face-Array ständig wächst, weil Faces nich gelöscht, sondern nur markiert werden. Erst nach einer Garbage Collection wird die Arraygröße ('Faces') auf die minimale Größe ('valid') komprimiert.
5.**Texture**: Name der Bitmapdatei, Mappingmethode und Anzahl der projezierten Faces.
6.**Reduce**: Anzahl der erfolgreich gelöschten Punkte und Faces beider Reduktionsstufen. Reduktionsrate.
7.**Output**: Name der Ausgabedatei, Anzahl der geschriebenen Trisets.

2.History Graph

Hier werden wichtige Reduktionsparameter über der Zeit aufgetragen. Diese Parameter sind: Anzahl der gelöschten Punkte, Effizienz, Temperatur und Energie (die letzten beiden Werte nur, wenn Simulated Annealing gewählt wurde).
Diese Werte können für spätere Auswertungen zusätzlich in eine Textdatei geschrieben werden (vgl. Kapitel VIII im Hauptband).

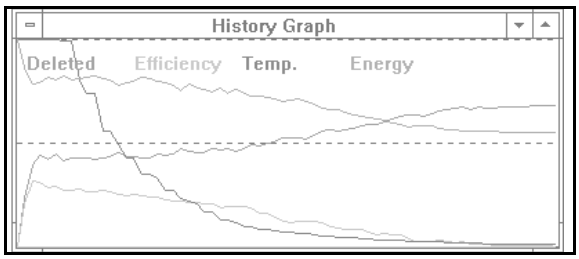

Abb. I.8 Historyfenster

3.Output Window

In dieses Textfenster werden Debuginformationen zu Testzwecken geschrieben.

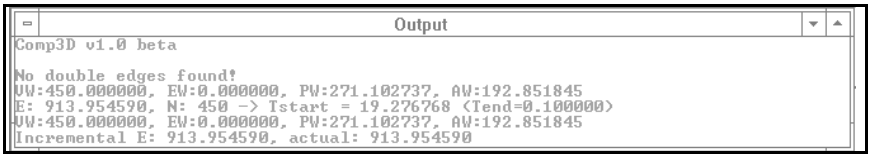

Abb. I.9 Output Window

4.Log-Dateien

Über die Menüleiste, Unterpunkt '**Debug**' können verschiedene Protokolldateien erzeugt werden:

1.**CEF-Log**: Während des Einlesens wird der Inhalt der binären Cef-Datei in Klartext geschrieben (→Dateiname: 'CefOut.dbg').
2.**History-Log**: Protokolliert Reduktionsparameter in regelmäßigen Abständen. Diese Textdatei kann z.B. von Tabellenkalkulationsprogrammen ausgewertet werden (→Dateiname wie Triset-Datei, Erweiterung: '.prn').
3.**Parameter-Log**: Protokolliert sämtliche eingestellten Parameter eines Reduktionslaufes (→Dateiname wie Triset-Datei, Erweiterung: '.log').
4.**Output-Log**: Über das lokalen Menü des Output Window kann dessen Inhalt in eine Textdatei kopiert werden (→Dateiname frei wählbar).

II.Mathematische Grundlagen

A.Abstand Punkt-Punkt, Punkt-Linie

Für den Abstand von zwei Punkten P_1 und P_2 im dreidimensionalen Raum gilt:

$$d_{12} = \left| \vec{p}_1 - \vec{p}_2 \right| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Der minimale Abstand zwischen einem Punkt R und einer Geraden $P_0+t\cdot v$ berechnet sich wie folgt (nach [Foley90], Seite 1100):

$$d_{\min} = \left| \vec{R} - \vec{P}_0 + \frac{(\vec{R} - \vec{P}_0 \bullet \vec{v})}{\vec{v} \bullet \vec{v}} \bullet \vec{v} \right|$$

B.Abstand Punkt-Ebene

Der minimale Abstand zwischen einem Punkt und einer Ebene, die durch die Vektoren k und l aufgespannt wird, ist entlang der Senkrechten N zwischen Punkt und Fläche zu messen.

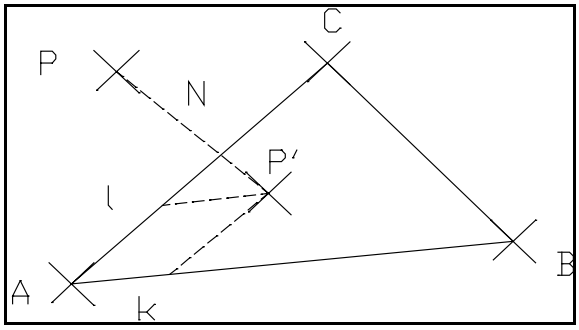

Abb. II.10 Abstand Punkt-Ebene

Es gilt:

und

$$\vec{P} = \vec{P}' + \gamma\vec{N}$$

mit

$$\vec{P}' = \vec{A} + \alpha\vec{k} + \beta\vec{l}$$

Umformung und Einsetzen ergibt:

$$\vec{k} = \vec{B} - \vec{A}$$

$$\alpha\vec{k} + \beta\vec{l} + \gamma\vec{N} = \vec{P} - \vec{A}$$

$$\vec{l} = \vec{C} - \vec{A}$$

$$\vec{N} = \vec{k} \times \vec{l}$$

Der Betrag von $\gamma\vec{N}$ ist der minimale Abstand d_{min}.

C.Abstand Punkt-Dreieck

Der minimale d_{min} Abstand zwischen einem Punkt P und einem Dreieck _ABC wird durch Fallunterscheidungen ermittelt.
Entscheidend ist, ob P' innerhalb oder außerhalb des Dreiecks liegt (11).
Wenn P senkrecht über _ABC liegt, dann ist d_{min} identisch mit dem Abstand zwischen P und der Ebene, die das Dreieck aufspannt.

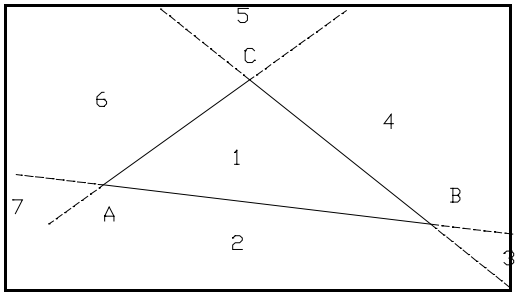

Abb. II.12 Sieben Zonen

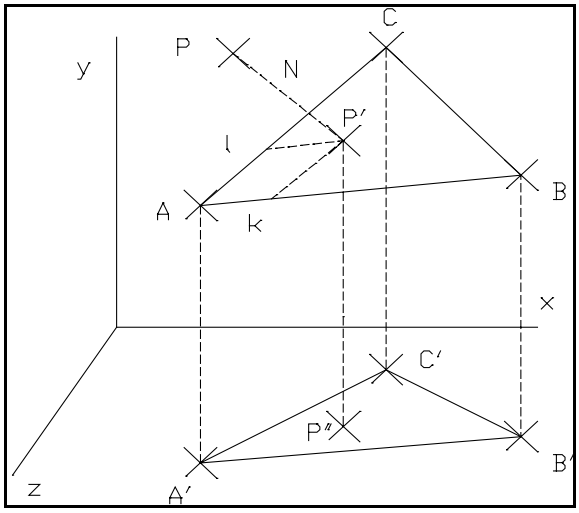

Abb. II.11 Abstand Punkt-Dreieck

Anderenfalls ist der kleinste Abstand zwischen P und dem Rand von _ABC entscheidend. Die möglichen Fälle sind in 12 zu sehen.
In Sonderfall Nr. 5 ist z.B. d_{min} gleich dem Abstand P,C. Für Fall 2 ist der Abstand zwischen P und der Strecke AB maßgebend usw.

D.Volumen eines Tetraeders

Das Volumen V eines Tetraeders ist ein drittel vom Produkt aus Grundfläche und Höhe darüber.

E.Volumen zwischen zwei Trisets

Da ich das Volumen ΔV zwischen einem Patch und seinem Tesselierten Nachbarpolygon berechnen möchte (vgl. Hauptband Kapitel VI.C.2.b.1), lauten die Randbedingungen:

- Beide Trisets sind durch das selbe Polygon begrenzt (in der Grafik: ABCD).
- Ein Triset trianguliert das Randpolygon (in der Grafik gepunktet).
- Das andere Triset hat einen zusätzliche Vertex V in der Mitte (in der Grafik gestrichelt dargestellt).

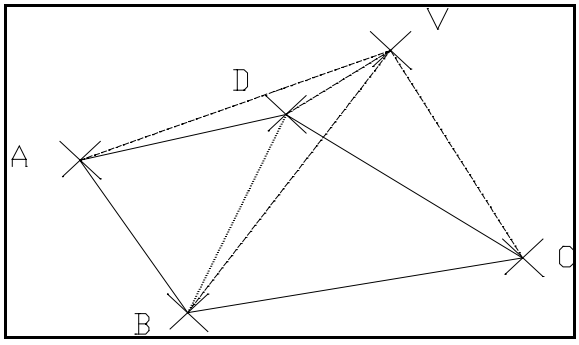

Abb. II.13 Volumendifferenz

Das Volumen läßt sich aus der Summe der Volumen der Tetraeder berechnen, die aus V und den Dreiecken des kleineren Patch bestehen (im Beispiel: V,ABD und V,BCD).

F.Dynamischer Mittelwert

Ich benötige an mehreren Stellen Mittelwerte über das Verhältnis von erfolgreichen Versuchen zu deren Gesamtzahl. Damit ich nicht eine Tabelle mit N einträgen verwalten und regelmäßig aufsummieren muß, begnüge ich mit einer Abschätzung.
Als Modell dient mir dabei ein Becken mit ständigem Zufluß ΔV und einem Überlauf. Das Gesamtvolumen V bleibt konstant. Bei einem erfolgreichem Versuch wird farbige Flüssigkeit zugeführt, sonst Wasser. Der abgeschätzte Mittelwert M entspricht dem Anteil der farbigen Flüssigkeit.
Für einen Zeitraum der Mittelwertsbildung von N Takten gilt:

1. Bei jedem Takt fließt 1/N des Beckeninhalts ab, und damit auch der gleiche Anteil von M (gleichmäßige Verteilung der Flüssigkeit vorausgesetzt).
2. Bei jedem erfolgreichen Takt erhöht sich M um 1/N.

Um das umzusetzen wird M bei **jedem** Takt neu berechnet:

$$M = \frac{N - 1}{N} M$$

zusätzlich gilt bei jedem **erfolgreichen** Takt:

$$M = M + \frac{1}{N}$$

III.Programmbeschreibung

Das Programm Comp3D wurde mit Borland C++ 3.1 unter Windows erstellt. Der Sourcecode umfaßt gut 13.000 Zeilen. Die kompilierte, lauffähige Datei hat eine Größe von ca.200 kBytes.

A.Datenflußplan

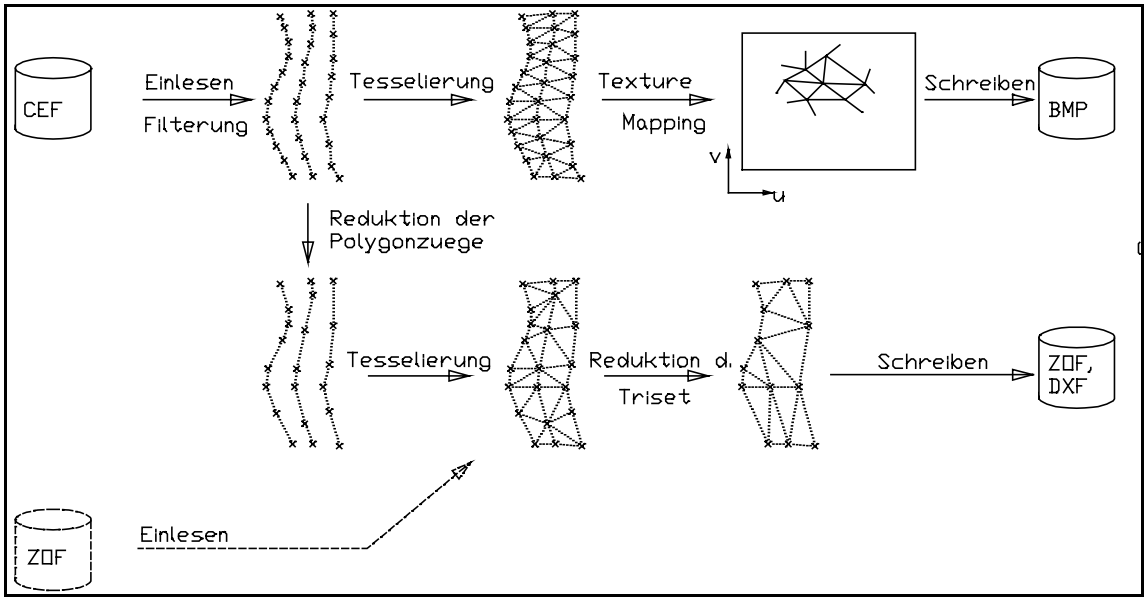

Abb. III.14 Comp3D: Datenflußplan

B.Programmorganisation

Die Aufteilung der Quelldateien, Module und Klassen ist in etwa identisch, d.h. fast jedes Modul wurde in einer Klasse gekapselt[4]. Diese Klassen sind in der Regel in eigenen Dateien definiert.

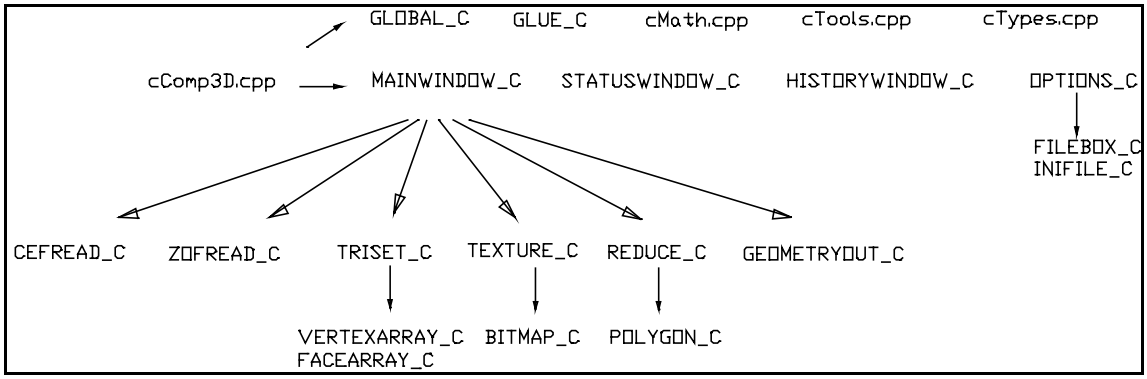

Abb. III.15 Klassenabhängigkeiten

[4] Ausnahmen: cComp3D.cpp (Startmodul und Message Loop), cMath.cpp, cTypes.cpp, cTools.cpp.

Comp3D ist als typisches Windowsprogramm ereignisgesteuert. Deswegen ist es nicht möglich, dafür einen strenge, baumartige Hierarchie anzugeben. In 15 versuche ich dennoch die logischen Abhängigkeiten grafisch aufzuzeigen. Die großgeschriebenen Namen mit dem Postfix '_C' bezeichnen Klassen[5]. Außerdem gibt es allgemeine nützliche Funktionen, die nicht als Methoden in Klassen definiert wurden. In diesen Fällen habe ich stattdessen die Datei angegeben, in der sie definiert sind (z.B.: cMath.cpp).
Alle Module, die in der obersten Zeile der Grafik 15 liegen, sind global für alle anderen Module zugänglich (z.B. mathematische Routinen, nützliche Standardfunktionen, PHIGS-Typen).
In der zweiten Zeile sind Klassen aufgeführt, die verschiedene Fenster verwalten. Sie arbeiten ereignisgesteuert quasiparallel. Die wichtigste Aufgabe kommt hierbei MAINWINDOW_C zu, wo der eigentliche Reduktionslauf koordiniert wird.
Die darunterliegenden Klassen sind für Teilaufgaben der Reduktion zuständig. Die Randbedingungen und Parameter der Reduktion werden zentral in OPTIONS_C verwaltet.
Der Eintrittspunkt von Comp3D liegt in cComp3D.cpp. Hier wird das Hauptfenster geöffnet und GLOBAL_C initialisiert. Danach wird bis zum Programmende der Windows Message Loop ausgeführt.
GLOBAL_C übernimmt die Initialisierung aller anderen Objekte.
Im Hauptfenster aus werden die Befehle des Benutzers ausgewertet und dadurch der Programmablauf gesteuert.

C.Beschreibung der Klassen

Ich benutze Objekte hauptsächlich zur Datenkappselung. Mehrfachvererbung, Friend-Classes und tiefe Klassenhierarchien habe ich vermieden.
Da ich ich nur in einem Fall Vererbung anwende (Face- und Vertexarray werden von DYNARRAY_C abgeleitet), erübrigt sich eine grafische Darstellung.
Es folgt nun eine Beschreibung der benutzten Klassen.

GLOBAL_C: Von dieser Klasse existiert genau ein statisches globales Objekt ('**Global**'). Hier sind Daten und Methode abgelegt, die für alle anderen Objekte zugänglich sein sollen. Dazu gehören:
- Methoden zur Initialisierung aller anderen Objekte
- Zeiger auf andere Objekte
- Semaphore, Flags zur Ablaufsteuerung und globale Variablen
- Methoden zur Ausgabe von Debugmeldungen

GLUE_C: Erleichtert die Zuordnung zwischen Fensterhandles und Objekten.

MAINWINDOW_C:
Messagehandler und Menü-Handler für das Comp3D Programmfenster, About-Box, Programmablaufsteuerung.

STATUSWINDOW_C:
Messagehandler und Ausgaberoutinen für das Statusfenster.

HISTORYWINDOW_C:
Grafische Ausgabe von Zahlenfolgen ('Fieberkurven').

OPTIONS_C: Messagehandler und Daten aller Optionsdialogboxen. Die meisten anderen Objekte greifen

[5] Von fast jeder dieser Klassen existiert nur eine Instanz (Objekt). Ein Zeiger darauf ist global in GLOBAL_C gespeichert, wo auch die Initialisierungen vorgenommen werden.

auf diese Parameter zu.

FILEBOX_C: Stellt eine Dialogbox zur Eingabe von Dateinamen bereit.

INIFILE_C: Methoden zum lesen und schreiben der Konfigurationsdatei.

CEFREAD_C: Lesen von CEF-Dateien, Erzeugen von Simulationsdaten.

ZOFREAD_C: Einlesen von ZOF-Dateien.

TRISET_C: Methoden und Datenstrukturen zur Verwaltung von beliebig großen Trisets. Zur Optimierung wurden Hashing-Algorithmen und Assemblerroutinen implementiert.

DYNARRAY_C: Allgemeine Klasse zur Verwaltung von dynamischen Arrays (als Template realisiert). Auf dieser Klasse basieren das VERTEXARRAY_C und FACEARRAY_C eines Trisets.

TEXTURE_C: Two-Part Mapping für verschiedene Methoden und Zwischenflächen.

BITMAP_C: Initialisierung und Zugriff auf Bitmaps, Schreiben von BMP-Dateien.

REDUCE_C: Hier sind die Algorithmen für die Reduktion von Polygonzügen, direkte Reduktion von Trisets und Simulated Annealing implementiert.

POLYGON_C: Klasse zur Verwaltung von Polygonzugdaten.

GEOMETRYOUT_C:
Methoden zum Schreiben von Trisets (ZOF- und DXF-Format)

D.Modulbeschreibungen

Folgende Dateien werden benutzt:

cBitmap.cpp: Implentierung von BITMAP_C.
cCefRead.cpp: Implentierung von CEFREAD_C.
cComp3D.cpp: WinMain, Messagelopp
cFileBox.cpp: Implentierung von FILEBOX_C.
cGeomOut.cpp: Implentierung von GEOMETRYOUT_C.
cGlobal.cpp: Implentierung von GLOBAL_C, Deklaration von 'Global'.
cGlue.cpp: Implentierung von GLUE_C.
cHistWin.cpp: Implentierung von HISTORYWINDOW_C.
cIniFile.cpp: Implentierung von INIFILE_C.
cMainWin.cpp: Implentierung von MAINWINDOW_C, About-Dialogbox.
cMath.cpp: Mathematische Funktionen
cMem32.asm: 32-Bit Assemblercode für schnelle Suche in den Trisetdaten
cOptions.cpp: Implentierung von OPTIONS_C.
cPolyLin.cpp: Implentierung von POLYGON_C.
cReduce.cpp: Implentierung von REDUCE_C.
cStatWin.cpp: Implentierung von STATUSWINDOW_C.
cTexture.cpp: Implentierung von TEXTURE_C.
cTools.cpp: Allgemeine Funktionen für Stringverarbeitung, Filehandling, etc.
cTriset.cpp: Implentierung von TRISET_C, FACEARRAY_C, VERTEXARRAY_C.
cTypes.cpp: Allgemeine Typdefinitionen
cZofRead.cpp: Implentierung von ZOFREAD_C.

Für alle '.cpp' Dateien exisiert außerdem ein Headerfile gleichen Namens mit der Erweiterung '.hpp'

E.Hauptschleife

```
PROZEDUR comp3dMain
PARAMETER:        -

Statusfenster initialisieren
IF Eingabe im ZOF-Format THEN
        ZOF-lesen
ELSE
        readNextPolyline (P1)
        filterPolyLine (P1)
        reducePolyline (P1)
        REPEAT
                readNextPolyline (P2)
                filterPolyLine (P2)
                mapTristrip (P1, P2)
                reducePolyline (P2)
                addTristrip (P1, P2)
                P1 ← P2
        UNTIL End of file
ENDIF
writeTexture
reduceTriset
writeTriset
```

Alg. III,1 Comp3D Hauptschleife

Der Algorithmus zeigt den prinzipiellen Ablauf eines Comp3D Reduktionslaufes. **comp3dmain()** ist als Methode von MAINWINDOW_C implementiert.

Programmlistings

in alphabetischer
Reihenfolge

[→ siehe Quellcodedateien im Anhang]

Quellen:

[Foley90] Computer Graphics, Principles and Practice
Foley, van Dam, Feiner, Hughes 1990 Addison-Wesley
Publishing Company 9

Abbildungen: