Hochschule für Architektur und Bauwesen Weimar
– Universität –
Fakultät Informatik und Mathematik

view4d

– ein Programm, das die vierte Dimension begreifbar macht –

entstanden im Rahmen der Lehrveranstaltung
Computergestützte Bauplanung III, Teilgebiet Virtuelle Realität in der Architektur
Prof. Dr. Ing. Dirk Donath, Dipl. Ing. Holger Regenbrecht

von
Marko Meister[‡]

Student der Informatik
Fakultät Informatik und Mathematik
4/91/$\mathcal{A}$

Weimar, den 25. Januar 1995

[‡] meister1@nessi.informatik.hab-weimar.de.

Zusammenfassung

Die vorliegende Arbeit zeigt, wie ein vierdimensionaler Körper anschaulich dargestellt werden kann. Diese Darstellung erfolgt mit Hilfe eines Computerprogrammes. Das Programm wurde in $\mathcal{C}$ unter **HP–UX** erstellt und mit dem **HP–UX** $\mathcal{C}$–Compiler `cc` compiliert. Für die Grafik wurde auf das Grafiksystem **PEX** zurückgegriffen, außerdem wurden diverse $\mathcal{X}$-Bibliotheken benutzt. Als Eingabemedium wurden Maus und Tastatur ausgewählt. Die Ausgabe erfolgt auf dem Bildschirm wahlweise als Stereobild (Rot/Grün) oder als Darstellung mit gefüllten Flächen.

Inhaltsverzeichnis

1 Theoretische Grundlagen

Will man sich im vierdimensionalen Raum[1] Körper vorstellen, so muß man zuerst verschiedene Fragestellungen beantworten. Die erste Frage ist die nach dem Körper, der dargestellt werden soll. Im Beispielprogramm wird ein Körper dargestellt, der das 4D–Analogon zum Würfel (im dreidimensionalen Raum[2]) bzw. zum Quadrat (im zweidimensionalen Raum[3]) ist.

1.1 Eckpunkte

Welche Punkte bilden nun die Eckpunkte eines solchen Körpers? Man geht vom zweidimensionalen Fall aus und erhält die folgenden Eckpunktkoordinaten eines Quadrates im 2D:

Punkt	$coord_x$	$coord_y$
p_0^{2D}	-1	-1
p_1^{2D}	-1	1
p_2^{2D}	1	-1
p_3^{2D}	1	1

Für die dritte Dimension sieht das dann so aus:

Punkt	$coord_x$	$coord_y$	$coord_z$
p_0^{3D}	-1	-1	-1
p_1^{3D}	-1	-1	1
p_2^{3D}	-1	1	-1
p_3^{3D}	-1	1	1
p_4^{3D}	1	-1	-1
p_5^{3D}	1	-1	1
p_6^{3D}	1	1	-1
p_7^{3D}	1	1	1

Im 4D sollte das ganze also analog so aussehen:

Punkt	$coord_x$	$coord_y$	$coord_z$	$coord_w$
p_0^{4D}	-1	-1	-1	-1
p_1^{4D}	-1	-1	-1	1
p_2^{4D}	-1	-1	1	-1
p_3^{4D}	-1	-1	1	1
p_4^{4D}	-1	1	-1	-1
p_5^{4D}	-1	1	-1	1
p_6^{4D}	-1	1	1	-1
p_7^{4D}	-1	1	1	1
p_8^{4D}	1	-1	-1	-1
p_9^{4D}	1	-1	-1	1
p_{10}^{4D}	1	-1	1	-1
p_{11}^{4D}	1	-1	1	1
p_{12}^{4D}	1	1	-1	-1
p_{13}^{4D}	1	1	-1	1
p_{14}^{4D}	1	1	1	-1
p_{15}^{4D}	1	1	1	1

Der Hyperwürfel hat also diese 16 Eckpunkte.

[1] im folgenden 4D gennannt
[2] 3D
[3] 2D

1.2 Begrenzungsflächen

Beim Körper im 3D werden diejenigen Eckpunkte zu einer Fläche zusammengefaßt, bei denen jeweils der Wert einer Koordinate identisch ist. Beim 3D–Würfel würden auf diese Weise 6 Flächen entstehen:

Fläche	p_1	p_2	p_3	p_4	
F_0^{3D}	p_0^{3D}	p_1^{3D}	p_3^{3D}	p_2^{3D}	$\Longleftrightarrow yz$ – Ebene $(x \equiv -1)$
F_1^{3D}	p_4^{3D}	p_5^{3D}	p_7^{3D}	p_6^{3D}	$\Longleftrightarrow yz$ – Ebene $(x \equiv 1)$
F_2^{3D}	p_0^{3D}	p_1^{3D}	p_5^{3D}	p_4^{3D}	$\Longleftrightarrow xz$ – Ebene $(y \equiv -1)$
F_3^{3D}	p_2^{3D}	p_3^{3D}	p_7^{3D}	p_6^{3D}	$\Longleftrightarrow xz$ – Ebene $(y \equiv 1)$
F_4^{3D}	p_0^{3D}	p_2^{3D}	p_6^{3D}	p_4^{3D}	$\Longleftrightarrow xy$ – Ebene $(z \equiv -1)$
F_5^{3D}	p_1^{3D}	p_3^{3D}	p_7^{3D}	p_5^{3D}	$\Longleftrightarrow xy$ – Ebene $(z \equiv 1)$

Die Punkte sind dabei so geordnet, daß sie in der Reihenfolge $p_1 \rightarrow p_2 \rightarrow p_3 \rightarrow p_4$ ein Quadrat ergeben.
Beim 4D–Würfel versuchen man es nun analog. Zuerst wird in 3D–Untergebilde eingeteilt, indem alle Punkte zusammengefaßt werden, bei denen eine Koordinate identisch ist. So entstehen 8 Untergebilde, die im vorliegenden Fall Würfel im 3D darstellen. Diese Würfel haben jeweils 6 Begrenzungsflächen. Insgesamt existieren also 48 Begrenzungsflächen. Dabei sind einige dieser Flächen identisch. Diese Flächen müßte man durch weitere Überlegung lokalisieren.
Eine anderer Weg, die Flächen zu bestimmen, geht dahin, daß eine Begrenzungsfläche des 4D–Würfels entsteht, wenn man die Koordinatenwerte zweier Koordinaten variiert, und die beiden anderen konstant hält.
Es gibt dann genau vier Möglichkeiten, die Werte zweier Koordinaten konstant zu setzen. Für ein Koordinatenpaar (x, y) also beispielsweise:

$(-1,-1); (-1,1); (1,-1); (1,1)$

Natürlich kann man dies nun für alle Grundfebenen im 4D tun. Diese Ebenen sind die

xw–Ebene, yw–Ebene, zw–Ebene, xy–Ebene, xz–Ebene und die yz–Ebene

Auf diese Art entstehen 24 Begrenzungsflächen. Diese Anzahl erscheint plausibel und wurde umgesetzt, obwohl man an dieser Stelle sicher noch weitere Überlegungen anstellen könnte.

1.3 Abbildungen

Die vierte Dimension kann man sich nicht vorstellen, weil der Raum der uns alle umgibt nur dreidimensional ist. Der 4D–Körper ist aber rein formal existent. Um diesen Körper anschaulich darzustellen behilft man sich mit einem Trick. Wie bei der Abbildung eines 3D–Körpers auf ein Blatt Papier (die Ebene hat nur zwei Dimensionen) kann man auch ein Gebilde aus einem vierdimensionalen Raum auf einen dreidimensionalen Raum abbilden. Die tägliche Erfahrung zeigt, daß solche Abbildungen zwar nicht exakt das wiederspiegeln, was real ist, trotzdem kann man aus einer zweidimensionalen Darstellung eines 3D–Körpers relativ gut auf das dreidimensionale Original schließen. Man denke nur an Isometrische Darstellungen, Fotografien und das Fernsehen. Der Gedanke ist nun eine bekannte, einfache Abbildung zu benutzen, um die vierte Dimension auf die drei uns bekannte abzubilden. Im Beispielprogramm wird an dieser Stelle eine einfache Parallelprojektion benutzt. Parallelprojektion heißt, daß ein 3D–Punkt aus einem 4D–Punkt gewonnen wird, indem einfach die w–Koordinate weggelassen wird. Das so entstehende dreidimensionale Gebilde kann man nun — so gut es irgend möglich ist — auf den zweidimensionalen Bildschirm bringen.

1.4 Rotation im 4D

Um sich die vierte Dimension noch anschaulicher zu machen, soll der 4D–Würfel bewegt werden. Jeder denkt sofort an die Drehung um die vier Achsen (die vier orthogonalen Einheitsvektoren). Eine Rotation um eine solche Achse ist aber im 4D nicht eindeutig bestimmt. Um im 4D ein Objekt um einen bestimmten Winkel zu drehen, muß man eine Fläche spezifizieren, um die gedreht

werden soll. Dies sieht man leicht ein, wenn man sich die analogen Fälle im 2D und 3D ansieht. In einer Ebene (2D) benötigt man für eine Rotation einen Drehpunkt. Es ist nicht möglich, um eine Gerade zu drehen. Im 3D werden Objekte gewöhnlich um Geraden (Rotationsachsen) gedreht. Eine Rotation um einen Punkt mit einem bestimmten Winkel ist nicht eindeutig auszuführen. Im 4D verhält es sich ganz analog. Eine Rotation um eine Achse ist nicht möglich, weil nicht eindeutig bestimmt. Deshalb muß eine Rotationsebene angegeben werden. Hierzu nimmt man einfacherweise die bereits erwähnten sechs Grundebenen.
Man kann diese Rotationen sehr einfach durchführen, indem man Rotationsmatrizen benutzt. Diese Matrizen sehen anlog den Rotationsmatrizen im 3D aus. Als Beispiel sei hier die Rotation um die zw–Ebene angegeben. Dabei steht zw jeweils für den Drehwinkel um die zw–Ebene.

$$\begin{pmatrix} cos(zw) & sin(zw) & 0 & 0 \\ -sin(zw) & cos(zw) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Solche Matrizen kann man für jede der sechs Hyperebenen aufstellen. Für die endgültige Rotation werden diese Matrizen einfach multipliziert. Die transformierten Punktkoordinaten ergeben sich durch Multiplikation der alten Koordinaten mit der endgültigen Rotationsmatrix. Wichtig ist hierbei, daß die Reihenfolge der Multiplikation im Allgemeinen nicht egal ist. Werden die Matrizen in einer anderen Reihenfolge multipliziert, dann ergibt sich eine andere Endlage. Dieser Fakt wurde im Beispielprogramm nicht beachtet, die Multiplikation erfolgt immer in der gleichen Reihenfolge. Im Beispielprogramm führt also das Betätigen der „Rotationstasten“ immer zum gleichen Ergebnis, auch wenn eine andere Reihenfolge der Betätigung gewählt wird.

2 Implementation

Bei der Implentation wurde darauf geachtet, den Körper möglichst interaktiv im 4D bewegen zu können. Da die Darstellung — wie bereits erwähnt — nur maximal im 3D erfolgen kann (War schon mal jemand im 4D ?), muß eine Abbildung in die uns vertrauten drei Dimensionen erfolgen. Diese Abbildung kann man auf einem Computerbildschirm mit verschiedenen Mitteln darstellen und betrachten. Eine Möglichkeit wäre, das 3D Gebilde durch light–shading oder texture–mapping realistisch aussehen zu lassen. Eine andere Möglichkeit wäre, das räumliche Sehen des Menschen zu unterstützen, und auf dem Bildschirm räumliche Bilder zu erzeugen (Betrachtung mit rot/grün–Brille oder shutterglasses). Aber auch durch einfache Drahtgitterdarstellung, kann man — im Zusammenspiel mit interaktiver Bewegung des Modells — die dritte Dimension auf dem Bildschirm ahnen. Diese Methode ist aber bei weitem nicht so gut geeignet wie z.B. die rot/grün–Darstellung.
Um die oben aufgeführten Anforderungen möglichst effizient zu erfüllen, mußte ein Grafiksystem als Basis benutzt werden. Es wurde in diesem Fall **PEX** gewählt. Die Grund hierfür war die Freistellung der Wahl des Grafiksystems in der Aufgabenstellung. Auch wenn die Einarbeitung in dieses System relativ schwierig ist, so sprechen die Erfolge der Implementation doch für dieses System. Desweiteren waren beim Autor noch fast keine Erfahrungen im praktischen Umgang mit Grafiksystemen wie **PEX**, **Phigs/Phigs+** oder der **Open GL** vorhanden. Es gibt sicherlich Grafiksysteme, mit denen die Implementation einfacher hätte gestaltet werden können, doch das soll hier nicht näher besprochen werden.
Für weitere Implementationsfragen sei an dieser Stelle auf den Quelltext verwiesen.

3 Nutzerdokumentation

Das Beispielprogramm wurde unter **HP–UX** getschrieben, compiliert und getestet. Portierungen auf andere Plattformen sollten — bei Vorhandensein der `PEXLib` — ohne weiteres möglich sein. Um ausreichende Geschwindigkeit bei der Ausgabe zu erreichen, ist es angebracht, Hardware–Z-Buffering zu benutzen. Jede gut ausgestattete CAD–Workstation sollte diese Möglichkeit bieten.

PEX prüft automatisch ob diese Möglichkeit nutzbar ist.
Zum Start des Programmes wird einfach ein Terminal geöffnet und in das Verzeichnis gewechselt, indem sich das ausführbare Beispielprogramm befindet. Nun kann das Programm durch Eingabe von `view4d` gestartet werden. Sollte das Programm noch nicht compiliert sein, dann einfach `make` tippen. Dieser Befehl compiliert das Programm. Sollten dabei Fehler auftreten, dann ist besser ein erfahrener **UNIX**-Kenner einzuschalten.
Die Bedienung des Programmes ist nun denkbar einfach. Im folgenden sind die möglichen Aktionen angegeben, die während des Programmablaufes ausgeführt werden können.
Um den Würfel in der anfgegebenen Richtung um die sechs Hyperebenen zu drehen betätige man die folgenden Tasten:

Richtung	xw–Ebene	yw–Ebene	zw–Ebene	xy–Ebene	xz–Ebene	yz–Ebene
+	1	2	3	4	5	6
−	y	x	c	v	b	n

Durch Drücken der linken Maustaste (Button1) und gleichzeitiges Bewegen des Mauszeigers, kann man die Sicht im 3D ändern. Hierbei wird nicht das Modell, das Gebilde gedreht, bewegt oder verändert, sondern man bewegt sich quasi um das **Abbild im 3D** herum.
Die Taste `r` bewirkt eine Reinitialisierung der 4D–Drehwinkel. Diese Winkel haben nach dem Drücken der Taste `r` den Wert 0. Die Sicht auf die 3D–Abbildung des Hyperwürfel wird auch zurückgesetzt.
Mit `a` kann man die drei Achsen des 3D ein- bzw. ausblenden. Diese Achsen helfen bei der Orientierung im 3D–Abbildungsraum.
Die Art der Darstellung kann durch Betätigung der Taste `t` geändert werden. Es wird immer zwischen **wireframe** (Drahtgitter), **solid** (gefüllte Flächen) und **stereo** (Rot–Grün) umgeschaltet.
Gerade bei der **solid** Darstellung ist es wichtig, verdeckte Flächen zu entfernen. Dieses **hidden line / hidden surface removal** wird mit der Taste `h` ein- bzw. ausgeschaltet.
Ebenfalls in der **solid** Darstellung kann man die Art der Flächenfüllung einstellen. Die Taste `s` bewirkt die Umschaltung zwischen einfacher Flächenfüllung mit einer Farbe und der Flächenfüllung mit Lichteffekten.
Mit der Taste `f` kann man die Farbe des Objekte verändern (nicht bei **stereo** und bei **solid** bzw. **lightshading**).
Mit Hilfe der Taste `?` kann man sich die Tastenbelegung noch einmal im Terminalfenster ansehen.
Die Taste `q` beendet das Programm.

Hier noch einmal eine Übersicht der Tasten:

`?` – ein Hilfetext wird auf dem Terminal ausgegeben
`a` – die 3D Achsen werden dargestellt
`f` – die Farbe des Würfel kann geändert werden
`h` – hidden line / hidden surface removal (HLHSR) ein- bzw. ausschalten
`q` – Programm beenden
`r` – alle Winkel auf Initialwerte setzen
`s` – Schattierung ein- bzw. ausschalten
`t` – der Darstellungstyp wird verändert (SOLID → STEREO → WIREFRAME ...)

`1` | `2` | `3` | `4` | `5` | `6` – positive Rotation um die Hyperebenen
`y` | `x` | `c` | `v` | `b` | `n` – negative Rotation um die Hyperebenen

Durch Halten der linken Maustaste und Bewegen des Zeigers kann die Ansicht der 3D Abbildung verändert werden. Die Abbildung[4] zeigt noch einmal das Ausgabefenster und das Terminalfenster nach Eingabe von `?`.

[4] aus Platzgründen nicht in diesem Dokument

A Literatur

Literatur

[1] **Scheiffler, Robert W.** und **Gettys, James**
X Window System; The Complete Reference to Xlib, X Protocol, ICCCM, XLFD
Digital Press, 1992,
INFMAT 24.544.- 14

[2] **Anand, Vera B.**
Computer Graphics and Geometric Modeling for Engineers
Wiley, 1993,
ISBN 0-471-59960-3, INFMAT 24.735.- 23

[3] **Hewlett Packard**
PEXLib OnLine Help und Beispielprogramme

HP Visual User Enviroment, Help System