# Anleitung für JONGL

**COLLABORATORS**

| | *TITLE* :<br><br>Anleitung für JONGL | |
|---|---|---|

| *ACTION* | *NAME* | *DATE* | *SIGNATURE* |
|---|---|---|---|
| WRITTEN BY | | August 23, 2022 | |

**REVISION HISTORY**

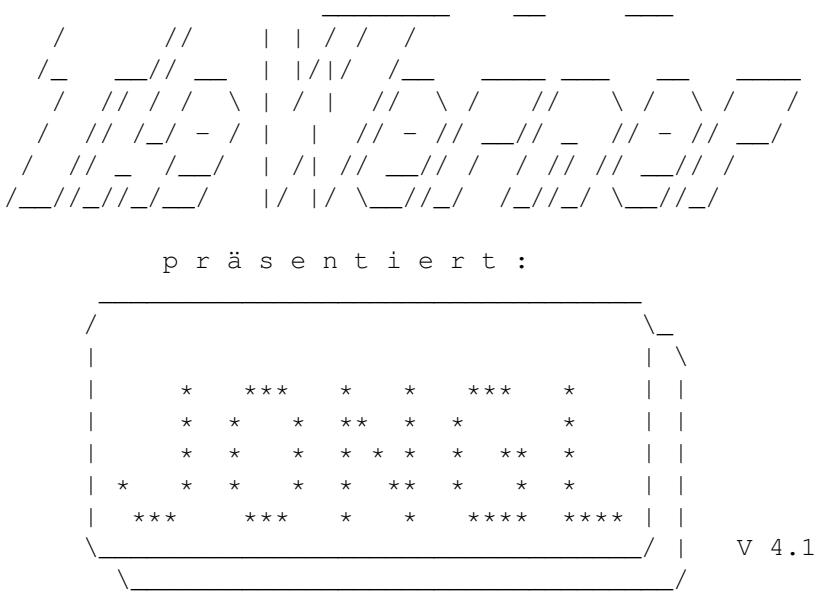
| NUMBER | DATE | DESCRIPTION | NAME |
|---|---|---|---|
| | | | |

# Contents

# Chapter 1

# Anleitung für JONGL

## 1.1 Anleitung für JONGL




das einzige (uns bekannte) Jonglier-Simulations-Programm, mit dem man Kettensägen passen und sich das in Echtzeit und 3D ansehen kann.

(Passen: Jonglieren mit 2 oder mehr Leuten)

Stand dieser Anleitung: 23.6.96

Inhaltsverzeichnis

## 1.2 Inhaltsverzeichnis

## 1.3 Einleitung

Die Einleitung zerfällt (vollautomatisch) in folgende Sachgebiete:

Was soll das Programm eigentlich?

Rechtlicher Kram

Systemvoraussetzungen

Neue features

Die Programmierer

## 1.4 Was soll das Programm eigentlich?

Jongl ist ein Jonglier-Simulations-Programm. Das bedeutet, daß man (fast) beliebige Jongliermuster mit 1 bis 10 Leuten ansehen kann. Im Gegensatz fast allen anderen Programmen dieser Art zeigt Jongl auch die Leute und den Boden mit an. Dabei erfolgt die Darstellung mit ausgefüllten Flächen und auf Wunsch auch mit schattierten Objekten oder mit Schatten auf dem Boden (sehr wichtig...). Da das Programm die Darstellung in Echtzeit berechnet, kann man, während das Muster jongliert wird, mit der Maus den Blickpunkt und -winkel verändern. So kann man sich auch in den Kopf eines Jongleurs versetzen und erleben, wie es wäre, wenn man z.B. mit 7 Tennisschlägern jonglieren könnte.
Hier sieht man noch ein weiteres feature von Jongl gegenüber anderen: Hier kann man mit jedem beliebigen Objekt werfen, vorausgesetzt, man hat es vorher definiert.
Mitgeliefert werden verschiedenfarbige Bälle, Ringe, Keulen, eine Topfpflanze, Kettensäge, Tennisschläger usw.

Jongl liest ASCII-Dateien ein, in denen die Objekte und die Muster defi-

niert sind. Durch Ändern oder Neuschreiben solcher Dateien ist Jongl beliebig erweiterbar; siehe bitte auch
Gegenleistung
.

Es gibt je eine Version für 68000 und 68020/68881. Letztere heißt "Turboversion" und hat einige zusätzliche features, die der Koprozessor oder der 68020 ermöglicht ↩
.

Beachten Sie auch die weiteren Anmerkungen zu den Themen

Rechtlicher Kram
sowie
Die Programmierer
!

## 1.5 Rechtlicher Kram

Jongl ist zwar frei vertreibbar, aber trotzdem bleiben alle Rechte bei uns. Es ist ausdrücklich erlaubt, dieses Programm im Ami- und Internet, auf Aminet-CD-ROMs und in Mailboxen zu führen. Nicht erlaubt ist das Verkaufen für einen höheren Preis als der Datenträger rechtfertigt.

Dieses Programm kostet Dich nichts, also kannst Du auch nicht erwarten, daß wir für irgendetwas, was es unter merkwürdigen Umständen anrichten könnte, aufkommen.

(Es hat bis jetzt noch nie irgendetwas angerichtet...)

## 1.6 Systemvoraussetzungen

Eigentlich sollte Jongl auf jedem AMIGA laufen. Wir konnten ↩
natürlich
nicht alle ausprobieren. Aber grundsätzlich gilt: Es ist umso toller, je schneller der AMIGA ist. Das sieht man schon daran, daß sich Martin (einer von

Die Programmierer
) während der Arbeit an seinen Routinen eine Turbokarte kaufte, weils dann einfach viiieel coooooler ist.

Geschwindigkeit ist viel wichtiger als RAM-Ausbau. Viel (eigentlich SEHR viel) RAM braucht man nur für den Aufzeichnungsmodus, wenn man Muster mit langer Periode und komplizierten Objekte aufnimmt. Alles andere läuft einwandfrei mit 1 MB freiem Speicher.

Wieviel stack das Programm braucht, weiß ich nicht, weil ich kein Vertrauen zur stack-overflow-Funktion des C-Compilers mehr habe. Ich habe 20000 Stack und Jongl läuft problemfrei.

Jongl ist momentan auf 68030 und 68882 mit je 50 MHz ausgelegt. Für diese Prozessorkombination ist die Darstellung auf dem Bildschirm in Echtzeit. AMIGAs mit anderer Rechengeschwindigkeit müssen die Muster mit G (siehe

Tastatur
) anpassen.

## 1.7 Neue features

Das hier ist die neueste Version V4.1, die die einzige bisher veröffentlichte Version V4.0 zu völlig überholter software macht.

Hier ist die Geschichte von JONGL:

V4.1

* im Blitter-Modus bei weniger bitplanes jetzt endlich vernünftige Farben für die Kugeln

* das Muster-Auswahl-Menü ist jetzt in der WB-Auflösung (auch auf Grafikkarten)

* in jongl.prefs kann man optional die Anzahl der Spalten dazu angeben

* auch ohne overscan ist das Bild richtig zentriert

* der obere braune Balken ist wech

* neue Taste H für hardcopies. braucht dafür die iff.library und eine neue Zeile in jongl.prefs

* verenglischt. Menüs, Meldungen, list_of_objects und die Objekte selber gips jetzt auch auf Englisch. Braucht nochne Zeile in jongl.prefs

* paar Fehler in der Anleitung ausgebessert

* n paar neue Jonglier-Muster

V4.0

* erste veröffentlichte Version.

## 1.8 Die Programmierer

Alles bis auf die Projektions-Routinen ist von:

Werner Riebesel

Dipl.-Ing. Univ. (booaah ey!)
Auenbruggerstr. 110
80999 München
Telefon 089 / 8 12 51 52
email: Katja.und.TheWerner@t-online.de
oder
bm964867@muenchen.org

Die komplette Projektion (in Assembler) incl. Sortierung nach Entfernung,
Darstellung mit Schatten und ausgefüllten Flächen ist von:

Martin Hoffmann
Löherweg 10b
80997 München
Telefon 089 / 145 911
email: Hoffmann@fhmrz4.rz.fh-muenchen.de

Siehe unbedingt auch
Gegenleistung
und
Martins Geschwafel zum Thema
Geschwindigkeit
.

## 1.9 vom Programm benötigte Dateien

JONGL läuft nicht oder nur eingeschränkt, wenn es nicht
folgende Dateien und Verzeichnisse findet:

o (dir): enthält
Objekt-Dateien
zum Rumschmeißen

o/basic (dir): enthält statische Objekte, wie Mensch & Bierkiste

m (dir): enthält
Muster-Dateien
, in denen die Jongliermuster
definiert werden

sounds (dir): enthält Fang-, Dotz- und sonstige Geräusche

irgendein Verzeichnis, in dem beliebige looping-fähige ILBM-8SVX-
Sounddaten liegen (siehe
jongl.prefs
)

jongl.prefs
Voreinstellungen

```
                list_of_objects
                 Liste der möglichen Objekte, zeilenweise definiert


                list_of_people
                  Liste der möglichen Leute


                list_of_sounds
                  Liste der Samples, die auf den Funktionstasten liegen


FONTS:Jongl/6   für die Texte während des Programmlaufs

irgendwelche serienmäßigen Mathe-libraries

iff.library     von Christian A. Weber (Aminet: util/libs/IFFLIB22.lha)


jongl.guide     das ist das, was Du gerade lesen tust, Mann!


für OS2.0 und OS3.1 zusätzlich

sys:prefs/presets (dir): muß die Datei 'LeererPointer' enthalten (im Liefer-
                         umfang

ENVARC:sys/pointer.ilbm  (V2.04)
ENVARC:sys/pointer.prefs (V3.1)



Folgende Dateien sind nicht zwingend nötig, aber durchaus sinnvoll:

boden           dient zum Einstellen der Bodens. Die aktuelle Liste der
                verfügbaren Böden gips mit "Boden info".

loo             = "type list_of_objects"



Das Programm ging Anfang 1996 mit V1.3 auch noch.
```

## 1.10 Programm starten

```
                JONGL ist ein CLI-Programm.  Man kann es mit 0 bis 2 Argumenten
starten:

> jongl       startet das Programm und zeigt die Auswahl aller verfügbaren
              Jongliermuster. In diesem Muster-Auswahl-Menü kann man sich
              mit der Maus oder den Cursor-Tasten bewegen und mit der linken
```

```
              Maustaste oder RETURN oder ENTER auswählen.
              Wenn man nix auswähln will, drückt man ESC.

> jongl -n    n steht für eine Zahl zwischen 0 und 45. Die Zahl gibt an,
              welche Sorte Objekte geworfen werden sollen. Was das
              im einzelnen ist, steht immer in
                list_of_objects
                .
              Außerdem kann man dort neue Objekte eintragen.

              Zahlen außerhalb dieses Bereichs führen dazu, das eine Auswahl
              aller derzeit verfügbaren Objekte gezeigt wird, aus der ausge-
              wählt werden kann.

              Diese Option überschreibt die weiter unten definierte 'a=', mit
              der in der Jonglier-Muster-Datei die Art der Objekte festgelegt
              wird.

              Beispiel: jongl -5
              führt dazu, daß alle dann ausgewählten Muster mit Keulen
              jongliert werden.

> jongl -name Das gleiche wie oben, nur daß man das Objekt mit Namen nennt,
              statt eine Nummer zu verwenden. Ist die schnellste Möglichkeit,
              neue Objekte zu testen, weil man die
                list_of_objects
                 nicht
              ändern muß. Als Name wird der Dateiname der Objektdefinition
              angegeben.

              Beispiel: jongl -Messer

> jongl x     x steht für die Datei, in der das gewünschte Jongliermuster
              definiert ist. Damit wird das Menü mit den Jongliermustern
              übersprungen. Wenn das Muster mit einer Zahl beginnt, wird
              vorausgesetzt, daß die Muster im Verzeichnis "m" stehen.
              Fängt es nicht mit einer Zahl an, dann wird das als Pfad
              gewertet.

              Beispiele: jongl 5_dotzen
                         jongl ram:schrott


> jongl ?     zeigt die Versionsnummer des Programms und meine Adresse. Es
              gibt je eine Programmversion für 68020/68881 und für 68000.

> jongl info  zeigt dieses jongl.guide an.


Die beiden ersten Optionen können auch gleichzeitig angegeben werden (aber nur
in dieser Reihenfolge. Beispiel: jongl -2 5


Nach dem Starten interessiert man sich mit Recht für die
                Steuerung des Programms
```

## 1.11 Steuerung des Programms

Das Programm kann mit
Maus
,
Tastatur
,
Menü
und
Joystick
gesteuert werden.

Damit es nicht so langweilig ist, wird während des Programmlaufs ein zufällig ausgewähltes Sample abgespielt. Das Verzeichnis, in dem diese Samples liegen, wird in
jongl.prefs
festgelegt. Die Samples müssen im ILBM-8SVX-Format vorliegen und looping-fähig sein. Derzeit verkraftet das Programm bis zu 100 Samples in diesem Directory.

Wenn man keine solchen Samples hat oder einen Tracker oder sonstige Hintergrundmusik hören möchte, oder wenn man einfach seine Ruhe habe will, dann gibt man in
jongl.prefs
statt des Verzeichnisses no_sound
an. Dann ist Ruhe.

## 1.12 Maus

Mit der Maus bewegt man sich auf konzentrischen Kreisen um den geometrischen Mittelpunkt aller Objekte. Daraus folgt, daß man bei hohen Mustern das Geschehen von einer höheren Position aus verfolgt. (Siehe Option Z=)

Also: Maus nach links bewegt den Beobachterstandpunkt nach links.

Durch Drücken der linken Maustaste während die Maus nach oben/unten bewegt wird, kann man den Abstand des Beobachters zum Geschehen verkleinern/ vergrößern.

Maus rauf/runter zusammen mit der rechten Maustaste verstellt den Blickwinkel in Richtung Weitwinkel/Zoom.

## 1.13 Tastatur

Es sind ziemlich viele Tasten mit mehr oder weniger sinnvollen ↩
Funktionen
belegt. Hier ist die Liste:

A: Aufzeichnungmodus an/aus. (Bringt vor allem bei komplizierten Mustern und ohne Turbokarte was.) Es wird ein ganzer Zyklus der Datei aufgezeichnet, der dann deutlich schneller wieder abgespielt wird. Sobald ein Zyklus komplett aufgezeichnet ist, beginnt die beschleunigte Darstellung. Zur Kontrolle erscheint am rechten Bildrand ein 'A'. Allerdings kann man dann nicht mehr mit der Maus den Blickwinkel verstellen.

Die Geschwindigkeitsanpassung (siehe G) funktiniert auch während des Abspielens. So kann man also das Muster so konfigurieren, daß es beim Abspielen mit der richtigen Geschwindigkeit dargestellt wird.

B: Schaltet die Maussteuerung von Drehung auf Bewegung um und zurück. Damit kann man sich auch auf anderen Bahnen als auf den konzentrischen Kugeln bewegen.
Man kann sich in allen 6 Raumrichtungen bewegen: links, rechts, oben und unten durch bewegen der Maus. Nach vorne und hinten kommt man durch Drücken der linken Maustaste und gleichzeitigem Hoch- oder Runterfahren der Maus.
Mit einem
Joystick
in Port 2 funktioniert diese Bewegung
genauso.
Wenn man sich verfahren hat, R drücken.

C: CPU statt Blitter fürs Malen verwenden. Bei schnellen Rechnern wird die Bildwiederholfrequenz dadurch beschleunigt. Da in diesem Modus triple-buffering verwendet wird, wird auch gleichzeitig WaitTOF() ausgeschaltet (siehe N).
Diese Option kann standardmäßig eingeschaltet werden, wenn man den entsprechenden Parameter in
jongl.prefs
auf 1 setzt.
Damit dieser Modus was bringt, muß man was schnelleres als einen AMIGA 3000 (68030 / 25 MHz) haben.
Noch wichtiger: Einige Funktionen gehen NUR im CPU-Modus, weil der Blitter dazu zu langsam ist (und wir zu faul zum doppelt programmieren...)
Am wichtigsten: Der CPU-Modus geht nur in der Turbo-Version, weil sonst keine Kugeln zu sehen wären.

E: Einzelbildmodus starten. Das Bild bleibt stehen und bei jedem weiteren Druck auf E wird ein Bild weitergeblättert. Wenn Informationen oder Zeichen gewünscht werden, dann müssen diese vorher eingeschaltet werden. (Vielleicht ändere ich das ja noch mal...)

(E->) W: Einzelbildmodus beenden. (weiter)

F: Frequenzanzeige: es wird die Anzahl der zwischen zwei aufeinanderfolgenden Bilder vergangenen TOFs (ca. 1/50s) angezeigt. (auch mit F wieder auszuschalten)
Also: + heißt 50 Bilder pro Sekunde (eher unwahrscheinlich)
++ 25
+++ 17
++++5++++0++ bedeutet 4 B/s

G: Geschwindigkeitsberechung durchführen. Der Rechner stoppt die Zeit, die das Programm braucht, um einen Durchgang der Musterdatei darzustellen und ändert die Parameter u= und dt= in der Musterdatei so, daß die Darstellung in Echtzeit erfolgt. Anschließend wird das Muster nochmal reingeladen, um die geänderten Parameter berücksichtigen zu können.
Das geht auch während des Abspielens eines mit A aufgezeichneten Musters.

H: hardcopy machen. Das Verzeichnis, in dem abgespeichert wird, ist in
jongl.prefs
anzugeben (unbedingt mit : oder / aufhören lassen!). Die Datei heißt genauso wie das dargestellte Muster. Am Ende wird noch eine durchnumerierte Zahl angehängt. Das ergibt folgendes Beispiel: "J:pics/9_Russen_Pass.2". Diese Funktion benötigt die "iff.library" von Christian A. Weber aus CH-Zürich ab Version V21.

I: Simuliert ein Nachtsichtgerät. Dazu wird sie Sonne (automatisch) abgeschaltet und ein verschwommenes und seltsamfarbiges Bild gezeigt. Geht nur im CPU-Modus; und der geht nur in der Turbo-Version.
Besonders lustig in Verbindung mit Y und V. Möglichst viel Bewegung ins Bild bringen verbessert den Effekt.

K: Kotz-Modus. Man fliegt 50 cm hinter einem zufällig ausgewählten Objekt her. Warum das dann Kotz-Modus heißt, merkt man spätestens in den scharfen Kurven, während das Objekt in der Hand ist. Geht nur in der Turbo-Version.

L: Liniendarstellung. Durch Drücken von L kann man abwexlnd Drahtgitter oder ausgefüllte Flächen sehen.
L wie langweilig.

M:
Menü
an & aus. Wie im richtigen Leben ist das Menü redundant. Es dient nur dazu, sich die Tastaturkürzel schneller einzuprägen. Genauer gesagt sind nicht alle Funktionen von der Tastatur auch im Menü. Aber das wäre zu unübersichtlich.

N: Schaltet WaitTOF() ein oder aus. Hiermit kann man direkt sehen, wie lange das Programm für den Bildaufbau braucht, weil nicht erst in der Rasterstrahlaustastlücke auf das andere Bild umgeschaltet wird, sondern sofort, wenn die Berechnung fertig ist. N wie No WaitTOF().
Wenn der CPU-Modus (C) an ist, ist WaitTOF automatisch aus.

O: Sonne verstellen. Hier kann man die Richtung des Schattens (siehe 'S') verändern. Maus links-rechts dreht den Winkel, unter dem die Sonne scheint. Maus hoch-runter verändert die Länge des Schattens.

P: Hier wird die aktuelle Position des Beobachters (B=..) und die Höhe des Beobachters (Z=..) in die Musterdatei reingeschrieben. Beim nächsten Programmstart erscheint dann genau der gleiche Blickwinkel. (Erklärung der Parameter siehe weiter unten, bei 'Muster-Dateien')
Das Programm läuft dann weiter.

R: Reset für den Blickwinkel (wenn man zuviel mit den obigen Funktionen rumgespielt hat...) und für den Mittelpunkt (siehe 'B')

S: Umschalter für die Schattierung-/Schattenmodi:
0: -- normal --
1: -- Objekte schattiert (falls sie so definiert wurden) --
2: -- alles wirft Schatten (siehe 'O') --
Das geht nur, wenn man in
jongl.prefs
4 bitplanes
eingestellt hat. Welcher Modus beim Starten aktiviert wird, kann man auch in
jongl.prefs
einstellen. Wie man die Objekte so definiert, daß sie richtig dargestellt werden, steht in unter anderem in
Flags
.

T: Zeitanzeige an- oder ausschalten. Hier sieht man, wie sich die im Programm verstrichene Zeit (P:) zu der Echtzeit (E:) verhält. Es sind nur die Sekundenzeiger dargestellt.

U: Umschalten der unten am Bildschirm befindlichen Statuszeile; und zwar zyklisch:
-- Belegung der Funktionstasten mit Samples
-- 1. Kommentarzeile des gerade dargestellten Musters und Restspeicheranzeige --
-- nichts --

Ü: Überflüssige Werte anzeigen. Dient hauptsächlich dem debugging, sieht aber auch ungemein wichtig aus.

V: Verfolgung. Die Position des Beobachters wird nicht verändert, sondern nur, wo er hinguckt. Er behält nämlich ein zufällig ausgewähltes Objekt im Auge. Welches Objekt man verfolgt, kann man mit TAB durchschalten.
Das heißt einerseits, daß man dann noch mit der Maus rumfahren kann, was dazu führt, daß man sich dann nicht mehr auskennt. Andererseits kann man auch vorher eine Zahl auf dem Zehnerblock drücken und ein Objekt aus der Sicht eines Jongleurs verfolgen. Auch das geht nur in der Turbo-Version.

Y: FisheYe. Dazu gibt es nicht viel zu sagen. Mal ein bißchen mit Zoom und Radius rumspielen. Und wenn einem dann noch nicht schlecht ist, dann drücke man zusätzlich folgendes:
1 (auf dem 10er-Block) und V. Und wem es jetzt noch nicht reicht, der drücke außerdem noch I. Na...?
Diese wesentliche Funktion geht auch nur mit der Turbo-

| | |
|---|---|
| | Version. |
| Z: | Zeichen statt Objekte oder umgekehrt; ist sozusagen ein Umschalter. (zum debugging, Erklärung später) |
| 1...9,0 über den Buchstaben | setzt den Mittelpunkt der Bewegung auf den Jongleur Nr. n Damit kann man beim Passing um jeden einzelnen Jongleure herumfahren. Mit 'R' oder 'B' wieder ausschalten. |
| 1...9,0 auf dem Zehnerblock | versetzt sich in den Kopf des jeweiligen Jongleurs. So sieht man, wie es ist, 15 Kettensägen alleine zu jonglieren. |
| TAB: | Zum Durchschalten der Objekte, die im "Verfolgungs-Modus" angeguckt werden; siehe auch V. |
| F1...F10: | irgendwelche Samples (für die Tastenbelegung U drücken) diese Samples sind in der Datei 'list_of_sounds' definiert |
| +: | Delay++ (der zeitliche Abstand zwischen den Einzelbildern wird vergrößert) |
| -: | Delay-- (dito, verkleinert) |
| [ 10erBlock: | Lautstärke der Hintergrundmusik reduzieren |
| ] 10erBlock: | dito lauter |
| SPACE: | Bewegung des Beobachters stoppen. Das ist zur Vereinfachung, weil es manchmal schwierig ist, die dreidimensionale Bewegung des Beobachters schnell und an der gewünschten Stelle zu stoppen. |
| HELP: | Zeigt die Tastenbelegung an. |
| RETURN oder ENTER | kehrt zum Auswahl-Menü der Muster zurück, damit man ein anderes Muster auswählen kann, ohne das Programm verlassen zu müssen |
| ESC: | Programm beenden |

Es ist dem user selber überlassen, wieviele dieser Modi gleichzeitig eingeschaltet werden.

## 1.14 Menü

Wenn es in jongl.prefs nicht "serienmäßig" aktiviert ist, dann schaltet man es mit M ein.

Weil die Maus schon für die Bewegung im Raum benutzt wird, kann man das Menü

mit den Cursor-Tasten steuern. Den ausgewählten Punkt aktiviert man mit
RETURN oder ENTER.

Das Menü bremst die Darstellung deutlich, aber man soll ja das Menü lesen
und nicht die Animation bewundern. Dazu kann man das Menü ja wieder aus-
machen.

## 1.15 Joystick

Mit dem Joystick in Port 2 kann man sich geradlinig bewegen. Die ↩
Steuerung
funktioniert genauso wie bei der
Maus
und vorher gedrücktem B auf der

Tastatur
.

## 1.16 Format der eingelesenen Dateien

Alle Dateien sind im ASCII-Format. Es können beliebige Mengen von ↩
Leerzeichen
eingefügt werden. Leerzeilen ebenfalls (fast) überall erlaubt, außer
innerhalb der eigentlichen Musterdefinition (nach dem *, siehe weiter unten).

In diesem Zusammenhang unterscheiden wir zwischen
Objekt-Dateien
und
Muster-Dateien

## 1.17 Objekt-Dateien

Objekt-Dateien befinden sich im Unterverzeichnis 'o' des aktuellen
Verzeichnisses. Hier sind diejenigen Objekte, die herumgeworfen werden können.
Es gibt noch ein weiteres Unterverzeichnis namens 'o/basic', in dem die
statischen Objekte definiert sind.

Beispiel: o/KeuleLight

```
.....................

 r=1

 *
```

```
 8 PUNKTE

 0    0     -.35     #0

 0    0     -.06     #1

 -.06 -.052  .1
  .06 -.052  .1      #2-4
 0     .104  .1

 -.03 -.026  .25
  .03 -.026  .25     #5-7
 0     .052  .25

 10 LINIEN

 (7 4)    0 1
 (4 4)    1 2
 (4 4)    2 5
 (4 4)    1 3
 (4 4)    3 6
 (4 4)    1 4
 (4 4)    4 7
 (4 4)    5 7
 (4 4)    7 6
 (4 4)    6 5
```

Die Keule ist in senkrechter Lage definiert. Der Griff ist in Richtung -Z.

.....

Die Datei ist in zwei logische Blöcke unterteilt. Der erste ist optional und kann wahlweise einen oder mehrere der folgenden Parameter enthalten:

h=... Das ist die Anzahl der Punkte pro Hand. Dieser Wert muß bei einem Jongleur angegeben werden, sonst darf er nicht vorkommen.

k=... Das gleiche, aber für die Anzahl der Punkte des Kopfs.

r=... Dieser Parameter ist nur für rumfliegende Objekte interessant, weil er die Rotation angibt. r=0 heißt, daß sich das Objekt nicht dreht (z.B. Topfpflanze); r=1 heißt, daß es sich wie eine Keule dreht.

Das Trennzeichen zwischen den beiden Blöcken ist ein *.

Anschließend kommen mindestens zwei der folgenden Abschnitte:

PUNKTe
LINIEn
DREIECKe
N-ECKe
KUGELn

Nicht vorhandene Abschnitte brauchen nicht angegeben zu werden. Auch ist die Mehrzahlform bei einzelnen Objekt-Teilen nicht nötig. Man kann also auch "1 DREIECK" schreiben.

Die Klammern in den Definitionen dürfen nicht weggelassen werden, sie dienen der Orientierung der Programms in zukünftigen Versionen.

Jetzt kommen also im Einzelnen:

Punkte

Linien

Dreiecke

n-Ecke

Kugeln

Flags

Rotation

Farbe

Objekt-Kommentare

## 1.18 Punkte

Die PUNKTE werden in 3D-Koordinaten in der üblichen Reihenfolge x, ↩ y,z einge-
geben. Es handelt sich natürlich um ein Rechts-System, also x nach rechts, y nach vorne (in den Bildschirm rein) und z nach oben.

Um die Achsen zu sehen, gibt kann man einen Boden mit Achsen auswählen, in dem man vor dem Starten von Jongl "BODEN 1" eintippt.

Alle Werte sind in Einheiten von METERN anzugeben. Objekte, die wie eine Keule fliegen, müssen mit dem Griff in Richtung -Z ausgerichtet sein. Das Programm dreht sie dann um 90\textdegree{}, so daß sie in der Hand waagrecht liegen.

Auf die Punkte wird in den nachfolgenden Teilen Bezug genommen. Deshalb muß man wissen, daß die Numerierung von 0 losgeht. 13 Punkte gehen also von 0 bis 12.

Danach kommen die
Linien
.

## 1.19 Linien

Anschließend werden die LINIEN definiert. Jede Linie ist definiert durch
ihre
Farbe
und ihre
Flags
. Woraus folgt, daß jede Linie eine andere Farbe
haben kann. Farbe und Flags sind eingeklammert, um bei Erweiterungen das
Objektformats die alten Dateien nicht ändern zu müssen.

Anschließend kommen die Anfangs- und Endpunkte der Linie. Im oberen Beispiel
hat die erste Linie die Farbe 7 und geht von Punkt Nr. 0 zu Nr. 1. Die
nächste Linie geht von Nr. 1 bis Nr. 2 und hat die Farbe 4.

```
(7 4) 0 1
(4 4) 1 2
```

Beide Linien haben als Flag 4, also ist bit 2 gesetzt. Das heißt, daß sie
bei Bedarf schattiert dargestellt werden können.

Dann kommen die
Dreiecke
.

## 1.20 Dreiecke

Die DREIECKE werden wie folgt definiert:
Farbe
auf der Vorderseite (siehe
bei Flags), Farbe auf der Rückseite,
Flags
. Anschließend kommen die 3
Eckpunkte des Dreiecks.

Beispiel:

```
(1 2 7) 5 6 7
```

Dieses Dreieck hat vorne Farbe 1 und hinten Farbe 2. Die Flags stehen auf 7,
also sind die bits 0,1 und 2 gesetzt. Somit ist es ausgefüllt, von beiden
Seiten sichtbar und ggf. in FarbeA schattiert.

Die drei Punkte des Dreiecks sind 5, 6 und 7.

Es folgen die
n-Ecke
.

## 1.21 n-Ecke

Auch hier kommen vor den Ecken die drei Parameter "vordere Farbe", "hintere Farbe" und Flags, die bei den Dreiecken schon erläutert wurden.
Anschließend kommen die Eckpunkte des n-Ecks, wobei der erste Punkt auch als letzter angegeben werden muß.

Beispiel:

```
(4 5 3)   11 6 7 8 9 11    (Farben 4 & 5; Flag 3: beidseitig ausgefüllt
                            das N-Eck hat 5 Ecken.)
```

Eine wichtige Einschränkung für korrekte Darstellung ist, daß nur konvexe Polygone verwendet werden. Es dürfen also keine nach innen zeigenden Ecken vorhanden sein. Außerdem muß das Polygon in einer Ebene liegen.

Witzige Effekte kann man erzielen, wenn man Polygone mit Löchern definiert. Dazu folgt der normalen Definition ein Minus (-) und dann die Punktnummern des Lochs.

Beispiel:

```
(4 5 3) 11 6 7 8 9 11 - 21 16 17 21
```

Man sieht, daß das Loch auch eine andere Anzahl von Punkten haben kann, aber nicht muß. Das Loch muß den gleichen Drehsinn wie das Polygon haben!

Zum Leidwesen der Nicht-Benutzer des CPU-Modus muß hier gesagt werden, daß die Löcher nur im CPU-Modus funktionieren. Schade auch...

Und dann kommen die Kugeln.

## 1.22 Kugeln

Kugeln: Eine Kugel ist durch 4 Werte definiert:

Farbe, Flags, Punktnummer, Radius.

Beispiel:

```
(4 0) 17 0.06    (Farbe 4, Flags=0, Punktnummer 17, Radius 0.06 m)
```

Für die Farbe gilt eine Besonderheit: gibt man -1 an, dann wird eine zufällige Farbe ausgewählt; siehe auch Objektdatei 'o/bunterBall'.

```
Die Punktnummer ist die Nummer des
                Punkte
                s, der die Koordinaten des Kugel-
mittelpunkts angibt.

Tja, und jetzt kommt die Einschränkung: Die Farben werden nur im CPU-Modus,
d.h. in der Turbo-Version dargestellt. Der Blitter zeichnet die Kugeln immer
in der gleichen Farbe (meist rot);
```

## 1.23 Flags

```
                Die FLAGS sind wie folgt definiert:
  bit 0 = 0: Drahtgitter
  bit 0 = 1: ausgefüllte Flächen

  bit 1 = 0: nur von der Vorderseite sichtbar (s.o.)
  bit 1 = 1: von beiden Seiten sichtbar (ggf. mit versch. Farben!)

  bits 2 und 3 = 00: Darstellung in normalen Farben
               = 01: beidseitig in Farbe A schattiert
               = 10: beidseitig in Farbe B schattiert
               = 11: Wenn es sich um Dreiecke oder N-Ecke handelt, dann
               gibt der Drehsinn die Farbe an, vorne Farbe A und hinten
               Farbe B. Für richtige 2farbigkeit also einfach den Drehsinn
               der Fläche umdrehen.
               Sind es Linien oder Kugeln, dann entscheidet die Farbe darüber,
               ob es mit Farbe A oder B schattiert wird. Ist die Nummer der
               Farbe ungerade, dann kommt Farbe A, sonst Farbe B.

Wie funktioniert die Schattierung jetzt?

Das Flugobjekt wird dunkler, wenn es weiter hinten rumfliegt. Es gibt nur 2

                Farbe
                n in je 4 Helligkeitsabstufungen: Farbe A und B (wer hätte das  ↩
                   gedacht?)
Die Farben A und B kann man in
                jongl.prefs
                 einstellen.

Die Vorderseite ist diejenige Seite, von der aus gesehen die Ecken im Uhr-
zeigersinn definiert sind. Das gilt natürlich nur für Dreiecke und n-Ecke.

Die bits 0 und 1 sind nur bei Dreiecken und n-Ecken sinnvoll;
bits 2 und 3 gelten auch für Linien und Kugeln.

Wichtig: Schattierung geht genau wie Schatten nur mit mindestens 4 bit-
planes. Also bei den Bedarf entsprechenden Wert in
                jongl.prefs
                 einstellen.
```

## 1.24 Rotation

Die Rotation des Objekts kann folgende Werte annehmen:

```
  0: rotiert nicht
  1: rotiert wie eine Keule
(-1: es ist ein Mensch: es neigt den Kopf und bewegt die Arme)
```

Menschen, oder, allgemeiner gesagt, Objekte, die andere Objekte herumwerfen MÜSSEN folgenden Aufbau haben:

Die ersten 'h=...' Punkte sind die linke Hand. Die nächsten 'h=...' Punkte sind die rechte Hand.
Anschließend folgen 'k=...' Punkte, die den Kopf definieren.
(h= und k= sind in
Objekt-Dateien
erklärt.)

Die ersten beiden Punkte des Kopfes bestimmen die Lage der Rotationsachse. Wenn der Jongleur den Kopf nach vorne neigt, müssen diese beiden Punkte ausgetauscht werden.

Diese Reihenfolge ist wichtig, weil das Programm bei der Berechnung der Hand- und Kopfbewegungen davon ausgeht.

## 1.25 Farbe

Hier wird von einer 16-Farben-Darstellung ausgegangen, weil nur ↩
bei dieser
Farbtiefe alle Funktionen möglich sind. Bei weniger bitplanes hört die Farbpalette entsprechend früher auf.

Farbe 0 ist der Hintergrund und der ist meistens so gut wie schwarz.

Farbe 1: gelb
Farbe 2: beige (Hautfarbe)
Farbe 3: dunkelgrün
Farbe 4: hellrot
Farbe 5: rosa
Farbe 6: hellblau
Farbe 7: hellgrau

Farben 0..7 sind unveränderlich.

Farbe 8: weiß
Farbe 9: orange
Farbe 10: braun
Farbe 11: hellgrün
Farbe 12: dunkelrot
Farbe 13: lila
Farbe 14: dunkelblau

Farbe 15: dunkelgrau

Die Farben 8 bis 15 können sich ändern, denn durch Drücken von S auf der

Tastatur
kommt man in 2 weitere Farbmodi:

schattiert:

Hier werden die oberen 8 Farben (die Nummern 8 bis 15) neu belegt, und zwar mit je 4 Abstufungen der Farben "FarbeA" und "FarbeB", die in
jongl.prefs
eingestellt werden können.

mit Schatten:

Auch hier werden die oberen 8 Farben geändert. Diesmal erhalten sie die gleichen Farbtöne wie die ersten 8, jedoch mit der halben Helligkeit. Diese Farben werden folglich für die Schatten benötigt.

(Der völlig sinnlose Infrarotmodus (I) soll hier nicht weiter erwähnt werden.)

## 1.26 Objekt-Kommentare

Kommentare in Objekt-Dateien darf man derzeit an drei Stellen anbringen:

a) im ersten Teil (vor dem *), wenn der Kommentar durch ein ! eingeleitet wird; also genau wie in der Muster-Datei, die gleich erklärt wird.

b) hinter den 3D-Koordinaten, bis die Zeile voll ist

c) nach der letzten Zeile des Objekts kann man noch beliebig viel hinschreiben

## 1.27 Muster-Dateien

Eine Muster-Datei legt das zu jonglierende Muster fest und ist damit das A & O des Inputs.

Am Anfang werden verschiedene notwendige und optionale Parameter festgelegt, dann kommen die Positionen der Leute und anschließend wird das Muster definiert.

Alle Parameter können in beliebiger Reihenfolge auftreten, mit der einzigen Ausnahme, daß 'h=' vor 'o=' definiert werden muß.

WICHTIG ist, daß innerhalb eines Parameters keine Leerzeichen vorkommen dürfen, also 'dt = 0.234' ist so nicht erlaubt, sondern nur als 'dt=0.234'.

Als Konvention (nicht Convention, das ist was anderes) möchte ich einführen, daß jeder Muster-Dateiname mit der Anzahl der Objekte beginnt und daß es mit Unterstrichen in sinnvolle Abschnitte gegliedert wird. Außerdem sollte man darauf achten, daß der Name nicht zu lang wird, weil es sonst im Auswahlmenü zu Überschneidungen kommt.

Muster-Dateien können somit aus folgenden Komponenten aufgebaut werden:

notwendige Parameter

veränderbare Parameter mit programm-interner Voreinstellung

optionale Parameter

Muster-Kommentare

Positionen der Jongleure

Musterdefinition

Dotzen

Multiplex und wrap around

## 1.28 notwendige Parameter

In jeder Muster-Datei müssen die folgende drei Parameter angegeben sein.

```
h=4       Anzahl der Hände
o=7       Anzahl der Objekte
t=22      Anzahl der Takte in der Musterdefiniton (= der Zahl der Zeilen)
```

## 1.29 veränderbare Parameter mit programm-interner Voreinstellung

Mit diesen Parametern hat man Einfluß auf das Geschehen. Man muß sie jedoch nicht verstellen, da sie schon sinnvolle Anfangswerte haben.

a=6 Art der zu werfenden Objekte (Numerierung siehe oben bei 'jongl -n')

B=1.0,1.0,3.5,185.0 Mit dieser Angabe kann die Beobachterposition einstellen. Die Werte beschreiben den Blickwinkel, von dem aus man auf die Szene schaut. (in der Reihenfolge Phi, Theta, Radius und Zoom). Wenn man interaktiv einen schönen Blickwinkel erzielt hat, dann drückt man einfach auf 'P'. Damit werden die obigen Parameter dann in der Musterdatei eintragen.

dt=.25 Das ist die Zeit, die zwischen 2 Zeilen der Musterdefinition vergeht. Mit diesem Parameter kann man auch die Wurfhöhe anpassen, solange das Programm das noch nicht selber kann.

eps=.95 Der Reflexionswert von Dotzbällen, standardmäßig 5% Verlust. (wichtig: eps>0) Es wird hier der Geschwindigkeitsverlust bei der Reflexion angegeben. Der Verlust an Flughöhe geht quadratisch ein, weil die potentielle Energie (=Höhe) mit m*g*h gleich der kinetischen Energie = (m*v^2)/2 ist. Wenn die kinetische Energie um 5% kleiner wird, dann erreicht der Ball nur noch 0,95^2 = 0,9025 seiner Anfangshöhe! Will man also, daß der Ball auf die halbe Höhe zurückprallt, muß man sqrt(0.5)=0.707 eingeben.

g=9.81 Erdbeschleunigung. Positive Werte entsprechen Beschleunigung nach unten.

l=1.2 Längere Handverweilzeiten. Normalerweise hat der Jongleur in diesem Programm niemals gleichzeitig in beiden Händen ein Objekt, sondern immer nur abwechselnd rechts und links. In der Realität ist das aber nicht so; da hat man meistens in beiden Händen was und schmeißt nur dann was weg, wenn man kurz darauf was fangen muß. Damit man das etwas besser simuliert, kann man die Länge der Handverweilzeit verstellen. Voreingestellt sind 20% früher fangen und später werfen (macht logischerweise zusammen 40%!).

q=0 Quick-Darstellung. Normalerweise aus, bei Bedarf auf 1 (=an) setzen. Beschleunigt die Grafikausgabe, wenn sehr viel auf dem Bildschirm los ist. (Genauer gesagt werden die Leute zu jonglierenden Kleiderständern und der Karoboden ist nur noch als Rand zu sehen.) Als weitere Beschleunigung ist die Verwendung von Wurfobjekten mit möglichst wenigen Punkten und Linien zu empfehlen.

s=.15 Streuung der Wurf- und Fangbewegungen. Damit das Jonglieren auch realistisch aussieht, werden die Bewegungen innerhalb eines Würfels mit der Kantenlänge ’2*s’ um die eigentliche Fang- und Wurfposition ausgeführt.

u=5 Unterteilungen pro Takt. Um flüssigere Bewegungen zu erreichen, wird jeder im Muster definierte Takt (siehe unten) in mehrere Einzelbilder unterteilt. Hier kann man angeben, wieviele das sein sollen. Der Wert hängt vom Prozessor und von der Komplexität der Szene ab. Zu große Werte ergeben eine Zeitlupendarstellung. Irgendwann soll das das Programm mal selber können...

Z=1.7 Mit diesem Parameter kann man die Höhe des Beobachters über dem Boden einstellen. Wenn man Z=0.0 eingibt, dann wird das ’Auge’ in die Mitte des von allen Objekten aufgespannten Raumes gebracht. Eine andere Erklärung ist: hiermit stellt man die Höhe des Mittelpunkts der Kugelschalen, auf denen man sich bewegt, ein. Dieser Mittelpunkt ist während des Programmlaufs mit ’B’ beliebig veränderbar.

Die hier in den Beispielen angegebenen Werte sind die Voreinstellungen im Programm.

## 1.30 optionale Parameter

```
                Hier werden Parameter beschrieben, die keine default-Werte haben,  ↩
                   weil
es keine sinnvollen default-Werte gibt.



                verschiedene Objekte in einem Muster
                 um was hervorzuheben


                Handmakros
                 für verschiedene Handpositionen


                Zusätzliche unbewegte Objekte
                 für diverses Gerümpel
```

## 1.31 verschiedene Objekte in einem Muster

```
O=5 1#2 2#6 2#4      Dieser Parameter ersetzt gleichzeitig 'o=' und 'a='.
                     Er dient dazu, verschiedenen Objekte gleichzeitig zu
werfen. Im obenstehenden Beispiel sind es 5, und zwar 1 mal Nr. 2 (Ring),
siehe oben bei 'jongl -n', gefolgt von 2 mal Nr. 6 (Kugel) und 2 mal Nr. 4
(Keule). Die Reihenfolge der Objekte ergibt sich aus der Reihenfolge des
Auftretens in der nachfolgenden Musterdefinition.
Ein Beispiel aus der Musterdefinition:
-a
d-
-b
e-
-c
....
Hier wäre also das Objekt mit dem Zeichen 'a' ein Ring, 'd' und 'b' wären
Kugeln und 'e' und 'c' Keulen.
Man sieht, daß sich das Programm nach der Reihenfolge des Auftauchens in der
Datei richtet und nicht nach dem Alphabet oder der Größe der Zahlen.
```

## 1.32 Handmakros

```
Mit Handmakros kann man für einzelne Würfe andere Handpositionen festlegen.


Beispiel:

@A p 0 .5 1

oder

@B P 0 .6 1  0 .3 1  Das @ am Anfang leitet eine Makrodefinition für die
```

Handposition ein. Das direkt nach dem @ folgende Zeichen ist das Kommando zum Aufrufen dieses Makros. (Zum Merken: @, engl. at: gibt eine Position an.) Ich empfehle Großbuchstaben zur Unterscheidung von den i. Allg. bei den Mustern verwendeten Kleinbuchstaben.
Danach kommt entweder ein kleines oder ein großes p. Dann folgt, wie weiter unten beschrieben die Definition der Position der Hand (egal, ob linke oder rechte). Man kann also eine Fang- und Wurfposition getrennt eingeben.
WICHTIG: Die unten definierten Positionen des Jongleurs müssen trotzdem angegeben werden!
Normalerweise macht jede Hand für sich immer die gleiche Bewegung, egal wohin sie was schmeißt. Dieser Befehl ermöglicht verschiedenen Handbewegungen bei verschiedenen Würfen, z.B. Spagat oder Tennis oder Boston Mess usw.
Beispiel: '4_Spagat'

```
h=2 o=4 t=4 dt=.22 u=6 s=.1
@A p 0  .3 1   % hier werden die beiden inneren Positionen
@B p 0 -.3 1   %   als Makros definiert
p 0
0  .6 1    % hier werden die beiden äußeren Positionen definiert
0 -.6 1
*
1 A2
-  -
B3 4
-  -
```

Mit etwas Geschick und Geduld kann man auch Führungstrix erreichen...

## 1.33 Zusätzliche unbewegte Objekte

Man kann auch zusätzliche Objekte definieren, die sich nicht bewegen:

```
z=2
o/basic/Bierkiste
  90
  -2 0 0
o/basic/Giraffe
  0
  2.1 0 0
```

Hinter 'z=' steht die Anzahl, anschließend folgt der Pfadname der einzulesenden Datei (im Objektformat wie oben beschrieben). Dann kommt die Rotation um die z-Achse in Grad, damit das Objekt beliebig ausgerichtet werden kann.
Als letztes die x,y,z-Koordinaten des Objektursprungs.
Momentan müssen die statischen Objekte nach der Position (p oder P) der Leute kommen!

## 1.34 Muster-Kommentare

In den Muster-Dateien gibt es Kommentare, die auf dem Bildschirm angezeigt werden und welche, die nur im Programm stehen.

!Alles, was hinter einem Ausrufezeichen steht, wird beim Programmlauf angezeigt

% Was hinter einem Prozent steht, ist zur Orientierung im Programm geeignet,
% und wird nicht angezeigt.

Die Kommentare enden jeweils am Ende der Zeile, müssen aber nicht am Anfang der Zeile anfangen. In der Musterdefinition dürfen sie nur rechts von der Defintion stehen. Beispiel siehe weiter unten.

## 1.35 Positionen der Jongleure

Um die Positionen der Jongleure festzulegen, gibt es zwei Möglichkeiten: 'p' und 'P'. Das kleine 'p' ist die einfachere Version, in der die Hände an der gleichen Stelle fangen und werfen. Mit 'P' kann man verschiedene Fang- und Wurfpositionen eingeben, womit man also Kaskade und Rückwärtskaskade unterscheiden kann.

Nach diesem Buchstaben kommt erst der Rotationswinkel um die z-Achse (in Grad) und dann die 3D-Koordinaten der Position. Bei einem Winkel von 0\textdegree{} guckt der Jongleur in Richtung +X, +Y ist nach links und +Z nach oben. Bei 90\textdegree{} glotzt er also Richtung +Y. Bei der Angabe der Position ist darauf zu achten, daß der Mittelwert der z-Koordinaten der Hände jedes Jongleurs möglichst genau 1 ergibt, weil die Leute vom Programm so hingestellt werden. Des weiteren wird die Figur in die Mitte der linken und rechten Hand gestellt. Das führt zu Problemen, wenn man z.B. "Spagat" definieren will. Man muß dann entweder die beiden inneren oder die beiden äußeren Hände normal definieren und die beiden anderen mit @ (siehe weiter oben).

Die Anzahl der Leute berechnet das Programm aus der Angabe 'h='.

Beispiele:

```
p 0
  -1  .5 1
  -1 -.5 1
*

P 0
  -1  .5 1.2  -1  .2 .8
  -1 -.5 1.2  -1 -.2 .8
  180
   1 -.5 1.2   1 -.2 .8
   1  .5 1.2   1  .2 .8
*
```

Wie man sieht, wird die Definition der Positionen mit einem '*' abgeschlossen.

```
Dann folgt die
                Musterdefinition
                .
```

## 1.36 Musterdefinition

Das Muster wird in genausovielen Spalten definiert, wie Hände angegeben wurden. Es können beliebig viele Leerzeichen zwischen den einzelnen Zeichen auftreten, JEDOCH SIND KEINE LEERZEILEN UND KEINE KOMMENTARZEILEN ERLAUBT.
Das einfachste Beispiel ist die Kaskade mit 3 Bällen alleine:

```
 a-  !linke Hand wirft Objekt 'a'
 -2
 E-
 -a  %rechte Hand fängt 'a'
 2-
 -E
```

Der Strich '-' bedeutet, daß zu diesem Zeitpunkt die betreffende Hand leer ist. Die Spalten sind von links nach rechts abwechselnd linke und rechte Hand. (Wichtig beim Eintippen von Passing-Mustern!). Welche Zeichen man für die Objekte wählt, ist fast beliebig. Alle Zeichen außer '-', '&', '!', '%', '*', '/' und '\' sind zulässig. Deshalb ist das obige Beispiel auch so uneinheitlich. Ich empfehle jedoch für die Benutzung in den Mustern die Reihenfolge 1,2,3,4,5,6,7,8,9,a,b,c,d,e,f,g,... Die großen Buchstaben kann man für die Makros (@) freihalten, damit etwas Übersicht gewahrt bleibt. Außerdem wird das Programm auch ständig erweitert, wobei man auch immer irgendwelche Steuerzeichen braucht. Dann ist es gut, wenn man nicht alle Muster durchforsten muß, um noch freie Zeichen zu finden.

Man sieht außerdem, wie man die Kommentare anwenden kann. Kommentare, die rechts neben dem Muster stehen, werden vom Programm grundsätzlich nicht angezeigt.

## 1.37 Dotzen

Wie in der Realität kann man auch im Programm nach oben oder nach unten loswerfen. Des weiteren kann man auch mehrere Dotzer eines Balles ausführen lassen, bevor man ihn wieder fängt. Das Dotzen wird durch zwei weitere Steuerzeichen eingeleitet:

Abwurf nach oben: '/'. Beispiel:

```
  - 1
  /2-
  - 3
  1 -
```

```
  - 2
  3 -
```

Hier wird das Objekt 2 nach oben geworfen, um einmal zu dotzen. Analog geht's nach unten: '\'. Mehrfache Dotzer werden durch mehrfaches Hintereinanderschreiben der entsprechenden Zeichen erzeugt. Beispiel: Dotz-Shower:

```
  1-
  -\\1
  2-
  -\\2
  3-
  -\\3
```

Hier dotzt jeder Ball zweimal und wird nach unten losgeworfen.

WICHTIG: DOTZEN GEHT NICHT MIT BELIEBIGEN PARAMETERN! Das Programm meckert, wenn es mit den Vorgaben nicht möglich ist, eine passende Abwurfgeschwindigkeit zu berechnen.

Noch wichtigerer Hinweis:
AmigaGuide und MultiView verhalten sich unterschiedlich, was die Interpretation von '\' angeht. MultiView betrachtet den backslash an Escape-Zeichen, weshalb man 2 backslashes schreiben muß, um einen angezeigt zu bekommen. AmigaGuide tut das nicht.
Also, AmigaGuide-user: im zweiten Dotzmuster sind es natürlich nur 2 backslashes pro Zeile.

## 1.38 Multiplex und wrap around

Es folgt ein komplizierteres Beispiel mit einigen neuen Befehlen:
'm/8_MPlex_Pass'

```
 *
   8 1&2   - 3
   8   -   5 3&4
 7&8   6   5 -
   -   6 5&1 2
   3 4&6   - 2
   3   -   7 2&8
 5&3   1   7 -
   -   1 4&7 6
 *
 12345678
 18475362
```

Hier gibts 2 neue features:

a) Multiplex. Mit dem Zeichen & kann man mehrere Objekte in einer Hand gleichzeitig halten. Genauergesagt beliebig viele. Man kann sie zu verschiedenen Zeiten fangen und gleichzeitig wieder abwerfen (realistisch) oder auch nicht (unrealistisch).

b) Wrap around. Das ist ein Prinzip, mit dem man Muster in kurzer Form einge-
ben kann, die zwar so aussehen, als ob sie sich nach kurzer
Zeit schon wiederholen, dann aber vertauschte Objektnummern haben. Klar?
Also: Passing im 4-count (jeden 2. rechten passen). Wenn man sich aufmalt,
wie lange es dauert, bis alle Objekte wieder da sind, wo sie angefangen haben,
dann merkt man, daß man eine ganze Menge schreiben muß. Einfacher ist das mit
wrap around:

```
 *
 -1 -4
 2- 5-
 -3 -6
 4- 1-
 -2 -5
 3- 6-
 -4 -1
 5- 2-
 *
 123456
 561234
```

Genaue Beschreibung:

Nach der Musterdefinition kommt ein weiterer *. In der nächsten Zeile stehen
alle im Muster verwendeten Objekte in beliebiger Reihenfolge. In der Zeile
darunter stehen die Ersatzbezeichnungen der jeweiligen Objekte.
Beispiel: In der ersten Zeile nach dem unteren * steht vorne eine 1. Darunter
steht eine 5. Das heißt, daß das Objekt 1, wenn es unten aus der Datei
rausfliegt, oben mit der Nummer 5 wieder reinkommt. Das Objekt 5 wird jedoch
durch 3 ersetzt. usw.. Das spart massenhaft Tipparbeit. Man muß nur noch
jeden Wurf mindestens einmal angeben und muß sich dann nur noch überlegen, wie
die wrap around-Tabelle aussehen muß.

Siehe auch '10_H_Mult_dotz', in dem z.B. /4&R9 vorkommt!

## 1.39 Listings von externen Dateien

Hier kommt eine (irgendwann einmal aktuell gewesene) Liste der ↩
folgenden
externen Dateien, die JONGL benötigt:

jongl.prefs
Voreinsteller

list_of_people
Liste der verfügbaren Jongleure

list_of_objects

```
                  Liste der werfbaren Objekte


                 list_of_sounds
                  Liste mit Krach
```

## 1.40 jongl.prefs

```
                ..............................................................
 4   : Anzahl bitplanes
 0   : Speicher für Aufzeichnungsmodus reservieren
 0   : wieviele Bits Fußzeile hochschieben
 25  : Puffer-Faktor
 0   : CPU-Modus an
 0   : Schatten (0:keiner; 1:schattiert=hinten dunkler; 2:mit Schatten)
 diamond.font 12 4 : Schriftart & -höhe f Mustermenü [Anzahl der Spalten]
 no_sound          : keine Hintergrundmusik (ausgeblendet)
 ! DH0:Audio/Sounds  : Hintergrundmusik
 f00 : FarbeA
 ff0 : FarbeB
 1   : Menü an
 N   : overscan (N: nicht   H: horiz   V: vert   B: beides)
 ram:  : Pfad zum Abspeichern der hardcopies
 deutsch : Sprache
.............................................................
```

Relevant sind nur die Daten vor den Doppelpunkten. Die Texte dahinter dienen zur Orientierung für den Anwender.

Kurze Erläuterung der einzelnen Zeilen:

Anzahl bitplanes: Werte zwischen 1 und 4 sind derzeit sinnvoll. Je bunter desto langsamer. Schatten und Schattierungen gehen nur bei 4 bitplanes.

Speicher ... reservieren: Wenn zuwenig Speicher vorhanden ist, kann man hier durch Eintragen einer 0 die Speicheranforderung für den Aufzeichnungs-modus abschalten.

wieviele Bits ...: falls die mit U einschaltbare Fußzeile nicht sichtbar sein sollte (weiß nicht warum), dann kann man sie hiermit hochschieben. (siehe notfalls

```
                Tastatur
                )
```

Puffer-Faktor: Ist ein experimentell zu wählender Faktor, der den Speicherverbrauch der dargestellten Objekte beschreibt. Wenn sehr komplexe Objekte rumfliegen, dann muß dieser Wert erhöht werden. Das Programm meldet dies aber bei Bedarf. Bei Speichermangel kann er verkleinert werden, was allerdings dazu führen kann, daß das Programm komplexe Objekte nicht mehr darstellt.

CPU-Modus an: Auf 1 setzen, wenn das Programm im CPU-Modus starten soll (siehe <C> auf der

Tastatur
und im
Menü
).

Schatten: Startwert für den Schatten.
0 heißt kein Schatten
1 bedeutet schattierte Objekte, die nach hinten dunkler werden. Diese Option funktioniert nur mit dafür vorgesehenen Objekten, siehe
Flags
.
2 aktiviert die Schatten

Schriftart: Hier schreibt man eine sich genehme Schriftart rein, die man gerne im Muster-Auswahl-Menü sehen möchte. Auf jeden Fall das ".font" mit hinschreiben! Anschließend kommt die Größe der Schriftart. Als optionaler dritter Parameter kann die Anzahl der Spalten kommen, wenn man mit den standardmäßigen 5 Spalten nicht zufrieden ist.

Hintergrundmusik: Es folgt der absolute Pfad auf das Verzeichnis, wo die looping-fähigen Hintergrundmusik-Samples liegen. Der Pfad darf mit einem ":" aufhören, aber nicht mit einem "/".
Wenn man hier no_sound angibt, dann läßt das Programm seine Finger von den Audiokanälen, womit man dann irgendwelche Tracker oder so Zeug im Hintergrund laufen lassen kann.

FarbeA und FarbeB sind die
Farbe
n, mit denen schattiert wird. Genaueres zu diesem Thema ist bei
Flags
nachzulesen. Die Angaben erfolgen, wie man sich denken kann in Hexadezimalschreibweise. f00 ist Rot, ff0 ist gelb, 0f0 ist grün usw.

Menü an: Steht hier eine 1, ist das Menü vom Start weg eingeschaltet. Während des Programmlaufs kann man es aber jederzeit mit M ein- und ausschalten.

overscan: einschalten, wenn man keine schwarzen Bildschirmränder haben möchte - ausschalten, wenns zu langsam ist.

Pfad für hardcopies: Nach dem Drücken auf H wird der aktuelle Bildschirminhalt abgespeichert. Hier kann man angeben, wohin. Der Pfad muß mit : oder / aufhören!

Sprache: Wem deutsch nicht zusagt, kann hier auch "english" reinschreiben. Dann werden die Menüs und die HELP-Seite auf auswärts angezeigt.

Um eine Zeile auszublenden, schreibt man ein ! davor. Der Parameter, der in der ausgeblendeten Zeile definiert wäre, muß aber trotzdem angegeben werden. Klingt komisch, hat aber folgenden Sinn: So kann man verschiedenene Versionen einzelner Zeilen haben, ohne sie immer löschen zu müssen. Die erste Anwendung ist bei der Hintergrundmusik, s.o.

Wenn "jongl.prefs" nicht gefunden werden kann, dann werden die im obigen Beispiel angegebenen Standard-Werte angenommen. Jongl geht dann also trotzdem.

## 1.41 list_of_people

```
.............................................
 6
 Peter
 Jongleuse_mit_Rock
 Blondine
 Skelett
 blauerJongleur
 rotgelberJongleur
 Roboter_B
.............................................
```

In der ersten Zeile kommt die Anzahl der Jongleure, danach pro Zeile ein Dateiname aus dem Verzeichnis 'o/basic'.

Hier sei noch eine Manipulationsmöglichkeit erwähnt: Wenn man einige Jongleur-Sorten nicht haben möchte, dann verschiebt man sie ans Ende der Liste und verringert den Wert in der ersten Zeile so, daß nur noch die gewünschten Leute ausgewählt werden.

## 1.42 list_of_objects

```
                .............................................
 0:  - Würfel
 1:  S Flagge
 2:  S Ring
 3:  S DoppelRing

 ...

 41: S HgrünerBall   )
 42: S DroterBall    )
 43: S lilaBall      )
 44: S DblauerBall   )
 45: S DgrauerBall   )
     ^--- S heißt SCHATTIERT
.............................................
```

Die Zahlen werden zeilenweise durchnumeriert. Die Texte sind die Dateinamen der Objekte im Verzeichnis 'o'. Das S in der zweiten Spalte ist nur eine Merkhilfe für den user. Es bedeutet, daß das Objekt so definiert wurde, daß es schattiert angezeigt werden kann. (Siehe auch

```
                Flags
                ). Da das Programm den dritten String
aus jeder Zeile für den Dateinamen hält, ist es wichtig

a) wenn kein S kommt, statt dessen was anderes zu schreiben (z.B. -)
b) keine Leerzeichen in den Objektnamen zu verwenden.


Diese Liste ist nicht auf dem aktuellen Stand. Um sich den anzusehen,
tippt man LOO ein. Wenn das nicht geht, tuts auch TYPE LIST_OF_OBJECTS.
```

## 1.43 list_of_sounds

```
.............................................
 9
 FANG
 DOTZ
 Yell
 Audience_yell
 Explosion
 DuSchaust
 Ruebennase
 Hohoho
 Jingle
.............................................


Es können bis zu 2+10 Sounds definiert werden. Die  ersten  beiden  werden
für  FANGen  und  DOTZen  verwendet.  Die  bis zu 10 danach liegen auf den
Funktionstasten  F1  bis  F10.  Aus  (c)-Gründen  können  wir  hier  keine
mitliefern.
```

## 1.44 Zusatzprogramme

```
                Für das an sich schon extrem tolle Programm Jongl gips noch 1 paar ↩
                    Zusatz-
programme, die die Möglichkeiten bis zum Extrem ausreizen...



                 Freestyle
                   erzeugt zufällige Muster


                 J2
                   erzeugt systematisch alle Muster nach Vorgaben


                 J2Konv
                   macht J2-output Jongl-lesbar
```

```
                Objekt-Konverter
                  gips noch nich, wäre aber toll
```

## 1.45 Freestyle

```
                Wie man sich anhand des Namens schon fast denken kann, handelt es ↩
                   sich hier-
bei um ein tool zur Erzeugung von Freestyle-Jonglier-Mustern.

Zum Starten einfach "freestyle" eintippen & die Fragen beantworten. Die
erzeugte Musterdatei kann man selbstverständlich im Nachhinein mit dem
Editor verbessern.

Hier sollte man eher wenig Objekte (1 oder 2 mehr als Hände) nehmen, aber
dafür möglichst lange Perioden, so z.B. 60 oder 80. Letzteres geht natür-
lich auf Kosten des Speichers.

Nicht von mir, aber trotzdem gut ist
                J2
                .
```

## 1.46 J2

```
                Wer keine Ahnung von site-swap-Notation hat, der wird diesen und ↩
                   den nächsten
Abschnitt leider nicht verstehen. Hierzu verweise ich auf die Dokumentation
zu j2, nämlich j2.guide und notation.guide.

Wer jetzt noch weiterliest, der behauptet wenigstens von sich, daß er was
mit site-swap anfangen kann. Also:

j2 berechnet alle möglichen Jonglier-Muster

* zu einer gegebenen Anzahl von Objekten
* mit einer maximalen Wurfhöhe
* und einer gegebenen Periode.

Um den output von j2 in jongl reinzukriegen, habe ich
                j2konv
                 geschrieben.

J2 ist (c) von Jack Boyce. Vielen Dank für dieses tolle Programm.
```

## 1.47 J2Konv

Das ist ein Konverter zwischen site-swap, wie sie von J2
erzeugt wird, und
dem, was jongl als input benötigt. Die möglichen Parameter werden beim Starten angezeigt, wenn man keine Parameter mitgibt.

Kleine Warnung: j2konv ist noch nicht so extrem ausgereift.

Hier gehts zur Anleitung von J2KONV.

## 1.48 Objekt-Konverter

So was gips leider noch nicht. Woraus folgt, daß wir alle Objekte mit der Hand definiert haben. Wem das zu blöd ist, der schreibe doch bitte einen (mehr oder weniger) komfortablen Editor für unsere tollen Objekte.

## 1.49 AGA-Version

AGA wäre schon nicht schlecht, aber leider hat keiner von uns einen entsprechenden Rechner. Wenn also jemand unbedingt Wert auf eine AGA-Version legt, dann möge er uns einen AMIGA mit AGA-Chipset schicken. Wir werden dann das Programm dahingehend anpassen.

Vielen Dank im voraus....

## 1.50 Sonstiges

Wie nicht anders zu erwarten, kommt im Abschnitt Sonstiges das ↩
wüste Durch-
einander von zusammenhanglosen Punkten, nämlich:

Geschwindigkeit

Bekannte Fehler

Ausblick

Internet

De-Installation

## 1.51 Geschwindigkeit

Vorweg: Wer einen 68060 Prozessor hat, der kann das hier ↩
wahrscheinlich
vergessen.

Da es bei einer Simulation von sich schnell bewegenden Objekte sehr wichtig ist, eine flüssige Animation zu erzeugen, gibt es hier ein paar Hinweise zur Geschwindigkeit des Programms.

Achtung! Wer kein sinnloses 'Lebensgeschichtengeschwalle' mag, bitte diesen Absatz überspringen.

Die Darstellung von mehreren hundert Grafikelementen auf einem AMIGA mit 7 Mhz Taktfequenz über die Standard-Grafikroutinen des Betriebsystems, die über eine Hochsprache (C) angesprochen werden, ist einfach zu langsam. Vor allem, wenn man keinen beschleunigten AMIGA hat. Auf dem 50 Mhz 68030-Rechner sah das ja noch ganz nett aus, aber auf dem 68000er gabs nicht mehr als 5 Bilder in der Sekunde selbst bei simpel-Mustern (5 Bälle) und Drahtgitterschattierung. Da ich (Martin von
Die Programmierer
) jedoch
schon andere lustige Sachen in Assembler programmiert hatte, dachten wir darüber nach, das Programm durch eine Assembler-Unteroutine aufzupeppen. Nach etwa 5 Monaten Programmierarbeit war die grafische Assembler-Unterroutine dann soweit fertig.

Also ganz kurz zusammengefaßt: Die Grafikroutinen sind in Assembler programmiert. (ganz was neues...)

Nun gibt es 2 Versionen der Grafikausgabe:

- Über den in jedem Amiga vorhandenen Grafikprozessor 'Blitter'

- der Hauptprozessor macht das alles

Ganz einfach kan man sagen, daß der Blitter sehr schnell ist, jedoch ergeben sich auf schnellen Systemen (ab Amiga 3000 mit 25 Mhz) leichte Vorteile durch den Prozessor. Wer nicht mindestens einen 68020 und Mathecoprozessor 68881 hat, der kann sowieso nur die Blitterversion nutzen. Da der Blitter parallel zum Hauptprozessor arbeiten kann, ist bei der Blitterversion die Geschwindigkeit nicht so stark vom Prozessor abhängig. Die Geschwindigkeit der Prozessor-Version, im folgenden CPU-Version genannt, verhält sich dagegen proportional zur Prozessorgeschwindigkeit. Auf unseren 68030 mit 50 Mhz ist sie doppelt so schnell wie die Blitterversion.

Nun folgt eine Tabelle, was auf die Geschwindigkeit welchen Einfluss hat. Das ganze ist von mir geschätzt, es basiert auf Erfahrungswerten, da ich das Programm kenne (es ist ja von mir).

+ = bremst ganz irre stark (bremst hervorragend (ganz klasse wirklich))
o = bremst schon auch (wie ne Handbremse - so zum Vergleich)
- = bremst nur n' bißchen (aber auch irgendwie)

+--------------------------------------+-------------+

| Version | Blitter | CPU |
|---|---|---|
| Viele Punkte | + | o |
| Viele Polygone | o | + |
| Große Flächen | + | o |
| Anzahl der Bitplanes | + | - |

Man sieht also, daß alles, was toll ist, auch einen bremsenden Einfluß auf die Geschwindigkeit hat. Nur im CPU-Modus sollten immer 16 Farben (4 Bitplanes) benutzt werden, da weniger eine kaum wahrnehmbare Beschleunigung bedeuten.

Was kann man nun daraus lernen?

Hier nun einige Tips:

Viele Punkte kann man reduzieren, indem man einfache Objekte verwendet. Am besten sind hier natürlich Bälle mit nur einem Punkt.

Viele Polygone sind auch nur in komplexen Objekten wie dem Hut oder der Topfpflanze zu finden. Einfachere nicht gefüllte Objekte sind schneller.

Große Flächen kommen hauptsächlich durch den Boden. Um dies zu vermeiden, den Boden auf Drahtgitter setzen. Schatten sind dann leider nicht sichtbar, werden aber berechnet. Also Schatten dabei unbedingt ausschalten.

Die Anzahl der Bitplanes ergibt sich durch den ersten Eintrag in den

jongl.prefs

. Weniger als die normalen 4 Planes verhindern Schatten und Schattierung. Weniger als 3 Bitplanes ist farblich nicht mehr so der Renner und 1 Bitplane schaut einfach schlimm aus (von den Farben her).

Abschalten von Anzeigen wie Frequenz (Taste F) und Zeitanzaige (Taste T) bingen auf langsameren Rechnern auch einiges, da dies Betriebssystem-routinen sind, die nunmal nich soo schnell sind.
Wenn das alles nichts hilft, gibt es noch die Möglichkeit, im Muster den Parameter "q=1" anzugeben, der aus den JongleurInnen KleiderständerInnen macht, und den Boden gaanz langweilig. Wenn man dann als Objekte noch die Minikeule verwendet, (-16) flitzt es ganz schön ab. Das ist jedoch nur für Geschwindigkeitsfreaks, da es total blöde und langweilig aussieht (nur im Notfall verwenden).

Als letzter und garnichtmal unwichtigster Hinweis: Der Aufzeichnungsmodus ist just zu dem Zweck ins Leben gerufen worden, um geknechteten Besitzern langsamer Rechner auch ein Geschwindigkeitserlebnis zuteil werden lassen zu können. Siehe hierzu im Abschnitt

Tastatur

den Punkt A.

Das war’s hierzu... nun geh’ mal schön ausprobieren!

Übrigens, wer hier Tips sucht, um die Ausgabe zu verlangsamen, der sollte folgendes ausprobieren:

```
Boden 11
Jongl -19 24_Brad_2Cnt
```

Jetzt sieht man einen Bradford-Circle mit 8 Leuten, die auf einem 100 Felder großen Boden Topfpflanzen jonglieren. Das sind dann ungefähr 1750 Punkte und 450 Flächen oder Linien. Das geht auf einem 68030 mit 50 Mhz mit 4 Bildern pro Sekunde (gähn). Für die anderen Muster empfiehlt sich ein oder mehrmals auf Minus (-) zu drücken, dann wird's auch langsamer.

## 1.52 Bekannte Fehler

Manchmal startet das Programm normal, schafft es aber nicht, die Grafik aufzubauen. Dann einfach ESC drücken und den Computer neu booten. Das tritt nur selten auf und vor allem weiß ich nicht, warum. Schicksal.

Der angeforderte Speicher wird in seltenen Fällen nicht komplett freigegeben, wenn das Programm an einer ungewöhnlichen Stelle wegen eines Problems abbricht.

FÜr alle unklaren Eigenschaften dieses Programms gilt:

```
          +--------------------------------+
          |                                |+
          | ITS NOT A BUG, ITS A FEATURE. ||
          |                                ||
          +--------------------------------+|
           +--------------------------------+
```

Z.B. Werner-Hände und andere Merkwürdigkeiten...

## 1.53 Ausblick

Als nächstes größeres Projekt werde ich Echtzeitdarstellung einbauen. Dann hat man immer die maximale Bildfrequenz und kann den Verlauf der Zeit z.B. zwischen 0 und 200% der Echtzeit einstellen. Endlich werden dann alle Muster auf allen Rechnern auf Anhieb in Echtzeit dargestellt werden.

## 1.54 Internet

Falls es jemand noch nicht weiß: Jonglieren gips auch im Internet. Und zwar unter der URL http://www.juggling.org.

## 1.55 De-Installation

Zuerst solltest Du prüfen, warum Du das Programm löschen willst. Hierbei gibt es mehrere Möglichkeiten:

1. Du findest das Programm einfach schrecklich.
2. Deine Festplatte ist schon mit vielen, vielen sehr, sehr sinnvollen Programmen vollgemüllt.
3. Du hast das Programm nur aus Neugier installiert, kannst aber gar nicht jonglieren, und hast daher keine Verwendung für eine so hochwertige Jongliersimulation.

Falls die dritte Möglichkeit zutrifft, so läßt sich das ändern. Es gibt in Deutschland viele Jongleure, die sich wöchentlich treffen (z.B. im Rahmen vom Hochschulsport). Dort gibt es genügend nette Leute, die einem gerne zeigen, wie man damit anfängt. Adressen von Jongliertreffen findet man unter anderem im Internet beim Juggling Information Service (JIS). Eine einfache Einleitung ins Jonglieren findet man auch im Aminet unter Docs/Help/JuggleGuide. Ausserdem finden fast jeden Monat Jonglierconventions statt, bei denen sich Jongleure aus ganz Deutschland treffen. Falls Du keinen Internetzugang hast, kannst Du dich auch bei Die Programmierer melden (toll formuliert, nicht?). Viel Spaß beim Jonglieren.

Falls die zweite Möglichkeit zutrifft, dann solltest Du Deine Programme darauf überprüfen, ob sie wirklich so sehr sinnvoll sind, und die, die es dann nich sind einfach löschen. (Jongl braucht komplett installiert weniger als 600 KB).

Bei ersterer Möglichkeit gibt es keine andere Möglichkeit als entweder deinen AMIGA zu verkaufen (denk nicht mal daran) oder das Programm zu löschen. Und das geht so:

1. Das Verzeichnis Jongl löschen (Delete jongl all quiet).
2. Die Pointer löschen
   2.1 Bei Kickstart 1.3 sind keine installiert (Glück gehabt)
   2.2 Bei Kickstart 2.0 und 3.0 LeererPointer in sys:Prefs/presets/ löschen.
3. Jongl Font aus Fonts: löschen
4. das war’s

## 1.56 Gegenleistung

Nachdem in diesem Programm mehrere Mann-Jahre Entwicklungsarbeit ↩
stecken,
erwarte ich nicht, daß jemand den entsprechenden Preis dafür zahlen möchte.

Wenn Du dieses Programm benutzt, dann schicke doch z.B.:

- eine email
- einen Brief
- eine Postkarte
- eine Diskette

mit Kommentaren, Anregungen, Fehlerbeschreibungen, ...
und vor allem: neuen Mustern und Objekten an
Die Programmierer
.

Danke.

Bei Fehlerbeschreibungen bitte unbedingt folgende Angaben machen:

- verwendete Hardware (Amigatyp, Prozessor, evtl. exotische Hardware)
- verwendetes Betriebssystem (Jongl lief mal ab OS 1.3)
- verwendete Version (68000 oder 68881)

Auch würde uns interessieren, wasfürn Rechner ihr benutzt, falls ihr keine Fehler findet, da wir dann in etwa wissen, welche Funktionen erweitert werden sollen. Just zu diesem Moment, wo ich das hier schreibe, benutzen ALLE Anwender dieses Programms einen Amiga 2000 mit 50Mhz 68030 Turbokarte und 68882 Coprozessor. Klasse nicht?