Inside CHAOS*ultd*

oder

The Programmers Guide To CHAOS*ultd*

Inhaltsverzeichnis

I Modul-Programmierung **1**

1 Vorbemerkungen **1**
1.1 Vorraussetzungen . . . 1
1.2 Unterstützung von Modulentwicklern . . . 1
1.3 Beispiel- und Include-Dateien . . . 1
1.4 Copyright . . . 2

2 Modulkonzept **2**
2.1 Übergabe der Parameterstrukturen . . . 2

3 ein paar CHAOS*ultd*-Interna **3**

4 Druckermodule **4**
4.1 Id's . . . 4
4.2 Routinen . . . 4
4.3 Parameter . . . 5

5 Berechnungsroutinen **5**
5.1 allgemeine Angaben . . . 5
5.2 Routinen . . . 7

6 XCHAOS-Routinen **12**

7 CHAOS*ultd*-Routinen für externe Module **14**
7.1 Bibliotheksroutinen . . . 14
7.2 Dialogverwaltung . . . 15
7.3 Info-Dialoge . . . 16
7.4 spezielle Dialoge . . . 16
7.5 Dialoge beim Bilder-Erzeugen . . . 16
7.6 Hilfsroutinen für die Parametereingabe . . . 16
7.7 Unterstüzung für gedrehte Blöcke . . . 18
7.8 Zufallszahlen-Generator . . . 18
7.9 Interpolation . . . 18
7.10 Tastatur-Abfrage . . . 18
7.11 Routinen zum Zeichnen . . . 19
7.12 weitere Hilfroutinen zur Berechnung . . . 20
7.13 Routinen zum Ändern des Objektstatus . . . 20
7.14 Speicherverwaltung . . . 20
7.15 32-Bit Festkomma-Arithmetik . . . 21
7.16 Einstellungen . . . 21
7.17 Unterstüzung für Hardcopy-Routinen . . . 21

II Dateiformate **23**

1 CHAOS*ultd*-Bilder **23**

2 Filme **24**

3 Einstellungen **24**

Teil I
Modul-Programmierung

1 Vorbemerkungen

CHAOS*ultd* ist bekanntlich modular aufgebaut. Dies ermöglicht es, daß auch andere Programmierer als ich Module für neue Fractaltypen oder Hardcopyroutinen schreiben können.
Bei Berechnungsroutinen hat das gegenüber eigenen Programmen den erheblichen Vorteil, daß man sich den ganzen Bildverwaltungskram sparen kann. Verwendet man die XCHAOS-Routinen, dann entfällt auch noch der Aufwand für die Parametereingabe, so daß praktisch nur noch die Berechnungsroutine für das Bild selbst geschrieben werden muß.
Im folgenden soll beschrieben werden, wie solche Module zu realisieren sind.

1.1 Vorraussetzungen

CHAOS*ultd* wurde mit PureC geschrieben und PureC hat (bekanntlich) etwas exotische Übergabekonventionen für Parameter. Dies schränkt die Möglichkeiten der verwendeten Programmiersprache auf PureC (bzw. TurboC) (und Assembler) ein. Für die Realisierung von MyDial-Dialogen, die im Sinne der Einheitlichkeit von CHAOS*ultd* natürlich vorzuziehen sind, sollte Interface vorhanden sein (muß im Moment, weil man nur dann die nötigen Include-Dateien für die MyDials hat).
Weitere Voraussetzungen existieren eigentlich keine, ein bißchen programmierern sollte man natürlich auch können.

1.2 Unterstützung von Modulentwicklern

Sollten bei der Programmierung von CHAOS*ultd*-Modulen irgendwelche Fragen auftauchen, so bin ich natürlich (im Rahmen meiner Möglichkeiten) gerne bereit weiterzuhelfen.
Und zwar:

- telephonisch: 0941/949802
  Wann ich da bin, ist schwer vorhersagbar. Ich bitte von Anrufen zwischen 22^{00} und 12^{00} abzusehen.
- brieflich:
  Th. Morus Walter
  Schulstr. 22
  93080 (W-8401 Pentling)
  (möglichst mit Rückumschlag)
- via e-Mail: morus.walter@rphs1.physik.uni-regensburg.de

Wenn jemand beim Programmieren auf den Source-Level-Debugger nicht verzichten will, so bin auch bereit, die Sourcen von CHAOS*ultd* zur Verfügung zu stellen. Es ist möglich, die externen Routinen auch direkt zum Hauptprogramm zu linken, so daß sie nicht mehr nachgeladen werden. (Dabei sind an den Routinen keine Änderungen nötig, im Hauptprogramm ist das durch bedingte Compilierung geregelt). Allerdings wüßte ich schon gerne, wer was mit meinen Sourcen vor hat.
Auch Erweiterungen der Schnittstelle stehe ich nicht prinzipiell ablehnend gegenüber (allerdings müßt ihr mir die Routinen schon liefern).

1.3 Beispiel- und Include-Dateien

Die wichtigste Datei ist die Include-Datei *COMMON.H*. In ihr sind alle Strukturen und Konstanten definiert, die die externen Routinen kennen müssen.
An Beispielen hat man für Berechnungsroutinen die Feigenbaumdiagramme (im Ordner *FEIGENBM*). Für XCHAOS-Routinen hat man die Popcorn-Routinen als Beispiel, wobei die restlichen Routinen und die XCHAOS-Verwaltungsroutinen[1] als Objektfiles vorhanden sind (Ordner *X_CHAOS*). Im Ordner *HCOPY* findet sich der Quellcode zur 24 Nadel-Hardcopy-Routine.

[1] bis auf den Teil zur Struktur-Übergabe an CHAOS*ultd*, so daß man eigene Routinen hinzufügen kann

Die zugehörigen Resource-Dateien finden sich im Ordner *CHS_ULTD.GEM*, im Ordner *INCLUDE* findet sich eine weitere Include-Datei *GEM.H*, die von *COMMON.H* nachgeladen wird und ein paar Makros zur (AES-)Objekt-Manipulation enthält. Darüberhinaus werden die Include-Dateien für die MyDial-Routinen benötigt (*portab.h*, *mglobal.h*, *mydial.h* und *nkcc.h*), die ich aus urheberrechtlichen Gründen nicht beilegen kann (ich bemühe mich um Zustimmung des Autors (die Mail ist raus, aber ich habe noch keine Antwort)).

Die Dateien *CHS_EMPT.C* und *XCH_EMPT.C* sind leere Quellen für Routinen bzw. XCHAOS-Routinen, in denen lediglich die zu schreibenden Funktionen vordefiniert sind. Man kann also diese Dateien als Ausgangspunkt eigener Routienen verwenden (alternativ zum Umschreiben der Beispiele). In *CHS_EMPT.C* sind optionale Funktionen auskommentiert. Mittels bedingter Compilierung werden diese Funktionen automatisch in die Modul-Struktur eingetragen, wenn man die Kommentare entfernt.

Anmerkungen zum Compilieren der Quellen:
In den Quellen müssen die Pfade für die Include-Dateien unter Umständen korrigiert werden. Dies gilt auch für die Projektdatei für die XCHAOS-Routine.
Ich verwende die folgenden Compiler-Optionen:
C Verschachtelte Kommentare
K default char ist unsigned
(Y Debug-Informationen)
(V Compilermeldungen)
max. Bezeichnerlänge 32
Beim Linken muß für die Module *kein* Stack erzeugt werden (unter Umständen ignoriert PureC die Einstellung auf 0, ich lasse den Stack immer 2 Byte groß sein). Der Stack von CHAOS*ultd*, den auch die Module verwenden, ist 10kByte groß, was reichen muß.
Beim Linken ist weiter darauf zu achten, daß man die Module *ohne* Startup-Code (*PCSTART.O*) linkt. Die Routine, die die Modul-Struktur an CHAOS*ultd* zurückgibt muß am Anfang des ersten Quellcodes stehen, so daß sie nach dem Linken die erste Funktion im Modul ist.

1.4 Copyright

Das Copyright der Beispiel-Sourcen liegt bei mir.
Unter der Bedingung, daß der Autor der Routinen (also ich) im Programminfo und/oder in der Dokumentation des Programmes genannt wird, dürfen die Sourcen ganz oder teilweise auch für eigene Programme verwendet werden, falls diese nicht komerziell sind (d.h. Public Domain, Freeware oder Giftware; Shareware würde ich hier schon als komerziell ansehen). Die Nutzung der Sourcen für komerzielle Programme bedarf der Zustimmung des Autors.
In jedem Fall würde ich mich freuen, wenn ich ein Belegexemplar erhielte (falls das Programm auf einem Internet-PD-Server liegt reicht mir auch ein Hinweis, wo ich es finden kann; der Wunsch bezieht sich hauptsächlich auf ST/TT/Falcon-Software).
Genauso würde ich mich natürlich freuen, wenn ich von externen Routinen ein Exemplar bekäme (wird die Routine auf dem Clausthaler Internet-PD-Server abgelegt, dann finde ich sie auch ohne Message).
Komerziell vertriebene externe Routinen (wenns denn unbedingt sein muß) unterstüze ich nur, wenn ich ein Exemplar erhalte.

2 Modulkonzept

2.1 Übergabe der Parameterstrukturen

Bei den Druckermodulen existiert nur eine Parameterstruktur. Die Initialisierungsroutine gibt einfach einen Zeiger auf diese Struktur zurück. Die nötigen Kennungen sind in dieser Parameterstruktur enthalten.
Bei den Berechnungsroutinen gibt es zwei Parameterstrukturen, außerdem ist es möglich, daß Berechnungsroutinen für mehrere Bildtypen zu einem Modul zusammengefaßt sind, so daß die Strukturen mehrerer Bildtypen zurückgegeben werden müssen. Es ist also nicht möglich direkt einen Zeiger auf die Parameterstruktur zurückzugeben, sondern es wird eine weitere Struktur benötigt, die dann die Parameterstrukturen (oder Zeiger auf diese) zusammenfaßt. An CHAOS*ultd* wird dann ein Zeiger auf eine solche Struktur zurückgegeben.
Dabei gibt es zwei Möglichkeiten: Entweder die Parameterstrukturen liegen bereits als Arrays vor. Dann genügt es, je einen Zeiger auf den Anfang dieser Arrays und die Anzahl der Routinen zurückzugeben. Dazu kommt noch eine Kennung in diesem Fall `CHSultdG`. Realisiert ist das Ganze in der Struktur `X_CHAOS`, die Kennung ist als `X_CHAOS_ID` definiert. Nach der Kennung ist noch die erwartete Version der Schnittstelle anzugeben (die aktuelle Versionsnummer ist in *COMMON.H* als `CHS_VERSION` definiert). Die Versionsnummer der Schnittstelle dient dazu, festzustellen ob die Version des Programmes und des Moduls zusammenpassen.

Da die Parameterstrukturen nicht unbedingt als Array vorliegen müssen gibt es noch die andere Möglichkeit, bei der zusätzlich ein Array mit Zeigern auf die verschiedenen Parameterstrukturen angelegt werden muß. In diesem Fall verwendet man die Struktur `X_CHAOS`, in der nur noch der Anfang der Arrays von Zeigern angegeben werden muß, und die Kennung `X_CHAOS_ID` (CHSultdg).

3 ein paar CHAOS*ultd*-Interna

Ich möchte vor allem die Verwaltung der Objekte in CHAOS*ultd* kurz erläutern, da diese Informationen im folgenden nützlich sein dürften. Zu jedem Objekt (Bild (mit oder ohne Fractal-Parameter) oder Film) wird eine Objektstruktur verwaltet:

```
typedef struct
{
    X_DATA_P col_tab;   /* zeiger auf farbtabelle */
                        /* (array of COLOR={int reg; int green; int blue}; */
    X_DATA_P pic_data;  /* zeiger auf bilddaten */

    int     width;      /* breite der bitmap */
    int     height;     /* soll-hoehe der bitmap */
    int     xheight;    /* ist-hoehe der bitmap (u.U. <height) */
    int     planes;     /* zahl der bildebenen */
    int     colors;     /* zahl der farben in col_tab */

    char    packed;     /* UNPACKED, UNPACKABLE, PACKED */
    char    raster;     /* 1, 2, 4 (oder sonstwas) */

    int     xor_offset; /* xor-offset in BYTE */
    long    pic_len;    /* laenge der daten in pic_data */
} BITMAP;

typedef struct
{
    long    null1;      /* muß 0 sein (in bitmap X_DATA_P!) */
    long    null2;      /* muß 0 sein (in bitmap X_DATA_P!) */
    SHOW_OPTS opts;     /* anzeigeoptionen */
    int     pic_anz;    /* zahl der bilder */
} FILM;

typedef union {
    BITMAP  bitmap;
    FILM    film;
} X_OBJC;

typedef struct
{
    int     nr;         /* nummer des objectes */
    int     typ;        /* object-typ: 0 film  1 pic  2... fractale */
    int     objc_flag;  /* flag für icon */
    int     objc_state; /* status für icon */
    char    name[10];   /* name des objectes */
    char    path[40];   /* pfad des objectes */
    char    ext[4];     /* extension */

    X_OBJC  x;          /* BITMAP oder FILM */

    X_DATA_P param;     /* speicherblock mit parametern */
    X_DATA_P x_param;   /* speicherblock mit zus. parametern */
} CHS_OBJC;
```

Diese CHS_OBJC-Struktur enthält alle Daten des Objektes (entweder selber oder in Form von Zeigern auf Speicherblöcke). Ein Zeiger auf die Struktur eines Objektes wird deshalb an alle Funktionen übergeben, die sich auf das Objekt beziehen.

4 Druckermodule

Zunächst sollen Druckermodule, die viel weniger umfangreich sind als Berechnungsmodule, beschrieben werden.

4.1 Id's

Die Modulstruktur (CHS_PRN) beginnt mit der Schnittstellen-Id (char chs_id[8], X_CHAOS_ID, d.h. CHSultdG), der Versionsnummer der verwendeten Schnittstelle int chs_version (CHS_VERSION) und der Hardcopy-Id (long hcopy_id, HCPY). An diesen beiden Id's erkennt CHAOS*ultd*, daß es sich tatsächlich um eine Hardcopy-Routine handelt. Die Versionsnummer der Schnittstelle dient dazu, festzustellen ob die Version des Programmes und des Moduls zusammenpassen.
Anschließend kommt die individuelle Kennung char id[8], die beim Speichern der Einstellungen verwendet wird. Hier ist im Prinzip vorgesehen, daß die ersten vier Zeichen für den Autor und die restlichen vier Zeichen für die Routine stehen[2].
Es folgen drei Zeiger auf Textstrings char *text, char *name und char *version, die für eine Beschreibung der Routine, den Namen des Autors und die Versionsnummer (als ASCII-Text) gedacht sind. Maximale Länge der Texte ist 32, 24 und 20 Zeichen. Bei den Drucker-Routinen wird bisher nur name verwendet, und zwar für den Init-Dialog.

4.2 Routinen

Als nächstes kommen die Routinen, die von CHAOS*ultd* aufgerufen werden. Als da wären:

do_init

Definition: `int do_init(void);`
Aufgabe: Initialisierung der Routine
Status: verpflichend
Parameter: keine
Rückgabewerte: RET_OK oder Fehlercode, insbesondere RET_INIT
Beschreibung: Die Routine do_init muß die Initialisierung der Hardcopy-Routine, das ist vor allem das Laden des Resourcefiles und die Anpassung der Objektbäume, übernehmen.
Zurückgegeben wird RET_OK falls Initialisierung fehlerfrei; falls die Initialisierung fehlerhaft ist RET_INIT oder irgend einen anderen von RET_OK abweichenden Wert. Im Fall von RET_INIT wird von CHAOS*ultd* eine Alertbox mit Hinweis auf einen Fehler beim Laden der Resourcedatei zum Modul hingewiesen (weil das eigentlich das einzige ist was schiefgehen kann), bei anderen Rückgabewerten wird keine Fehlermeldung ausgegeben.

hardcopy

Definition: `void hardcopy(int ks,int dm,void *pw,int objc);`
Aufgabe: Hardcopy ausgeben.
Status: verpflichend
Parameter: die Parameter dienen der Festlegung der betroffenen Objekte
Rückgabewerte: keiner
Beschreibung: Die Funktion hardcopy ist die eigentliche Hardcopy-Routine.
Mit Hilfe der Funktionen first_ob (Modus GET_PICS) und next_ob werden die betroffenen Bild-Objekte ermittelt. Dabei können die Objekte auch mehrfach durchgesucht werden, etwa um erst einmal nur die Anzahl der Objekte festzustellen (geht leider nur so, daß man alle Objekte der Reihe nach mit first_ob und next_ob durchgeht).
Mit der Routine get_std_planes können die Bilddaten eines Objektes geholt werden.
Mit den Routinen get_konv_info, make_konv_str und get_hcopy_planes ist darüberhinaus die Einstellung der Konversion, die Anzeige derselben und das holen der Bilddaten mit Konversion möglich.
Zu jedem Bild (soweit für den Bildtyp vorgesehen) kann durch Aufruf der Routine pr_info ein Info zum Bild gedruckt werden.

[2] diese Kennung muß für jede Routine (Berechungsroutine oder Hardcopyroutine) anders sein, da es sonst mit der Parameterverwaltung Probleme gibt. Alle Kennungen, die mit TMMW für den Autor beginnen sind für mich reserviert

Zum Ausdrucken der Daten stehen die Funktionen `pr_byte` und `pr_bytes`, zum Feststellen der Empfangsbereitschaft des Druckers `check_printer`[3] zur Verfügung.
Ein Info-Dialog kann mit `init_xinfo` aufgerufen, mit `exit_xinfo` verlassen werden. Zur Abfrage der Tastatur (Abbruch) steht `check_key` zur Verfügung.

dr_param

Definition: `void dr_param(void);`
Aufgabe: Einstellen der Druckerparameter
Status: verpflichend
Parameter: keine
Rückgabewerte: keine
Beschreibung: Die Routine wird durch Anwahl des Menüpunktes Druckereinstellung aufgerufen.

conv_param

Definition: `void conv_param(param,void (*conv_dpos)(DIAL_POS *dp));`
Aufgabe: Umrechnen der Dialogpositionen in den Druckereinstellungen
Status: optional
Parameter: `param` Zeiger auf die Parameter
`conv_dpos` Umrechenroutine für eine Dialogposition
Rückgabewerte: keine
Beschreibung: Diese Routine wird nach dem Laden und vor dem Speichern der Einstellungen aufgerufen, und erlaubt das Umrechnen der Dialogpositionen, so daß diese halbwegs auflösungsunabhängig abgespeichert werden können.
Wichtig ist, daß *nicht* die Parameter der Routine selbst sondern die, auf die `param` zeigt, umgerechnet werden (beim Speichern wird ein Duplikat angelegt). Zum Umrechnen ist die Routine `conv_dpos` zu verwenden, die für jede Dialogposition einmal aufgerufen werden muß.

4.3 Parameter

Zur Verwaltung der Parameter der Druck-Routine existieren noch `unsigned int set_len` und `void *set`, wobei `set_len` die Länge der Einstellungen angibt, und `set` auf die Einstellungen selbst zeigt. Ist `set_len` Null, so gibt es keine Einstellungen.

Für Erweiterungen der Druckroutinen-Schnittstelle sind noch vier Langworte vorgesehen, die auf 0 zu setzen sind.

5 Berechnungsroutinen

Die Berechnungsroutinen sind deutlich umfangreicher und die Schnittstelle auf zwei Datenstrukturen `CHAOS_1` und `CHAOS_2` verteilt. (In der ersten findet sich alles, was für einfache Bilder und Filme nicht gebraucht wird, für diese beiden Typen gibt es eine solche Struktur nicht.) Deshalb werde ich im folgenden *nicht* wie bei den Hardcopy-Routinen streng in der Reihenfolge der Struktur-Deklaration vorgehen.

5.1 allgemeine Angaben

In `CHAOS_1` sind die folgenden Angaben zu machen:

- `char id[8];`
  Kennung der Routine (um Bilder und Einstellungen zu markieren). Wie bei den Druckerroutinen ist hier im Prinzip vorgesehen, daß die ersten vier Zeichen für den Autor und die restlichen vier Zeichen für die Routine stehen.
- `char *menue;`
  Eintrag der Routine im Neu-Menü (max. 16 Zeichen)
- `char *name;`
  Name des Autors für Programm-Info (max. 24 Zeichen)

[3] **muß** vor `pr_byte` und/oder `pr_bytes` mind. einmal aufgerufen werden

- `char *version;`
  Versionsnummer (und Datum) der Routine (ASCII, max. 20 Zeichen)

- `char *text;`
  Allgemeine Beschreibung der Routine oder sonstiger Kommentar (zweite Zeile im Programminfo), max. 64 Zeichen

- `unsigned int set_len`, `void *set`, `ALLG_PAR *allg_par`
  Länge der Voreinstellungen, Zeiger auf die Voreinstellungen. Falls `set_len` und `set` Null ist: keine Voreinstellungen. Zusätzlich zu den Voreinstellung werden die allgemeinen Parameter für eine Routine behandelt (die deshalb nicht in `set` liegen und bei `set_len` nicht mitgezählt werden), falls `allg_par` auf einen Speicherbereich für die allgemeinen Parameter zeigt (`allg_par` kann Null sein (keine allgemeinen Parameter verwalten), umgekehrt kann `allg_par` aber auch existieren, wenn sonst keine Voreinstellungen vorhanden sind).

- `unsigned int par_len`
  Länge der Parameter *eines* Bildes, so wie sie mit dem Bild verwaltet werden sollen.

- `unsigned int make_len`
  Länge der Eingabeparameter beim Erzeugen neuer Bilder durch Abänderung der Parameter. Hier muß die Möglichkeit von Bilderfolgen vorgesehen werden, so daß diese Parameter sicher länger sind als die Parameter eines Bildes. In den Parametern sollte darüberhinaus Platz für die Parameter eines einzelnen Bildes vorgesehen sein (vgl. Routine `get_picpar`).

- `unsigned int neu_len`
  Länge der Parameter für das Neu-Erzeugen bestehender Bilder. Dabei sollte ebenfalls Platz für die Parameter eines einzelnen Bildes vorgesehen sein.

- `X_DATA_P *neu_par`, `int neu_flag`
  Zeiger auf Zeiger auf Speicherblock mit Parametern zum Neu-Berechnen von Bildern (vgl. Routinen zum Neuberechnen) und Flag für das Neuberechnen. Folgende Möglichkeiten: -2 kein Neu-Berechnen möglich[4]. 0 Neu-Berechnen möglich (während des Ablaufes des Neu-Berechnens muß dann 1 für Ok und -1 für Abbruch eingetragen werden).

- `char xor_lines`
  Flag, um wieviele Punkt-Zeilen (nicht Pixel-Zeilen, eine Raster-Punkt-Zeile entspricht mehreren Raster-Pixel-Zeilen) beim Packen der Bilder diese mittels zeilenweisem Xor verknüpft werden. Sinnvoll ist hauptsächlich 0 (kein Xor) oder 1 (Xor mit einer Zeile), wobei Xor immer dann sinnvoll ist, wenn im Bild größere (nichtweiße) Flächen existieren (z.B. Fractale, nicht aber Feigenbaumdiagramme).

- `char xget_flag;`
  Mögliche Vorgaben für die Eingabe der Parameter:
  `_GET_OBJC` Objektdaten
  `_GET_BLOCK` Block
  `_GET_KONST` Konstante
  `_GET_RBLOCK` rotierter Block
  (Bei Vorgaben für die Eingabe von Parameter ist der Objekttyp des Bildes aus dem die Daten stammen und des zu erzeugenden Bildes natürlich gleich).

- `LIMITS limits;`
  Vorgabe der minimalen und maximalen möglichen Bildhöhe und -breite sowie der minimalen und maximalen Farbzahl. In `limits.raster` wird noch Angegeben, ob Raster erlaubt sind (1) oder nicht (0).
  (Die Angaben werden *nicht* auf Konsistenz gecheckt. Also sinnvolle Daten eingeben!)

- `X_RES_DATA *x_mem;`
  Zeiger auf halbresidente Speicherblöcke, die zwar nicht gleich freigegeben werden, die aber von der Speicherverwaltung freigegeben werden können falls Speicherbedarf herrscht. (Siehe Speicherverwaltung)

In `CHAOS_2` hat man dann noch:

[4] Dies sollte man nur Eintragen, wenn das Neu-Berechnen tatsächlich nicht möglich ist. Ein Wert ungleich -2 bedeutet *nicht*, daß die Routine Parameteränderungen beim Neu-Berechnen ermöglichen muß, die Routine `get_neu_param` kann fehlen. Es bleibt dann die Möglichkeit, Bildgröße usw. zu verändern. Bei den bisherigen Routinen erlauben nur die IFS-Kopierer kein Neu-Berechnen, da hier kein Quellbild mehr vorhanden ist.

- `ICONBLK *icon;`
  Zeiger auf das Icon für den Bildtyp
- `char *typ;`
  Zeiger auf String mit Namen des Bildtypes (wird bei Bildparametern verwendet)
- `char last_flag`
  Flag, das angibt ob das Bild zeilenweise berechnet wird oder nicht. Im ersteren Fall kann CHAOS*ultd* bei teilweise berechneten Bildern sich darauf beschränken nur den oberen Teil des Bildes abzuspeichern (intern im Speicher), wobei die bisherige Höhe von der Routine zur Berechnung des Bildes zurückgegeben wird. Dazu ist `last_flag` auf 1 zu setzen. Ist `last_flag` 0, dann wird stets das ganze Bild gespeichert. (In diesem Fall geht eine von der Berechungs-Routine eventuell zurückgelieferte Höheninformation verloren!)
- `char block_flag`
  Flag, das angibt ob im Bild ein Block markiert werden kann (`block_flag` = 1).

5.2 Routinen

Initialisierung

do_init

vergleiche `do_init`-Routine der Hardcopy-Routinen.

conv_param

vergleiche `conv_param`-Routine der Hardcopy-Routinen.

Bilderzeugung

Die Routinen, die zur Erzeugung von Bildern (Parametereingabe und eigentliche Berechnung) aufgerufen werden, werden in `CHAOS_1` verwaltet (die Reihenfolge untereinander wurde beibehalten).

Ablauf der Erzeugung neuer Bilder

Bei der Erzeugung neuer Bilder wird (schematisch) so vorgegangen:
als erstes läßt CHAOS*ultd* die allgemeinen Bildparameter eingeben. Dann werden – via externer Routine – die speziellen Parameter des Bildtypes eingelesen und zwar für eine ganze Bildfolge. Mit einer weiteren externen Routine werden dann Bild für Bild aus den Bildfolgenparametern die Bildparameter erzeugt und die Berechnungsroutine aufgerufen.

get_param

Definition: `int get_param(int typ,AP_DATA *ad,int mode,int ks,GET_DATA *gd);`
Aufgabe: Eingabe der Parameter für das Erzeugen neuer Bilder
Status: obligatorisch
Parameter: `typ`: Bildtyp
`ad`: Zeiger auf Eingabedatenstruktur
`mode`: Flag für Vorgabe von Daten
`ks`: Sondertastenstatus
`gd`: voreinzustellende Daten
Rückgabewerte: 0: Abbruch sonst: Ok
Beschreibung: Die Routine dient der Eingabe der Parameter für neue Bilder. Dabei kann optional ein Objekt oder Block oder eine Konstante vorgegeben werden, deren Daten anstelle der Vereinstellungen gesetzt werden sollen (ob dies möglich ist, regelt das Flag `xget_flag` in `CHAOS_1`).
Für die Parameter steht mit `ad->sp` ein Zeiger auf einen bereits reservierten Speicherblock (vgl. Speicherverwaltung) zur Verfügung. Dort müssen die Parameter (für erstes und letztes Bild einer Bilderfolge) abgelegt werden. `ad->ap` zeigt auf die allgemeinen Parameter (die schon eingestellt sind). `ad->set_flag` ist 0, falls Parameter eingegeben werden sollen, 1 falls die Parameter voreingestellt werden sollen und 2 beim Einstellen neuer Parameter. `local1-3` werden vor Aufruf auf 0 gesetzt und können von der Routine beliebig verwendet werden.
nützliche Unterstützungsroutinen: `write_double` zur Ausgabe von Fließkommazahlen
`get_allg_par` um die allgemeinen Parameter nochmal ändern zu lassen
`write_ap_info` um eine Info über die allgemeinen Parameter anzuzeigen

`set_ipol_pbutton`, `do_ipol_pbutton`, `get_ipol_pbutton` und `do_get_popup` zur Unterstützung von Popupmenüs
`show_aratio` zur Ausgabe und Korrektur des Seitenverhältnisses
`get_object` und `get_obblk` zur Übernahme von Daten aus bestehenden Objekten (auch Block, Konstante)
`conv_sc_mm` und `conv_mm_sc` zur Umrechung zwischen minimalen und maximalen Koordinaten einerseits und Skalierungsfaktor und Offset andererseits

get_picpar

Definition: `void *get_picpar(int typ,void *param_in,int nr,int anzahl);`
Aufgabe: Erzeugung der Parameter für ein Bild aus den Bildfolgenparametern
Status: obligatorisch
keine Aufrufe der Speicherverwaltung
Parameter: `typ` Bildtyp
`param_in` Zeiger auf Parameter
`nr` Nummer des aktuellen Bildes (0 . . .anzahl-1)
`anzahl` Zahl der insgesammt zu Erzeugenden Bilder
Rückgabewerte: Zeiger auf erzeugte Parameter (i.A. irgendwo in der `param_in` Struktur liegend
Beschreibung: Diese Routine muß aus den eingegeben Parametern, die ja unter Umständen eine ganze Bilderfolge definieren, die Parameter der einzelnen Bilder erzeugen
nützliche Unterstützungsroutinen: `ipol` und `xipol` zur Interpolation zwischen den Anfangs- und Endwerten

set_param

Definition: `void set_param(int typ,AP_DATA *ad);`
Aufgabe: Voreinstellung der Parameter
Status: optional
Parameter: `typ` Bildtyp
`ad` Parameter
Rückgabewerte: keine
Beschreibung: Die Routine soll die Voreinstellung der Parameter erlauben. Die Parameter können zunächst wie bei `get_param` in `ad` eingelesen werden. Die berechnungsroutinen-spezifischen Parameter müssen anschließend von der Routine selber in die Voreinstellungen kopiert werden. Für die allgemeinen Parameter übernimmt dies, soweit Voreinstellungen existieren, CHAOS*ultd*.

extended

Definition: `void extended(int typ);`
Aufgabe: keine
Status: optional
Parameter: keine
Rückgabewerte: keine
Beschreibung: Die Routine wird aufgerufen, falls der Menüpunkt der Berechnungsroutine mit `CONTROL` aufgerufen wird und dient beliebigen Zwecken.

do_draw

Definition: `int do_draw(CHS_OBJC *objc,int width,int height,int colors,BITMAP *bitmap,int *xheight);`
Aufgabe: Zeichnen eines Fractals
Status: obligatorisch
Parameter: `objc` Objekt, das gezeichnet werden soll
`width`, `height`, `colors` Breite, Höhe (in Punkten, *nicht* Pixeln) und Zahl der Farben (oder Raster)
`bitmap` Bitmap, in die gezeichnet werden soll
`xheight` Höhe, bis zu der schon gezeichnet wurde (falls Bild zeilenweise berechnet wird); dient auch zur Rückgabe dieses Wertes!
Rückgabewerte: Sondertastenstatus beim Berechnungsabbruch (wird von `check_key` geliefert) oder 0 (falls kein Abbruch). Im Fehlerfall kann ein Wert ¡0 zurückgegeben werden, dann wird die Berechnung (auch für folgende Bilder) abgebrochen
Beschreibung: Jetzt gehts zur Sache. Mit dieser Funktion werden die Bilder tatsächlich berechnet. Um die berechneten Punkte oder Linien auch tatsächlich auszugeben bediene man sich *ausschließlich* derjenigen Punktsetz-Routinen,

die man mittels `init_draw` und `init_xdraw` übergeben bekommen hat (zum Linienzeichnen gibts `draw_line`, dem eine solche Punktsetz-Routine übergeben werden muß).
Zum Parameter `xheight`: `xheight` zeigt auf ein Flag, mit dem angegeben wird, ob der Aufruf für ein völlig neues oder ein schon teilweise berechnetes Bild erfolgt. Im ersten Fall ist `xheight` 0, im zweiten ungleich 0, wobei bei zeilenweise berechneten Bildern (`last_flag` gesetzt) die Raster-Zeile, bei der die Berechnung fortgesetzt werden soll angegeben wird.
`xheight` dient gleichzeitig zur Rückgabe des berechneten Bildteiles: wird -1 zurück gegeben, so heißt das, daß das Bild fertigberechnet wurde, ein Wert ungleich 0 entspricht einem teilweise berechneten Bild, wobei bei gesetztem `last_flag` die letzte berechnete Rasterzeile und sonst die Bildhöhe (`height`) zurückgegeben werden muß. Der Wert 0 bedeutet (unabhängig vom `last_flag`), daß kein Bild existiert (bzw. ein leeres, was das selbe ist).

dont_draw

Definition: `int dont_draw(CHS_OBJC *objc);`
Aufgabe: Parameter verarbeiten, wenn Berechung eines Bildes wegen Abbruchs nicht aufgerufen wird (um später weiter zu ermöglichen)
Status: optional
Parameter: `objc` Objekt, das nicht berechnet wird
Rückgabewerte: 0 Ok, ¡0 Fehler (wie bei `do_draw`)
Beschreibung: Diese Routine wird unter Umständen gebraucht, um bei Bildern, für die die Zeichenroutine nicht aufgerufen wird, die Parameter zu korrigieren.

Ablauf beim Neu-Berechnen von Bildern

Beim Neu-Berechnen von Bildern werden zunächst die neuen Einstellungen für die allgemeinen Parameter eingelesen, wobei auch gleich festgelegt wird, ob die Bilder überschrieben oder neu erzeugt werden sollen.
Anschließend werden alle betroffenen Bilder durchgegangen und für den entsprechenden Bildtyp die Eingabe der neuen Parameter `get_neu_param` aufgerufen, falls das noch nicht geschehen ist. Die Eingaberoutine muß – um zu kennzeichnen, daß sie aufgerufen wurde – das `neu_flag` entsprechend setzen.
Sind die Parameter alle eingesammelt, so werden nochmals alle betroffenen Bilder durchgegangen und diesmal neu berechnet. Dazu läßt CHAOS*ultd* die exteren Routine zunächst aus den Änderungsparametern für den Bildtyp und den Bildparameter des alten Bildes die Parameter des neuen Bildes erzeugen (mit `make_neu_param`). Werden neue Bilder erzeugt, so werden die geänderten Parameter in das neue Objekt geschrieben, andernfalls werden sie ins alte übernommen. Wenn das neue Bild neu gezeichnet werden muß (dies ist eigentlich nur dann nicht der Fall, wenn nur die Farbtabelle geändert wurde), wird vor dem eigentlichen Aufruf der Berechnungsroutine noch die `xneu_param`-Routine aufgerufen, so sie für den Bildtyp existiert. Berechnet werden die Bilder natürlich wie immer mit `do_draw`.

get_neu_param

Definition: `int get_neu_param(int typ,X_DATA_P *param,ALLG_PAR *ap);`
Aufgabe: Parameter für Neu-Berechnen von Objekten eingeben
Status: optional
Parameter: `typ` Objekttyp
`param` Zeiger auf Zeiger auf Speicherbereich für Parameter
`ap` Allgemeine Parameter
Rückgabewerte: 0 Ok, 1 Abbruch
Beschreibung: mit dieser Routine werden die Parameter für das Neu-Berechnen von Bildern aufgerufen. Die Routine muß die Eingabe der Parameter übernehmen und anschließend `neu_par` (Zeiger auf Speicherbereich mit den Parameter, also `param`) und `neu_flag` Flag, ob Eingabe Ok oder Abbruch (1 bzw. -1) setzen (dies ist notwendig, um die Parameter für mehrere Bildtypen gleichzeitig einstellen zu können (wie bei Fractalen)).

make_neu_param

Definition: `void *make_neu_param(CHS_OBJC *objc,void *param,int *change);`
Aufgabe: Erzeuge Parameter für neues Bild aus Änderungsparametern und Parametern des alten Bildes
Status: optional, muß existieren, wenn get_neu_param ex.
keine Speicherverwaltungsaufrufe
Parameter: `objc` Objekt des alten Bildes
`param` Änderungsparameter

Rückgabewerte: `change` Flag, ob sich Parameter geändert haben (1 sonst 0)
neue Parameter (Platz für die Parameter sollte in param sein)
Beschreibung: die Routine muß (analog `get_picpar`) die Parameter für das neue Objekt ermitteln. In `change` ist zurückzugeben, ob sich die Parameter gegenüber den alten geändert haben.

xneu_param

Definition: `void xneu_param(CHS_OBJC *objc);`
Aufgabe: passe Parameter eines Objektes für neue Berechnung an
Status: optional
Parameter: `objc` Objekt
Rückgabewerte: keine
Beschreibung: diese Routine wird (falls existent) aufgerufen, bevor ein Objekt neu berechnet wird. Damit kann die Routine womöglich nötige Anpassungen in den Parametern vornehmen (z.B. Zahl der Iterationen auf Null setzen). Die Routine ist besonders dann wichtig, wenn Neu Berechnen nicht explizit unterstützt wird (keine `get_neu_param`-Routine), aber nicht unmöglich ist (`neu_flag` nicht -2). Dann kann mit Neu-Berechnen beispielsweise noch die Bildgröße verändert werden. Wird das Bild dann neu berechnet, so sind womöglich Anpassungen an den Parametern nötig.

Parameter anzeigen und Infos ausgeben

Diese Funktionen finden sich in der `CHAOS_2`-Struktur (Reihenfolge ebenfalls beibehalten).

show_par

Definition: `int show_par(CHS_OBJC *ob);`
Aufgabe: Fractalparameter eines Bildes anzeigen
Status: optional, sollte existieren
Parameter: `ob` Bildobjekt
Rückgabewerte: 0: nichts 1: Zeichne Objekt-Icon neu
Beschreibung: Anzeige der Fractalparameter. Die Routine darf die Parameter ändern (wenn dies mit der Berechnungsroutine vereinbar ist). Zur Änderung des Objektstatus stehen die Funktionen `set_unready` (für unfertig erklären) und `set_ready` (für fertig erklären) zur Verfügung.

show_info

Definition: `void show_info(CHS_OBJC *ob);`
Aufgabe: Zeige Berechnungsinfo an
Status: optional
Parameter: `ob` Bildobjekt
Rückgabewerte: keine
Beschreibung: Die Routine wird aufgerufen, wenn das Bildinfo für *ein* Bild des Types aufgerufen wird.
Fehlt die Routine, so kann ein Bildinfo auch durch CHAOS*ultd* selbst ausgegeben werden, soweit die Routine `get_info` existiert.

get_info

Definition: `void get_info(CHS_OBJC *ob,P_INFO *pi);`
Aufgabe: Gebe Informationen über Berechnung an CHAOS*ultd* zurück
Status: optional
Parameter: `ob` Bildobjekt `pi` Zeiger auf Struktur für die Daten
Rückgabewerte: keine
Beschreibung: In `pi` sind Informationen über die Berechnung einzutragen: `points`, die Zahl der Bildpunkte, `time`, die Berechnungszeit in Sekunden, `iter`, die Zahl der Iterationen für das Bild. Alle drei Angaben sind als Fließkommazahlen einzutragen, für ein Bild können die Angaben natürlich auch in Langworten verwaltet werden (Zeit in Timer-Steps).

get_objcblk

Definition: `void get_objcblk(CHS_OBJC *objc,R_REC *r);`
Aufgabe: Gebe Koordinaten des Bildrechtecks zurück
Status: optional, sollte existieren, wenn ein Bildrechteck existiert **Parameter:** `objc` Bildobjekt
Rückgabewerte: `r`
Beschreibung: Die Koordinaten der linken oberen Bildeck und der rechten unteren werden zurückgegeben
`R_REC` ist in *COMMON.H* definiert und umfaßt neben minimalem und maximalem x bzw. y auch noch einen Winkel `phi`. Unterstützt die Routine keine gedrehten Blöcke, so ist `phi` auf 0.0 zu setzen.

get_konst

Definition: `void get_konst(CHS_OBJC *objc,double *x,double *y);`
Aufgabe: Gebe Konstante aus den Fractalparametern zurück
Status: optional
Parameter: `objc` Bildobjekt
Rückgabewerte: `x`, `y`,
Beschreibung: wie `get_block`, nur daß eine Konstante zurückgegeben wird

pkt_info

Definition: `void pkt_info(CHS_OBJC *object,int x,int y);`
Aufgabe: Zeige Punktinfo für die (Raster)koordinaten x,y an
Status: optional
Parameter: `object` Bildobjekt
`x`, `y`,
Rückgabewerte: keine
Beschreibung:

get_dr_info

Definition: `void get_dr_info(CHS_OBJC *object,DR_INFO *di);`
Aufgabe: Gebe Info-Strings für Hardcopy zurück
Status: optional
Parameter: `object` Bildobjekt
`di` Zeiger auf String-Struktur
Rückgabewerte: keine
Beschreibung: In die Strings `di->str[]` dürfen maximal 5 Zeichenketten (jeweils max. 80 Zeichen) mit Informationen zum Bild in `object` geschrieben werden, die dann unter die Hardcopy gedruckt werden.

X_TYP *x_typ

Mit Hilfe des Zeigers `x_typ` ist es möglich Routinen zur Konversion der Parameter alter Versionen einer Routine anzugeben. Ist eine solche Möglichkeit nicht vorgesehen, so ist 0l einzutragen.
`x_typ` zeigt auf eine folgendermaßen definierte Struktur:

```
typedef struct {
    char id[8];
    uint par_len;
    void (*conv_param)(void *out,void *in,X_DATA_P *x_param,BITMAP *bitmap);
} _X_TYP;

typedef struct {
    int anzahl;
    _X_TYP *x_typ;
} X_TYP;
```

In der `X_TYP`-Struktur gibt man die Zahl vorhandenen Routinen an, `x_typ` zeigt auf eine entsprechende Zahl von `_X_TYP`-Strukturen. In diesen wird die `id` der alten Version, die Länge der Parameter und eine Konversionsroutine `conv_param` angegeben. Der Konversionsroutine wird (direkt) ein Zeiger auf die Parameter im alten Format `in`, ein

Zeiger auf Speicher für die Parameter im neuen Format (ebenfalls direkt) `out` sowie ein Zeiger auf den Speicherblock mit den zusätzlichen Parametern und die Bitmap-Struktur für das geladene Bild übergeben. *Achtung!* nach Speicherverwaltungszugriffen darf auf die Parameter in `in` und `out` nicht mehr zugegriffen werden.

6 XCHAOS-Routinen

Mit Hilfe der XCHAOS-Routinen kann man wie gesagt besonders einfach neue Bildtypen realisieren. Außer der eigentlichen Berechnungsroutine, die der normaler Berechnungsroutinen entspricht, muß man nur noch einige allgemeine Angaben zu den Parametern machen.
Dazu werden in der Headerdatei *XCHS.H* zunächst einige Konstante und Strukturen definiert:

```
typedef enum        /* Variablentypen */
{
    P_EMPTY=0,
    P_INTEGER,
    P_LONG,
    P_DOUBLE
} VAR_TYPE;

typedef struct {    /* Variablendefinition */
    VAR_TYPE type;
    char *text;
} XCHAOS_VAR;

typedef enum        /* Flagtypen */
{
    F_EMPTY=0,
    F_BUTTON,
    F_RBUTTON,
} FLAG_TYPE;

typedef struct {    /* Flagdefinition */
    FLAG_TYPE type;
    char *text;
    int anz;
    char *texte[5];
} XCHAOS_FLAG;

typedef struct {    /* voreinstellungen für popupmenü in parametereingabe */
    char *name;
    void *set;
} DEF_SET;

typedef struct {
    int         koords;     /* 0 keine, 1 x/y min/max */

    XCHAOS_VAR  par[6];     /* parameter */
    XCHAOS_FLAG flag[4];    /* flags */

    int         def_set;    /* zahl möglicher voreinstellungen */
    DEF_SET     *sets;      /* zeiger auf voreinstellungen */

    char        *x_param;   /* hilfs-datei für parametereingabe */
    char        *x_neu;     /* hilfs-datei für neu */
    char        *x_spar;    /* hilfs-datei für zeige parameter */
} XCHAOS_DEF;
```

Die zentrale Struktur ist `XCHAOS_DEF`, in ihr wird festgelegt, welche Parameter, Voreinstellungen und Hilfsdateien zu verwenden sind. Im einzelnen:

`koords` ist ein Flag, das festlegt, ob Koordinaten benötigt werden oder nicht. Koordinaten sind nur als minimales/maximales x/y möglich.
In `par` gibt man an, welche weiteren Parameter benötigt werden (maximal 6). `par[].type` ist ein Flag für den Typ des Parameters, der Integer (2 Byte), Long (4 Byte) und Double (10 Byte) sein kann, `par[].text` zeigt auf den Text, der im Eingabedialog zu diesem Parameter erscheinen soll. Wird ein Parameter nicht gebraucht, so setzt man `par[].typ` auf `P_EMPTY` und gibt 0l für den Dialogtext an.
Analog beschreibt man in `flag` bis zu vier Flags für den Bildtyp. Dabei kann man zwischen normalen Button und Radiobutton wählen. Dazu dient wieder `flag[].type`. Der Wert `F_EMPTY` bedeutet, daß das Flag unbenutzt ist. `flag[].text` ist der Dialogtext für alle Button, `flag[].anz` die Zahl der Button (1-5) und `flag[].texte[]` die Dialogtexte für die einzelnen Button.
Mit `def_set` und `sets` kann man Parametervoreinstellungen, die im Eingabedialog mittels Popup-Menü aufgerufen werden können, definieren. `def_set` ist die Zahl solcher Voreinstellungen (0-16), `sets` zeigt auf ein Array mit Voreinstellungen, für die deren Name (der dann im Popup erscheint und sonst keine Bedeutung hat) und die Voreinstellung selbst anzugeben ist. Voreingestellt werden die Parameter *eines* Bildes, also nur eine Parameterhälfte.

Für die Parameter definiert man sich vernünftigerweise eine Struktur, damit man in der Zeichenroutine einfach darauf zurückgreifen kann.
Dabei geht man entsprechend den Angaben in `XCHAOS_DEF` so vor: ist das Koordinaten-Flag gesetzt, so beginnen die Parameter mit vier double-Variablen, für xmin, xmax, ymin und ymax. Es folgen – in der gleichen Reihenfolge wie in `par` – die maximal sechs Parameter, `int`, `long` oder `double`, für nicht vorhandene Parameter gibt man natürlich nichts an. Für jedes Flag wird eine `char`-Variable verwendet. Dabei wird bei normalen Button dem ersten Button das Bit 0, dem zweiten das Bit 1 usw. zugeordnet, bei Radiobutton enthält die Variable die Nummer des selektierten Button (0-4). Es folgen zwei oder drei Long-Integer Variable für die Berechnungszeit, die Zahl der Iterationen und die Zahl der Punkte (oder sonstigen Bildelemente). Die Zahl der Punkte fehlt, wenn das Bild zeilenweise berechnet wird und deshalb das `last_flag` auf 1 gesetzt ist.
Für die Vorbelegung der Voreinstellungen (nicht die schon erwähnten für die Parametereingabe, sondern die, die man gleich in der `CHAOS_1`-Struktur angeben muß) definiert man sich noch eine weitere Struktur, die immer aus einer Parameterstruktur `start`, eine Parameterstruktur `ende` und einer der Zahl von Parametern (nicht Flags) entsprechenden Flags für die Interpolation entspricht (vgl. `P_SET` in den Popcorn-Routinen). Bei der Zahl von Parametern sind die Koordinaten (so vorhanden) als vier Parameter mitzuzählen).
Um den XCHAOS-Routinen die nötigen Informationen zu liefern, übergibt man ihnen eine `XCHAOS`-Struktur:

```
typedef struct {
    XCHAOS_DEF def;
    CHAOS_1 chaos_1;
    CHAOS_2 chaos_2;
    PAR_OFFS offsets;
} XCHAOS;
```

`def` enthält die erläuterte Definition der Parameter. `chaos_1` und `chaos_2` sind die chaos-Strukturen, die dann an das Hauptprogramm übergeben werden (wie bei normalen Berechnungsroutinen), die man aber nur zu einem kleinen Teil ausfüllen muß. `offsets` wird gänzlich von XCHAOS ausgefüllt, die zusätzliche Routine muß nur den Speicherplatz dafür zur Verfügung stellen.
Welche Parameter müssen nun in den beiden chaos-Strukturen ausgefüllt werden (zu den Inhalten siehe die Beschreibung der normalen Routinen):
In `chaos_1` ist einzutragen:

```
char id[8];
char *menue;
char *name;
char *version;
char *text;
uint set_len;
void *set;
ALLG_PAR *allg_par;
...
char xor_lines;
...
LIMITS limits;
X_RES_DATA *x_mem;
...
```

```
int (*do_draw)(CHS_OBJC *objc,int width,int height,int colors,BITMAP *bitmap,int *xheight);
int (*dont_draw)(CHS_OBJC *objc);
...
```

In `chaos_2`:

```
ICONBLK *icon;
char *typ;
char last_flag;
...
```

Anmerkungen: die Pünktchen deuten an, daß hier andere Variable stehen, die von XCHAOS ausgefüllt werden; bei der Vorbelegung der Definition als Variableninitialisierung (nötig!, es wird keine Routine zu Vorbelegung aufgerufen) muß man hier natürlich entsprechen Nullen eintragen.
Die `dont_draw`-Routine ist natürlich optional und wird für XCHAOS-Routinen wohl auch nicht gebraucht, also 0l eintragen.
Beim `icon` kann man für die Icon-Maske 0l eintrage, dann verwendet XCHAOS eine Maske in der Größe des gesammten Icons (so wie bei bisher allen Bildtypen), das spart einige Byte.

7 CHAOS*ultd*-Routinen für externe Module

Wie schon gesagt stellt CHAOS*ultd* seinerseits den externen Modulen eine Reihe von Routinen zur Verfügung.

7.1 Bibliotheksroutinen

Teilweise handelt es sich bei diesen Routinen einfach um Bibliotheksroutinen, die vor allem deshalb in die `COMMON`-Struktur aufgenommen wurden damit sie nicht zu jedem Modul neu gelinkt werden müssen (deshalb sollte man hier auch die Routinen von CHAOS*ultd* verwenden und nicht die Bibiliotheken dazulinken).
Diese Routinen sollen zuerst erklärt werden:

MyDial-Routinen

```
void dial_fix(OBJECT  *tree,BOOLEAN is_dialog);
long get_obspec(OBJECT *tree, WORD obj);
void set_obspec(OBJECT *tree, WORD obj, LONG obspec);
int popup_select(OBJECT *dialtree, WORD btn, OBJECT *poptree, WORD obj,
                         BOOLEAN docheck, WORD docycle, BOOLEAN *ok);
int popup_menu(OBJECT *tree, WORD obj, WORD x, WORD y, WORD center_obj,
                         BOOLEAN relative, WORD bmsk, BOOLEAN *ok);
int do_alert(int defbut, CONST char *txt);
int dial_alert(BITBLK *alicon,CONST char *string,int defbut,int undobut,int align);
```

Diese Routinen entsprechen (fast) exakt den gleichnamigen MyDial-Routinen. Einziger Unterschied: bei den Routinen `popup_menu`, `do_alert` und `dial_alert` wird gegebenenfalls die Farbe vom angezeigten Bild auf die Desktopfarbe umgeschaltet.

AES-Routinen

```
int rsrc_load(const char *path);
int rsrc_gaddr(int type,int index,void *addr);
int objc_offset(OBJECT *tree,int ob,int *x,int *y);
int graf_mkstate(int *,int *,int *,int *);
```

Alle in meinen Routinen bisher vorkommenden AES-Routinen (VDI kommt gar nicht vor) wurden übernommen, um so 1600 Byte Übergabearrays in jedem Modul zu sparen.

printf-Routinen

```
int sprintf( char *string, const char *format, ... );
int sscanf( const char *string, const char *format, ... );
int fprintf( FILE *file, const char *format, ... );
int fscanf( FILE *file, const char *format, ... );
```

Auch hier war die Sparsamkeit Grund für die Aufnahme der Routinen, sie belegen doch ca. 5 kByte pro Modul, falls man sie zu jedem Modul extra dazulinken muß.

7.2 Dialogverwaltung

CHAOS*ultd* stellt eine einfach, aber leistungsfähige Möglichkeit für Fenster-Dialoge zur Verfügung. Dabei wir eine zentrale Dialogroutine aufgerufen, der eine Routine zur Bearbeitung von Exit-Objekten übergeben werden kann (fehlt diese Routine, so beendet jedes Exit-Objekt die Dialogeingabe).
Für die Dialoge gibt es eine kleine Einschränkung: das (der Objektnummer nach!) erste Objekt *muß* ein Eselsohr zum Fliegen sei (oder alternativ eine Ibox, deren verstecken im Falle von Fensterdialogen nicht stört, die für Dialog aber angezeigt wird (also nicht einfach ein Objekt mit 'hide' verschwinden lassen). Das zweite Objekt muß eine Titelzeile sein, die bei Fenstern zum Fenstertitel wird (bei Fensterdialogen wird nur der Teil des Dialoges unterhalb dieser Titelzeile angezeigt!).

xdialwind

Definition: `int xdialwind)(int flag,OBJECT *objc,DIAL_POS *ob_pos,int ob_nr, int (*do_exit)(int objc,void *data,void *dw),void *data,int *redraw,char *help_file);`
Aufgabe: Zentrale Dialogroutine
Parameter: `flag` Flag ob Fenster-Dialog möglich (1), sonst 0
`objc` Objektbaum des Dialoges (Adresse, nicht Nummer)
`ob_pos` Zeiger auf Speicher für Dialogposition (am Anfang auf 0 setzen oder mit den Voreinstellungen laden)
`struct int x, int y DIAL_POS;`
`ob_nr` erstes zu edierendes Object (==Position des Text-Cursors)
`do_exit` Routine um exit-Object zu bearbeiten (s.u.)
`data` Zeiger auf Daten, der `do_exit` übergeben wird
`redraw` Zeiger auf Flag, in dem zurückgegeben wird, ob die aufrufende Funktion einen Redraw durchführen muß oder:
0l -> nichts
-1l -> dialwind führt Redraw für alle Fenster und den Desktop selbst aus (nicht für nicht-Fenster-Dialoge)
`helpfile` Name der Hilfsdatei zum Dialog, die Hilfs-Funktion wird bei Aufruf des Hilfsbutton automatisch aufgerufen (kann auch 0l sein -> keine Hilfe).

Rückgabewerte: Nummer des Exit-Objectes mit dem der Dialog beendet wurde
Beschreibung: Der Dialog wird dargestellt, je nach Einstellungen von CHAOS*ultd* als normaler Dialog oder als Dialogfenster (modal). Dabei werden die MyDial-Routinen verwendet, entsprechende Erweiterungen also unterstützt. (Eine Versionsnummer finde ich gerade nicht, aber ich habe Interface V2.01). Werden Exit-Objekte angewählt, so wird entweder die Routine `do_exit` aufgerufen oder, falls eine solche Routine nicht vorhanden ist, der Dialog beendet. Das `redraw`-Flag ist eigentlich nur im Zusammenhang mit verschachtelten Dialogen (wärend ein Dialog dargestellt ist, wird von diesem aus ein weiterer aufgerufen) von Interesse (max. Schachtelungstiefe ist übrigens 4).
Die Routine do_exit muß sich um alle Exit-Button kümmern, die nicht zum Beenden des Dialoges führen:
`int do_exit(int objc,void *data,void *dw)`
`objc` exit-object
`data` irgendwelche Parameter (beim Aufruf an dialwind übergeben)
`dw` zeiger auf structur für dialwind (intern, eventuell für Redraw's benötigt)
Rückgabewerte:
-3 -> weiter, kein redraw
-2 -> weiter, redraw alles falls redraw_flag (internes flag)
-1 -> weiter, redraw alles
0 -> ende
>0 -> redraw objc nr.

Zum Neuzeichnen des Dialoges darf keinesfalls direkt `objc_draw` oder ähnliches verwendet werden (es könnten ja Fenster auf dem Dialog draufliegen). Zu diesem Zwecke gibt es deshalb eine Redraw-Funktion:
`void xredraw_dialwind(int objc,int flag,void *dw)`
Gezeichnet wird das Objekt `objc`. Ist `flag` gesetzt, so wird (bei NICHT-Fenster-Dialogen) nur dann neu gezeichnet, wenn das interne redraw_flag gesetzt ist (ich weiß im Moment wirklich nicht wieso, und wofür ich das gebraucht habe; generell gilt bei den Redraw-Flags wohl am besten die Regel, sich am bestehenden Code zu orientieren, ich habe hier etwas den Überblick verloren und mache es meist genauso).

7.3 Info-Dialoge

Neben Eingabe-Dialogen gibt es in CHAOS*ultd* noch Info-Dialoge, die irgend eine Aktivität des Programmes anzeigen (z.B. das Berechnen von Bildern). Zum Handling dieser Dialoge ist die `xdialwind`-Routine natürlich absolut ungeeignet. Daher stehen eigene Funktionen zur Verfügung:
`int init_infodial(OBJECT *objc,DIAL_POS *ob_pos,int flag);`
Starte einen Info-Dialog. `objc` ist der Objektbaum, `ob_pos` die Position und `flag` das Fensterdialog-Flag.
Mit `int exit_infodial(void);` wird man den Info-Dialog wieder los (nicht vergessen, bei Dialogboxen wird der ganze Bildschirm gesperrt, bei Fenstern immerhin noch die Menüleiste von CHAOS*ultd*), mit `void redraw_infodial(int objc);` kann man ein Objekt `objc` neu zeichnen lassen. Info-Dialoge lassen sich NICHT schachteln (deshalb reicht bei `redraw_infodial` die Angabe der Objekt-Nummer).
Um das ganze noch etwas komfortabler zu gestalten gibt es noch einen anderen Zugang zu Info-Dialogen. Dabei wird stets der gleiche Dialog verwendet, in dem man einen Titel und einen weitern Text (beide kurz) angeben kann. Alle Info-Dialoge (außer dem Init-Dialog) in CHAOS*ultd* sind so realisiert. Der Aufruf erfolgt mit `void init_xinfo(char *titel,char *text,int esc_flag);`, wobei diese Funktion auch mehrfach aufgerufen werden kann. Beim ersten Aufruf wird der Dialog gezeichnet, bei allen weiteren Aufrufen wird der Dialog neugezeichnet. `titel` und `text` sind natürlich der Titel und der Text des Infos, `esc_flag` gibt an, ob die Meldung Abbruch mit 'ESC' erscheinen soll oder nicht. `void exit_xinfo(void);` beendet den Dialog (nicht vergessen, wie oben).

Keine Dialoge aber ebenfalls die Möglichkeit auf interne Aktivitäten hinzuweisen bieten die Routinen `maus_busy(MFORM *mform)` und `maus_unbusy(int flag)`. Mit `maus_busy` kann man die Mausform `mform` setzen; ist `mform` 0, so wird die Biene angezeigt. Mit `maus_unbuys` wird der alte Mauszeiger wiederhergestellt. `flag` gleich Null hebt einen `maus_buys`-Aufruf auf, `flag` ungleich Null alle. (maximale Schachtelungstiefe ist 8, bei mehr `maus_buys`-Aufrufen wird `mform` nicht mehr beachtet und immer die Biene angezeigt.

7.4 spezielle Dialoge

Für mehrere Fälle stellt CHAOS*ultd* noch spezielle Dialoge zur Verfügung:
Mit `int do_error(int nr,int mode,int wind);` kann man eine Fehlermeldung ausgeben. Die Fehlernummern `nr` sind in *COMMON.H* einigermaßen selbsterklärend definiert. `mode` gibt an, ob man unter Umständen weitermachen soll (dann kann in der Fehlermeldung 'weiter' ausgewählt werden) (1, sonst 0), `wind` gibt an, ob der Dialog auch im Fenster erscheinen darf (1, sonst 0). Rückgabewert der Routine ist 1 für weiter und 0 sonst.
Mit `void do_help(char *name,int flag,int *redraw);` kann ein Hilfstext ausgegeben werden. `name` ist der Dateiname des Hilfstextes (relativ zum *HELP*-Ordner), `flag` und `redraw` die Fenster- und redraw-Flags (wie bei `xdialwind`).
`int fsel(char *path,char *sel,int *button,char *label);` ruft den Dateiselektor auf, wobei `label` nur bei hinreichend modernen TOS-Versionen als Titel hergenommen wird.

7.5 Dialoge beim Bilder-Erzeugen

Wegen der Anzeige des Mauszeigers (und anderer Eventualitäten) sollte man für Fehlermeldungen während des Zeichnens von Bildern NICHT `do_alert`, `form_alert` oder `do_error` hernehmen. Statt dessen gibt es zwei gleichwertige Funktionen, `int make_alert(int defbut, CONST char *txt);` und `int make_error(int nr,int mode,int wind);`, die sich nur durch die korrekte Erledigung der Eventualitäten von den Routinen `do_alert` und `do_error` unterscheiden.

7.6 Hilfsroutinen für die Parametereingabe

Zur Unterstützung der Parametereingabe gibt es eine Reihe von Funktionen zu unterschiedlicher Zwecken.
Die Routine `int get_allg_par(AP_DATA *ad,int typ);` erlaubt es, die Eingabe der allgemeinen Parameter nochmal aufzurufen. Für die Parameter `ad` und `typ` sind die Parameter, mit der die Eingaberoutine (`get_param`) aufgerufen wurden, zu übergeben. Vor Aufruf der Routine muß der eigene Eingabedialog geschlossen werden.
Mit `void write_ap_info(OBJECT *tree,int object,ALLG_PAR *ap);` kann man in ein (Button- oder String-) Objekt einen Infotext mit Bildnamen, Größe und Bilderzahl schreiben lassen. `ad` ist wieder der Parameter aus dem Eingaberoutinen-Aufruf, `tree` und `object` bestimmen das Objekt in dem der Infotext stehen soll.

Mit den Routinen `set_ipol_pbutton`, `do_ipol_pbutton` und `get_ipol_pbutton`, kann man die Interpolations-Popups verwalten. `void set_ipol_pbutton(OBJECT *tree,int object,int value,int mode);` dient dazu, im Dialog `tree` den Button `object` auf den Wert `value` zu setzen. Ist `mode` 0, werden die Interpolationstypen '–','0' und '+', bei 1 '–', '0', '+' und 'l', bei 2 '–', '0', '+' und 'r' angeboten.

`void do_ipol_pbutton(OBJECT *tree,int object,int mode,int cycle,int *ok);` dient zum Bedienen des Popups. `tree`, `object` und `mode` entspricht den Parametern von `set_ipol_pbutton`, `cycle` gibt an, ob das Popupmenü zur Auswahl dargestellt werden soll (`cycle` 0) oder ob der nächste Eintrage des Popupmenüs verwendet werden soll (`cycle` 1). `ok` gibt an, ob der Dialog neu gezeichnet werden muß (`ok` gleich 0), weil nicht genug Speicher zum Retten des Popup-Hintergrundes zur Verfügung stand.
Mit `int get_ipol_pbutton(OBJECT *tree,int object,int mode);` kann man schließlich den Zustand eines Interpolations-Popups auslesen: die Parameter entsprechen denen von `set_ipol_pbutton`, die Nummer des ausgewählten Eintrages wird zurückgegeben.

Neben den Interpolationspopups gibt es in den Eingabedialogen noch die Popups zum Löschen der Eingabe, Übernahme von Daten usw.
Diese werden ebenfalls vom Hauptprogramm verwaltet und können mit der Routine `int do_get_popup(OBJECT *tree,int object,int mode,int *ok);` aufgerufen werden. `tree` und `object` geben wieder den Button zum Popup-Aufruf an. Mit `mode` kann man wählen ob der Popup, der sich auf beide Parameterhälften bezieht, angezeigt werden soll (`mode` 0) oder der für die linke oder rechte Parameterhälfte (vgl. z.B. die Fractal-Routinen). Im letzten Fall ist `mode` ein Bitvektor für die verschiedenen Einträge, die bei gesetztem Bit aktiv sind (in jedem Fall erhält man hier einen Wert ungleich 0, da ja mindestens ein Eintrag aktiv sein muß, sonst ist der Popup-Aufruf sinnlos). Die Werte für die einzelnen Einträge sind in *COMMON.H* mit den Konstanten `_PLR__*` definiert (ungleich den Rückgabewerten `_PLR_*`!!!). `ok` ist das übliche Redraw-Flag für den aufrufenden Dialog.
Um das Seitenverhältnis der eingestellten Koordinaten anzuzeigen, hat man die Routine `int show_aratio(ALLG_PAR *ap,OBJECT *tree,int ob_xmin,int ob_xmax,int ob_ymin,int ob_ymax,int *ok);` (vgl. ebenfalls die Fractal-Routinen). Die Routine arbeitet nur mit Koordinaten im Minimum/Maximum (nicht Scale/Offset) Modus zusammen. `ap` sind die übergebenen Parameter, `tree` der Objektbaum des Dialoges, `ob_xmin` usw. die Objektnummer für den minimalen X-Wert usw. Aus diesen Objekten werden die jeweiligen Werte ausgelesen (es müssen natürlich Tedinfos sein), unter Umständen werden auch korrigierte Werte hineingeschrieben (15 Zeichen lang, 11 Nachkommastellen). `ok` ist ...(you know it).

Die den Routinen
`void conv_sc_mm(double scale,double offset,int delta,double *min,double *max);`
`void conv_mm_sc(double min,double max,int delta,double *scale,double *offset);`
kann man Koordinaten im Minimum/Maximum-Format ins Scale/Offset-Format wandeln lassen (jeweils für x und y getrennt). Die zugrundeliegenden Formeln nachzuschauen bin ich jetzt zu faul, deshalb die Routinen selbst:

```
static void conv_sc_mm(double scale,double offset,int delta,double *min,double *max) {
    *min=(-delta)/scale+offset;
    *max=(delta)/scale+offset;
}

static void conv_mm_sc(double min,double max,int delta,double *scale,double *offset) {
    *offset=(min+max)/2;
    *scale=delta/(max-*offset);
}
```

`delta` gibt irgendwie eine Skalierung an, wobei ich (don't remember why) bisher 320 horizontal und 200 vertikal angegeben habe.

Für den Fall, daß Zeiten von Millisekunden in Stunden, Minuten und Sekunden umgerechnet werden sollen gibt es `void conv_time(unsigned long ms,int *h,int *m,double *s);`. `ms` ist die Zeit in Millisekunden (ein `unsigned long` reicht für fast 20 Tage, sollte also hinreichend groß sein), in `h`, `m` und `s` werden die Stunden, Minuten und Sekunden zurückgegeben.

Die Routine `write_double` ermöglicht die Ausgabe von Fließkommazahlen ohne Nachkommanullen. In `void write_double(char *s,double z,int l,int p);` ist `s` der Zielstring, `z` die Zahl, `l` die gewünschte Länge der Ausgabe und `p` die Zahl der Nachkommastellen. Die Zahl wird prinzipiell *ohne* Exponent ausgegeben. Die Ausgabe erfolgt mit `sprintf(str,``%*.*lf``,l,p,z)`, wobei das Ergebnis nicht länger als 128 Zeichen werden darf, sonst gibt es vermutlich einen Absturz. Wird die Zahl länger die Ziellänge, so wird sie abgeschnitten, was zu falschen Angaben führen kann.

Mit der Routine `int get_object(int mode,int typ,GET_DATA *gd,char *info_str);` kann man ein Objekt oder einen Block (usw.) für die Übernahme von Daten aus bestehenden Objekten auswählen lassen. `mode` gibt an, welche Arten von Daten (Objekte, Blöcke und/oder Konstanten) ausgewählt werden dürfen. Die Flags sind in *COMMON.H* unter `_GET_*` definiert, und dienen auch als Rückgabewerte (bei Abbruch der Funktion wird 0 zurückgegeben). Mit `typ` kann man angeben, welchen Typ das Objekt, auch das, das dem Block / der Konstante zugrundeliegt, haben soll. Ist dies egal, so gibt man -1 an. `gd` ist eine Struktur, in der die Daten zurückgegeben werden und `info_str` ein String, der im Hilfs-Dialog während der Eingabe angezeigt wird.

Hat man ein Objekt via `get_object` zurückerhalten, so möchte man unter Umständen dessen Objekt-Rechteck ermitteln. Dazu bediene man sich der Routine `int get_obblk(CHS_OBJC *objc,R_REC *r);`, die im Erfolgsfall 1 zurückgibt und die Daten nach `r` schreibt.
Möchte man auf die Bilddaten aus einem Objekt zurückgreifen, so kann man dies mit `int get_std_planes(BITMAP *bitmap,MFDB *dest,X_DATA_P *mem,int h_flag,int planes);` (die Routine ist auch für Hardcopy-Routinen brauchbar, wenn man auf die Konvertierung der Bilder verzichtet). `bitmap` ist die Bitmap, in der die Daten gespeichert sind (bei Objekten: `objc.x.bitmap`) `dest` ein `MFDB` für das (entpackte) Zielbild, in dem die Adresse `fd_addr` *nicht* gesetzt ist. Das Zielbild wird nämlich in den Speicherblock `mem` geschrieben, so daß man die Adresse noch auf `mem->data` setzen muß (anschließend keine Speicherverwaltungszugriffe; wenn man die Adresse nochmal braucht ist sie neu zu setzen). `h_flag` gibt an, ob das Bild mit der bestehenden Höhe (also bei teilweiser Berechnung nur teilweise) oder mit der vollständigen Höhe geholt werden soll. Im ersten Fall wird bei völlig leeren Bildern `mem` auf -1 gesetzt. `planes` ist die Zahl der Bitebenen, aus denen das Zielbild bestehen soll.
Die Routine `int xsel_load(char *ext,char *titel,X_DATA_P *mem,long *off,int *ok);` dient dem Laden von Daten aus Parameterdateien. `ext` ist das Datei-Extension der der Parameterdateien, `title` gibt den Titel für den Dialog an. `mem` zeigt auf den Speicherblock für die Datei (der vom aufrufenden Programm freizugeben ist, der aber auch schon eine Datei beinhalten kann). `off` gibt – im Erfolgsfall – den Offset zu den ausgewählten Daten an, wobei der Offset relativ zu `mem->data` zu verstehen ist. `ok` ist ...

7.7 Unterstüzung für gedrehte Blöcke

Um gedrehte Blöcke zu realisieren muß man – in zeilenweise berechneten Bildern – lediglich bei jedem Schritt nach rechts (innerhalb einer Zeile) einen Offset zum x-Wert und einen Offset zum y-Wert dazurechnen. Genauso muß hat man beim Sprung in die nächste Zeile zwei solche Offsets (die relativ zum *ersten* Punkt in der Zeile drüber bestimmt werden). Diese Offsets kann man mit
`void init_rrec(R_REC_DXY *out,R_REC *in,int width,int height,int xheight,int flag);`
berechnen lassen. `flag` gibt an, ob man von links oben (0) oder links unten (1) anfangen will, `xheight` gibt an, wieviele Zeilen übersprungen werden sollen (i.a. weil sie schon berechnet wurden). `in` legt das Koordinatenrechteck fest, `width` und `height` die Bildgröße (in Punkten). In `out` werden dann die Offsets zurückgegeben, `x0` und `y0` sind die Anfangswerte, die Offsets `xdx` und `xdy` dienen dem Schritt nach rechts, die Offsets `ydx` und `ydy` dem Zeilenvorschub.

7.8 Zufallszahlen-Generator

Mit den Funktionen `long irand(long dummy);` und `double xrand(long dummy);` lassen sich Zufallszahlen erzeugen. `irand` liefert Zahlen zwischen 0 und 10^9, `xrand` zwischen 0 und 1. `dummy` ermöglicht die Initialisierung des Zufallszahlengenerators: ist `dummy` -1, so wird zufällig initialisiert (mit der xBios-Routine), ist `dummy` >0, so wird mit `dummy` initialisiert (nach gleicher Initialisierung spuckt der Generator natürlich auch gleiche Zufallszahlen aus). Setzt man `dummy` gleich 0, so wird nicht initialisiert (Normalfall!). Erfolgt keine explizite Initialisierung, so wird beim allerersten Aufruf (von welcher Routine auch immer) zufällig initialisiert.

7.9 Interpolation

Mit den Routinen `ipol` und `xipol` kann für die Erzeugung von Bildfolgen interpoliert werden.
`double ipol(double anf,double end,int x,int cnt,int flag);`
Mit dieser Routine wird zwischen `anf` und `end` interpoliert. `x` ist die Nummer des aktuellen Wertes (0 bis `cnt`-1), `cnt` die Gesammtzahl der Werte die interpoliert werden sollen. `flag` gibt den interpolationstyp an, bei 0 wird degressiv, bei 1 linear und bei 2 progressiv interpoliert. 3 bedeutet die Berechnung eines zufälligen Wertes zwischen `anf` und `end`
Die Routine `xipol`
`void xipol(double anf1,double end1,double anf2,double end2,double *anf,double *end,int act,int n);`
dient der Interpolation zwischen Rechtecken und zwar so, daß die Fläche linear wächst (wenn ich mich recht entsinne). Angegeben werden mit `anf1`, `end1`, `anf2` und `end2` die minimalen bzw. maximalen x- bzw. y-Werte des Rechtecks (man braucht zwei Aufrufe für ein ganzes Rechteck). `act` und `n` sind die Nummer des aktuellen Wertes (0 bis `n`-1) und die Gesammtzahl der Werte die interpoliert werden sollen. In `anf` und `end` werden die interpolierten minimalen bzw. maximalen Werte zurückgegeben.

7.10 Tastatur-Abfrage

Mit der Funktion `check_key` kann die Tastatur während der Berechnung oder bei anderen abbrechbaren Aktionen (z.B. Hardcopy) abgefragt werden.

`int check_key(MAKE_INFO *mi);`
Rückgabewert ist 1, wenn die `Esc`-Taste gedrückt wurde (oder wenn im Berechnungsinfo `Abbruch` gewählt wurde), 0 sonst (die Funktion erlaubt dem Programm gleichzeitig das Reagieren auf GEM-AES-Messages (z.B. Fensterredraw) und sollte schon deshalb gelegentlich aufgerufen werden).
`mi` ist speziell für die Tastaturabfrage bei Berechnungen vorgesehen. `mi` ist ein Zeiger auf eine Info-Struktur über den Berechnungsstand des Bildes. Ist `mi` 0l, so wird nur auf `Esc` geprüft, andernfalls kann mit `Space` das Berechnungs-Info aufgerufen werden.

```
typedef struct
{
    int         width;      /* breite des bildes */
    int         height;     /* höhe des berechneten bildes */
    int         xheight;    /* bereits berechnete zeilen
                               (-1, falls nicht zeilenweise berechnet wird) */
    long        iter;       /* zahl der bereits berechneten iterationen */
    long        iter_ges;   /* -1 -> unknown */
    long        time;
    int         time_iter;  /* -1 -> unknown */
    long        pkte;       /* -1 -> xheight*width */
    long        pkte_ges;   /* -1 -> height*width */

    int         key;
    int         state;
} MAKE_INFO;
```

In `mi.key` und `mi.state` werden Tastaturcode und Sondertastenstatus (wie von `evnt_multi` geliefert) zurückgegeben.

7.11 Routinen zum Zeichnen

Beim Zeichnen von Fractalen genügt es natürlich nicht, die Fractale mit VDI-Routinen auf den Bildschirm auszugeben, sie müssen auch in die interne Bitmap, die dann gespeichert wird geschrieben werden (sonst wären ja auch übergroße Bilder unmöglich).
CHAOS*ultd* stellt deshalb einige Routinen zum Zeichnen von Punkten und Linien zur Verfügung. Dabei wird so vorgegangen, daß die Zeichenroutine eine Punktsetz-Routine anfordert, die dabei gleich auf die angegebene Bitmap initialisiert wird. Diese Routine gibt (falls die Berechnung nicht verdeckt erfolgt) auch auf den Bildschirm aus.
ACHTUNG! nach dem Initialisieren der Punktsetz-Routine darf die Zeichenroutine *keine* Speicherverwaltungsaufrufe mehr tätigen!
Es stehen grundsätzlich zwei Typen von Punktsetz-Routinen zur Verfügung, die mit den Funktionen

```
_draw_pkt init_draw(BITMAP *bitmap,int mode,int puffer);
_xdraw_pkt init_xdraw(BITMAP *bitmap,int x,int y,int mode,int puffer);
```

initialisiert werden.
`_draw_pkt` und `_xdraw_pkt` sind Zeiger auf die Zeichenroutinen:
`typedef void (*_draw_pkt)(int x,int y,int col); typedef void (*_xdraw_pkt)(int col);`
Der Unterschied zwischen den beiden Typen ist, daß im ersten Fall beim Zeichnen des Punktes dessen Koordinaten angegeben werden, im zweiten Fall gibt man die Koordinaten beim Initialisieren an, es wird dann immer ein Punkt nach dem anderen (und zwar zeilenweise) ausgegeben, so daß die Angabe von Koordinaten beim eigentlichen Zeichnen nicht mehr nötig ist. `col` ist in jedem Fall die Farbe, die der Punkt erhalten soll.
So jetzt bleibt noch `mode` und `puffer` in der Init-Routinen zu erklären (`bitmap` ist natürlich die Bitmap in die gezeichnet werden soll). `mode` ist ein Flag, das zwischen dem reinen Setzen von Punkten (`mode`=0) und dem Setzen und Löschen (`mode`=1) unterscheidet. Übermalt man keine Punkte, dann genügt das reine Setzen (was natürlich schneller geht).
`puffer` gibt an, ob – für die Ausgabe auf den Bildschirm – ein Punktpuffer verwendet werden darf (was die Ausgabe sehr beschleunigt). Dies ist momentan nur bei monochromen Monitor oder Bild möglich. `puffer` gibt die Zahl der maximal zu puffernden Punkte (genaugenommen Pixel) an (vorausgesetzt der Puffer ist groß genug), -1 steht für die maximale Puffergröße und 0 bedeutet die Pixel stets unmittelbar auszugeben.
Im Zusammenhang mit dem Ausgabepuffer gibt es noch die Funktionen `void clr_pktpuffer(void);` und `void flush_pktpuffer(void);`, die erste löscht den Punktpuffer (ohne ihn auszugeben), die zweite gibt ihn aus.

Die Routine `void (*clr_screen)(void);` löscht den Bildschirm, nicht die interne Bitmap. (Man darf nie direkt auf den Bildschirm zugreifen, ihn also auch nicht direkt löschen, weil die Zeichenroutine ja nicht wissen kann, ob die Berechnung verdeckt oder nicht erfolgt.

Zum Zeichnen von Linien gibt es die Routine
`void draw_line(int X1, int Y1, int X2, int Y2, int color,_draw_pkt draw);`
der eine Punktsetz-Routine übergeben werden muß.

Analog zu den Punktsetz-Routinen kann man auch eine Punkt-Abfrageroutine initialisieren, mit der Routine `_get_pkt` `(*init_getpkt)(BITMAP *bitmap);`; `typedef int (*_get_pkt)(int x,int y);`. (Hier gilt der gleiche Hinweis zur Speicherverwaltung; zwischen Initialisierung und Aufruf der `_get_pkt`-Routine darf nicht auf die Speicherverwaltung zugegriffen werden.

7.12 weitere Hilfroutinen zur Berechnung

`long stackavail(void);`
Diese Routine gibt die Größe des *freien* Stacks zurück, so daß man bei rekursiven Routinen vor einem Stack-Überläuf rechtzeitig abbrechen kann.
`long gettime(void);`
liefert den Stand des 200 Hertz-Zählers

7.13 Routinen zum Ändern des Objektstatus

Mit `void set_unready(CHS_OBJC *objc);` und `void set_ready(CHS_OBJC *objc)` kann man Objekte für unfertig oder für fertig erklären. Der direkte Zugriff auf den Objekt-Status ist *nicht* erlaubt!
Objekte dürfen *nicht* für fertig erklärt werden, wenn sie nur teilweise berechnet sind und nur der berechnete Teil gespeichert wurde (vgl. `last_flag`).

7.14 Speicherverwaltung

CHAOS*ultd* reserviert sich am Anfang einen großen Speicherblock und teilt diesen dann intern weiter auf. Um eine Reorganisation der Speicherblöcke (garbage collection) zu ermöglichen, darf auf jeden Speicherblock nur *ein* Zeiger, dessen Adresse die Speicherverwaltung kennt und dessen Inhalt von der Speicherverwaltung geändert werden kann, zeigen. Alle anderen Referenzen auf den Speicherblock müssen als Zeiger auf diesen Zeiger auf den Speicherblock verwaltet werden (es seie denn, man weiß, daß zwischen dem Kopieren des Zeigers auf den Speicherblock und dem Benutzen der Kopie *keine* Zugriffe auf die Speicherverwaltung stattfinden). Speicher sollte in CHAOS*ultd* immer von der internen Speicherverwaltung angefordert werden, der Zugriff auf den Systemspeicher (via GEMDOS) sollte unterbleiben. Speicherblöcke umfassen neben dem Dateninhalt noch den Rückpointer und die Länge des Blockes:

```
typedef struct
{
    long    len;
    long    *mother;
    char    data[];
} X_DATA;
typedef X_DATA *X_DATA_P;
```

Hat man einen Zeiger auf einen Speicherblock `X_DATA_P p;` dann kann man auf die eigentlichen Daten mit `p->data` zurückgreifen. `p->len` und `p->mother` darf man nicht verändern!! `p->len` ist *nicht* die Länge des Speichers, sondern hier werden die 8 Byte Verwaltungsinformation noch mitgezählt. Außerdem wird die Länge des Speicherblocks ganzzahlig aufgerundet.
Die Speicherverwaltung umfasst die folgenden Funktionen:
`int (*x_malloc)(long len,X_DATA_P *mother);`
Speicher reservieren; `len` ist die Länge des Speichers, `mother` ein Zeiger auf den Zeiger auf den Block.
`int (*x_mshrink)(X_DATA_P *data,long new_len);`
Speicher verkürzen; `new_len` ist die neue Länge, `data` ein Zeiger auf den Zeiger auf den Block
`int (*x_mfree)(X_DATA_P *data);`
Speicherblock freigeben; `data` ist ein Zeiger auf den Zeiger auf den Block
`int (*x_ch_mem)(X_DATA_P *new,X_DATA_P *old);`
Speicherblock von einem Zeiger auf einen anderen Übertragen ('verschiebe' Speicherblock); `new` ist der neue Zeiger, `old` der alte Zeiger, der dann auf 0l gesetzt wird, also nicht mehr gültig ist.
Alle Routinen liefern im Erfolgsfall 0 (`RET_OK`) zurück, sonst eine Fehlermeldung. `x_malloc` setzt im Fehlerfall `mother` auf 0l.

7.15 32-Bit Festkomma-Arithmetik

CHAOS*ultd* stellt zwei Typen von Rechenroutinen für Festkommazahlen zur Verfügung, die einen Rechnen nur mit 8 Bit Vorkomma- und 24 Bit Nachkomma-Anteil, bei den anderen kann man die Zahl der Nachkommabits explizit angeben.
Zum ersten Typ gehören:
`long mult32(long a,long b)` und
`long quad32(long a)`
Eine Fehlerflag gibt es nicht, falls ein Überlauf stattfindet, wird das Ergebnis auf den Maximalwert gesetzt.
Beim zweiten Typ hat man:

```
long multiply(long a,long b,int n);
long divide(long a,long b,int n);
long add(long a,long b);
int overflow;
```

`n` gibt die Zahl der Nachkommabits an, die zwischen 0 und 32 liegen kann (0 und 32 selbst dürfte nicht funktionieren); für `add` spielt `n` keine Rolle (falls man sich nicht für Überläufe interessiert kann man auch direkt `a+b` statt `add(a,b)` verwenden). `overflow` muß vor der Rechnung gelöscht werden und wird (auf 1) gesetzt falls ein Überlauf auftritt; bei Divisionen wird der Überlauf nicht immer erkannt.
Mit der Routine `sincos68(long phi,long *s,long *c)` kann man den Sinus und Cosinus eines Winkels (natürlich in Radiant) berechnen lassen. Hier ist die Zahl der Nachkommabits fest auf 16 eingestellt.

7.16 Einstellungen

Mit `CHS_INST *inst;` steht ein Zeiger auf die Einstellungen von CHAOS*ultd* zur Verfügung. Die `CHS_INST`-Struktur ist in *COMMON.H* definiert.

7.17 Unterstüzung für Hardcopy-Routinen

Mit den Routinen `first_ob` und `next_ob` können die Objekte ermittelt werden, für die die Hardcopy aufgerufen wurde.

```
CHS_OBJC *first_ob(OB_MODE mode,int ks,int dm,void *pw,int objc);
CHS_OBJC *next_ob)(void);
```

`mode` setzt man auf `GET_PICS`; `ks`, `dm`, `pw` und `objc` bekommt man beim Aufruf der Hardcopy vom Hauptprogramm übergeben. Zurückgegeben wird jeweils ein Zeiger auf die `CHS_OBJC`-Struktur des Objektes, sind alle Objekte durch, so wird 0l zurückgegeben.

Ausgabefunktionen:
Mit `int check_printer(void);` wird abgefragt, ob der Drucker bereit ist. (Rückgabewert `RET_OK`, sonst `RET_ERROR`.) Verwendet man die CHAOS*ultd*-Routinen zum Ausgeben der Daten (was man tun sollte) so *muß* diese Funktion vor dem Drucken mindestens einmal aufgerufen werden. Außerdem ist es natürlich nicht sinnvoll auf einen nicht bereiten Drucker auszugeben, im Fall der Rückgabe von `RET_ERROR` ist die Hardcopy-Routine also abzubrechen.
Zur eigentlichen Ausgabe von Daten dienen die Routinen `void pr_byte(char byte);` und `void pr_bytes(char *byte,long len);` deren erste ein Byte ausgibt, die zweite gleich eine ganze Anzahl von Bytes.

Zur Ausgabe von Bild-Infos unter die Hardcopy dient `int get_dr_info(CHS_OBJC *objc,DR_INFO *dr);`. Im Erfolgsfall (Rückgabewert `RET_OK`) wird in die Stringstruktur `dr` die Information zum Objekt `objc` geschrieben. Diese müssen von der Hardcopyroutine noch ausgedruckt werden. Die Strings enthalten kein CR LF. Ist der Rückgabewert `RET_ERROR` (oder sonst eine Fehlermeldung, bisher nicht möglich), so wurde die Stringstruktur nicht ausgefüllt (etwa weil die zugehörige Routine die Funktion nicht unterstüzt).

Zur Unterstützung des Konvertierens vor dem Ausdrucken gibt es drei Funktionen:

```
int get_konv_info(KONV_INFO *ki,char *info,char *titel,int *ok);
void make_konv_str(KONV_INFO *ki,int planes,char *str1,char *str2); int get_hcopy_planes(KONV_INFO
*ki,CHS_OBJC *ob,MFDB *dest,X_DATA_P *mem,X_DATA_P *c_tab,int max_planes);
```

Prinzipiell wird die eingestellte Konversion durch eine `KONV_INFO`-Struktur beschrieben, deren Details hier aber keine Rolle spielen. Mit der Routine `get_konv_info` kann man die Einstellung für die `KONV_INFO`-Struktur (im folgenden KI) aufrufen. Dazu übergibt man einen Zeiger auf die Struktur, eine Info und eine Titelzeile die im Dialog angezeigt werden. `ok` ist das Redraw-Flag, das man beim Aufruf von Dialogen aus Dialogen zum korrekten Redraw benötigt.
Die Routine `make_konv_str` liefert zu einer KI zwei Info-Strings, die man im Einstelldialog der Hardcopy-Routine ausgeben kann, damit klar ist wie vor dem Drucken konvertiert wird. Die Routine benötigt zusätzlich einen Parameter `planes`, den man stets auf 2 setzen sollte.

Mit der Routine get_hcopy_planes kann man schließlich ein konvertiertes Bild anfordern. Anzugeben ist die KI, das Objekt (ob), ein Memory Form Definition Block dest, eine Speicherverwaltungszeiger mem (vgl. die Routine get_std_planes) ein weiterer Speicherverwaltungszeiger c_tab für die Farbtabelle (die stets kopiert wird) und die Angabe max_planes, wieviele Bildebenen das Zielbild maximal umfassen darf (wenn man keine Farbhardcopy schreibt 1). Das Bild wird stets vollständig d.h. inclusive nicht berechneter Teile (die dann halt leer sind) geholt.

siehe auch 'Info-Dialoge', 'Tastatur-Abfrage' und 'Daten aus Objekten holen'.

Teil II
Dateiformate

Dieser Teil wurde der Dokumentation zu Version 5.0x entnommen. Ich bitte mögliche Unterschiede im Layout zu entschuldigen.

An sich sollte es selbstverständlich sein, daß eine Programmdokumentation auch eine Beschreibung der verwendeten Dateiformate enthält. Wie so vieles ist es das nicht, was mich aber nicht hindern soll, hier mit gutem Beispiel voranzugehen.

1 CHAOS*ultd*-Bilder

CHAOS*ultd* verwendet für Bilder ein eigenes Format[5]
Eine Bilddatei besteht aus einem Header, der Farbtabelle, den Parametern, zusätzlichen Parametern und den Bilddaten, wobei letztere immer gepackt werden, auch dann, wenn sie dadurch länger werden[6].
Der Header sieht (als C-Struktur) folgendermaßen aus:

```
typedef struct
{
/* id's */
    char chaos_id[8]; /* 'CHSultd5' */
    char frac_id[8];
        /* größe, planes und colors */
    unsigned int size_x;
    unsigned int size_y;
    unsigned int planes;
    unsigned int colors;
        /* zus. parameter */
    int lastline;
    int last_flag;
    int xor_offset;
        /* länge der folgenden daten */
    long par_len;
    long xpar_len;
    long pic_len;
    /* coltab in long */
    /* parameter */
    /* zus. parameter */
    /* bilddaten */
} FRAC_HEAD;
```

`chaos_id` ist eine Kennung, die die Zeichen `CHSultd5` beinhalten muß. `frac_id` ist eine weitere Kennung, die die Berechnungsroutine, die für das Bild verantwortlich ist, kennzeichnet.
`size_x`, `size_y`, `planes` und `colors` legen die Bildgröße, die Anzahl der Bitebenen und die Zahl der Farben fest. Die Bildgröße ist – unabhängig davon, ob und wieweit das Bild schon berechnet wurde – die Größe des fertig berechneten Bildes.
`lastline` gibt an, ob das Bild fertig ist oder nicht. Ist `lastline` 0, so gibt es gar keine Bilddaten, -1 und jeder andere Wert kleiner 0 bedeutet, daß das Bild fertig berechnet ist. Ist `lastline` positiv, so hängt die Bedeutung von `lastline` davon ab, ob das unterste Bit in `last_flag` gesetzt ist[7]. Ist dies der Fall, so gibt `lastline` Ist dieses Bit gesetzt, so gibt `lastline` die Zahl der vorhandenen Zeilen an. Andernfalls ist für `lastline`$\neq 0$ stets das gesammte Bild gespeichert.
`xor_offset` ist für das Packen der Daten von Bedeutung und wird gleich erklärt; `par_len`, `xpar_len` und `pic_len` gibt die Länge der drei Datenblöcke an. Die Parameter und zus. Parameter sind natürlich Bildtypabhängig, bleiben also die Bilddaten.

[5] Wozu Standartformate verwenden, wenn man auch eigene definieren kann?
Aber im Ernst: die verschiedenen Bildgrößen und die Verwaltung der Parameter ließen mir ein spezielles Format sinnvoll erscheinen und wenn man sowieso ein eigenes Format verwendet, ist es auch schon egal, wie speziell dieses Format dann ist (denkbar wäre höchstens noch ein GEM-Image-Format gewesen, mit einem erweiterten Header; vielleicht führe ich das mal als alternatives Format ein).
[6] das ist kein Scherz, das kann wirklich passieren, die Daten werden aber nicht nennenswert länger
[7] die anderen Bits sind reserviert

Im Farbmodus werden zunächst die Bitplanes, die im Screen-Format ja wortweise hintereinanderstehen getrennt, d.h. es kommt erst die ganze Bitplane 0, dann 1 usw. (man hat also gewissermaßen ein GEM-Standartformat, wobei die Größe des Blockes durch `size_x` und `size_y` festgelegt ist).
Dann werden die Bilddaten von hinten mit dem `xor_offset` mit sich selbst via xor verknüpft. Der Offset entspricht im Allgemeinen einer oder mehreren Bildschirmzeilen. Dadurch entsteht ein Bild, dessen Zeilen (außer der ersten) immer nur die Differenz zur darüberliegenden Zeile beinhalten und das deshalb i.a. besser zu packen ist. Ist `xor_offset` 0 so entfällt die Xor-Verknüpfung. Um die Xor-Verknüpfung beim Laden rückgängig zu machen muß man die Daten erneut via xor miteinander verknüpfen, diesmal allerdings von vorne.
Anschließend werden die Bilddaten im Stad-Format gepackt und gespeichert.
Das heißt die gepackten Daten beginnen mit den drei Bytes Kennbyte, Packbyte und Spezialbyte, anschließend folgen die gepackten Graphikdaten, wobei folgende Kodierungen Anwendung finden:
Das Kennbyte und das nachfolgende Byte n bedeuten, daß das Packbyte n+1 mal zusammengefaßt wurde.
Das Spezialbyte und die beiden nachfolgenden Bytes a und n bedeuten, daß das Byte a n+1 mal zusammengefaßt wurde.
Alle anderen Bytes müssen so wie sie in der Datei stehen in das Bild übernommen werden.

2 Filme

Film-Dateien haben sich gegenüber FRACTAL Version 4.1 (und 4.3) nicht geändert.
Sie bestehen aus dem Header 'film', der (als 2 Byte Integer abgelegten) Anzahl verschiedener Bilder sowie der Anzahl der Bilder im Film. Es folgen die Anzeigeoptionen (eine SHOW_OPTS-Struktur), daran schließen sich die Informationen über die im Film enthaltenen Bilder an. Und zwar für jedes Bild Dateipfad, Name und Dateiextension, jeweils als nullterminierter String (unter diesen Dateinamen werden die Bilder beim Einladen eines Filmes gesucht). Deshalb ist es auch nötig, daß die Bilder beim Speichern des Filmes gespeichert sind, FRACTAL könnte sonst höchstens raten, wohin man die Bilder abspeichern wird. Dateipfade können (seit FRACTAL V4.3) relativ sein, sie beginnen dann nicht mit einem Laufwerk. In diesem Fall ist der Pfad relativ zum Verzeichnis der Filmdatei zu verstehen.
Zuletzt folgt eine Liste mit der Reihenfolge der Bilder im Film, wobei sich die Nummern (1 Byte Integer) auf die Reihenfolge der Dateinamen beziehen. Die Liste darf nicht länger als 32000 Einträge lang sein.

3 Einstellungen

Die Einstellungsdateien will ich hier nicht im Detail erläutern, da sie sicher nicht sonderlich interessant sind.
Eventuell doch von Interesse ist allerdings der prinzipielle Aufbau dieser Dateien, die ja nicht nur die Voreinstellungen von CHAOS*ultd* selbst, sondern auch die der diversen Berechungsroutinen – letztere in nicht festgelegter Reihenfolge und womöglich wechselnder Anzahl.
Realisiert wird dies durch eine Block-Struktur. Jeder Block beginnt mit einem 12 Byte langen Header: die ersten 8 Byte erhalten eine Kennung (CHAOS*ultd* verwendet für seine Einstellungen `CHSultd5`, für die Berechnungsroutinen wird die gleiche Kennung verwendet wie in Bilddateien. Es folgt als Langwort (4 Byte) die Länge der folgenden Daten. Anschließend kommen die Einstellungsdaten sebler, deren Inhalt von den Berechnungsroutinen abhängt; die CHAOS*ultd*-Einstellungen bestehen aus einer `CHS_SET`-Struktur, die in der Headerdatei `XCOMMON.H` definiert ist.