Borland Graphics Interface for X Window System



Document Manual

山田 泰司[1]

[1] 東京電機大学 工学部 電子工学科 taiji@henon.d.dendai.ac.jp

目次

1 入門 XBGI **5**

1.1 hello.c のコンパイルと実行 5

1.1.1 xflush の利用 7

1.1.2 getch の利用 7

1.1.3 xbgiloop の利用 7

1.2 bgidemo.c の実行 7

2 オリジナル BGI との相違点 **9**

2.1 関数における相違点 9

2.2 グラフィックドライバとモード 9

2.3 補足ライブラリ 10

3 XBGI 拡張機能について **11**

3.1 X ウィンドウのための拡張機能 11

3.1.1 環境設定のためのいくつかの方法 11

3.1.1.1 関数による方法 11

3.1.1.2 primary design 指定子による方法 13

3.1.1.3 環境変数による方法 15

3.1.1.4 環境ファイルによる方法 15

3.1.1.5 環境設定の優先順位 15

3.1.2 イベント指向のサポート 16

3.1.2.1 XBGI イベントループ 16

3.1.2.2 イベントプロシージャの設定 16

3.2 XBGI システムのための拡張機能 18

3.2.1 XBGI の ID によるアクセス 20

3.2.2 オブジェクト xbgi 22

3.2.3 XBGI の設計思想 23

3.3 BGI グラフィックスのための拡張機能 24

3.3.1 倍精度グラフィックス 24

3.4 拡張機能を使用したコードの可搬性 26

4 内部仕様 **27**

4.1 XBGI 構築手続きに関して 27

4.1.1 XBGI 環境設定ファイルの利用 27

4.2 構築系関数 28

4.2.1 初期化関数 28

4.2.2 ドライバの登録 28

4.3 ツールキットとの融合プログラミング 29

4.4 XBGI のページ 29

4.4.1 ページの構造 29
4.4.2 アクティブページの特性 30
4.4.3 ビジュアルページの特性 31
4.5 ビジュアルクラスやカラーマップとの関係 34
4.5.1 デフォルトカラー設定 35
4.5.2 RGB 設定 35
4.6 フォントおよび国際化機能の利用 36

A XBGI 拡張関数 **41**

B BGI 標準関数 **43**

はじめに

使用機種をパーソナルコンピュータ (PC) からワークステーション (WS) へ移行した人や, 数値解析を目的に WS の C 言語を使用している人などは,「X ウィンドウシステム上でもっと気軽にグラフィックが扱えないものか.」,「計算結果をグラフで簡単に表示したい.」,「家の PC でつくってきたプログラムを研究室の WS でそのまま動かしたい.」などと感じることは多いでしょう. 本ライブラリ (XBGI) はパーソナルコンピュータで普及している Borland 社製の C/C++ 言語処理系に付属しているグラフィックライブラリ Borland Graphics Interface (BGI) をワークステーションの X ウィンドウシステムでサポートするものです. この XBGI は X ウィンドウに非常に手軽なグラフィックライブラリを提供するとともに, PC で普及率が高い BGI を C ソースレベルでシミュレートします[1].

本ドキュメントは Turbo C のマニュアルではありませんので, ここでは, X ウィンドウにおける XBGI 使用上の注意やオリジナル BGI と XBGI システムの相違点, XBGI で拡張された機能およびオリジナル BGI と XBGI の双方に対応したライブラリについて説明します. BGI の詳しい使用方法については, Borland 社のマニュアルまたは市販の Turbo C 関係の入門書を御覧下さい.

[1] Turbo C 言語処理系自体をサポートするわけではないので, PC 機種依存した C プログラムを WS で動作させることはできません.

第 1 章

入門 XBGI

1.1 hello.c のコンパイルと実行

まず, PC 上で動いていたプログラム hello.c をコンパイルして, X ウィンドウ上で実行させることを考えましょう.

サンプルプログラム hello.c

```
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
main()
{
  int gd = DETECT, gm;
  int cx, cy, l = 300, tri[8];

  initgraph( &gd, &gm, "" );
  cx = (getmaxx()+1)/2;
  cy = (getmaxy()+1)/2;
  setcolor( getbkcolor() );
  setlinestyle( SOLID_LINE, 0, THICK_WIDTH );
  settextjustify( CENTER_TEXT, CENTER_TEXT );
  settextstyle( TRIPLEX_FONT, HORIZ_DIR, 6 );
  setfillstyle( SOLID_FILL, LIGHTBLUE );
  bar( cx-l/2, cy-l/2, cx+l/2, cy+l/2 );
  setfillstyle( SOLID_FILL, LIGHTRED );
  fillellipse( cx, cy, l/2, l/2 );
  setfillstyle( SOLID_FILL, LIGHTGREEN );
  tri[0] = cx;
  tri[1] = cy - l/2;
  tri[2] = cx + l/4*3/1.732;
  tri[3] = cy + l/4;
  tri[4] = cx - l/4*3/1.732;
  tri[5] = cy + l/4;
  tri[6] = cx;
  tri[7] = cy - l/2;
  fillpoly( 3, tri );
  drawpoly( 4, tri );
  outtextxy( cx, cy, "Hello World !" );
  getch();
  closegraph();
}
```

これは画面に四角形, 三角形, 円とグラフィック文字を描くプログラムです (図 1.1). このサンプルプログラムのように graphics.h をインクルードして下さい. また, グラフィックドライバ指定子 gd には DETECT を代入し, 関数 initgraph を呼び出して下さい. コマンドラインからつぎのようにタイプすると, hello.c がコンパイルされ, 実行可能ファイル hello が生成されます.

```
% gcc -o hello hello.c -lXbgi -lXpm -lX11 -lm
```

また, XBGI のインストールディレクトリがコンパイラのデフォルトディレクトリでないときには, つぎのようなシェルスクリプトを用意してもよいでしょう (もちろん, Makefile でコンパイルしてもかまいません).

```
#!  /bin/csh -f
set IPATH = "-I/usr/local/xbgi/include"
set LPATH = "-L/usr/local/xbgi/lib"
set LINKS = "-lXbgi -lXpm -lX11 -lm"
gcc -o $1 $1.c $IPATH $LPATH $LINKS
```

ただし, ピックスマップライブラリがなく ENABLE_XPMLIB を定義しないで, XBGI ライブラリのインストールをした場合は, リンカーオプション -lXpm は必要ありません.

コンパイルがうまくいかない場合, コンパイラに XBGI インクルードパスが正しく指定されているか, リンカに XBGI ライブラリが正しく指定されているかを確認して下さい.

無事コンパイルが成功したら, このサンプルプログラムを実行してみましょう.

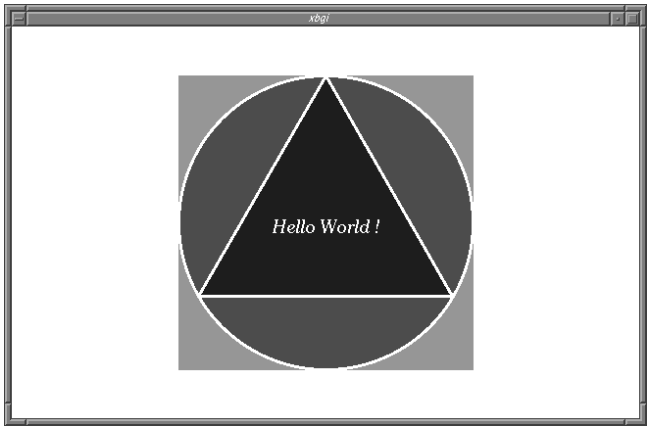


図 1.1: プログラム hello.c の実行画面

図 1.1 のような X のウィンドウが開けば, あなたは今後 WS 上の C 言語で BGI のグラフィックスを描くことが出来るようになります.

ではサンプルプログラム hello.c から, XBGI 使用上の注意を解説していきましょう. このプログラム最後の行にある closegraph の直前に, getch という関数が実行されています. これは何の目的で使われているのかというと, 描画が終了した時点で文字を入力する関数 getch を用いて, プロセスが終了してしまうのを防いでいるのです. もしこの getch がないとすると, XBGI はウィンドウを開いてグラフィックを描画しますが, 画面をながめる間も与えずウィンドウを消滅させてしまいます. タイミングが悪ければウィンドウを開くことなくプログラムが終了することもあります. これがオリジナル BGI と XBGI の違いです. PC ではプログラムの実行前も実行後も, 描画のためのディスプレイは存在しますが, X ウィンドウでは, しかるべき準備をすることでディスプレイが接続され, プロセスが終了することでディスプレイが切り離されます. ですから, XBGI ではプロセスが終了するのを抑止することが必要となります.

また, サンプルプログラムの getch を使うかわりに stdio.h の文字列入力関数 getchar を用いた場合にはどうなるでしょう. 今度は getchar が呼ばれた時点で終了しているはずの描画が描かれません. これは, X ウィンドウシステムがネットワーク指向であることに起因します. 簡単にいえば, ある程度の仕事量がバッファ (リクエストバッファ) にたまらなければ, それらのリクエストは実行されません. ですから, 描かれなかったリクエストを強制的にフラッシュする必要があります.

プロセスの終了を抑止し, 描画リクエストを強制的にフラッシュする代表的な方法を 3 種類紹介します.

1.1.1 xflush の利用

```
#include <stdio.h>
#include <graphics.h>
main()
{
  (中略)
  xflush();
  getchar();
  closegraph();
}
```

XBGI には描画リクエストをフラッシュする xflush が用意されています. この xflush を実行してから, 文字列を入力する関数 getchar をプロセス終了を抑止するために使います. このとき getchar はプログラムを実行したコンソールから入力を待ちます.

1.1.2 getch の利用

```
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
main()
{
  (中略)
  getch();
  closegraph();
}
```

先のサンプルプログラムで行われている方法です. 文字を入力する関数 getch を使います. getch は conio.h で宣言されています. このとき getch はウィンドウから入力を待ちます. つまり, この getch は XBGI が用意した関数であり, 内部で xflush を済ませています.

1.1.3 xbgiloop の利用

```
#include <stdio.h>
#include <graphics.h>
main()
{
  (中略)
  xbgiloop();
  closegraph();
}
```

XBGI 専用のイベントループである xbgiloop という関数を使います. この関数にはいろいろと便利な機能が用意されており, 'Q' キーで終了, マウスボタン 2 でメニューを表示します. このメニューの中で " グラフィック画面のファイルへの保存 " などが用意されています. xbgiloop の詳細については §3.1.2 を御覧下さい.

1.2 bgidemo.c の実行

Borland 社製の C 言語処理系には, BGI のサンプルプログラム bgidemo.c が付属しています. XBGI システムでは, このプログラムをほとんど無変更でコンパイルすることができました[1]. ただし, bgidemo.c を動かすために補足的にサポートしたヘッダファイル (conio.h や dos.h) では, 本来 Turbo C に備わっている機能のすべてをサポートしたわけではありませんので注意して下さい.

[1] bgidemo.c で使用されている関数 itoa (整数から文字へ変換する関数) は ANSI 標準関数ではないので, bgidemo.c の中で, sprintf(str, "%d", val) で代用しています.

第 2 章

オリジナル BGI との相違点

XBGI では, BGI が供給しているグラフィックス機能のほとんどをサポートしていますが, すべての機能を完全にシミュレートしているわけではありません. 本章ではオリジナル BGI と XBGI との若干の相違点について記載します. それらの違いは通常の使用にはほとんど関係ありませんが, ユーザーによってはこの章を一読しておく必要性はあるかもしれません. そして, XBGI はオリジナル BGI の完全なシミュレートというよりも, PC プログラミングからウィンドウプログラミングへのゆるやかな移行, そのためのインターフェースとしての役割を担うようなコーディングを心がけています. この点について何か良いアイデアがおありでしたら御教示くだされば幸いです.

2.1 関数における相違点

提供されている関数のなかで, オリジナル BGI を完全にシミュレートしていない関数を表 2.1にあげます. それらは, 未サポート関数, リザーブド関数, 機能が異なる関数, 要注意関数に分類されます.

未サポート関数は, 今後もサポートする予定のない関数です. これらの関数が実行されると標準エラー出力に, メッセージ `"Not Supported"` を通知します.

リザーブド関数は, 現段階でサポートしてない関数です. これらの関数は, どのようにコーディングすべきか考察中の関数なので, 将来サポートされる可能性があります. もし, これらの関数が実行されると, 標準エラー出力にメッセージ `"Reserved Future"` を通知します.

機能が異なる関数は, オリジナル BGI と若干機能が違う関数です. `setbkcolor` は現在, 動的に背景色を変更することができず, `cleardevice` 命令が実行されてはじめて背景色が変更されます. また `setfillpattern` では, ユーザーフィルパターンがオリジナル BGI に対して左右反転します. これは, X に用意されている `bitmap` コマンドを使って, ユーザーフィルパターン (8x8) を作製できるような仕様になっているからです.

要注意関数は, 利用状況によっては止まってしまう関数です. `getpixel` と `getimage` は内部で `XGetImage` を利用しているので, XBGI ウィンドウがアンマップ状態 (主にアイコン状態) の時に実行されると, X のエラーメッセージを発行して完全に停止します.

2.2 グラフィックドライバとモード

オリジナル BGI では, グラフィック画面を初期化する `initgraph(&gd, &gm, pd)` の引数である `gd` と `gm` にグラフィックドライバとそれに対応したモードを指定します. また, `gd` に自動検出 `DETECT` を指定した場合には, そのマシン最高のグラフィック能力が検出されます. 一方 XBGI では, そのドライバとモードがオリジナル BGI と異なります. 現在, XBGI におけるドライバは `DETECT` で `BGI` になります. 他のグラフィックドライバは Version 2 の段階ではサポートされていません. また, XBGI におけるグラフィックモードは自動検出において `BGICOLOR` になります. しかし, ユーザーが使用している WS のディスプレイにその能力がない場合には, XBGI はそれを自動的に検出して使用可能なモードへと変更します. `BGICOLOR` 以外には `BGIMONO` や `BGIGRAY` などがあります.

表 2.1: オリジナルに相違な関数

関数	機能
未サポート関数	
restorecrtmode	画面を初期化前の状態にリストア
setgraphbufsize	バッファ容量の設定
_graphgetmem	グラフィックバッファ容量の取得
_graphfreemem	グラフィックバッファ容量の解放
リザーブド関数	
installuserdriver	ドライバの登録
installuserfont	フォントの登録
setusercharsize	フォントサイズの設定
機能が異なる関数	
setbkcolor	背景色の設定
setfillpattern	ユーザーフィルパターンの設定
要注意関数	
getpixel	(X,Y) 位置のピクセルカラー取得
getimage	指定領域のビットイメージ取得

2.3 補足ライブラリ

include ディレクトリに conio.h と dos.h が存在しますが, サポートしているのは getch と kbhit および delay だけです. 将来, 他の機能がサポートされる可能性があります.

表 2.2: 補足ライブラリ

関数	機能
conio.h で宣言されている機能	
int kbhit()	キーボード入力があったときに真
int getch()	キーボードから入力した文字を返す
dos.h で宣言されている機能	
delay(ms)	プログラム実行を ms ミリ秒止める

第 3 章

XBGI 拡張機能について

この章では, XBGI 製作者が独自に拡張した機能について解説します. ここでは, 拡張機能を

1. X ウィンドウのための拡張機能 (§3.1)
2. XBGI システムのための拡張機能 (§3.2)
3. BGI グラフィックスのための拡張機能 (§3.3)

の 3 つに分類し, 順に解説していきます.

3.1 X ウィンドウのための拡張機能

X ウィンドウのための拡張機能は,

1. グラフィック環境のための拡張機能 (§3.1.1)
2. イベント処理のための拡張機能 (§3.1.2)

に分けられます.

3.1.1 環境設定のためのいくつかの方法

本節では, XBGI のグラフィック環境に関するさまざまな設定 (サイズ, タイトルおよびアイコン名, ページ, カラー, バッキングストア機能) を設定するための一連の手法について解説します.

3.1.1.1 関数による方法

X ウィンドウでは, さまざまなサイズのウィンドウが開けるため, オリジナル BGI と違ってグラフィック画面の大きさが固定されていません. また, X がサポートしているビジュアルクラス (モノクロ, グレイスケール, カラーなど WS のディスプレイ表示能力のクラス) のなかでは, 非常にたくさんの色数を扱うことが可能なクラスがあります. また, 一般的なウィンドウシステムでは, アプリケーションウィンドウが他のウィンドウの上になったときなどに必要になる, 画面の再描画は, ウィンドウシステムがそれを行うのではなく, 各々のアプリケーションに任されています. しかし, X ウィンドウシステムでは画面の再描画を, ウィンドウシステムに依頼する方法 (バッキングストア属性) があります.

XBGI では, これらさまざまな機能を扱うための拡張関数が用意されています (表 3.1).

これらの関数は, `initgraph` によって開かれたウィンドウに対して作用しますので, `initgraph` でウィンドウを構築するまえに実行してはなりません. これらの関数を利用したサンプルプログラム `colors.c` を示します.

サンプルプログラム `colors.c`

```
#include <conio.h>
#include <stdio.h>
#include <graphics.h>
int WIDTH, HEIGHT, NCOLOR;
bars()
{
```

表 3.1: 拡張関数

関数	機能
setmaxx(int x)	X 軸最大値の指定
setmaxy(int y)	Y 軸最大値の指定
getsizexy(int *width, int *height)	画面サイズの取得
setsizexy(int width, int height)	画面サイズの指定
getgeometry(int *x, int *y, int *width, int *height)	画面ジオメトリの取得
setgeometry(int x, int y, int width, int height)	画面ジオメトリの指定
setmaxcolor(int n)	使用可能な色の最大値の設定
settitle(char *name)	ウィンドウ名の設定
seticonname(char *name)	アイコン名の設定
seticon(char *filename)	アイコンファイルの設定
setbackingstore(int flag)	バッキングストアの設定

```
  int i, j;

  for ( i=0; i<NCOLOR/8; i++ ){
    for ( j=0; j<8; j++ ){
      setfillstyle( SOLID_FILL, 8*i+j );
      bar( WIDTH*j/8,
          HEIGHT*i/(NCOLOR/8),
          WIDTH*(j+1)/8-1,
          HEIGHT*(i+1)/(NCOLOR/8)-1 );
    }
  }
}
main()
{
  char c;
  int gd = DETECT, gm;

  initgraph( &gd, &gm, "c:/borlandc/bgi" );
  settitle( "colors" );
  seticonname( "colors" );
  WIDTH = getmaxx()+1;
  HEIGHT = getmaxy()+1;
  NCOLOR = getmaxcolor()+1;
  cleardevice();
  bars();
  while ( ( c=getch() ) != 0x1b ){
    switch ( c ){
    case '+':
      NCOLOR = min( NCOLOR+8, 256 );
      setmaxcolor( NCOLOR-1 );
      break;
    case '-':
      NCOLOR = max( 8, NCOLOR-8 );
      setmaxcolor( NCOLOR-1 );
      break;
    case '4':
      WIDTH = max( 8, WIDTH-8 );
      setmaxx( WIDTH-1 );
      break;
    case '6':
      WIDTH = min( WIDTH+8, 1024 );
      setmaxx( WIDTH-1 );
      break;
    case '8':
```

```
      HEIGHT = max( 8, HEIGHT-8 );
      setmaxy( HEIGHT-1 );
      break;
    case '2':
      HEIGHT = min( HEIGHT+8, 800 );
      setmaxy( HEIGHT-1 );
      break;
    }
    bars();
  }
  closegraph();
}
```

このプログラムは, 使用可能な色数をすべて表示するプログラムです. キーボード操作で色数やウィンドウのサイズが変化します.

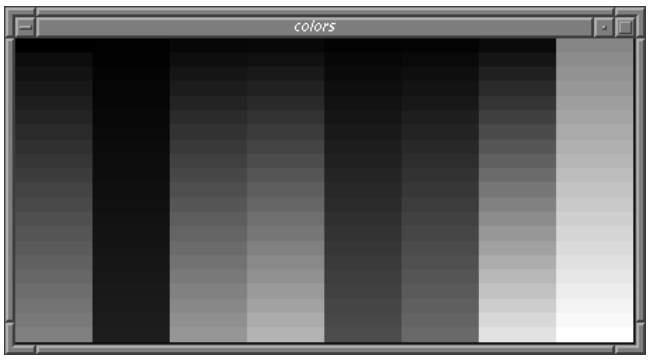


図 3.1: プログラム colors.c の実行画面

関数による方法は, initgraph によって構築されたグラフィック環境に対して作用します. つまり, XBGI システムの状態変数のなかで, 構築した後に変更可能な属性に対してこのような設定が可能なわけです. しかし, グラフィック環境の構築時に決定されてしまう状態変数も存在しますので, 関数による方法では不都合な場合があります. よってグラフィック環境を構築するまえに属性を設定する方法が用意されています.

3.1.1.2 primary design 指定子による方法

BGI では, initgraph(&gd, &gm, pd) の pd で示す文字列はグラフィックドライバが置かれているディレクトリを示しますが, XBGI では, pd にグラフィック環境を設定するための文字列 (オプション文字列) を記述することができ, グラフィック環境はオプション文字列に従って構築されます. XBGI では pd を primary design 指定子と呼びます.

まずは, サンプルプログラム clock.c を御覧下さい.

サンプルプログラム clock.c

```
#include <stdio.h>
#include <time.h>
#include <sys/types.h>
#include <graphics.h>
void main( void )
{
  int i, j, gd = DETECT, gm;
  time_t clock;
  struct tm *tm;
  char str[64], fmt[64] = "%Y/%m/%d %R";

  initgraph( &gd, &gm, "-geometry 200x100 -title clock" );
```

```
  settextstyle( SANS_SERIF_FONT, HORIZ_DIR, 5 );
  settextjustify( CENTER_TEXT, CENTER_TEXT );
  cleardevice();
  while ( !0 ){
    time( &clock );
    tm = localtime( &clock );
    strftime( str, 64, fmt, tm );
    outtextxy( getmaxx()/2, getmaxy()/2, str );
    xflush();
    sleep( 60 );
    clearviewport();
  }
  closegraph();
}
```

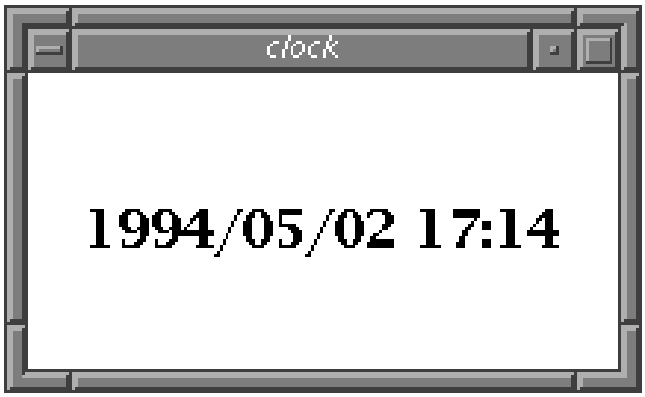


図 3.2: プログラム clock.c の実行画面

このように, primary design 指定子 pd に希望のオプション文字列を記述して下さい. また, 表 3.2 に提供されているオプション文字列を示します.

表 3.2: オプション文字列

オプション文字列	設定対象
-display *displayname*	ディスプレイ文字列
-geometry *geom*	ジオメトリ文字列
-title *name*	ウィンドウ名
-name *name*	アイコン名
-icon *filename*	アイコンファイル名
-gd *num*	グラフィックドライバ番号
-gm *num*	グラフィックモード番号
-npage *num*	ページ数
-color *num*	前景色番号
-bkcolor *num*	背景色番号
-ncolor *num*	色数
-backing	バッキングストア真
-backing-	バッキングストア偽
-only_ascii	ASCII 文字しか使用しない
-only_ascii-	日本語文字なども使用する
-fn *fontname*	描画以外で使用するフォント
-fg *colorname*	描画以外で使用する前景色
-bg *colorname*	描画以外で使用する背景色

この方法は, PC で動いていたユーザーソースコードを書き換えることで実現されますので X 専用プログラムに向いていますが, 条件つきコンパイルによって容易に PC, WS 汎用コードを書くことができるでしょう. 定義済みマクロ __BORLANDC__ や __TURBOC__ (TC, BC 処理系で宣言), _XBGI_ (XBGI で宣言) がそれぞれ宣言されているので利用して下さい. 条件つきコンパイルのサンプルとして, 先の clock.c を PC, WS 汎用にすると以下のようになります.

```
void main( void )
{
  (前略)
  initgraph( &gd, &gm,
#if defined(_XBGI_)
            "-geometry 200x100 -title clock"
#else
            "c:/borlandc/bgi"
#endif
            );
  (後略)
}
```

3.1.1.3 環境変数による方法

環境変数 XBGISET にオプション文字列を記述しても, 同様にグラフィック環境を設定することができます.

```
% setenv XBGISET "-geometry 640x400 -npage 2"
```

この方法はソースコード無変更でさまざまな属性を設定できるので, PC と WS の両方を使用する人にお勧めできます.

3.1.1.4 環境ファイルによる方法

環境ファイル .xbgisets にオプション文字列を記述しても, 同様にグラフィック環境を設定することができます. 例えば, 以下のような環境ファイル .xbgisets がユーザーのホームディレクトリに在るとします.

```
XDEF -color 0 -bkcolor 15 -geometry 500x500
A:/BGI -title PC98 -npage 1 -geometry 640x400
A:/BC  -title DOSV -npage 1 -geometry 640x480
C:/TC  -title HIGH -npage 1 -geometry 1120x750
```

ここで, ユーザープログラムの initgraph(&gd, &gm, pd) の primary design 指定子 pd に, 環境ファイル中で記述された環境認識子 (この場合 XDEF や C:/TC) を指定すると, XBGI はその環境認識子を検索して環境を設定します. この環境認識子は任意の文字列を指定できます. PC におけるディレクトリパス文字列はもちろん, PC マシン名やグラフ名など, ユーザーの目的に応じて記述してください. この方法は, 複数の PC と WS を使用する人には特にお勧めでしょう.

3.1.1.5 環境設定の優先順位

ここでは, XBGI が管理するグラフィック環境を設定する一連の方法 (関数, primary design 指定子, 環境変数, 環境ファイル) を解説しました. これらを組み合わせて使用した場合, その優先順位は以下のようになります.

関数 > pd **指定子** > **環境変数** > **環境ファイル**

このように, XBGI にはグラフィック環境を設定するさまざまな方法が用意されていますので, ユーザーの用途に応じて使い分けてください.

3.1.2 イベント指向のサポート

X ウィンドウシステムは, イベント指向, オブジェクト指向そしてネットワーク指向のシステムです. このようなウィンドウシステムのためのコーディングは, そのための方法論に従った手法に移行しつつあります. ウィンドウプログラミングでは, 例えばツールキットなどを用いてアプリケーションを構築する場合, プログラマはメインループを明け渡し, すべてのイベント処理を依頼します. すなわち, オブジェクト指向やイベント指向のプログラミングでは, オブジェクト間のメッセージの伝播が重視され, 手続きや関数が主導のオーソドックスなプログラムとは思想が異なります.

だからといって, これらオブジェクト指向などがすべてのプログラムに絶対的な方法論ではありません. 例えば, アルゴリズムをプログラムを通して伝えるためのサンプルプログラムなどは, そのアルゴリズムのみを記述したシンプルさが要求されますし, アプリケーション作成目的ではなく, 数値解析のためのプログラムにイベント指向はほとんど必要ありません.

しかし, ウィンドウシステム上でコードを動作させる以上, 最低限の作法は守らなければなりません. XBGI 専用のイベントループである `xbgiloop` は, それらの作法を内部的に隠蔽し, イベント指向をオーソドックスなプログラムへと導きます.

3.1.2.1 XBGI イベントループ

XBGI の標準イベントループ `xbgiloop` は, マウスイベント, キーイベント, コンフィギアイベント (ウィンドウの大きさがユーザーによって変化したときなどに発生するイベント), エクスポーズイベント (隠されていたウィンドウがマップされたときなどに発生するイベント) などに対して, 各々のイベントに用意されたプロシージャを呼びだします. そして, それらのプロシージャはあらかじめデフォルト値が与られているので, ユーザーは細かなイベント処理を必要としないのなら, そのまま `xbgiloop` を使用することが出来ます. 表 3.3に `xbgiloop` イベント処理のデフォルト設定値を示します.

表 3.3: xbgiloop イベント処理

イベントの種類	内容	デフォルト設定値
ボタン押下	button 1	未定義
	button 2	メニュー
	button 3	xbgiloop 終了
ボタン解放	button 1-3	未定義
マウス移動	button_state	未定義
マウスのページ侵入	page_number	未定義
マウスのページ離脱	page_number	未定義
キー押下	ctrl+c	プロセス停止
	ctrl+a	システム情報
	ctrl+i	オブジェクト情報
	ctrl+o	オプション情報
	ctrl+f	すべてのページをマップ
	q	xbgiloop 終了
	n	次ページ
	b	前ページ
	その他	未定義
ウィンドウ構造の変化	リサイズ	未定義
ウィンドウの出現	マップ	未定義
イベントなし	イベントなし	未定義

3.1.2.2 イベントプロシージャの設定

`xbgiloop` にユーザー定義のイベントプロシージャを設定するための関数群を, 表 3.4に示します.

これらの手続きによって, イベントプロシージャの既定値を変更することができます. 返り値は偽の時にイベントループを抜け出し, 結果, `xbgiloop` の制御下から離れることになります. `xbgiloop` にユーザー定義のイベントプロシージャを設定したサンプルを `draw.c` に示します.

表 3.4: イベントプロシージャの設定関数

設定関数 set_*event*_handler	関数の型・引数 int handler()
set_button_press_handler	int button,int x,int y
set_button_release_handler	int button,int x,int y
set_mouse_motion_handler	int button_state,int x,int y
set_page_enter_handler	int page
set_page_leave_handler	int page
set_key_press_handler	int key
set_configure_handler	void
set_expose_handler	void
set_noevent_handler	void

サンプルプログラム draw.c

```
#include <stdio.h>
#include <graphics.h>
#include <outtext.h>

int line_width = 1;
int line_color = 0;

int pressed_flag = 0;
int start_x, start_y;
int button_press( int b, int x, int y )
{
  if ( b == 1 && !pressed_flag ){
    pressed_flag = !0;
    start_x = x;
    start_y = y;
  }
  if ( b == 1 && pressed_flag ){
    setcolor( line_color );
    line( start_x, start_y, x, y );
    pressed_flag = !0;
    start_x = x;
    start_y = y;
  }
  if ( b == 3 )
    pressed_flag = 0;
  return !0;
}
int button_release( int b, int x, int y )
{
  return !0;
}
int mouse_motion( int b, int x, int y )
{
  static char mouse_info[64] = "640,480";

  setcolor( BLACK );
  setfillstyle( SOLID_FILL, WHITE );
  bar( 0, 0, textwidth(mouse_info), textheight(mouse_info) );
  sprintf( mouse_info, "%3d,%3d", x, y );
  outtextxy( 0, 0, mouse_info );
  return !0;
}
int key_press( int c )
```

```
{
  switch ( (char)c ){
  case '8': line_width = min(line_width+1,10); break;
  case '2': line_width = max( 0,line_width-1); break;
  case '6': line_color = min(line_color+1,14); break;
  case '4': line_color = max( 1,line_color-1); break;
  case 0x1b: return 0;
  }
  setlinestyle( SOLID_LINE, 0, line_width );
  setcolor( line_color );
  return !0;
}
int configure( void )
{
  int width, height;

  getsizexy( &width, &height );
  setsizexy(  width , height );
  cleardevice();
  pressed_flag = 0;
  return !0;
}
int noevent( void )
{
  return !0;
}
main()
{
  int gd = DETECT, gm;

  initgraph( &gd, &gm, "" );
  setcolor  ( BLACK );
  setbkcolor( WHITE );
  setlinestyle( SOLID_LINE, 0, line_width );
  cleardevice();
  set_button_press_handler  ( button_press   );
  set_button_release_handler( button_release );
  set_mouse_motion_handler  ( mouse_motion   );
  set_key_press_handler     ( key_press      );
  set_configure_handler     ( configure      );
  set_noevent_handler       ( noevent        );
  xbgiloop();
}
```

このプログラムは, マウスが動いたときの座標を随時表示しながら, マウスでラインを描くプログラムです. また, 4, 6 キーで色の変更, 2, 8 キーでラインの太さの変更をします. 実行画面例は図 3.3のようになります.

3.2 XBGI システムのための拡張機能

オリジナル BGI では initgraph という手続きによってグラフィック環境の初期化をおこない, 描画に関するさまざまな振る舞いを扱えるようになります. しかし, PC では接続されている画面はたかだか 1 つという制限があるため, initgraph で始まり closegraph で終わるグラフィック環境は, やはりたかだか 1 つだけです. この XBGI システムでは記憶域の許すかぎり環境を構築することが可能です.

次のサンプルプログラム recurse.c は複数の環境の構築を (なんら新しい関数を用いることなしに) 実現しています.

サンプルプログラム recurse.c

```
#include <stdio.h>
```

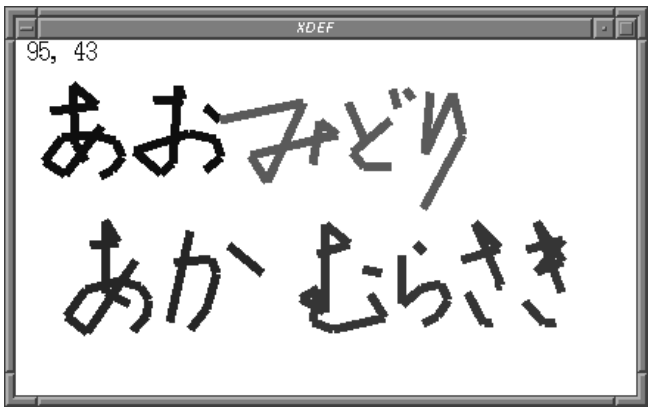


図 3.3: プログラム draw.c の実行画面

```
#include <graphics.h>
void carpet( int n, int x, int y, int l )
{
  int i, j;

  if ( n < 0 )
    return;
  setfillstyle( SOLID_FILL, (9+n)%16 );
  bar( x+l/3, y+l/3, x+l/3*2-1, y+l/3*2-1 );
  rectangle( x+l/3, y+l/3, x+l/3*2-1, y+l/3*2-1 );
  for ( i=0; i<3; i++ )
    for ( j=0; j<3; j++ )
      if ( i!=1 || i!=j )
        carpet( n-1, x+l/3*i, y+l/3*j, l/3 );
}
void mandara( int n, int x, int y, int l )
{
  int i, j;

  if ( n < 0 )
    return;
  setfillstyle( SOLID_FILL, (9+n)%16 );
  fillellipse( x+l/2, y+l/2, l/2-1, l/2-1 );
  circle( x+l/2, y+l/2, l/2-1 );
  for ( i=0; i<3; i++ )
    for ( j=0; j<3; j++ )
      if ( i==1 || j==1 )
        mandara( n-1, x+l/3*i, y+l/3*j, l/3 );
}
void main( void )
{
  int gd = DETECT, gm;
  int n = 3, x = 20, y = 20, l = 216;

  initgraph( &gd, &gm, "c:/borlandc/bgi" );
  cleardevice();
  carpet( n, x, y, l );

  initgraph( &gd, &gm, "c:/borlandc/bgi" );
  cleardevice();
  mandara( n, x, y, l );
```

```
  xbgiloop();
  closegraph();
  closegraph();
}
```

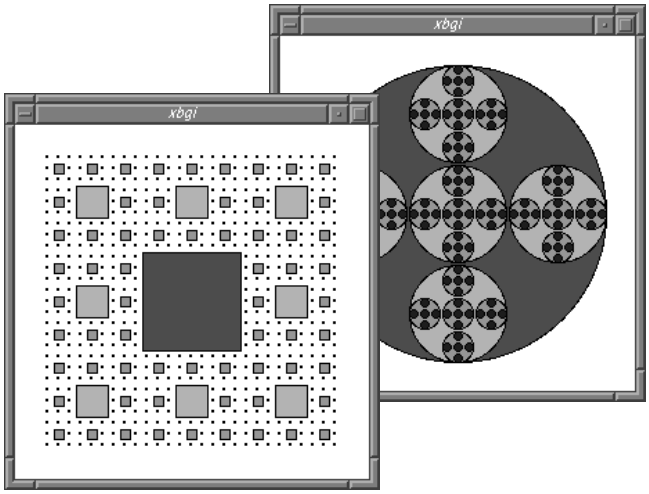


図 3.4: プログラム recurse.c の実行画面

XBGI では, このプログラムのように過去の PC プログラムソースをただ結合させるだけで, 複数の環境の構築が可能です. このメリットはグラフィック画面が複数扱えるということだけではありません. BGI ではページというものがあり, XBGI では任意数のページが用意できるので, グラフィック画面を複数扱うには単一のグラフィック環境でも可能です. 複数の環境の構築のメリットは, 各々の環境が完全に独立していることです. たとえば, 環境 1 で変更されたパレットは, 環境 1 が管理する全てのページに影響をおよぼしますが, 環境 2 には影響をおよぼしません. その他の状態変数においても完全に独立しています.

本章では, グラフィック環境を 1 つのプロセスから複数扱うための手法を解説します. それには以下の方法が用意されています.

1. XBGI の ID を取得する方法 (§3.2.1)
2. オブジェクト xbgi を取得する方法 (§3.2.2)

1 番目の方法は, グラフィック環境の ID を取得し, ターゲットであるグラフィック環境をアクティブにアクセスする手法です. 2 番目の方法はグラフィック環境の実体 (オブジェクト xbgi) をオープンし, ターゲットであるグラフィック環境を高速にアクセスする手法です.

3.2.1 XBGI の ID によるアクセス

XBGI では initgraph でグラフィック環境を構築するときに, そのグラフィック環境にユニークな番号 (ID) を割り当てます. ID を取得するには graphnumber を, 取得した ID でカレントのグラフィック環境を指定するには selectgraph を用います. 表 3.5 にそれらの関数をあげます. また, プログラム chaos.c は ID を用いて複数のグラフィック環境にアクセスしたサンプルです.

サンプルプログラム chaos.c

```
#include <stdio.h>
#include <math.h>
#include <graphics.h>
#define B 1.5
double chaos( double x )
```

表 3.5: ID に関する拡張関数

関数	機能
int graphnumber()	ID の取得
selectgraph(int id)	ID でカレント BGI を指定
termgraph(int id)	ID で BGI を破壊

```
{
  if ( 0.0<=x && x<0.5 )
    return( x+pow(2.0,B-1)*pow(x,B) );
  else
    return( x-pow(2.0,B-1)*pow(1.0-x,B) );
}
void main( void )
{
  int i, gd = DETECT, gm;
  int graph1, graph2;
  double x0 = 0.2, x1;

  initgraph( &gd, &gm, "-geometry 800x200" );
  graph1 = graphnumber();
  cleardevice();
  initgraph( &gd, &gm, "-geometry 400x400" );
  graph2 = graphnumber();
  cleardevice();
  for ( i=0; i<800; i++ ){
    x1 = chaos( x0 );
    selectgraph( graph1 );
    line( i-1, 200-200*x0, i, 200-200*x1 );
    selectgraph( graph2 );
    line( 400*x0, 400-400*x0, 400*x0, 400-400*x1 );
    line( 400*x0, 400-400*x1, 400*x1, 400-400*x1 );
    x0 = x1;
  }
  xbgiloop();
  termgraph( graph1 );
  termgraph( graph2 );
}
```

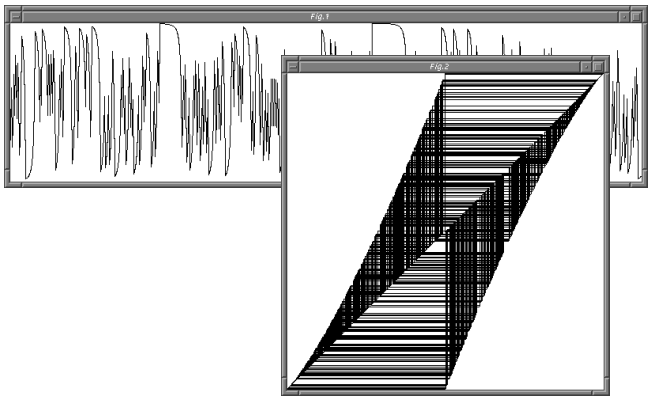


図 3.5: プログラム chaos.c の実行画面

3.2.2 オブジェクト xbgi

ID を取得する方法では, 登録されている オブジェクト xbgi の中から指定された ID を持つオブジェクトを検索するため, それほど高速にカレントオブジェクトを切替えられるわけではありません. そこで, 検索時間の要しない方法が用意されています. それがオブジェクト xbgi を取得する方法です. オブジェクト xbgi を取得するには graphics.h のかわりに graphix.h をインクルードして下さい. すると, プログラムのコンパイル時に, X に関するヘッダファイルもインクルードされるので, BGI 環境を利用しながら X に直接アクセスするプログラマに適しています. 表 3.6 にオブジェクト xbgi 関連の関数をまとめます. 使い方については, サンプルプログラム message.c を参照して下さい.

表 3.6: オブジェクトに関する拡張関数

関数	機能
graphix.h で宣言	
creategraph	BGI 環境の構築とオブジェクト取得
thisgraph	構築されているオブジェクトを取得
currentgraph	オブジェクトでカレント BGI を指定
destroygraph	オブジェクトで BGI を破壊

サンプルプログラム message.c

```
#include <stdio.h>
#include <graphix.h>
void main( int argc, char *argv[] )
{
  xbgi *gr1, *gr2;
  int gd = DETECT, gm;
  int i, x = 220, meswidth;
  char message[64] = "usage ...  % message mes_str";

  if ( argc > 1 ){
    strcpy( message, argv[1] );
    for ( i=2; i<argc; i++ ){
      strcat( message, " " );
      strcat( message, argv[i] );
    }
  }

  gr1 = creategraph( &gd, &gm, "-geometry 100x50" );
  settextstyle( TRIPLEX_FONT, HORIZ_DIR, 6 );
  settextjustify( LEFT_TEXT, CENTER_TEXT );
  cleardevice();

  gr2 = creategraph( &gd, &gm, "-geometry 100x50" );
  settextstyle( TRIPLEX_FONT, HORIZ_DIR, 6 );
  settextjustify( LEFT_TEXT, CENTER_TEXT );
  cleardevice();

  meswidth = textwidth( message );
  while ( !kbhit() ){
    if ( --x < -meswidth )
      x = 220;

    currentgraph( gr1 );
    cleardevice();
    outtextxy( x, 25, message );
    xflush();

    currentgraph( gr2 );
```

```
    cleardevice();
    outtextxy( x-120, 25, message );
    xflush();
    delay( 100000 );
  }
  destroygraph( gr2 );
  destroygraph( gr1 );
}
```

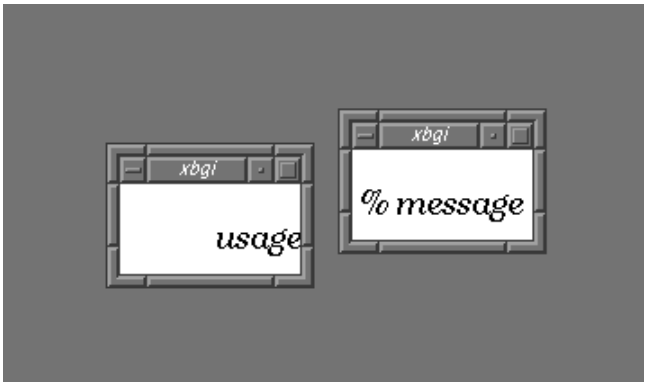


図 3.6: プログラム message.c の実行画面

3.2.3 XBGI の設計思想

Xbgi システムは オブジェクト xbgi の リスト構造になっており, このような複数のグラフィック環境の構築を実現しています. さらに, 各々のオブジェクトが X のディスプレイ接続をおこなうので, ディスプレイ 1 にはグラフィック 1 を表示し, ディスプレイ 2 にはグラフィック 2 を表示する, といった使い方も可能となっています. そして, オープンした XBGI をすべて掌握しているのは, 先に解説した xbgiloop です. 複数のグラフィック環境を利用している場合において, xbgiloop の利用価値は非常に高いといえます. そして, xbgiloop はプログラマブルに扱えるので, プログラムソースの PC 互換を考慮しないのであれば, ある程度実用的なアプリケーションを構築することができます.

Xbgi システムはオブジェクト指向を目指したプログラムですが, ユーザーにオブジェクト指向を強いることのない設計となっています. その思想は非オブジェクト指向というよりも, オブジェクト指向のオブジェクトさえも隠蔽したいという発想から成り立っています. オブジェクト指向的な考え方だけでは, “描画” という一連の作業において, 常に “キャンバス” を意識することを強要し, それは, しばしばコーディングスタイルにわずらわしさをもたらします. グラフィックインターフェースにおけるオブジェクト指向のあり方は, オブジェクト指向からアクションのみを抽出したアクト指向であると考えます. アクト指向の資産を構築するためには, オブジェクト指向の概念を必要としますが, それを扱うプログラマには従来の手続き型のプログラミングしか要求しません. この考え方は, 比較的数の少ないオブジェクトしか扱わない枠組みであれば, 他のオブジェクト指向にも適用可能かと思われます.

この Xbgi システムでは, Bgi のシミュレートという題材を通じてこのアクト指向を試みました. そして将来的にはオブジェクトをさらに継承し, 描画対象を X ウィンドウだけでなく, ドロー系ツールやポストスクリプトをターゲットに計画しています. 特に描画対象がポストスクリプトの場合, 整数座標系で構築されている X や Bgi では, ポストスクリプトやプリンタの性能を生かしきれません. つまり, アルゴリズムによって生成されるグラフィックを高精度の出力で得るには, このようなグラフィックインターフェースレベルで, 整数空間でないワールド座標系 (2 次元の実数空間) をサポートする必要性があるのです.

次の節ではそれに関連して, グラフィック機能の拡張をおこないます.

3.3 BGI グラフィックスのための拡張機能

この章では Bgi に倍精度グラフィックスや, ユーティリティなどを供給する Bgi ライブラリの解説をします.

倍精度グラフィックスでは, Bgi のビューポートを 2 次元実数空間のワールド座標系として表現するためのライブラリ graphr2_.h をベースとして, 対数座標系や投影変換による 3 次元空間などを供給します. また, ユーティリティライブラリでは, 関数 outtext で書式付き出力を可能にしたり, グラフィック画面のファイルへの保存をサポートします. 表 3.7 にインクルードファイル名と提供する機能をあげます.

表 3.7: Bgi ライブラリ

ファイル名	機能
倍精度グラフィックス	
graphr2_.h	倍精度によるワールド座標系
gturtle_.h	タートルグラフィックス
graphl2_.h	両対数，片対数座標系
graphr3_.h	投影変換による 3 次元空間
graphp3_.h	透視変換による 3 次元空間
graphs3_.h	ステレオグラフ 3 次元空間
ユーティリティ	
axes.h	グラフィック画面に座標軸を描画
outtext.h	グラフィック画面に書式付き outtext
outtext_.h	ワールド座標画面に書式付き outtextxy
gsave.h	グラフィック画面を保存

これらのファイルはインクルードディレクトリ include/graphics にあります. これらのインターフェースはオリジナル Bgi, Xbgi の双方に対応しており, いずれもインクルードファイルによってその機能を使うことが可能になります. つまり, 高速性と簡便さのために, インクルードファイルにコードやマクロが書かれており, 正式なライブラリとしての手続きを経ていません. よって, これらのインクルードファイルを Borland 社製 C のインクルードディレクトリに複写しておけば, これらの機能を共有することが出来ます. ただし, これらは ANSI スタイルで書かれているので, 古い C 処理系ではコンパイルできない場合があるかもしれません. よって UNIX 上では gcc をお使いになることをお勧めします.

3.3.1 倍精度グラフィックス

このライブラリの最も中核をなすインクルードファイル graphr2_.h を用いたサンプルを attract.c に示します. サンプルプログラムのように setwindow_で浮動小数点によるワールド座標を指定して下さい. ワールド座標でグラフィック画面にアクセスするには, Bgi 標準関数にアンダースコア (_) を付加した関数を用いて下さい. アンダースコア (_) は座標に関する引数が倍精度であることを表しています.

サンプルプログラム attract.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <setjmp.h>
#include <conio.h>
#include <graphics.h>
#include <graphr2_.h> /* R^2 空間をサポートするインクルードファイル */
#include <coloring.h> /* パレットを扱うためのツール */

#define BTIME 500
#define ITIME 100
int MAXCOLOR;
jmp_buf jumper;

/*
```

```
 * 参考文献
 * F. Aicardi and S. Invernizzi:
 *  ``Memory effects in discrete dynamical systems'',
 *  International Journal of Bifucation and Chaos, 2, 4, 815-830 (1992)
 * R. M. May:
 *  ``Simple mathematical models with very complicated dynamics'',
 *  Nature, 261, 459-467 (1976)
 */
double A, B = 0.8, X0, X1, X2;
#define F(x) A*(x)*(1.0-(x))
#define G(x1,x0) F((1.0-B)*x1+B*x0)

void iterate( void )
{
  int t;

  for ( t=0; t<10; t++ ){
    X2 = G(X1,X0);
    X0 = X1;
    X1 = X2;
  }
  for ( t=0; t<ITIME; t++ ){
    X2 = G(X1,X0);
    putpixel_( X1, X2, t%(MAXCOLOR-1)+1 ); /* 論理座標への描画 */
    X0 = X1;
    X1 = X2;
    if ( kbhit() && getch() == 0x1b )
      longjmp( jumper, 1 );
  }
}

void bifurcate( void )
{
  int b;

  for ( b=0; b<BTIME; b++ ){
    A = 3.0 + (4.0-3.0)*b/BTIME;
    X0 = 0.1;
    X1 = 0.2;
    iterate();
  }
}

int main( void )
{
  int gd = DETECT, gm, i;
  int jump_val;

  initgraph_( &gd, &gm, "-geometry 400x400" );
  MAXCOLOR = getmaxcolor();
  coloring( 1, MAXCOLOR-1 ); /* RGB グラデーションカラーの作成 */
  setviewport_( 0, 0, 400-1, 400-1, !0 ); /* 物理座標を指定 */
  setwindow_( 0.0, 1.0, 1.0, 0.0 ); /* 論理座標を指定 */

  jump_val = setjmp( jumper );
  if ( jump_val == 0 ){
    bifurcate();
    (void)getch();
  }

  closegraph_();
  return 0;
```

}

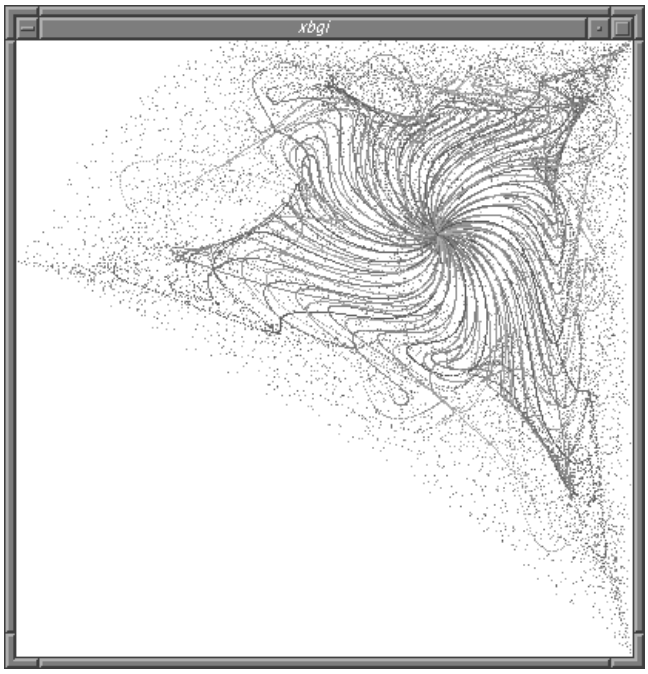


図 3.7: プログラム attract.c の実行画面

3.4 拡張機能を使用したコードの可搬性

この章では, X ウィンドウのための拡張機能, XBGI システムのための拡張機能, BGI グラフィックスのための拡張機能に関して順に解説してきました. 3 番目の拡張機能はオリジナル Bgi, Xbgi に汎用なので可搬性は問われませんが, 1,2 番目の機能を用いたプログラムは, 再び PC 上でコンパイルすることは出来ません. そのためオリジナル Bgi 上でこれらの拡張機能を無効にするためのインクルードファイルが用意されています. オリジナル Bgi 用の GRAPHICS.H に

```
#include <xbgi2bgi.h>
```

の 1 行を書き加えておけば, 1,2 番目の拡張機能を無効にすることが出来、かつイベント指向プログラミングの xbgiloop と set_*event*_handler が PC 上にサポートされます. 実際にサンプルプログラムを PC 上でコンパイル, 実行してみてください. さらに, PC のインクルードディレクトリにこの GRAPHIX.H を加えておけば, 2 番目のオブジェクト xbgi に関する拡張機能を無効にすることが出来ます.

表 3.8: BGI 用インクルードファイル

ファイル名	機能
XBGI2BGI.H	拡張関数の無効化
GRAPHIX.H	オリジナル Bgi 用 graphix.h

これらはディレクトリ exportpc にあります.

第 4 章

内部仕様

この章では XBGI の内部仕様に関して, 前章で説明した内容と重複する事柄を含めて解説していきます.

4.1 XBGI 構築手続きに関して

4.1.1 XBGI 環境設定ファイルの利用

ホームディレクトリの XBGI 環境設定ファイル .xbgisets について詳しく解説します. この設定ファイルがなくても XBGI はなんの支障もなく動作しますが, XBGI のカスタマイズをしたい方, 複数の PC と WS を併用して利用している方には, これは大変便利なファイルです. 私は主に 4 種類の機種を同時に利用しています. そのうちの 3 機種は PC で IBM 互換機, PC-9801, FMR-70 です. あとの 1 機種はもちろん WS で SUN を使っています. そのような製作者が使っている .xbgisets をご覧にいれましょう.

```
      -title XBGI -ncolor 16 -color 0 -bkcolor 15 -geometry  500x500 -npage 1
A:/TC -title PC98 -ncolor 16 -color 15 -bkcolor 0 -geometry  640x400 -npage 2
C:/BC -title DOSV -ncolor 16 -color 15 -bkcolor 0 -geometry  640x480 -npage 1
D:/TC -title HIGH -ncolor 16 -color 15 -bkcolor 0 -geometry 1120x750 -npage 1
```

これは簡単な画面情報データベースとなっていることがおわかりでしょうか. XBGI ではさまざまな PC の画面情報データベースを内部には持たず, すべてユーザーに任せることによって環境設定できるようにしています. この環境設定ファイル .xbgisets がホームディレクトリにある場合, initgraph の第 3 引数が "A:/TC" と書かれた実行プログラムは PC-9801 の画面構成で X ウィンドウが開くようになっています. また, "C:/BC" では IBM マシンの VGA 画面, "D:/TC" では ハイレゾ画面のウィンドウが開くようになっています. この "D:/TC" などを「環境認識子」と呼びます. 環境認識子には機種名などを書いてもよいのですが, PC では機種の特性からハードディスクのデフォルトドライブ名がばらばらになるので, C 言語処理系を格納しておく場所が, ちょうど上記の認識子のようになるのではないでしょうか.

ところで, この .xbgisets の最初の行の設定には認識子がありません. これは「ヌル認識子」と呼ばれ, 認識子を用いなかったときに有効な設定です. 以下の例ではいずれも, ヌル認識子の設定が有効になります.

```
initgraph( &gd, &gm, "" );
initgraph( &gd, &gm, "-geometry 300x200" );
```

但し, 設定内容が重複する場合は initgraph の第 3 引数である pd (primary design) 指定子の方が有効になりますので, 上の 2 番目の例では画面の大きさが 300x200 になります. 以下の例でも pd 指定子の方が有効になりますので, 画面の大きさが 200x200 になります.

```
initgraph( &gd, &gm, "A:/TC -geometry 200x200" );
```

4.2 構築系関数

オリジナル BGI ではグラフィックドライバの初期化を initgraph という関数を用いて実行します. それは多くの機種に互換性をもたせるため, たいへん巧妙な仕組みでドライバの検出や初期化が遂行されるよう設計されています. XBGI でも, X ウィンドウの能力が非常に高いおかげで, 柔軟なグラフィック画面の構築を行えるよう設計されています. この章では, さまざまな構築系関数の解説をしていきます.

4.2.1 初期化関数

XBGI にはオリジナル BGI の initgraph に相当する以下のような構築系関数が用意されています.

表 4.1: 構築系関数

関数	機能
void initgraph(int *gd, int *gm, char *pd)	オリジナル BGI 互換関数
void _initgraph(int argc, char *argv[], char *pd)	コマンドラインオプションが引数
xbgi *creategraph(int *gd, int *gm, char *pd)	返り値がオブジェクト xbgi
xbgi *_creategraph(int argc, char *argv[], char *pd)	上記のコマンドラインオプション版

init 系と create 系関数の違いは, 関数の型が void であるか, オブジェクト xbgi を返すかの違いです. アンダーバー (_) 系の関数については, 引数がコマンドラインオプションを渡すことが通常の関数と違うところです. すなわち, プログラムを起動したときの設定が最優先で有効になるのです. よって, あなたが作成するコードは, いかなる画面の大きさ, 色数でも動作するように作成されていなければなりません. そして, デフォルト設定は primary design 指定子 pd に記述しておくとよいでしょう.

4.2.2 ドライバの登録

オリジナル BGI には installuserdriver という関数が用意されており, ユーザーグラフィックドライバを外部リンク出来るようになっています. XBGI では現在この installuserdriver をサポートしていませんので, "Reserved Future" をエラー出力に発行し, 実行は無視します. ただし, ユーザーのプログラマブルなカスタマイズができないわけではありません. 次に説明する registerbgidriver を用いることによって, 柔軟なグラフィック用ウィンドウの構築が可能になります.

グラフィックドライバの登録を行う registerbgidriver は XBGI に特有な関数ではなく, オリジナルの BGI にも用意されています. XBGI の registerbgidriver の説明をする前に, オリジナルの registerbgidriver の説明をしなければなりません. この関数は, BGI ドライバを実行プログラムに内部リンクするための関数です. 使い方をまったく知らないという方のために, その利用方法を紹介します. まず, PC 用のサンプルプログラムをご覧ください.

オリジナル BGI のためのサンプル REGIST.C

```
#include <stdio.h>
#include <graphics.h>
main()
{
  int gd = DETECT, gm;

  registerbgidriver( DOSVGA_driver );
  initgraph( &gd, &gm, "" );
  (中略)
  closegraph();
}
```

このように initgraph の前に内部リンクしたいドライバの public name を登録します. そしてコンパイルは DOS コマンドライン上で以下のようにします.

```
> bcc regist.c dosvga.obj graphics.lib
```

但し, その前に BGI ドライバをオブジェクト形式に変換しておく必要があります. それには DOSVGA.BGI が置いてあるディレクトリで, 以下のように作成してください.

```
> bgiobj dosvga
BGI to OBJ Converter Version 3.01 Copyright (c) 1991 Borland International

9214 bytes from 'DOSVGA.BGI' converted into 'DOSVGA.OBJ',
public name = '_DOSVGA_driver'.
```

この方法で作成されたプログラムは, 自動検出 (DETECT) でマシンの検出や, pd(path to driver) からドライバ検索をせずに, すでにコンパイルリンク時に登録されたドライバを呼び出します. 上の例では DOS/V 用を想定した例ですが, 他のマシン用のドライバも同様に行います.

XBGI でも同様のことが可能であり, グラフィックドライバの作成方法を理解すれば, オリジナルのグラフィックドライバで, あなただけのグラフィックシステムを構築することができます.

4.3 ツールキットとの融合プログラミング

XBGI はユーザーインターフェースを軽視しています. なぜなら, XBGI オリジナルのウィジット (ボタンやメニュー) を拙作で用意しなくても, そのようなコーディングはすでに用意されている X ウィンドウ用のツールキットに任せてしまえばいいからです. そしてさきほど, 「オリジナルのグラフィックドライバで, あなただけのグラフィックシステムを構築することが出来る」と述べましたが, その最たる例としても, またユーザーインターフェースを補強する意味でも, 「ツールキットとの融合プログラミングテクニック」に興味がある方はたくさんいることでしょう. XBGI 付属のサンプルプログラムでは, オリジナルのグラフィックドライバを作成し, ツールキットの一例として X View との融合プログラミングを実現しています. 興味のある方は, ぜひサンプルプログラムを解読してみてください. このサンプルプログラムは xvsample にあります。実行画面のみ図 4.1に示します。

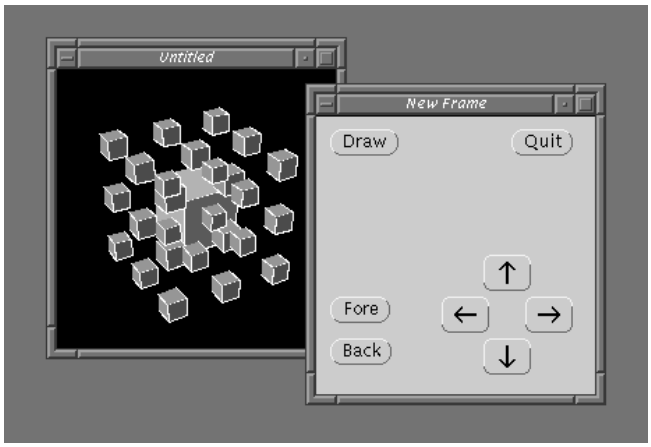


図 4.1: X View とのマルチプラットフォームプログラミング例

4.4 XBGI のページ

4.4.1 ページの構造

XBGI では複数のページを用意することが出来ます. それでは最大限どのぐらいのページを用意できるのでしょうか. 原理的には記憶域の許す限りページを用意できるのですが, そのページに対してバッキングストアを設定している (ユーザーの多くはこれを想定して利用している) 場合, ユーザーの報告では, 総計 200 枚のページを開いたところ, 何枚かの

ページにバッキングストアの欠損が生じたそうです. バッキングストアをオフにした場合にはもっとオープン可能な枚数が増えますが, その場合には, バッキングストアを補うような, X ウィンドウプログラミングに則したプログラミングが必要です.

XBGI における 1 枚のページは 1 枚の X ウィンドウによって構築されています. そして, ウィンドウマネージャのリサイズイベントによって画面が消滅しないように, 以下の図のような構成になっています.

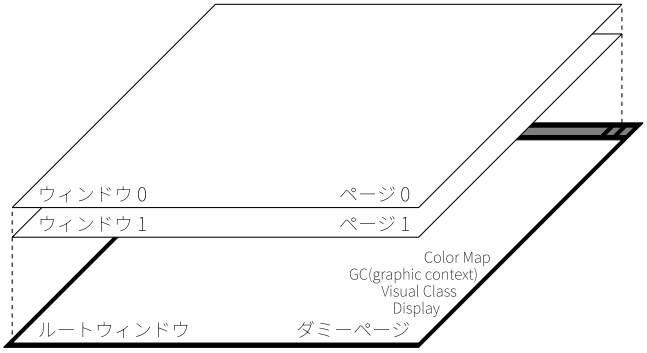


図 4.2: ページとウィンドウの関係

このように, n 枚のページに対応して 0 から $n-1$ までのウィンドウが用意されます. そして, それらのウィンドウの親であるダミーページを 1 枚有し, このダミーページにはさまざまなオブジェクトが接続されています. さまざまなオブジェクトとは, ディスプレイ, ビジュアルクラス, グラフィックコンテキスト, カラーマップなどです. なぜ, このようにダミーページを用意しているのかというと, その子ウィンドウであるすべてのページにさまざまな属性を共有させるためです. よって, 例えばアクティブページ 0 のときにパレットを変更しても, ルートウィンドウが属性のすべてを所有しているので, その属性変更はすべてのページに応答します. この特性は, オリジナル BGI になぞらえたものです.

ただし, グラフィック画面の大きさの設定や取得においては, 0 番目のページにある程度の特権が与えられており, その他のページに対するものとは応答が異なる機能があります. そのページの特性に関して次の節で解説します.

4.4.2 アクティブページの特性

`getmaxx, getmaxy` はグラフィック画面の大きさ (正確には最大値) を, x 座標, y 座標それぞれに関して取得するものです. また, `getsizexy` はグラフィック画面の大きさを, x 座標, y 座標について同時に取得するものです. ただし, アクティブページ 0 においてこれらの関数を実行した場合には, ダミーページ, すなわちルートウィンドウの画面のサイズ情報を返します. また, `setmaxx, setmaxy` はグラフィック画面の大きさ (正確には最大値) を, x 座標, y 座標それぞれに関して設定するもので, `setsizexy` はグラフィック画面の大きさを, x 座標, y 座標について同時に設定するものですが, この場合も取得関数と同様に, アクティブページ 0 においてこれらの関数を実行した場合には, ルートウィンドウとすべてのページの画面サイズを変更します.

表 4.2: 画面サイズ関数

関数	機能
`int getmaxx()`	x 座標最大値の取得
`int getmaxy()`	y 座標最大値の取得
`void setmaxx( int x )`	x 座標最大値の設定
`void setmaxy( int y )`	y 座標最大値の設定
`getsizexy( int *width, int *hight )`	画面サイズの取得
`setsizexy( int width, int height )`	画面サイズの設定

実際の動作特性を確認するために, 2 つのサンプルプログラム pages.c, pages2.c をあげます. 両方のプログラムとも, マウスでウィンドウサイズを変更しながら実行してみてください. アクティブページ 0 のときのみ, すべての画面サイ

ズが変更されることが確認できます.

サンプルプログラム pages.c

```
#include <stdio.h>
#include <graphics.h>
main()
{
  int gd = DETECT, gm;
  int i, mxx, mxy, mxx2, mxy2;
  char str[64];

  initgraph( &gd, &gm, "-npage 8 -color 0 -bkcolor 15" );
  setcolor( 0 );
  setbkcolor( 15 );
  mxx = getmaxx();
  mxy = getmaxy();
  i = 0;
  do {
    setactivepage( i );
    setvisualpage( i );
    mxx2 = getmaxx();
    mxy2 = getmaxy();
    if ( i==0 && ( mxx!=mxx2 || mxy!=mxy2 ) ){
      setmaxx( mxx2 );
      setmaxy( mxy2 );
      mxx = mxx2;
      mxy = mxy2;
    }
    cleardevice();
    sprintf( str, "%d:%dx%d", i, mxx2+1, mxy2+1 );
    outtextxy( 10, 10, str );
    if ( ++i==8 ) i = 0;
  } while ( getch() != 0x1b );
  closegraph();
  exit( 0 );
}
```

サンプルプログラム pages2.c

```
(前略)
    setactivepage( i );
    setvisualpage( i );
    getsizexy( &xsize2, &ysize2 );
    if ( i==0 && ( xsize!=xsize2 || ysize!=ysize2 ) ){
      setsizexy( xsize2, ysize2 );
      xsize = xsize2;
      ysize = ysize2;
    }
(後略)
```

4.4.3 ビジュアルページの特性

アクティブページとは描画対象であるページを指し，BGI の描画プロセスはすべてこのページを対象に遂行されます．一方，ビジュアルページとは表示対象であるページを指し，このページはディスプレイ上に表示されるのみで，いかなるアクションもビジュアルページには遂行されません．`setvisualpage` はビジュアルページ番号を指定する関数で，設定されたページが可視化の対象となります．オリジナル BGI にしたがって，原則として可視化されているペー

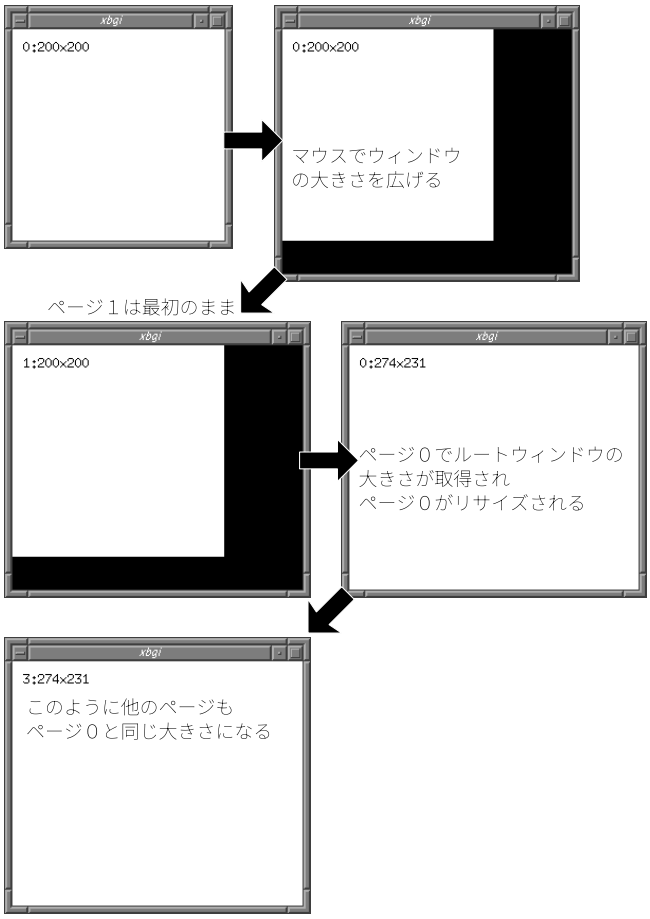


図 4.3: プログラム pages.c の実行画面: すべてのページのサイズを取得し表示する. ただし, ページ 0 のときは取得するサイズはページ 0 のサイズではなく, ウィンドウマネージャによって変更された可能性のある, ルートウィンドウのサイズを取得するので, そのサイズが変更されていれば, すべてのページのサイズをその大きさで設定し直す.

ジはたかだか 1 枚です. しかし, すべてのページを一度にみたいという欲求もあろうかとおもいますので, XBGI ではすべてのページを可視化させる 2 種類の方法を用意してあります. 1 つはただ単に, すべてのページにウィンドウマネージャを介入させる方法で, もう 1 つはユーザーの管理のもと, ページをルートウィンドウに配置する方法です.

まず, すべてのページを可視化させる 1 つめの方法を説明します. ページの構造は図 4.2で説明しましたが, これらのページを, 親であるダミーページから自立させてしまうことのよって, すべてのページが可視化されます. 方法は簡単で, setvisualpage にページ数 n を指定するだけです (サンプルプログラム pages3.c).

サンプルプログラム pages3.c

```
#include <stdio.h>
#include <graphics.h>
main()
{
  int gd = DETECT, gm;
  int i, c;
  char str[64];

  initgraph( &gd, &gm, "-npage 3 -color 0 -bkcolor 15" );
  setcolor( 0 );
  setbkcolor( 15 );
  for ( i=0; i<3; i++ ){
```

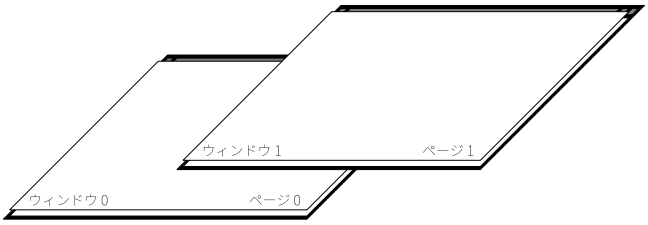


図 4.4: すべてを可視化した場合のページとウィンドウの関係

```
    setactivepage( i );
    sprintf( str, "page:  %d", i );
    outtextxy( 10, 10, str );
  }
  i = 0;
  while ( (c=getch()) != 0x1b ){
    switch( c ){
    case '8':
      if ( i < 3 )
        setvisualpage( ++i );
      break;
    case '2':
      if ( 0 < i )
        setvisualpage( --i );
      break;
    }
  }
  closegraph();
  exit( 0 );
}
```

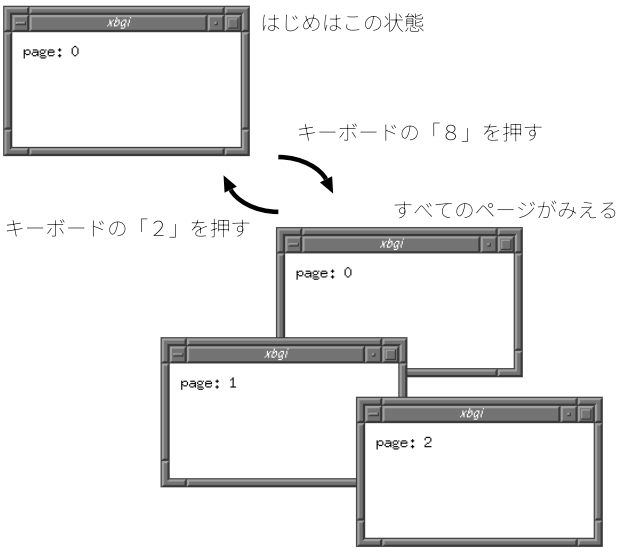


図 4.5: サンプルプログラム pages3.c の実行画面: すべてのページにページ番号を書いておく. キーボード 2 と 8 のキーでビジュアルページを変更, 3 枚すべてのページが表示されるのは, setvisualpage(3) が実行されたとき.

すべてのページが可視化される, もう一つの方法は, まず次の新しい関数を知ってもらわなければなりません.

表 4.3: ページジオメトリ関数

関数	機能
void getgeometry(int *x, int *y, int *width, int *height)	ページジオメトリ取得
void setgeometry(int x, int y, int width, int height)	ページジオメトリ設定

これらはアクティブページのジオメトリを取得設定する関数です. 引数の width, height はページのサイズを示しますが, x, y はルートウィンドウに対する位置を示します. つまり, ルートウィンドウにページを配置することによって, すべてのページが可視化されます (サンプルプログラム pages4.c).

サンプルプログラム pages4.c

```
#include <stdio.h>
#include <graphics.h>
main()
{
  int gd = DETECT, gm;
  int i;
  char str[64];
  struct sizeset {
    int x, y, w, h;
  } page[4] = {
    { 0, 0, 300, 300 },
    { 100, 0, 100, 150 },
    { 50, 100, 200, 100 },
    { 150, 150, 150, 150 },
  };

  initgraph( &gd, &gm, "-npage 4 -geometry 300x300" );
  for ( i=0; i<4; i++ ){
    setactivepage( i );
    setvisualpage( i );
    setcolor( 0 );
    setbkcolor( 15 );
    cleardevice();
    setgeometry( page[i].x, page[i].y, page[i].w, page[i].h );
    rectangle( 0, 0, getmaxx(), getmaxy() );
    sprintf( str, "page:  %d", i );
    outtextxy( 10, 10, str );
  }
  xbgiloop();
  closegraph();
  exit( 0 );
}
```

ページの重なり具合は, あとから可視化したページが上になるので, setvisualpage の指定順によって決まります. また, これによって配置されたページを, setvisualpage(n) ですべて可視化すると, もとに配置された始点情報を失うので, 再配置する必要があります.

4.5 ビジュアルクラスやカラーマップとの関係

WS にはいろいろな種類があり, そのグラフィック能力もさまざまです. X はそのバリエーションを「ビジュアルクラス (visual class)」として分類し, サポートしています. そのビジュアルクラスは, 色を表現できるか, 色を同時に多く表現できるか, 濃淡や色の変更が可能かによって決まります. WS の彩色能力は, グレイスケール, 限定色カラー, フルカラーの 3 種類に分けられ, 濃淡や色の変更能力は, カラーマップの内容の変更が可能か不可能かによって決まります.

これらの能力によって分類されるビジュアルクラスを以下の表にまとめます.

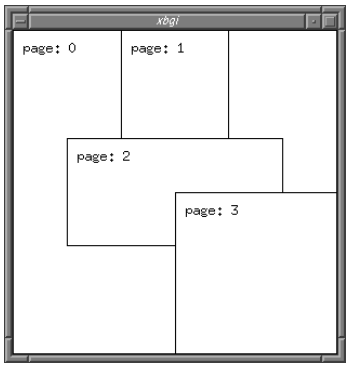


図 4.6: プログラム `pages4.c` の実行画面: それぞれのページの始点と大きさを `setgeometry` で設定しながら, ページ番号を書いていく. `xbgiloop` の `B` と `N` キーで, ビジュアルページが変更できるので, ページ同士の重なり具合が変更される.

表 4.4: X のビジュアルクラス

表示色	カラーマップ	
	変更不可	変更可
グレースケール	StaticGray	GrayScale
限定色カラー	StaticColor	PseudoColor
フルカラー	TrueColor	DirectColor

また, どのくらいの濃淡階調を表現できるかは,「デプス (depth)」によって表されます. デプスは 1 画素を表現するために必要なビット数です. 例えばモノクロディスプレイなら, 黒か白かを表現できればいいのでデプスは 1 です.

Xbgi では, これらのビジュアルクラスをグラフィックモードに対応させています. ただし現在, フルカラーはサポートしておりません. ただし, モノクロとは, デプスが 1 で StaticGray の場合のことですが, わかりやすさのため, あえて `BGIMONO` という列挙定数名を用意してあります.

4.5.1 デフォルトカラー設定

Xbgi では `setmaxcolor` 命令によって, 表示可能色を指定できるようになっています. PseudoColor なら最大 256–2 色のカラーマップを取得できます. しかし, もしあなたが非常にカラフルな色で X ウィンドウを使用中, Xbgi プログラムを起動しますと, 1 度 Xbgi はカラーマップの取得に失敗することがあります. そして, Xbgi はしかたがないので, オリジナルのカラーマップを作成し, カラーマップの取得をしなおします. この状態は *false colors* 現象を引き起こしますので, 他のウィンドウはおかしな配色になることでしょう. ウィンドウマネージャが正しく動作していれば, マウスカーソルが Xbgi ウィンドウから出ることによって, 他のウィンドウが正しく配色されます.

Xbgi は `setmaxcolor` が実行されると, デフォルトカラーをとりあえず配色しておきます. そのデフォルトカラーについては, 先に紹介したサンプルプログラム `colors` によって知ることができます.

4.5.2 RGB 設定

Xbgi の RGB 設定は `setrgbpalette` によって行われますが, オリジナル Bgi との互換性から, 16 色のときに 16 階調になるようになっています. つまり, 64 色取得したときは 64 階調, 128 色のときは 128 階調で RGB 設定します.

カラー取得数が少ないときでも，階調を高く RGB 設定したいときは，65536 階調の _setrgbpalette を使ってください.

4.6 フォントおよび国際化機能の利用

オリジナル BGI の日本語圏を想定した処理系では日本語のテキスト出力が可能です. XBGI でも X の国際化機能を利用することによってマルチリンガルに使うことができますので, 当然日本語のテキストも outtext, outtextxy で出力することができます. 日本語のコードは EUC, JIS, シフト JIS のいずれでも可能です.

サンプルプログラム hello2.c

```
#include <stdio.h>
#include <graphics.h>
main()
{
  initgraph( &gd, &gm, "" );
  (中略)
  settextstyle( DEFAULT_FONT, HORIZ_DIR, 1 );
  outtextxy( cx, cy, "ようこそ XBGI へ" );
  cy += textheight("H");
  outtextxy( cx, cy, "漢字 (JIS,MS 漢字,EUC)" );
  xbgiloop();
}
```

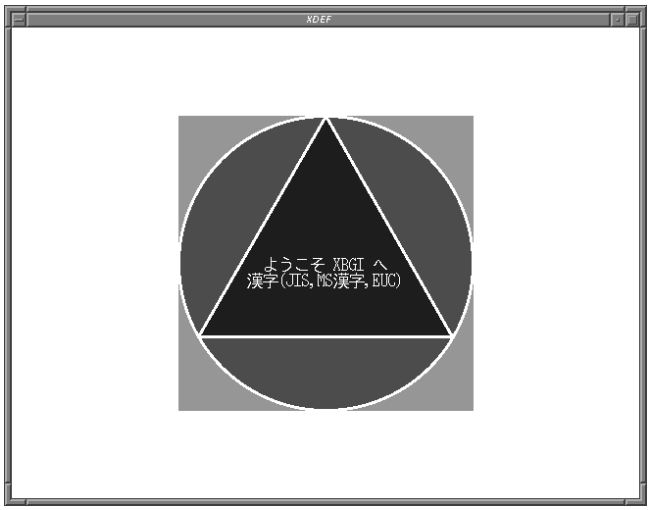


図 4.7: プログラム hello2.c の実行画面

このサンプルプログラムを動かしてみてください. 正しく漢字が表示されない場合, あなたの WS の X は国際化機能が正しくインストールされていない可能性があります. くわしくはシステム管理者に,「ここの X は国際化機能が使えますか」と尋ねるとよいでしょう. もし, X の国際化機能が使えない場合でも, X11R4 の方法で日本語の出力が可能なように, XBGI は作成されています. この場合, XBGI システムをインストールするとき, ENABLE_MULTILINGUAL を定義するかわりに ENABLE_BILINGUAL を定義してインストールしてください. デフォルトで日本語の出力ができるようになります. いずれも, X が日本語のフォントをロードするのに若干の時間がかかりますので了承ください.

おわりに

XBGI は PC で市販されている Turbo C/Borland C が供給する Borland Graphics Interface (BGI) を WS へ移植しました. BGI は PC 上で非常に互換性の高いグラフィックインターフェースです. XBGI は BGI を X Window System の Xlib を利用してシミュレートしました. そのため, さまざまな WS 上でも BGI を稼働させることができるでしょう. さらに X Window がウィンドウシステムであることから, XBGI でも複数の BGI 環境の構築を可能にしました. その結果, 非常に簡便な方法で複数のグラフィック画面への描画が可能になりました.

BGI のウィンドウシステム上でのシミュレートが可能であったのは, ひとえに Xlib が非常にきめの細かい処理をサポートしているからです. もう一段上位レベルのライブラリでは, BGI のシミュレートは不可能です. 例えば Motif や X View などでは, さまざまなウィジットのためのイベント処理を行なわせるため, システムにメインループを明け渡してしまいます. BGI 用に書かれたプログラムはそのようなコーディングを想定していないので, 標準的なテクニックでは Motif や X View への BGI の実現は難しいでしょう.

XBGI の目的は X Window のようなウィンドウシステムに簡便な描画手段を提供することにあります. 例えば, ユーザーが数値計算などで C 言語を扱う場合, そのごく近い延長上にグラフィックインターフェースを提供すれば, ユーザーはプログラムの主題から離れることなくコーディングを行なうことができます. 多くのアルゴリズム入門書や数値計算の解説本, 画像処理の基本などを解説する書物などが Turbo C を使ってコーディングされていることからも, XBGI はそのようなベーシックなニーズに答えうるシステムであると確信します. もちろんこのシステムは, 数値計算を主体とする研究目的にも非常に適しているので, グラフィックスを TEX などに取り込むためのファイル形式へ保存したいものです. 現バージョンでもビットイメージ (bitmap, pixmap) をファイルに保存するための関数は用意されていますが, 今後のバージョン (Ver.3 以後) では, ライン情報などを扱うドロー系ツールやページ記述言語との親和性を高めたいと思っています. その場合, より高精度の出力を得たいときには, §3.3 で解説した BGI ライブラリの倍精度グラフィックスが必要になります. さらに, BGI,XBGI 対応の BGI ライブラリも今後バラエティを増やしていく予定です.

XBGI Version 2 や BGI ライブラリの不都合な点や提案がありましたら, taiji@henon.d.dendai.ac.jp までメールを下されば幸いです.

参考文献

[1] Borland International 著, 『Turbo C++ ライブラリ リファレンスガイド Vol 1 』 株式会社ボーランド ジャパン 1990

[2] Borland International 著, 『Turbo C++ ライブラリ リファレンスガイド Vol 2 』 株式会社ボーランド ジャパン 1990

[3] Oliver Jones 著, 西村 亨 監修, 三浦 明美・ドキュメントシステム 訳 『X Window ハンドブック』 アスキー出版局 1990

[4] 安居院 猛, 永江 孝規 共著, 『X アプリケーションプログラミング 1 — Xlib 編 』 新紀元社 1992

[5] 千葉 則茂, 村岡 一信, 小沢 一文, 海野 啓明, 『C アルゴリズム 全科 — 基礎からグラフィックスまで』 啓学出版 1992

[6] 河西 朝雄 著, 『C 言語による はじめてのアルゴリズム入門』 技術評論社 1992

[7] Douglas Young 著, 川手 恭輔 訳, 『X Toolkit プログラミング』 トッパン 1990

[8] Adobe Systems 著, 野中 浩一 訳, アスキー出版技術部監修, 『Post Script チュートリアル & クックブック』 アスキー出版局 1989

[9] Adobe Systems 著, アドビ システムズ ジャパン 監訳, 『Post Script リファレンス・マニュアル 第2版 』 アスキー出版局 1991

[10] Stephen Wolfram, "Mathematica — A System for Doing Mathematics by Computer" Addison Wesley 1990

[11] the Math Works Inc., "MATLAB — High - Performance Numeric Computation and Visualization Software" the Math Works Inc. 1992

[12] William H. Press, Brian P. Flannery, Saul A. Teukolsky and William T. Vetterling, "Numerical Recipes in C — the Art of Scientific Computing" Cambridge 1988

[13] Thomas H. Cormen, Charles E. Leiserson and Ronald L. Rivest, "Introduction to Algorithms" the MIT press 1992

[14] Ian Stewart, "Does God Play Dice ? — The New Mathematics of Chaos" Penguin Books 1989

付録 A

XBGI 拡張関数

表 A.1: **graphics.h** で宣言されている拡張関数とその機能

関数	機能
イベント，リクエスト処理	
xbgiloop()	XBGI 標準イベントループ
xflush()	リクエストのフラッシュ
環境設定	
setmaxx(int x)	X 軸最大値の指定
setmaxy(int y)	Y 軸最大値の指定
getsizexy(int *width, int *height)	画面サイズの取得
setsizexy(int width, int height)	画面サイズの指定
getgeometry(int *x, int *y, int *width, int *height)	画面ジオメトリの取得
setgeometry(int x, int y, int width, int height)	画面ジオメトリの指定
setmaxcolor(int n)	使用可能な色の最大値の指定
settitle(char *title)	ウィンドウ名の設定
seticonname(char *name)	アイコン名の設定
seticon(char *filename)	アイコンの設定
setbackingstore(int flg)	バッキングストアの設定
_initgraph(int argc, char *argv[], char *pd)	コマンドライン引数による構築
_setrgbpalette(int cnum, unsigned red, unsigned green, unsigned blue)	パレットの RGB 値の設定
XBGI の ID の取得，選択，破壊	
unsigned graphnumber()	XBGI の ID の取得
selectgraph(unsigned id)	XBGI の ID でカレント BGI を指定
termgraph(unsigned id)	XBGI の ID で BGI を破壊
ユーティリティ	
savegraphtoxbm(char *filename)	画面を X Bitmap 形式で保存
savegraphtoxpm(char *filename)	画面を X Pixmap 形式で保存

表 A.2: xbgiloop 用のイベントプロシージャの設定関数

関数	機能
set_button_press_handler(int (*proc)(int button,int x,int y))	ボタンが押された時の処理関数を設定
set_button_release_handler(int (*proc)(int button,int x,int y))	ボタンが離された時の処理関数を設定
set_mouse_motion_handler(int (*proc)(int button,int x,int y))	マウスが動いた時の処理関数を設定
set_page_enter_handler(int (*proc)(int page))	マウスがページに入った時の関数を設定
set_page_leave_handler(int (*proc)(int page))	マウスがページから出た時の関数を設定
set_key_press_handler(int (*proc)(int key))	キーが押された時の処理関数を設定
set_configure_handler(int (*proc)(void))	ウィンドウのリサイズ時の処理関数を設定
set_expose_handler(int (*proc)(void))	ウィンドウが現れた時の処理関数を設定
set_noevent_handler(int (*proc)(void))	何のイベントも無い時の処理関数を設定

表 A.3: **graphix.h** で宣言されている拡張関数とその機能

関数	機能
オブジェクト xbgi の取得，選択，破壊	
xbgi *creategraph(int *gd, int *gm, char *pd)	グラフィックシステムの構築とオブジェクト xbgi 取得
xbgi *_creategraph(int argc, char *argv[], char *pd)	コマンドライン引数による構築
xbgi *thisgraph()	カレントオブジェクト xbgi を取得
currentgraph(xbgi *this)	オブジェクト xbgi でカレント BGI を指定
destroygraph(xbgi *this)	オブジェクト xbgi で BGI を破壊

付録 B

BGI 標準関数

表 B.1: BGI システム関数とその機能

関数	機能
BGI システム構築，設定，破壊	
initgraph(int *gd, int *gm, char *pd)	グラフィックシステムの構築
detectgraph(int *gd, int *gm)	グラフィックシステム環境の自動検出
graphdefaults()	グラフィック設定値をデフォルトに設定
closegraph()	グラフィックシステムの破壊

列挙型 graphics_drivers

列挙定数名	意味
DETECT	自動検出
BGI	XBGI ドライバ

列挙型 graphics_modes

列挙定数名	意味
BGIMONO	モノクロ
BGIGRAY	グレイスケール
BGICOLOR	カラー

エラー処理	
int graphresult()	エラーコードの取得
char *grapherrormsg(int errcode)	エラーメッセージの取得

列挙型 graphics_errors

列挙定数名	意味
grOk	エラーなし
grNoInitGraph	BGI 環境が構築されていない
grError	グラフィックエラー

BGI システム環境の設定，取得	
int getmaxx()	X 軸最大値の取得
int getmaxy()	Y 軸最大値の取得
char *getdrivername()	グラフィックドライバ名の取得
setgraphmode(int mode)	グラフィックモードの設定
int getgraphmode()	グラフィックモードの取得
char *getmodename(int mode)	グラフィックモード名の取得
int getmaxmode()	グラフィックモード最高値の取得
getmoderange(int gd, int *lmode, int *hmode)	グラフィックモードレンジの取得
unsigned setgraphbufsize(int bufsize)	グラフィックバッファ容量の設定（未サポート）
int installuserdriver(char *name, int (*func)(void))	グラフィックスドライバの登録（リザーブド）
int registerbgidriver(void (*driver)(void))	グラフィックスドライバの登録

表 B.2: グラフィックス画面のための関数とその機能

関数	機能
ページ	
`cleardevice()`	グラフィック画面のクリア
`setactivepage( int page )`	描画ページの指定
`setvisualpage( int page )`	表示ページの指定
`restorecrtmode()`	初期化前の画面モードにリストア（未サポート）
ビューポート	
`clearviewport()`	ビューポートのクリア
`setviewport( int l, int t, int r, int b, int clip )`	ビューポートの設定
`getviewsettings( struct viewporttype *viewport )`	ビューポート情報の取得

構造体 `viewporttype`

```
struct viewporttype {
    int left, top;       左上の座標 x, y
    int right, bottom;   右下の座標 x, y
    int clip;            クリッピング真偽
};
```

表 B.3: 描画のための関数とその機能

関数	機能
ラインスタイル	
`setlinestyle( int style, unsigned pat, int thick )`	ラインスタイルの設定

列挙型 line_styles

列挙定数名	意味
SOLID_LINE	実線
DOTTED_LINE	点線
CENTER_LINE	一点鎖線
DASHED_LINE	破線
USERBIT_LINE	ユーザー定義のラインスタイル

構造体 linesttingstype

```
struct linesettingstype {
    int linestyle;          ラインスタイル
    unsigned upattern;      ラインパターンビット
    int thickness;          ライン幅
};
```

`getlinesettings( struct linesettingstype *lineinfo )`	ラインスタイル情報の取得
`setwritemode( mode )`	直線の描画モードの設定

列挙型 write_ops

列挙定数名	意味
COPY_PUT	上書き
XOR_PUT	排他的論理和
OR_PUT	論理和
AND_PUT	論理積
NOT_PUT	否定

カレント座標	
`int getx()`	カレント座標の X 座標の取得
`int gety()`	カレント座標の Y 座標の取得
`moveto( int x, int y )`	カレント座標の移動
`moverel( int dx, int dy )`	カレント座標の相対移動
直線，矩形	
`line( int x1, int y1, int x2, int y2 )`	直線描画
`lineto( int x, int y )`	カレント座標から指定位置へ直線描画
`linerel( int dx, int dy )`	カレント座標から相対位置へ直線描画
`rectangle( int l, int t, int r, int b )`	直方形の描画
`drawpoly( int numpoints, int *polypoints )`	多角形の描画
円，弧など	
`arc( int x, int y, int stangle, int endangle, int r )`	円弧の描画
`circle( int x, int y, int r )`	円の描画
`ellipse( int x, int y, int stangle, int endangle, int xr, int yr )`	楕円弧の描画
`getarccoords( struct arccoordstype *arccoords )`	arc で描いた円弧情報の取得

構造体 arccoordstype

```
struct arccoordstype {
    int x, y;             円弧の中心座標 x, y
    int xstart, ystart;   円弧の開始点座標 x, y
    int xend, yend;       円弧の終了点座標 x, y
};
```

`setaspectratio( int xasp, int yasp )`	アスペクト比の設定
`getaspectratio( int *xasp, int *yasp )`	アスペクト比の取得

表 B.4: フィリングのための関数とその機能

関数	機能
フィルパターン	
setfillstyle(int pattern, int color)	フィルスタイルの設定
setfillpattern(char *pat, int color)	ユーザーフィルパターンの設定
getfillsettings(struct fillsettingstype *fillinfo)	フィルスタイルの取得

構造体 fillsettingstype

```
struct fillsettingstype {
    int pattern;            塗りつぶし種類
    int color;              塗りつぶし色
};
```

getfillpattern(char *pat)	フィルパターンの取得

列挙型 fill_patterns

列挙定数名	意味
EMPTY_FILL	バックグラウンド色でフィル
SOLID_FILL	ベタフィル
LINE_FILL	線模様フィル
LTSLASH_FILL	スラッシュフィル
SLASH_FILL	太いスラッシュフィル
BKSLASH_FILL	太いバックスラッシュフィル
LTBKSLASH_FILL	バックスラッシュフィル
HATCH_FILL	淡い格子フィル
XHATCH_FILL	太い斜め格子フィル
INTERLEAVE_FILL	インターリーブ線フィル
WIDE_DOT_FILL	間隔の広い点フィル
CLOSE_DOT_FILL	間隔の狭い点フィル
USER_FILL	ユーザー定義フィル

矩形	
bar(int l, int t, int r, int b)	直方形のフィル
bar3d(int l, int t, int r, int b, int depth, int topflag)	3 次元バーの描画
fillpoly(int numpoints, int *polypoints)	多角形のフィル
円, 楕円など	
fillellipse(int x, int y, int xr, int yr)	楕円形のフィル
pieslice(int x, int y, int stangle, int endangle, int r)	扇形のフィル
sector(int x, int y, int stangle, int endangle, int xr, int yr)	楕円の扇形の描画とフィル
領域フィル	
floodfill(int x, int y, int border)	指定色閉領域のスキャンフィル

表 B.5: ピクセル, イメージのための関数とその機能

関数	機能
ピクセル	
unsigned getpixel(int x, int y)	(X,Y) 位置のピクセルカラーの取得
putpixel(int x, int y, int color)	ピクセルのプロット
イメージ	
unsigned imagesize(int l, int t, int r, int b)	ビットイメージの格納に要するバイト数の取得
getimage(int l, int t, int r, int b, void *bitmap)	指定領域のビットイメージの取得
putimage(int l, int t, void *bitmap, int op)	ビットイメージの画面への出力

表 B.6: グラフィックテキストのための関数とその機能

関数	機能
settextstyle(int font, int dir, int charsize)	テキストスタイル（フォント，方向，サイズ）設定
settextjustify(int horiz, int vert)	テキスト位置合わせ（水平，垂直）設定

列挙型 font_names

列挙定数名	意味
DEFAULT_FONT	デフォルトフォント
TRIPLEX_FONT	トリプレックスフォント
SMALL_FONT	スモールフォント
SANS_SERIF_FONT	サンスセリフフォント
GOTHIC_FONT	ゴチックフォント
SCRIPT_FONT	スクリプトフォント
SIMPLEX_FONT	シンプレックスフォント
TRIPLEX_SCR_FONT	トリプレックススクリプトフォント
COMPLEX_FONT	コンプレックスフォント
EUROPEAN_FONT	ヨーロピアンフォント
BOLD_FONT	ボールドフォント

定数 text_dirs

定数名	意味
HORIZ_DIR	左から右
VERT_DIR	下から上

列挙型 text_just

列挙定数名	意味
CENTER_TEXT	中央合わせ
LEFT_TEXT	左合わせ
RIGHT_TEXT	右合わせ
BOTTOM_TEXT	下合わせ
TOP_TEXT	上合わせ

gettextsettings(struct textsettingstype *texttypeinfo) | テキストスタイルの取得

構造体 textsettingstype

```
struct textsettingstype {
    int font;           フォント種類
    int direction;      テキストの向き
    int charsize;       フォントサイズ
    int horiz;          水平位置合わせ
    int vert;           垂直位置合わせ
};
```

int textheight(char *textstring)	テキストの高さの取得
int textwidth(char *textstring)	テキストの幅の取得
outtext(char *textstring)	カレント座標へのテキストの出力
outtextxy(int x, int y, char *textstring)	指定座標へのテキストの出力
setusercharsize(int mx, int dx, int my, int dy)	テキストフォントサイズの設定（リザーブド）
int installuserfont(char *name)	テキストフォントの登録（リザーブド）
int registerbgifont(void *(font)(void))	テキストフォントの登録（リザーブド）

表 B.7: カラー, パレットのための関数とその機能

関数	機能
カラー	
`setcolor( int color )`	前景色の設定
`setbkcolor( int color )`	背景色の設定
`int getcolor()`	前景色の取得
`int getbkcolor()`	背景色の取得
`int getmaxcolor()`	使用可能な最大のカラー値の取得

列挙型 COLORS

列挙定数名	意味	列挙定数名	意味
BLACK	黒	DARKGRAY	暗い灰色
BLUE	青	LIGHTBLUE	明るい青
GREEN	緑	LIGHTGREEN	明るい緑
CYAN	水色	LIGHTCYAN	明るい水色
RED	赤	LIGHTRED	明るい赤
MAGENTA	紫	LIGHTMAGENTA	明るい紫
BROWN	茶	YELLOW	黄色
LIGHTGRAY	灰色	WHITE	白

パレット	
`struct palettetype *getdefaultpalette()`	デフォルトのパレット情報の取得
`getpalette( struct palettetype *palette )`	パレット情報の取得

構造体 palettetype

```
struct palettetype {
    unsigned char size;                 パレットのサイズ
    unsigned char colors[MAXCOLORS+1];  割り当てられている色番号
};
```

`int getpalettesize()`	使用可能なパレット数の取得
`setallpalette( struct palettetype *palette )`	パレット全色の設定
`setpalette( int cnum, int color )`	パレット色の設定
`setrgbpalette( int cnum, int red, int green, int blue )`	パレットの RGB 値の設定