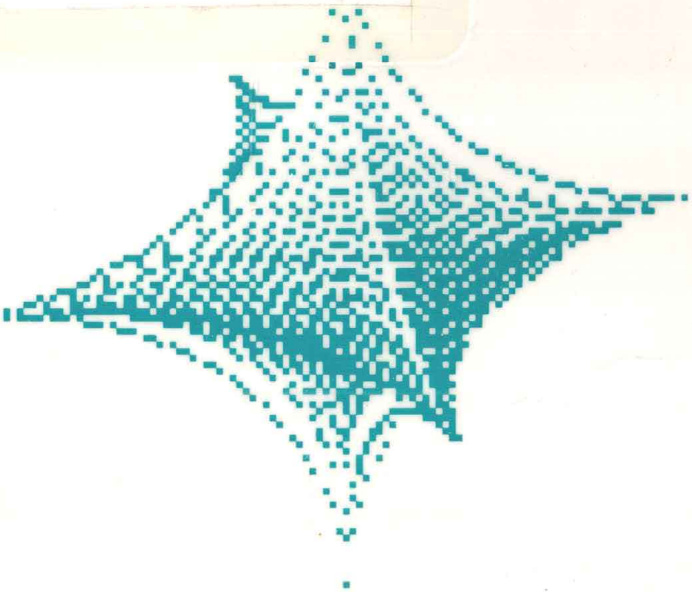
コンピュータグラフィックス講義

工学博士 吉木田直 著



コロナ社

コンピュータ
グラフィックス講義

工学博士 青木 由直 著

コロナ社

——著者略歴——

1964年　北海道大学工学部電子工学科卒業
1966年　北海道大学大学院修士課程修了（電子工学専攻）
　　北海道大学講師
1967年　北海道大学助教授
1972年　工学博士（北海道大学）
1979年　北海道大学教授
　　現在に至る

コンピュータグラフィックス講義
Lectures on Computer Graphics

1997年11月10日　初版第1刷発行

検印省略

著　者　青木由直（あおきよしなお）
札幌市西区西野6条10丁目10-27
発行者　株式会社　コロナ社
代表者　牛来辰巳
印刷所　壮光舎印刷株式会社

112　東京都文京区千石4-46-10
発行所　株式会社　**コロナ社**
CORONA PUBLISHING CO., LTD.
Tokyo　Japan
振替　00140-8-14844・電話(03)3941-3131(代)

ISBN 4-339-02347-7　（青田）　（製本：愛千製本所）
Printed in Japan

まえがき

近年，コンピュータグラフィックス（CG）技法の利用の拡大に従って，大学においてもCGの講義の必要に迫られている。著者もこの2, 3年，学部と大学院でCGの講義を行ってきており，本書はその講義を土台にして書かれている。講義では，C言語を利用してスクリーンに点を打ち，線を描くことから始めて，2次元図形，3次元立体モデルのCGについて，実際にプログラムを実行しながら技法の原理の説明を行っており，本書でもこのスタイルを踏襲した。したがって，本書の目的は，まずコンピュータにCGの原理に基づいてCG画像を描くことであり，CGの実際を学ぶことに配慮したつもりである。

著者の使用しているパソコンはノートタイプのもので，MS-DOSを利用している。したがって，本書のプログラムはその制約のもとにある。特に，MS-DOSにおけるメモリの制約は，得られるCG画像の分解能に大きく制限を与え，最近のCGのツールを利用して質の良いCG画像を作成することに比べると，本書のCG画像は見劣りのするものである。しかし，CG技法の原理に基づいて，手持ちのパソコンで実際にプログラムを実行させながらCG技法を理解していくことが，CGの原理や用語の解説のみでは得られないCGの理解につながることなのだと著者の経験から強調できる。

このような事情も手伝って，本書では画像の質にはこだわっていない。これに関連して，カラー画像に関しても本書では触れておらず，この点もCG技法としては十分ではない点をお断りしておく。とはいえ，CG技法の一つであるマッピング技法等では，メモリを大量に使用せねばならず，本書でも一部UNIXシステムを用いて処理を行っており，不十分ではあるが，UNIXによる画像処理についても述べている。また，C言語のプログラムにスペースをとられ，CG技法の全体を網羅してはいないきらいもある。この点は巻末にも引用してある

他のCGの書籍を参考にしていただきたい。

本書は，外国旅行中にもノートパソコンで執筆するなど時間のとれないなかで書かれており，このためC言語のプログラム等が精練されたものとはいえず，不備な点が十分予想される。しかし，手持ちのパソコンでCGの勉強ができる点については，著者自らノートパソコンでプログラムの作成やデバッグを行っているので，この点では最初のもくろみは実現できており，読者の役に立つものと確信している。

執筆に際しては，2, 3の方々に著者のプログラム能力不足を補っていただいており，著者の研究室の北海道大学工学部情報メディア工学講座の棚橋 真助手にはプログラムに関していろいろ教えていただいた。同講座の川嶋稔夫助教授をはじめ，北海道大学工学研究科大学院生，青木直史君，姜 錫君，工藤庸二君，その他の学生諸君にも直接的，間接的に本書の随所でお世話になっている。図面作成やTEXの入力作業には著者の秘書役の宮口広美嬢に負うところが大きい。本書の出版では，コロナ社の方々には著者の希望を最大限通していただいた。これらの方々への謝辞を最後に記して，著者の感謝としたい。

1997年9月

スクリーンの白黒反転を実社会の事件に垣間見ながら

青木 由直

目　　次

1.　C言語と点および線のグラフィックス

2.　直線と曲線の表現

3.　アフィン変換と2次元図形処理

4. 2次元光線追跡

5. 3次元図形表現と透視変換

6. 隠線および隠面処理

7.　3次元形状のモデリングとレンダリング

8.　シェーディングと影付け処理

9.　画像データ表現とマッピング

10. フラクタル図形処理

1

C言語と点および線のグラフィックス

本章では，グラフィックスの基本となる点や線をC言語でどのように表現するのか，さらにC言語の利用に関して，その基本的な事柄について解説する。

1.1 画素と画面

コンピュータグラフィックス (computer graphics: CG) とは，コンピュータの表示装置であるCRT (cathod ray tube) やLCD (liquid crystal display) のスクリーン面に点や線で図形を描くことである。スクリーン面の図形は，**ピクセル** (pixel) と呼ばれる図形表示の最小単位の画素に，白黒 (場合によっては階調のある灰色) やカラーの画面データを与えることで描かれる。

ここで，画素の数はコンピュータにより異なってくる。本書の大方のプログラムはノートパソコンでも処理できるものであり，ここで用いているノートパソコンのスクリーン面の大きさは，**図 1.1** に示すように，横640，縦400のピクセル数のものである。したがって，これがグラフィックス画面の大きさとなる。ただし，画像処理に関係する部分では，大容量のメモリが必要になるため，一部ワークステーションを用いており，X-windowなどの使用に際しては，さらに大きなサイズの画面の利用が可能である。これらの画面の大きさがどのくらいのメモリ容量を必要とするかは，簡単な計算で得られるので演習問題としておく (演習問題 [1.1])。

2次元の画面上で画素の位置を定めるためには，座標軸を定める必要があり，スクリーン画面では，図 *1.1* に示すように左上隅が原点 (0, 0) で，X, Y 両軸が図に示すように定められている。この座標系はスクリーンに固定されており，**物理座標系** (physical coordinate system) と呼ぶ。

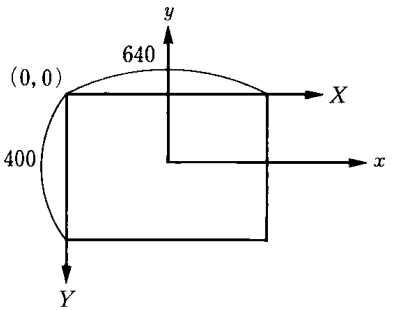


図 *1.1*　スクリーンと座標系

これに対して，原点を画面の任意の点に定めたほうが図形を描く上で便利であり，図 *1.1* の x, y 両軸のように座標軸を移して図形処理をする場合が多い。このような座標系を**論理** (logical) **座標系**と呼んでいる。本書では論理座標系で図形を記述し，それがそのままスクリーン座標に表示されるようにプログラム上で変換を行っている。

1.2　C 言 語 と CG

本書では各種 CG 技法に関して，**C 言語**によるプログラミングを行っている。しかし，C 言語の解説書ではないので，最初に C 言語の簡単な説明を行い，引用プログラムが現れるたびに新しい関数や処理方法の説明を加えていく。

C 言語では，プログラムに使用する変数や関数 (の演算結果) の型を，使用に先立ち定義しておく必要がある。本書で用いている **MS-DOS** 上の C 言語 (Microsoft C および IBM C2) では，グラフィックスに必要な定数，変数や関数名とその型を graph.h という**ヘッダー・ファイル**にまとめている。このヘッダー・ファイルに名前と型が定義されている**グラフィックス関数**を利用しようと思えば，プログラムの最初に# include <graph.h>と書くとよい。

そのほかのヘッダー・ファイルとしては，スクリーンにプログラムを表示するような標準的な入出力に関する処理については stdio.h，数学的な関数の値を求める処理は math.h を利用するとよい。

グラフィックス処理に関して，利用上便利な自家製の関数を作成し，これを一

つのファイルとしてまとめておき，別のプログラムでも利用する場合には，新しくヘッダー・ファイル，例えばgraphlib.h を設けて自家製の関数名と型の定義をこのヘッダー・ファイルに書いておく。ヘッダー・ファイルに定義された関数の処理の本体は，例えば，graphlib.c のように別のファイルとしておき，コンパイル時に main 関数が書かれたファイルと一緒にコンパイルして，実行形式のファイルを作成する。

この処理操作を簡単にするため make コマンドが用意されており，**make ファイル**を書式に従って作成し，make コマンドを実行することにより実行形式のファイルを作ることができる (演習問題 [1.2])。

ここで，スクリーン面上の i, j で与えられる位置に点を表示するグラフィックス関数 (命令) _setpixel(i, j) を用いて，半径 r の円を描く C 言語のプログラムを**リスト 1.1** に示してみる。

リスト 1.1 点により円を描くプログラム

```
/* en_dot.c */
#include <stdio.h>
#include <graph.h>
#include "hcopy.h"
struct videoconfig far *config;
void main()
{
  int i,j,r;
  printf("Input radius of circle r=");
  scanf("%d",&r);
  g_init();
  for(i=0;i<640;i++){
    for(j=0;j<400;j++){
      if((i-320)*(i-320)+(j-200)*(j-200)>r*r-50  &&
         (i-320)*(i-320)+(j-200)*(j-200)<r*r+50 )
         _setpixel(i,j);
  } }
  g_end();
}
```

```
g_init()
{
  int mode;
  mode=_VRES16COLOR;
  _setvideomode(mode);
  _getvideoconfig(config);
}

g_end()
{
  getch();
  _displaycursor(_GCURSORON);
  _displaycursor(_GCURSOROFF);
  _setvideomode(_DEFAULTMODE);
}
```

リスト 1.1 のプログラムの hcopy( ) は，ハード・コピーを得る関数であり，この関数はヘッダー・ファイル hcopy.h を介して利用できる (演習問題 [1.3])。

C 言語は関数の組み合わせで処理が行われ，例えば，処理の全体の流れは関数 main( ) で表現され，main 関数中に個々の処理に対応した関数が書かれている。リスト 1.1 の例では，printf(＊)(＊印は ( ) 内の引数等を省略して表している) は，( ) 内のダブルクォーテーション「 」で挟まれたメッセージをスクリーン面に表示し，scanf("%d",&r) は，キーボードから入力される整数型の数値を変数 r に取り込む関数である。ここで&r と書かれているのは変数 r のアドレスを表している。このように変数のアドレスを介して変数にデータを与える方法は C 言語の特徴的なもので，後出のプログラムにおいて再度説明する。

関数 g_init( )，g_end( ) ではそれぞれグラフィックス処理を開始する場合と終了する場合のパラメータ設定などを行っている。関数 g_init( ) 中の _setvideomode (＊) はスクリーンのモードを決定し，_getvideoconfig(＊) はグラフィックス処理に際しての各種パラメータ情報を得るものである。後出のプログラムでは，g_init( )，g_end( ) はファイル graphlib.c にまとめられており，ヘッダー・ファイル graphlib.h を介して用いられるようにしている。

C 言語では，変数の型をあらかじめ定義しておく必要がある。リスト 1.1 の

main 関数中の int は−32 768 ～ 32 767 の範囲の整数を表している。リスト 1.1 は中心がスクリーン画面の中央 ($X = 320$, $Y = 200$) にあり，整数 r の値をキーボードから入力して $r*r-50 < X^2+Y^2 < r*r+50$ を満足する画面上の点 (X, Y) を表示していくプログラムである。

例えば，$r = 100$ とした場合の出力結果を**図 *1.2*** に示す。図 *1.2* には半径 100 の円が描かれているが，その外側にも円が描かれている。これは int 型整数の範囲が，前述のように制限があるためで，例えば $i = 200$, $j = 100$ の点では $X^2+Y^2 = 200\times 200+100\times 100 = 50\,000$ となり，int 型整数の範囲を超え，32 768 からは改めて−32 768 として数値を増加させて (−32 768 からは 32 767 として数値を減少させて)，計算し直すためである。

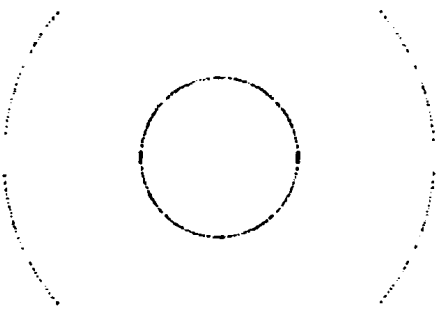


図 *1.2* 点で描かれた円

この int 型の範囲の制限に対処するためには，変数の型を変えればよく，例えば，倍長整数型にして int → long int とすればよい。これに従って，リスト 1.1 において scanf("%d",&r) → scanf("%ld",&r) と変える必要がある。

反復処理 for 文中の条件文である if 文を論理積 && から論理和 || に変更して手直しすれば，図 *1.2* の白黒を反転させたものが描ける。これについては演習問題としておく (演習問題 [1.4])。

1.3 座標点の表現

リスト 1.1 に struct で定義されている変数の型がある。これは**構造体**と呼ばれるもので，複数の変数をひとまとめにした変数の型である。図 *1.1* のスクリーン面での点は座標軸を定め，x, y 座標を与えて表現できるので，座標点を構造体で表してみる。構造体 p はつぎのように定義される。

```
struct {int x,y;} p;
```

この構造体は，二つの整数の変数の組 (x, y) から構成されていることを意味している。もし，上で定義された構造体 p をスクリーン面上の点に対応させると，この点の x, y 座標はそれぞれ p.x, p.y で表される。

リスト 1.1 のプログラムを構造体を用いて書き直すと，**リスト 1.2** のようになる。

リスト 1.2　構造体と関数のヘッダー・ファイルへの収納

```
/* en_dot1.c */
#include <stdio.h>
#include <graph.h>
#include "graphlib.h"
#include "hcopy.h"
struct {int x,y;} p;
void main()
{
  int i,j,r;
  printf("Input radius of circle r=");
  scanf("%d",&r);
  g_init();
  for(i=-320;i<320;i++){
    p.x=i;
    for(j=-200;j<200;j++){
       p.y=j;
       if(p.x*p.x+p.y*p.y>r*r-50
         && p.x*p.x+p.y*p.y<r*r+50 )
         _setpixel(i,j);
```

```
    } }
  g_end();
}
```

構造体の構成要素を一度定義して，つぎに使用するときに構造体のタグ名 Pixel を書くだけで構造体変数 p を定義するつぎのような書き方もある。

```
struct Pixel {int x,y;};
struct Pixel p;
```

要素構造体を構成する型の指定の際に，struct と書かなくても構造体変数を定義できるようにしたつぎの定義法もある。

```
typedef struct {int x,y;} Pixel;
Pixel p;
```

本書では基本的にはこの定義法に従った構造体の型定義と変数を用いている (演習問題 [1.5])。

1.4 グラフィックス関数

グラフィックスの基本は点と線である。スクリーン面上で 2 点を定め，2 点間に直線を引く操作では，ペン先を最初の点の上まで運び，ペンをおろしてつぎの点まで線を引く動作と同じことをコンピュータに行わせる。このため，C 言語のグラフィックス関数として，ペン先を移動させる _moveto(*) と，線を描く _lineto(*) の関数があり，これを利用して線を描けばよい。しかし，図 *1.1* に示されたスクリーン座標 X, Y より論理座標 x, y のほうが便利なので，論理座標で考えても描画の際にはスクリーン座標に変換されるようにしておく。

このような座標変換を行った例として，**リスト 1.3** に円の数 n と最終半径 radius を与えて，ら線† を描くプログラムを示す。リスト 1.1 の場合，物理座標系を採用しており，原点が画面の左上にあるのに対して，リスト 1.3 の g_int( )

† 本書では「螺旋」を「ら線」と表記した。

には_setlogorg(org_x,org_y) を加え，原点を org_x, org_y に移動している。具体的な org_x, org_y の値はリスト 1.3 では define 文で与えている。

このプログラムで新しく定義されたペン先を移動させる関数 MoveTo(p) (p はリスト 1.3 において POINT2 として定義された構造体の変数) と，線を描く関数 LineTo(p) において，点 p の y 座標 p.y の符号を変えているのは，図 *1.1* にも示したようにスクリーン座標と論理座標の y 軸の向きが逆であるので，論理座標と同じ y 軸の向きで図形が表示されるようにしているためである。**図 *1.3*** はリスト 1.3 において円の数を $n = 10$，半径を $r = 200$ とした場合に描かれるら線を示している (演習問題 [1.6])。

リスト 1.3　ら線を描くプログラム

```
/* spiral1.c */
 #include <stdio.h>
 #include <math.h>
 #include <graph.h>
 #include "hcopy.h"
 #define org_x 320
 #define org_y 200
 #define PI 3.14159265
 void MoveTo();
 void LineTo();
 int xvar,yvar;
 typedef struct {double x, y;} POINT2;
 main()
 {
   int i,n,ptnumber;
     POINT2 Center,p;
   double theta=0,r,radius,thinc;
   g_init();
   printf("Type in number of circle \n");
   scanf("%d",&n);
   printf("Type in radius of spiral \n");
   scanf("%lf",&radius);
   Center.x=0, Center.y=0;
   thinc=2*PI/100;
   ptnumber=100*n;
```