CG プログラミング入門 (9)

Introduction to CG Programming (9)

長島 忍 Sinobu NAGASHIMA

keywords

CG/プログラミング/C言語

1. はじめに

この講座もそろそろ終わりに近づいて、どのような内容を取り上げようか迷っている。本連載は入門者向けということもあり、そのためにプログラムの行数をできるだけ短くしている。面白い内容を取り上げようとするとしてもプログラムの行数が長くなり、処理が煩雑になってくる。そのためなるべく基本的で意味のある内容を考えた結果、今回はマクロ定義を利用した円の描画、円の描画を利用したネフロイドの作図、マウスのドラッグ、メモリ描画の利用、ドラッグ領域のクリップボードへのコピーを取り上げてみたい。たくさんあるようだが、今までの内容と一部重複しているので、過去のまとめと復習を兼ねながら新しいことをやってみたい。

2. マクロの利用

よくマクロという言葉が使われる。Microsoft Office の Excel や Access など、処理を自動的に行うスクリプトの意味でよく使われる。C 言語でのマクロはマクロ定義と呼ばれているもので、プログラムステートメント中の文字列を別の文字列に置き換える機能である。たとえば

```
#define PI 3.1416
```

というように、プログラム中の PI という文字列を 3.1416 という数値に置き換えてからコンパイルする。また単なる文字列の置換だけではなく、計算を含めた置き換えが可能である。

```
#define SANJOU(x) x*x*x
```

このように関数を作成する代わりにマクロ定義で済ませることもできる。なおマクロ定義では関数名と区別するために大文字がよく使われるが小文字だけでもエラー

にはならない。

このマクロ定義を利用して、円の描画をわかりやすく記述してみたい。円の描画は以前にも取り上げたが Arc という Windows API を利用する。Arc 呼び出し方は、Arc (hdc, x1, y1, x2, y2, x3, y3, x4, y4) のように 9 個もの引数がある。最初の引数は、デバイスコンテキストであり、2 番目から 4 組の xy 座標が並ぶ。それぞれ、円に外接する正方形の左上の座標、右下の座標、円弧開始の座標、円弧終了の座標になっている。

円を描画する場合は、中心の xy 座標と半径 r の 3 つだけを指定したいので、次のようなマクロを考えた。

```
#define CIRCLE(x, y, r) Arc(hdc, x-r, y-r, x+r, y+r, x, y-r, x, y-r);
```

このようなマクロ定義で

```
CIRCLE(x, y, r)
```

と記述するだけで円が描画できる。これを利用してネフロイドという図形を描いてみることにする。ネフロイドという図形は、次のように表せる。なおマクロ定義は、関数と違うのでコンパイルしたときに展開される。

```
int i, x, y;
for(i=0; i<20; i++) {
    x = (int)(100*sin(i*9*3.14159265/180));
    y = (int)(100*cos(i*9*3.14159265/180));
    CIRCLE(x+300, y+200, x);
    CIRCLE(300-x, y+200, x);
}
```

なお繰り返し回数の 20 は、片側の円の数で、180 度を 20 で割ると 9 度になるので、計算式に 9 という数字が出てくる。100 というのは基準になる円の半径で、この円は表示されない。

まずここまでを作成し、コンパイルして確認したい。

Visual C++を起動し、ファイルメニューから新規作成を選び、プロジェクトタブを開いてWin32 Applicationを選ぶ。プロジェクト名にはcgex9など適当な名前を入力してOKボタンを押す。次に「標準的な“Hello World”アプリケーション」を選んでいくととりあえずHelloという文字列を表示するプログラムが出来上がる。

ソースファイルのcgex9.cppまたはクラス表示のWndProcをダブルクリックして表示処理部分を探す。

```
case WM_PAINT:
```

という行が表示処理の始まりである。図1のように文字列表示3行分をコメントにするか削除し、ネフロイド表示部分を追加する。この他、sin, cosを利用しているので、ヘッダーファイルのStdAfx.hを開き、math.hというファイルをインクルードするよう指示を出す。

```
// C ランタイム ヘッダー ファイル
#include <stdlib.h>
#include <malloc.h>
#include <memory.h>
#include <tchar.h>
#include <math.h>
```

最後にマクロ定義を入れる場所である。プログラム文中でもかまわないが、体裁をよくするため、他の定義文と一緒にするとわかりやすい。さきほど表示部分を作成したcgex9.cppを開き、先頭にマクロ定義があるので、そこに入れるといいだろう。

```
#define MAX_LOADSTRING 100
#define CIRCLE(x,y,r) Arc(hdc,x-r,y-r,x+r,y+r,x,y-r,x,y-r);
```

```
break;
case WM_PAINT:
    hdc = BeginPaint( hWnd, &ps);
    // TODO: この位置に描画用のコードを追加してく
    // RECT rt;
    // GetClientRect( hWnd, &rt );
    // DrawText( hdc, szHello, strlen(szHello), &rt,
    int i,x,y;
    for (i=0; i<20; i++) {
        x=(int)(100*sin(i*9*3.14159265/180));
        y=(int)(100*cos(i*9*3.14159265/180));
        CIRCLE(x+300,y+200,x);
        CIRCLE(300-x,y+200,x);
    }
    EndPaint( hWnd, &ps );
    break;
```

図1 ネフロイド表示プログラム

図2 ネフロイド表示結果

図2が表示結果である。なおウィンドウの大きさを固定したい場合は、InitInstanceという関数にある、CreateWindowという関数呼び出しで,CW_USEDEFAULT, 0という引数(2つ並んでいる後ろの方)を600, 450などに変更すればよい。

3. マウスのドラッグ

マウスを利用する場合は、イベントというメッセージを処理するようにプログラムを作る。マウスのドラッグというイベントはないので、ボタンダウンとボタンアップという2つのイベント処理の連携を取って、ドラッグに対する処理を行う。本連載の第6回(図学研究101号)にマウスドラッグによる円描画を取り上げたので、それと同じようにプログラムを作ってみる。

まず、ドラッグする場合は、ボタンを押してそのまま別の場所に移動してからボタンを離す。したがってボタンを押した座標を(ox, oy)とする。またボタンを押したという合図を記憶しておく。そのため、

```
static int ox, oy, updown = 0;
```

という変数宣言をしておく。

ボタンを押したときの処理は次のようになる。

```
case WM_LBUTTONDOWN:
    ox = LOWORD(lParam);
    oy = HIWORD(lParam);
    updown = 1;
    break;
```

ここでox, oyはボタン押下時のxy座標である。画面全体に対する座標ではなく、本アプリケーションウィン

ドウの表示画面の相対座標なので、これが一番処理しやすい。ウィンドウがどこにあっても同じ処理で済むからである。

ドラッグ処理というのは、ボタンを離したときに処理が始まる。したがって

```
case WM_LBUTTONDOWN:
```

のあとに処理する内容を記述する。ボタンを離した場所における xy 座標はボタン押下時と同じ、

```
LOWORD(lParam);
```

```
HIWORD(lParam);
```

にそれぞれ xy 座標が入っている。ここではボタンを離したときに、ドラッグした矩形領域をクリップボードに転送する処理を行う。その処理方法について次に述べる。

4. メモリ描画とクリップボード

メモリ描画については、本連載の第7回（図学研究102号）、第8回（同103号）でも取り上げたので、一度読まれておくと理解が深まると思う。今回はドラッグ領域をメモリの画像に移し、クリップボードとして再利用できることをやってみる。

これらの処理を図3に示す。12種類もの Windows API を利用している。このプログラムの挿入場所は、先ほど case WM_PAINT: ~break; の次の部分である。ボタンダウン（押す）とボタンアップ（離す）のイベントに対応する処理になる。ボタンダウン時は上に述べたとおりである。ボタンアップの処理について説明する。

```
case WM_LBUTTONDOWN:
    ox=LOWORD(lParam); oy=HIWORD(lParam); updown=1;
    break;
case WM_LBUTTONUP:
    if (updown == 0) break;
    int width,height;
    HBITMAP hBitmap; HDC hdc,hMemDC;
    width=abs(ox - LOWORD(lParam));
    height=abs(oy - HIWORD(lParam));
    hdc=GetDC(hWnd);
    hBitmap=CreateCompatibleBitmap(hdc,width,height);
    hMemDC=CreateCompatibleDC(hdc);
    SelectObject(hMemDC,hBitmap);
    BitBlt(hMemDC,0,0,width,height,hdc,ox,oy,SRCCOPY);
    if (OpenClipboard(hWnd)) {
        EmptyClipboard();
        SetClipboardData(CF_BITMAP,hBitmap);
        CloseClipboard();
    }
    ReleaseDC(hWnd,hdc);
    DeleteObject(hBitmap);
    DeleteDC(hMemDC);
    updown=0;
    break;
```

図3 クリップボード設定処理

まず変数 updown が0のときは何もしない。ドラッグというのは、ボタンを押して離す操作なので、ボタンを押してなければ処理をしない。ボタンを押してあれば、xy 座標が ox, oy という変数に代入されている。

次に処理に必要な変数の宣言を行う。文中における宣言は C++ のやり方なので、気になる人は宣言場所を前の方に統一するといいい。

変数 width, height はドラッグした矩形領域の幅と高さである。ボタンを離したときの xy 座標と押下時の xy 座標の差を計算すれば幅と高さがわかる。

次に hdc はデバイスコンテキストである。WM_PAINT とは異なり、GetDC(hWnd); という Windows API を用いて取得する。次に、CreateCompatibleBitmap という長い名前の API を用いて hdc と同じ属性のビットマップ画像領域を用意する。画像の幅と高さにドラッグした矩形領域の幅と高さを用いる。

次に CreateCompatibleDC という API を用いてメモリ描画用のデバイスコンテキストを画面表示と同じ属性で作成する。さらに SelectObject(hMemDC, hBitmap); でビットマップをメモリ描画に割り当てれば準備完了である。

```
BitBlt(hMemDC, 0, 0, width, height, hdc, ox, oy,
SRCCOPY);
```

が画面上でドラッグした領域をメモリに転送する処理である。

次に、OpenClipboard(hWnd) というのは、クリップボードが利用可能かどうかということである。利用可能であれば、EmptyClipboard(); で一度消去し、SetClipboardData(CF_BITMAP, hBitmap); でビットマップを設定する。CloseClipboard(); でクリップボードの利用を中止する。

最後の処理は、利用した資源を解放する処理である。ReleaseDC(hWnd, hdc); は GetDC に対する開放、

DeleteObject(hBitmap); は作成したビットマップを破棄し、DeleteDC(hMemDC); は CreateCompatibleDC に対する開放である。

なお、これら図1や図3のプログラムを含む WndProc という関数の最初に、

```
static int ox, oy, updown = 0;
```

という静的変数宣言が必要である。

プログラムの実行結果は図2と同じである。ここでウィンドウ上をマウスでドラッグする。画面上では何の

図4 ペイントへの貼り付け

図6 Excel への貼り付け

図5 Word への貼り付け

変化も起きないが、ペイントを起動して編集の貼り付けを行うと、図4のようにドラッグした領域が出てくる。同じようにWord（図5）やExcel（図6）、その他Photo ShopやPaint Shopなどの画像作成ソフトへの貼り付けが可能となる。

5. おわりに

今回のプログラムではドラッグ中に何の表示もなく、他のアプリケーションに貼り付けをしてはじめて効果が確認できる。プログラムが長くなるのでこれらの処理を省略したが、WM_MOUSEMOVEというイベントに対応して長方形を描画すれば、ドラッグしている領域がわかって親切である。

●2004年6月8日受付

ながしま しのぶ

立教大学経済学部・教授

1979年東京大学工学部精密機械工学科卒、1985年工学博士。東京大学教養学部図学教室助手、講師、立教大学一般教育部助教授を経て1999年より現職。