CG プログラミング入門 (7)

Introduction to CG Programming (7)

長島 忍 Sinobu NAGASHIMA

keywords

CG/プログラミング/C言語

1. はじめに

最近はパソコンの能力も非常に高くなり、アニメーションやDVDの映像などがごく普通に行われるようになった。1994年だったか日本のアップル社を見学させてもらって当時はまだ珍しかったMacintoshのQuickTimeを見て、これからは動画の時代がやってくるのかと思ったことがあった。それから着実に技術が進歩して、現在では動画は普通の処理に変わった。今回はアニメーション処理で一般的に使用されるメモリへの描画というのを行ってみる。

2. メモリへの描画

アニメーションのような動画が見られるようになったのは、CPUの処理速度が向上したほかに、グラフィックボード、ビデオボード、アクセラレータボードと言われるグラフィック関係の処理が向上したためであろう。それに伴い一般ユーザもアニメーションのプログラムが作れるようになった。アニメーションのプログラムでは1秒間に数回〜30回ほど表示を繰り返すことにより2次元、3次元に限らずいろいろな動きが表現できるようになる。アニメーションでは画面への描画・表示が頻繁に行われ、このような処理をユーザが見ると画面がチカチカして見えると言われている。そのため一般にはメモリに描画し、それを画面のバッファに転送するようにプログラムを作成する。

実際にアニメーションを行うとなると、プログラムがかなり長くなり、特に3次元では座標変換やらシェーディングやらめんどろな処理が必要になるので、ここでは一番基本にあるメモリへの描画ということでいくつかの実験を試みたい。

処理の手順をみていく。まず初めに表示のデバイス属性と同じ画像領域を取得する。これにはCreate

Compatible Bitmap という Windows の API を利用する。

```
hBitmap = CreateCompatibleBitmap(hdc, 100, 100);
```

引数の hdc は表示用のデバイス属性である。その属性と同じ画像領域を作り出す。次の引数、100, 100は作成する画像の幅と高さのピクセル数である。実際には動画を表示する領域の幅と高さを指定する。よく見かけるのは現在の Window の幅と高さを取得して同じ大きさの画像を生成する。

次にデバイスコンテキストと同じ属性をコピーし、これをメモリ用コンテキストとして使用する。さらにこのコンテキストに先ほどのビットマップ画像を割り当てる。

```
hMemDC = CreateCompatibleDC(hdc);
```

```
SelectObject(hMemDC, hBitmap);
```

次にビットマップ画像を初期化する。ここでは白で埋め尽くす。前回にも使用した PatBlt という Windows API で画像の初期化を行う。

```
PatBlt(hMemDC, 0, 0, 100, 100, WHITENESS);
```

0, 0, 100, 100というのは画像の塗りつぶしを行う領域の左上の座標と幅と高さである。WHITENESSは白で塗りつぶしを行うことを意味するが、特定の色で塗るときは、その色のブラシを作成して塗りつぶしを行う。これらの処理でメモリ上の描画領域が生成できた。次にこの描画領域に何か図形の表示を行う。ここでは簡単な描画として放射状の図を作成した。

```
for(i = 0; i < 30; i++){ //メモリへの描画
    MoveToEx(hMemDC, 50, 50, NULL);
    LineTo(hMemDC, cos(i * 6.28/30) * 50 + 50,
            sin(i * 6.28/30) * 50 + 50);
}
```

繰り返しの中にある2つのAPIは始点設定と線の描画を行う。注意するのは両者の一番最初の引数である。一般のデバイスコンテキストではなく、メモリ用デバイスコンテキストを使用する。始点はビットマップ画像の中心座標である(50, 50)である。終点はcosとsinで円周上の座標を計算する。

これでメモリ領域への描画が終わったので、これを表示領域へマッピングする。マッピングには、Windows APIのBitBltを使用する。

```
for(i= 0 ;i<20;i++) //メモリを表示領域に移す
    BitBlt(hdc, (int)((rt.right-100)*rand()/
        RAND_MAX),
        (int)((rt.bottom-100)*rand()/RAND_MAX),
        100, 100, hMemDC, 0, 0, SRCCOPY) ;
```

単に1回だけ表示するのはつまらないので、乱数を用いていろいろな場所に20個表示することにした。

最初の引数は転送先の表示用デバイスコンテキスト、次の2つは転送先表示領域の左上の座標値、次の2つの引数は幅と高さ、次は転送元のメモリデバイスコンテキスト、次の2つは転送元の座標値、最後はラスターオペレーションコードである。ラスターオペレーションコードはSRCCOPYがよく使われる(実際には対応する整数)。あとでいくつかの例を示す。

3. プログラムの作成

この連載講座で毎回やっているように、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Win32Applicationを選ぶ。プロジェクト名にはcgex7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。

表示処理のWM_PAINT:部分を探し出し、上に述べた処理を追加すればよい。この部分のプログラムを図1に示す。なお使用する変数の宣言をまず行う。整数型のi, HDC型のhMemDC, HBITMAP型のhBitmapという変数宣言である。HDC, HBITMAP型はハンドルと呼ばれているポインタの一種であり、整数型と考えてよい。ビットマップ生成時にはかなりのメモリが動的に使用される。このようにビットマップ画像の生成はメモリリソースを使用するので、DeleteObjectでリソースを開放する必要がある。プログラムが終了すればリソースは開放されるが、終了しないまま多量のビットマップ画

図1 プログラムの表示部分

像を生成するとシステムの異常終了などの事態を生じかねない。

最後に、cos, sinを使用するにあたり、math.hというヘッダファイルのインクルードが必要になるので、StdAfx.hというファイルを開き、1行追加する。既存の、

```
#include <stdlib.h>
#include <malloc.h>
#include <memory.h>
#include <tchar.h>
// ...
#include <math.h>
```

を追加する必要がある。これでプログラムが完成する。

4. 実行例

上記プログラムの実行例を図2に示す。これは画面領域に描画したのではなく、メモリ領域に描画してそれを転送したものである。それを示すのが、ところどころ線が消えている部分である。転送元は正方形なので、転送

図2 表示例 (SRCCOPY)

図3 表示例 (SRCAND)

後に元々あった図を隠していることがわかる。

次に画像を転送するときの SRCCOPY というラスターオペレーションコードを SRCAND に変更してみる。この結果を図3に示す。SRCAND は転送元の画像と転送先の画像の画素どうしの論理 AND を取る操作を意味している。白黒の場合は元の線画が表示されてうまくいくが、カラーの中間色の場合はさらに複雑な処理が必要になるのでここでは省略する。ラスターオペレーションは17種類あるので、詳細は Windows API の書籍

図4 表示例 (SRCINVERT)

などを参照されたい。一つの例として、論理 XOR 演算を行う SRCINVERT の表示を図4に示す。

5. おわりに

DVD のビデオ表示などでは、今回紹介した WindowsAPI は使わずに DirectX を使用している。従来の Windows API は処理速度が遅いので、より本格的なアニメーションを実現されたい方は Direct X などを検討されるといいと思う。

●2003年11月9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。