ゲームプログラミング

▶　DirectDraw－第１１回　リージョン　◀

リージョンはDirectDrawの機能ではなくデバイスコンテキストの機能ですが、円形や星形といった矩形以外の図形でクリッピングを行うことができます。

リージョン

DirectDrawのクリッパーは、矩形(長方形)しか設定することができません。矩形以外、たとえば円形や星形といった図形でクリップをすることはできません。このような図形でクリップするには、１ピクセルずつ計算して描画する方法もありますが、デバイスコンテキストのリージョンという機能を使うと簡単に実現することができます。

リージョンを扱うAPIはいくつかあり、矩形、円形、多角形といった基本的なものから、リージョン同士を合成するものもあります。複雑な図形は、リージョンを組み合わせることによって作成することができます。

リージョンはデバイスコンテキストに設定することによって有効になります。DirectDrawの機能ではないため、BltメソッドやBltFastメソッドで転送しても効果が現れません。従来のBitBlt関数やStretchBlt関数で転送する必要があります。そのため、フリップやカラーキーなどといったDirectDrawの機能使用しながら転送することはできません。

リージョンの作成と設定

リージョンは、Windows内部で管理されており、HRGN型のハンドルをとおしてアクセスします。プログラムからはAPIを使ってアクセスし、直接操作はできません。

リージョンを作成するAPIは、以下のように基本図形ごとに用意されており、頂点座標を与えるだけで簡単に作成することができます。

CreateEllipticRgn関数	だ円形リージョンを作成します
CreatePolygonRgn関数	多角形リージョンを作成します
CreateRectRgn関数	長方形リージョンを作成します
CreateRoundRectRgn関数	角の丸い長方形のリージョンを作成します
CombineRgn関数	２つのリージョンを結合して新しいリージョンを作成します

基本図形以外のリージョンは、CombineRgn関数でリージョン同士を合成することによって任意の形状のリージョンを作成することができます。

これらのAPIでリージョンを作成すると、リージョンのハンドルが得られます。このハンドルを使い、SelectClipRgn関数で転送先デバイスコンテキストにリージョンを設定します。設定後にBitBlt関数などで画像を描画すると、リージョンで設定されている領域だけ描画されるようになります。

リージョンを解除するにはSelectClipRgn関数の２つ目の引数にNULLを渡します。必要なくなったリージョンの解放はDeleteObject関数で行います。

○ 課題 ○

リファレンスでリージョンを扱うAPIについて調べながら、リージョンの効果を確認しましょう。

(１)以下のプログラムは、CreateRoundRectRgn関数で作成したリージョンを使ってオフスクリーンサーフェイス１の画像をバックバッファに転送する関数です。どのような結果になるか確認しましょう。

```
/*****************************************************************************************************/
/*                                        リージョン付き転送                                         */
/*****************************************************************************************************/
void RgnBitBltBG()
{
    DDColorFill(DDS_BACKBUF, NULL, RGB(0, 0, 0));                  // バックバッファクリア

    HRGN   hRgn = CreateRoundRectRgn(80, 60, 560, 420, 20, 20);     // リージョン生成
    if(NULL != hRgn) {
```

```
        HDC   hDestDC = DDGetDC(DDS_BACKBUF);
        HDC   hSrcDC  = DDGetDC(DDS_OFFSCRN1);
        if(NULL != hDestDC && NULL != hSrcDC) {
            SelectClipRgn(hDestDC, hRgn);                                  // リージョン設定
            BitBlt(hDestDC, 0, 0, 640, 480, hSrcDC, 0, 0, SRCCOPY); // 転送
            SelectClipRgn(hDestDC, NULL);                                  // リージョン解除
        }
        DDReleaseDC(DDS_OFFSCRN1);
        DDReleaseDC(DDS_BACKBUF);

        DeleteObject(hRgn);                                                // リージョン解放s
    }
}
```

(２)(１)のプログラムを変更し、CreateEllipticRgn関数で作成しただ円形のリージョンが使われるようにしましょう。なお、だ円の大きさは適当でかまいません。

(３)以下のプログラムは、CreatePolygonRgn関数で作成したリージョンを使ってオフスクリーンサーフェイス１の画像をバックバッファに転送する関数です。どのような結果になるか確認しましょう。

```
/*****************************************************************************************************/
/*                                          リージョン付き転送                                         */
/*****************************************************************************************************/
void RgnBitBltBG()
{
    DDColorFill(DDS_BACKBUF, NULL, RGB(0, 0, 0));                     // バックバッファクリア

    // 頂点座標設定
    POINT   ptVertex[5] = {{320, 6}, {457, 429}, {98, 168}, {542, 168}, {183, 429}};

    HRGN    hRgn = CreatePolygonRgn(ptVertex, 5, ALTERNATE);          // リージョン生成
    if(NULL != hRgn) {
        HDC   hDestDC = DDGetDC(DDS_BACKBUF);
        HDC   hSrcDC  = DDGetDC(DDS_OFFSCRN1);
        if(NULL != hDestDC && NULL != hSrcDC) {
            SelectClipRgn(hDestDC, hRgn);                              // リージョン設定
            BitBlt(hDestDC, 0, 0, 640, 480, hSrcDC, 0, 0, SRCCOPY); // 転送
            SelectClipRgn(hDestDC, NULL);                              // リージョン解除
        }
        DDReleaseDC(DDS_OFFSCRN1);
        DDReleaseDC(DDS_BACKBUF);

        DeleteObject(hRgn);                                            // リージョン解放s
    }
}
```

(４)(３)のプログラム中にあるCreatePolygonRgn関数の３つ目の引数を"WINDING"に変更し、リージョンがどのように変更されるかを確認しましょう。

(５)以下の関数を入力してTestProc関数で呼び出すようにプログラムを変更し、動作を確認しましょう。

```
/*****************************************************************************************************/
/*                                            円形ワイプ                                               */
/*****************************************************************************************************/
void CircleWipeBG()
{
    static int      nWipeCnt  = 0;                                 // ワイプカウンタ
    const  int      WIPE_TIME = 300;                               // ワイプ時間
    const  double   RATIO     = (double)nWipeCnt / WIPE_TIME;      // ワイプ比率

    const int   cx = 640 / 2;                                      // 中心ｘ座標
    const int   cy = 480 / 2;                                      // 中心ｙ座標
    const int   r  = (int)sqrt(cx * cx + cy * cy) + 2;             // 最大半径
    const int   r1 = (int)(r * RATIO);

    DDColorFill(DDS_BACKBUF, NULL, RGB(0, 0, 0));
```

```
    HRGN   hRgn = CreateEllipticRgn(cx - r1, cy - r1, cx + r1, cy + r1);
    if(NULL != hRgn) {

        HDC   hDestDC = DDGetDC(DDS_BACKBUF);
        HDC   hSrcDC  = DDGetDC(DDS_OFFSCRN1);
        if(NULL != hDestDC && NULL != hSrcDC) {
            SelectClipRgn(hDestDC, hRgn);
            BitBlt(hDestDC, 0, 0, 640, 480, hSrcDC, 0, 0, SRCCOPY);
            SelectClipRgn(hDestDC, NULL);
        }
        DDReleaseDC(DDS_OFFSCRN1);
        DDReleaseDC(DDS_BACKBUF);

        DeleteObject(hRgn);
    }

    nWipeCnt++;
    if(WIPE_TIME < nWipeCnt)
        nWipeCnt = 0;
}
```

(６)以下の関数を入力してTestProc関数で呼び出すようにプログラムを変更し、動作を確認しましょう。

```
/*****************************************************************************************************/
/*                                          星形ワイプ                                               */
/*****************************************************************************************************/
void StarWipeBG()
{
    static int      nWipeCnt  = 0;                                  // ワイプカウンタ
    const  int      WIPE_TIME = 300;                                // ワイプ時間
    const  double   RATIO     = (double)nWipeCnt / WIPE_TIME;       // ワイプ比率

    const int   wd = 640;                                           // 背景幅
    const int   ht = 480;                                           // 背景高さ
    const int   r  = (int)(sqrt(wd * wd + ht * ht) * 1.1);          // 最大半径
    const int   r1 = (int)(r * RATIO);

    // 星座標設定
    POINT   ptStar[5];
    for(int i = 0; i < 5; i++) {
        const double   PI    = 3.14159265358979323846264338327795;  // 円周率
        const double   ANGLE = 2.0 * PI * i * 2.0 / 5.0;            // 座標角度
        const int      cx    = wd / 2;                              // 中心ｘ座標
        const int      cy    = ht / 2;                              // 中心ｙ座標

        ptStar[i].x = cx + (int)(sin(ANGLE) * r1);
        ptStar[i].y = cy - (int)(cos(ANGLE) * r1);
    }

    HRGN   hRgn = CreatePolygonRgn(ptStar, 5, WINDING);
    if(NULL != hRgn) {
        DDColorFill(DDS_BACKBUF, NULL, RGB(0, 0, 0));

        // 背景転送
        HDC   hDestDC = DDGetDC(DDS_BACKBUF);
        HDC   hSrcDC  = DDGetDC(DDS_OFFSCRN1);
        if(NULL != hDestDC && NULL != hSrcDC) {
            SelectClipRgn(hDestDC, hRgn);
            BitBlt(hDestDC, 0, 0, 640, 480, hSrcDC, 0, 0, SRCCOPY);
            SelectClipRgn(hDestDC, NULL);
        }
        DDReleaseDC(DDS_OFFSCRN1);
        DDReleaseDC(DDS_BACKBUF);

        DeleteObject(hRgn);
    }

    nWipeCnt++;
    if(WIPE_TIME < nWipeCnt)
        nWipeCnt = 0;
}
```