オブジェクト指向と
ゲームプログラミング

▶ コンポーネント編 ― 第６回　リージョン ◀

リージョン

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、画像を長方形や円形といった図形でクリップする(切り抜く)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図形を持つリージョンを作成するものから、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。リージョンを設定したデバイスコンテキスト対してBitBlt関数などのGDI関数で描画を行うと、リージョン内で図形が描画されている部分だけに画像が描画されるようになります。これを利用することで、複雑な図形に沿ってクリッピングできるというわけです。

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、直接アクセスはできません。
リージョンを扱う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。

CombineRgn	２つのリージョン(領域)を合成します
CreateEllipticRgn	楕円形のリージョンを作成します
CreateEllipticRgn Indirect	楕円形のリージョンを作成します
CreatePolygonRgn	多角形のリージョンを作成します
CreatePolyPolygon Rgn	複数の多角形で構成されたリージョンを作成します
CreateRectRgn	長方形のリージョンを作成します
CreateRectRgn Indirect	長方形のリージョンを作成します
CreateRoundRectRgn	角の丸い長方形のリージョンを作成します
EqualRgn	２つのリージョンが同一かどうかを調べます
FillRgn	指定されたブラシを使用してリージョンを塗りつぶします
FrameRgn	リージョンの境界を描画します
InvertRgn	リージョンの色を反転します
OffsetRgn	リージョンを移動します
PaintRgn	選択されているブラシを使用してリージョンを塗りつぶします
PtInRegion	点がリージョンの内側にあるかどうかを判定します
RectInRegion	指定の長方形の一部がリージョンの内側にあるかどうかを判定します
SelectClipRgn	指定のリージョンをクリッピング領域にします
SetPolyFillMode	多角形塗りつぶしモードを設定します
SetRectRgn	リージョンの形を指定の長方形に設定します
SetWindowRgn	ウィンドウのリージョンを設定します

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。
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はDeleteObject関数で解放します。

課　　題

ヘルプやMSDNライブラリ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ビットマップをブロック転送するCTestScene::RgnBitBlt関数です。どのような結果になるか確認しましょう。

・CTestScene.cppに追加

```
/*****************************************************************************************************/
/*                                          リージョン付き転送                                       */
/*****************************************************************************************************/
void CTestScene::RgnBitBlt(const HDC hDestDC, const int DestX, const int DestY,
                           const int Width, const int Height,
                           const HDC hSrcDC,  const int SrcX,  const int SrcY)
{
    HRGN   hRgn = ::CreateRoundRectRgn(80, 60, 560, 420, 20, 20);   // リージョン生成
    if(hRgn != NULL) {
        ::SelectClipRgn(hDestDC, hRgn);                             // リージョン設定
        ::BitBlt(hDestDC, DestX, DestY, Width, Height, hSrcDC,  SrcX,  SrcY,  SRCCOPY);
        ::SelectClipRgn(hDestDC, NULL);                             // リージョン解除
        ::DeleteObject(hRgn);                                       // リージョン解放
    }
}
```

・CTestSceneクラスのメンバに追加

```
 void RgnBitBlt(const HDC hDestDC, const int DestX, const int DestY,
                const int Width,   const int Height,
                const HDC hSrcDC,  const int SrcX,  const int SrcY);
```

・CTestScene::ActivateProc関数を以下のように変更

```
int CTestScene::ActiveProc()
{
    // ここに、機能テストのメイン処理を記述します
    // 内部処理

    // 描画処理
    if(FPSTimer().IsSkip() == false) {
        // ここに、機能テストの描画処理を記述します
        // フレーム構築
        ::BitBlt(m_pBackBuf->GetDC(), 0, 0, m_pBackBuf->GetWidth(), m_pBackBuf->GetHeight(),
                 m_pBG->GetDC(), 0, 0, SRCCOPY);
        ::TransparentBlt(m_pBackBuf->GetDC(), 0, 0, m_pBackBuf->GetWidth(), m_pBackBuf->GetHeight(),
                         m_pChara  ->GetDC(), 0, 0, m_pChara  ->GetWidth(), m_pChara  ->GetHeight(),
                         RGB(0, 0, 0));
        FPSTimer().DrawFPS(m_pBackBuf->GetDC());

        // バックバッファからディスプレイへ転送
        HDC   hDestDC = ::GetDC(GameApp().GetHWnd());
        RgnBitBlt(hDestDC, 0, 0, m_pBackBuf->GetWidth(), m_pBackBuf->GetHeight(),
                  m_pBackBuf->GetDC(), 0, 0);
        ::ReleaseDC(GameApp().GetHWnd(), hDestDC);
    }

    return 0;
}
```

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

(３)次のプログラムは、CreatePolygonRgn関数で作成したリージョンを使ってビットマップをブロック転送するCTestScene::RgnBitBlt関数です。どのような結果になるか確認しましょう。

```
void CTestScene::RgnBitBlt(const HDC hDestDC, const int DestX, const int DestY,
                           const int Width, const int Height,
                           const HDC hSrcDC,  const int SrcX,  const int SrcY)
{
    // 頂点データ
```

```
    POINT   Vertex[5] = {{320, 6}, {457, 429}, {98, 168}, {542, 168}, {183, 429}};

    HRGN   hRgn = ::CreatePolygonRgn(Vertex, 5, ALTERNATE);         // リージョン生成
    if(hRgn != NULL) {
        ::SelectClipRgn(hDestDC, hRgn);                             // リージョン設定
        ::BitBlt(hDestDC, DestX, DestY, Width, Height,hSrcDC,  SrcX,  SrcY,  SRCCOPY);
        ::SelectClipRgn(hDestDC, NULL);                             // リージョン解除
        ::DeleteObject(hRgn);                                       // リージョン解放
    }
}
```

(４)(３)の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)CTestSceneクラスを以下のように追加、変更して動作を確認しましょう。

・CTestScene.cppに追加

```
/*****************************************************************************************************/
/*                                          円形ワイプ                                               */
/*****************************************************************************************************/
void CTestScene::CircleWipe(const HDC hDestDC, const HDC hSrcDC)
{
    const  double   WipeRatio = (double)m_WipeCnt / m_WipeTime; // 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 * WipeRatio);                      // 半径

    HRGN   hRgn = ::CreateEllipticRgn(cx - r1, cy - r1, cx + r1, cy + r1);
    if(hRgn != NULL) {
        ::SelectClipRgn(hDestDC, hRgn);
        ::BitBlt(hDestDC, 0, 0, 640, 480, hSrcDC, 0, 0, SRCCOPY);
        ::SelectClipRgn(hDestDC, NULL);
        ::DeleteObject(hRgn);
    }
}
```

・インクルードファイルを追加

```
#include <cmath>
```

・CTestSceneクラスのメンバに追加

```
void CircleWipe(const HDC hDestDC, const HDC hSrcDC);

int         m_WipeCnt;     // ワイプカウンタ
const int   m_WipeTime;    // ワイプ時間
```

・CTestSceneクラスのコンストラクタ初期化子に追加

```
m_WipeCnt(0), m_WipeTime(300)
```

・CTestScene::ActivateProc関数を以下のように変更

```
int CTestScene::ActiveProc()
{
    // ここに、機能テストのメイン処理を記述します
    // 内部処理
    m_WipeCnt = (m_WipeCnt + 1) % m_WipeTime;

    // 描画処理
    if(FPSTimer().IsSkip() == false) {
        // ここに、機能テストの描画処理を記述します
        // バックバッファのクリア
        ::PatBlt(m_pBackBuf->GetDC(), 0, 0, m_pBackBuf->GetWidth(), m_pBackBuf->GetHeight(), BLACKNESS);

        CircleWipe(m_pBackBuf->GetDC(), m_pBG->GetDC());
        FPSTimer().DrawFPS(m_pBackBuf->GetDC());

        // バックバッファからディスプレイへ転送
```

```
            HDC   hDestDC = ::GetDC(GameApp().GetHWnd());
            ::BitBlt(hDestDC, 0, 0, m_pBackBuf->GetWidth(), m_pBackBuf->GetHeight(),
                     m_pBackBuf->GetDC(), 0, 0, SRCCOPY);
            ::ReleaseDC(GameApp().GetHWnd(), hDestDC);
        }

        return 0;
    }
```

(７)CircleWipe関数の代わりに以下の関数を使用してみましょう。

```
    /*****************************************************************************************************/
    /*                                          星形ワイプ                                               */
    /*****************************************************************************************************/
    void CTestScene::StarWipe(const HDC hDestDC, const HDC hSrcDC)
    {
        const  double   WipeRatio = (double)m_WipeCnt / m_WipeTime;     // 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 * WipeRatio);                          // 半径

        // 星座標設定
        POINT   Star[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

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

        HRGN   hRgn = ::CreatePolygonRgn(Star, 5, WINDING);
        if(hRgn != NULL) {
            ::SelectClipRgn(hDestDC, hRgn);
            ::BitBlt(hDestDC, 0, 0, 640, 480, hSrcDC, 0, 0, SRCCOPY);
            ::SelectClipRgn(hDestDC, NULL);
            ::DeleteObject(hRgn);
        }
    }
```

応用問題　リージョンを管理するクラスCRgnを作成しましょう。