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５回　ＧＤＩ ◀

GDIとデバイスコンテキスト

文字やグラフィックをAPIで描画する場合は、GDI(Graphic Device Interface)と呼ばれる描画を行う関数群を利用します。GDIでは、すべての描画はデバイスコンテキストを使って行われます。

デバイスコンテキストは、ディスプレイやプリンタなどの出力デバイスの描画属性に関する情報を管理するWindowsのデータ構造です。デバイスコンテキストが取得できるデバイスなら、どのようなデバイスに対しても同じ関数で描画することができます。これは、デバイスの違いをほとんど意識することなく、同じ操作で描画を行えることを意味します。

デバイスコンテキストをサポートする代表的なデバイスとして、ディスプレイ、プリンタ、メモリ、DirectDrawサーフェスが挙げられます。デバイスコンテキストを取得する方法はデバイスによって異なり、GetDC関数やCreateDC関数のほか専用の関数を用いる場合もあります。

GDI関数

GDIは多くの機能を提供しています。簡単な処理から、複雑で高度な処理までさまざまな関数が用意されています。たとえば、文字を描画するTextOut関数とDrawText関数、線を引くLintToEx関数、円を描くEllipse関数、多角形を描くPolygon関数、ビットマップを転送するBitBlt関数とStretchBlt関数などです。これらを含め、代表的なGDI関数には以下のようなものがあります。

ビットマップ関数

AlphaBlend	画像全体もしくはピクセルごとのα値に対応したビットマップを描画します
BitBlt	ビットマップのブロック転送を実行します
ExtFloodFill	領域をブラシで塗りつぶします
GdiFlush	現在蓄積されている(未完了の)GDI関数をまとめて呼び出します
GetDIBits	ビットマップのビット列をバッファにコピーします
GetPixel	指定された座標のピクセルのRGBカラー値を取得します
GradientFill	グラデーションのかかった三角形もしくは長方形形を描画します
MaskBlt	マスクビットマップを使用して２種類のラスタオペレーションを行います
PlgBlt	平行四辺形に変形してブロック転送を実行します
SetDIBits	DIBにビット列を設定します
SetDIBitsToDevice	DIBを指定されたデバイスの長方形内に描画します
SetPixel	ピクセルの色を設定します
SetStretchBltMode	ビットマップ伸縮モードを指定します
StretchBlt	ビットマップをコピーし、拡大または縮小します
StretchDIBits	DIBを指定されたデバイスの長方形内に描画し、拡大または縮小します
TransparentBlt	透過色を用いてビットマップをコピーし、拡大または縮小します

デバイスコンテキスト関数

BeginPaint	WM_PAINTメッセージの描画処理を準備します
CreateDC	指定デバイスのデバイスコンテキストを作成します
DeleteDC	デバイスコンテキストを削除します
DeleteObject	ペン、ブラシ、ビットマップなどのオブジェクトを削除します
EndPaint	WM_PAINTメッセージの描画処理の終了を示します
GetDC	クライアント領域に対応するデバイスコンテキストのハンドルを取得します
GetObject	グラフィックオブジェクトの情報を取得します
GetWindowDC	ウィンドウ全体のデバイスコンテキストを取得します
ReleaseDC	デバイスコンテキストを解放します
SelectObject	デバイスコンテキストの指定のオブジェクトを選択します
SetBkColor	デバイスコンテキストの背景色を設定します
SetBkMode	デバイスコンテキストの背景モードを設定します
SetROP2	前景混合モードを設定します

DrawDib関数

DrawDibClose	DrawDibデバイスコンテキストを閉じ、リソースを解放します
DrawDibDraw	DIBを描画します
DrawDibOpen	描画のためのDrawDibデバイスコンテキストを作成します

塗りつぶし関数

Chord	楕円と線分で囲まれた領域(弓形)を描画します
Ellipse	楕円を描画します
FillRect	ブラシを使用して長方形を塗りつぶします
FrameRect	長方形の境界線を描画します
InvertRect	長方形の内部の色を反転します
Pie	楕円と２つの半径で囲まれた領域(扇型)を描画します
Polygon	多角形を描画します
PolyPolygon	複数の閉じた多角形を描画します
Rectangle	四角形を描画します
RoundRect	角の丸い四角形を描画します

ブラシ関数

CreateBrushIndirect	ブラシを作成します
CreateHatchBrush	ハッチブラシを作成します
CreatePatternBrush	ビットマップパターンを持つブラシを作成します
CreateSolidBrush	ブラシを作成します
PatBlt	ブラシで長方形の範囲をペイントします

フォント関数

CreateFont	フォントを作成します
CreateFontIndirect	フォントを作成します
DrawText	長方形の中に指定された形式でテキストを描画します
GetGlyphOutline	デバイスコンテキストに選択されたTrueTypeフォントの文字のアウトラインやビットマップを取得します
GetTabbedTextExtent	タブを含む、文字列の幅と高さを取得します
SetTextColor	文字列の色を設定します
TextOut	文字列を特定の場所に書き込みます。

直線および曲線関数

AngleArc	線分と円弧を描画します
Arc	楕円弧を描画します
ArcTo	楕円弧を描画します
LineTo	現在の位置と指定された点を結ぶ直線(点は含まない)を描画します
MoveToEx	現在の位置を更新します
PolyBezier	１つまたは複数のベジェ曲線を描画します
PolyBezierTo	１つまたは複数のベジェ曲線を描画します
PolyDraw	線分とベジェ曲線の組み合わせを描画します
Polyline	指定された配列内の点を結び、１つの連続直線を描画します
PolylineTo	指定された配列内の点を結び、１つの連続直線を描画します
PolyPolyline	連続した線分を複数描画します

ペン関数

CreatePen	ペンを作成します
CreatePenIndirect	ペンを作成します

リージョン関数

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

パス関数

BeginPath	パスをオープンします
CloseFigure	パス内の開かれている図形を閉じます
EndPath	パスをクローズし、デバイスコンテキストに選択します
FillPath	パスの内部を塗りつぶします
FlattenPath	パスの曲線を座標変換し、直線の集合に変換します
PathToRegion	パスからリージョンを作成します
SelectClipPath	パスを利用してクリッピング領域を変更します
SetMiterLimit	マイター接合の制限値さを設定します
StrokeAndFillPath	パスの輪郭を描画し、内部を塗りつぶします
StrokePath	パスの輪郭を描画します
WidenPath	パスを塗る領域として再定義します

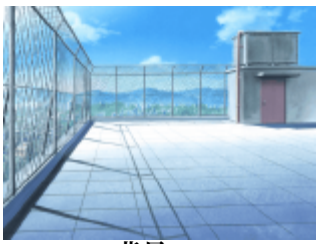
ディスプレイ関数

ChangeDisplay Settings	ディスプレイ設定を指定したグラフィックスモードに変更します

課　題

GDI関数を用いてビットマップを合成してみましょう。

(１)TransparentBlt関数で２つのビットマップを合成します。以下のような「背景ビットマップ」と「キャラクタビットマップ」を用意しましょう。


背景.BMP


キャラクタ.BMP

(２)CTestSceneクラスを以下のように変更しましょう。

・CTestSceneクラスのメンバに追加

```
CBitmap*    m_pBG;      // 背景
CBitmap*    m_pChara;   // キャラクタ
```

・CTestScene::CTestScene関数を以下のように変更

```
CTestScene::CTestScene()
    : m_pBG(new CDDB("BMP\\背景.bmp")), m_pChara(new CDDB("BMP\\キャラクタ.bmp"))
{
    // ここに、機能テストの初期化処理を記述します
    FPSTimer().Reset();         // タイマリセット
}
```

・CTestScene::~CTestScene関数を以下のように変更

```
CTestScene::~CTestScene()
{
    // ここに、機能テストの解放処理を記述します
    delete m_pChara;
    delete m_pBG;
}
```

・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
		HDC	hDestDC = ::GetDC(GameApp().GetHWnd());

		::BitBlt(hDestDC, 0, 0, 640, 480, m_pBG->GetDC(), 0, 0, SRCCOPY);
		::TransparentBlt(hDestDC, 0, 0, 640, 480,
				m_pChara->GetDC(), 0, 0, 640, 480, RGB(0, 0, 0));
		FPSTimer().DrawFPS(hDestDC);

		::ReleaseDC(GameApp().GetHWnd(), hDestDC);
	}

	return 0;
}
```

・ライブラリを追加

```
#pragma comment(lib, "msimg32.lib")
```

(３)(２)を実行すると、画面がかなりちらつきます。これは、描画途中の未完成なフレームを表示しているために起こります。この問題を解決するため、「バックバッファ」という概念を導入します。バックバッファとは、見えない裏画面のことで、ここに描画することにより、描画中の未完成なフレームを隠すことができます。フレームが完成したら、バックバッファからディスプレイへ転送するようにします。

CTestSceneクラスを以下のように追加、変更しましょう。

・CTestSceneクラスのメンバに追加

```
CBitmap*	m_pBackBuf; // バックバッファ
```

・CTestSceneクラスのコンストラクタ初期化子に追加

```
m_pBackBuf(new CDDB(NULL, 640, 480))
```

・CTestSceneクラスのデストラクタに追加

```
delete m_pBackBuf;
```

・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
		// バックバッファへ描画
		::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());
		::BitBlt(hDestDC, 0, 0, m_pBackBuf->GetWidth(), m_pBackBuf->GetHeight(),
				m_pBackBuf->GetDC(), 0, 0, SRCCOPY);
		::ReleaseDC(GameApp().GetHWnd(), hDestDC);
	}

	return 0;
}
```