.NET における画像処理＆表示のキホン

～PictureBox に自前で描画したらコロス～

Microsoft MVP for Visual Developer – Visual C#

渋木宏明(ひどり)

はじめに

対象

- Windows.Forms アプリケーション
- 一部、ASP.NET アプリケーションにも適用
- WPF アプリケーションは対象外

「画像」の定義

- 一般に、計算機で扱うことができるのは「デジタル画像」
- 今回は .NET の System.Drawing.Bitmap クラスで扱える「デジタル画像」
- せいぜい表示画面サイズ程度の静止画

画像表示のキホン

問 – この実装は適切？

- シナリオ
  - 画像ファイルを読み込み、PictureBox コントロールに表示する
- 実装例
  - Sample1 参照

答 – 適切とは言えない

- 問題点
  - 僅かながらも、陰で無駄な処理が行われる
  - 実行環境によっては、いわゆる「チラつき」が生じる
  - 「描画結果を保存」等の応用で苦しむことになる

問題の背景

- Windows の、ウィンドウおよびコントロールの描画手順が大元の原因
- 原則として、Paint イベントは「背景色による塗りつぶし」が行われた後に発生する
- 画面外や他のウィンドウと重なった箇所などは、描画が行われない

結果的に…

- 後から「画像」によって上書きされる個所への「背景色による塗りつぶし」はまったくの無駄な処理となる
- 実行環境によっては、「背景色の塗りつぶし→画像の描画」が順に実行さる過程が「チラつき」として認識されることがある
- 画面外や他のウィンドウと重なった箇所に「表示されたはず」の領域を含む表示結果をファイル保存したりはできない

さらにもう1点

- Image.FromFile() メソッドは問題アリ
  - Image.FromFile() メソッドを使用して Bitmap クラスのインスタンスを作成すると、画像ファイルが「使用中」のままになる
  - この問題は .NET Framework 2.0 SP1 でも改善されていない
  - .NET Framework 2.0 SP1 をコアとする、.NET Framework 3.0, 3.5 でも同様

よりよい実装は？

- 戦略
  - 「画像」は Bitmap クラスを使用して扱う
  - PictureBox コントロールには「画像の表示」だけを任せる
  - MemoryStream クラスと Image.FromStream() メソッドを使用して、Image.FromFile() メソッドの問題点を回避
- 実装例
  - Sample2 参照

おまけ

- Windows.Forms のダブル バッファリングを利用するのも１案
  - MSDN ライブラリ
    「方法：フォームとコントロールのダブル バッファリングを行うことによってグラフィックスのちらつきを軽減する」
    http://msdn2.microsoft.com/ja-jp/library/3t7htc9c(VS.80).aspx
- WPF ではこの辺の事情がガラッと変わってしまう！

画像処理のキホン

簡単な画像処理 – その1

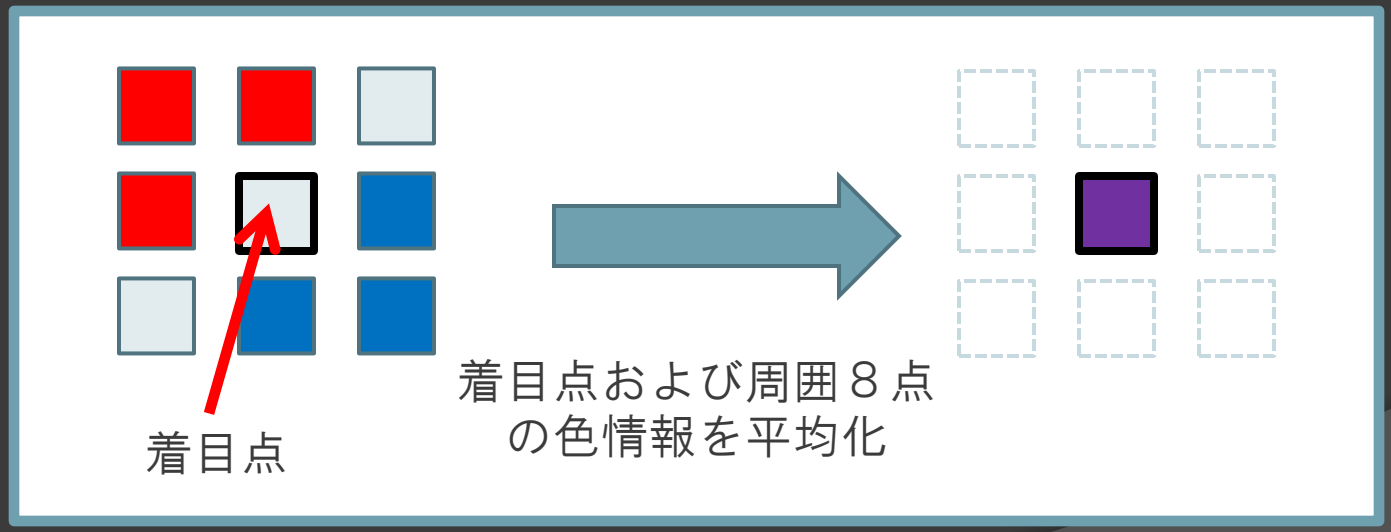
- シナリオ
  - 画像ファイルを読み込み、PictureBox コントロールに表示する
  - 画像に適当な図形を上書き描画する
  - 描画結果をファイル保存する
- 実装例
  - Sample3

簡単な画像処理 – その2

- シナリオ
  - 画像ファイルを読み込み、PictureBox コントロールに表示する
  - 画像に「ぼかし（簡略化した平滑化）」処理を行う
  - 「ぼかし」処理に要した時間を表示する
- 実装例
  - Sample4

「ぼかし」処理の概要

- 着目点およびの周囲８点の Color を取得
- R, G, B それぞれの平均値を求め、新たに着目点の Color とする



結果

- とても遅い…
- Bitmap.GetPixel(), Bitmap.SetPixel() メソッドが「遅い」から！？

問題の背景

- Bitmap.GetPixel(), Bitmap.SetPixel() メソッドが「遅い」のは定説
- 多量のピクセル処理が必要なシナリオには不適

解決策は？

- 戦略
  - Bitmap.LockBits() メソッドを使用して、Bitmap クラスが保持するビットマップデータを直接操作
- 実装例
  - Sample5 参照
  - 注意: 長くなるので、エラー処理などかなり簡略化してます

実装のポイント

- unsafe を避けるため、Bitmap クラスが保持するビットマップデータを Marshal.Copy() メソッドで int 型の配列にコピーしている
- 多くの場合、ビットマップデータのコピーのコストよりもフィルタ処理のコストの方が高いため、元が取れる
- 処理性能の限界に挑戦する場合、局所的に unsafe を使用することになるが、そこまでするなら C++ 等で記述した方が早くね？

まとめ

- 「画像」は Bitmap クラスで扱う
- 「画像表示（静止画像の表示）」は PictureBox に任せる
- 多量のピクセル処理を伴う「画像処理」には Bitmap.LockBits() を使用する

Q&A

ご清聴ありがとうございました

HIDORI on The Web
http://hidori.jp/

渋木宏明(ひどり) blog
http://hidori.spaces.live.com/