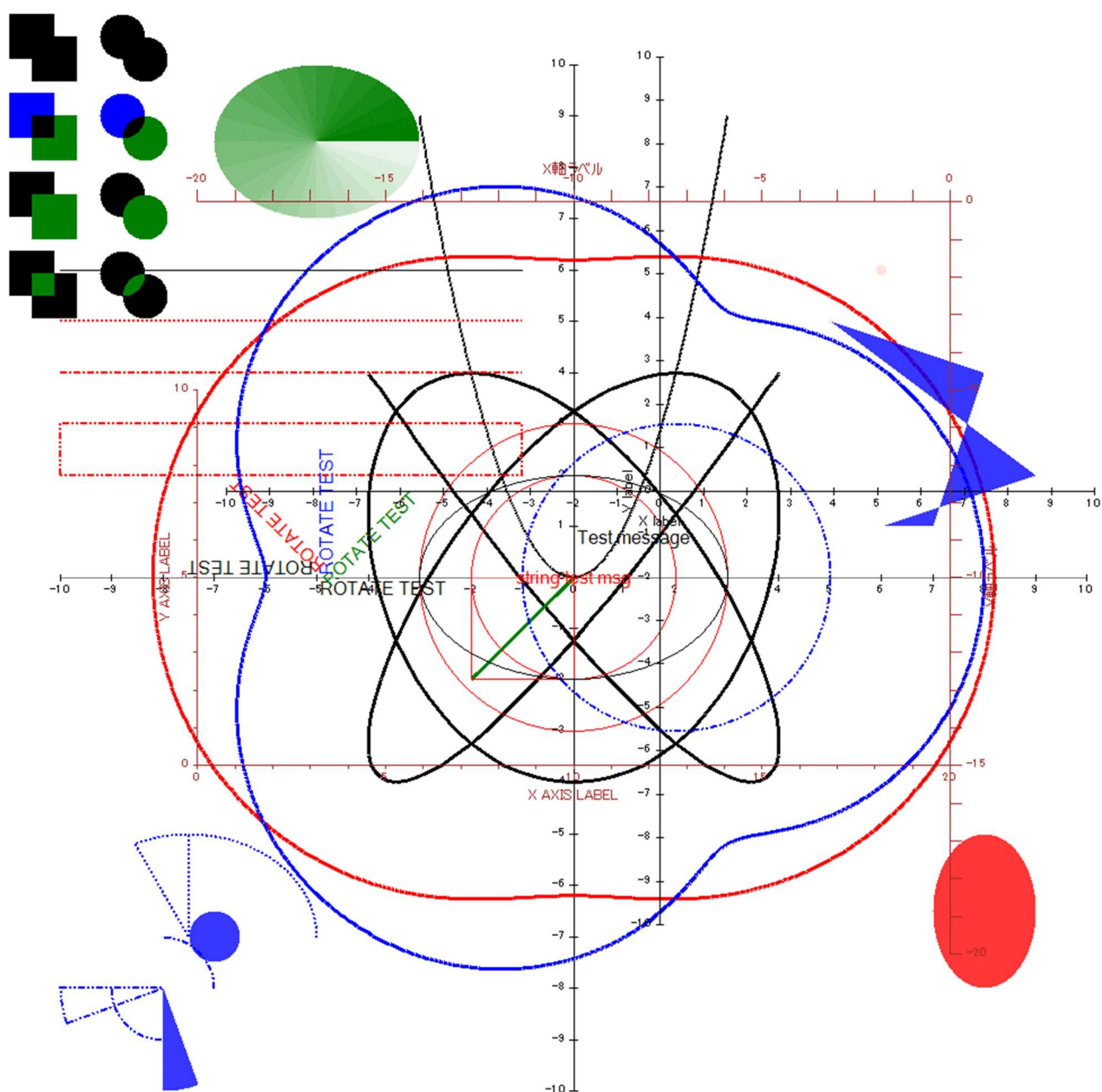


利用の手引き



GraphLibrary Rev 3.0

Written by H.Matsushima 2018/10/13

はじめに

Windows フォームアプリケーションでプログラムを作成し、図形を作成する場合、.NET Framework で提供されている、`DrawLine` 文などの関数を使ってフォーム上やパネル上に描画します。しかし、フォーム上に作成したその図形をプリンタに出力しようとするプログラム的大幅な変更が必要となり、途方に暮れたことがあります。

この **GraphLibrary** は、図形を描画するユーザー座標系と出力デバイスごとに異なるデバイス座標系とを分離します。ユーザーは、まず、任意のユーザー座標系を定義し、この座標系に **GraphLibrary** の提供する描画関数を使って図形を描画します。すると **GraphLibrary** がユーザー座標系からデバイス座標系に変換し指定された出力先に描画します。

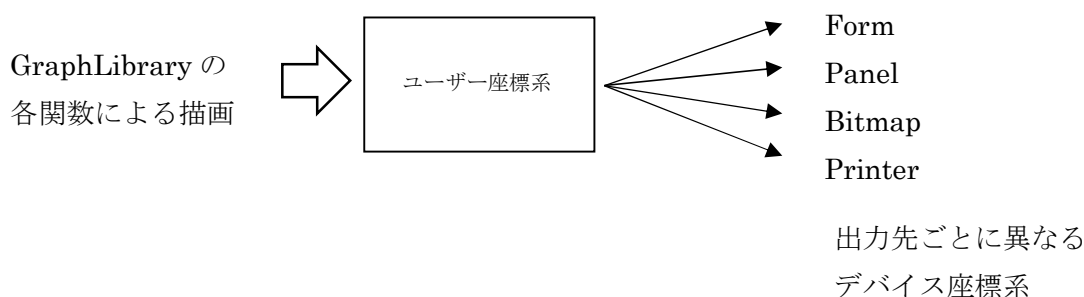
このため、描画先を変更する場合、描画するユーザープログラムをまったく変更せずに、驚くほど簡単に切り替えることを可能としたライブラリです。

出力先として **Graphics** オブジェクトに出力しますので、**Graphics** オブジェクトが作成できるものであれば指定可能です。具体的には、ウィンドウズのフォーム、フォーム上のパネル、**Bitmap** 及びプリンタなどです。

このライブラリは、**Visual C#**で作成していますが、**Visual Basic** 等 .NET Framework 上で動作するいずれの言語からも利用も可能です。

プログラム開発は **Visual Studio 2022** を使用しています。サンプルプログラムを実行するためには同環境を準備して下さい。

では、サンプルプログラムを例に具体的な使用方法を説明します。



シンプルな図形表示プログラム (CSLissajousFiguer)

Visual Studio を起動し、配布ファイルの **CSLissajousFiguer** にあるプロジェクトを開いてください。標準的なプロジェクトと異なる点が 2 か所ほどあります。まず、ソリューションエクスプローラーでプロジェクトを表示するとその中に参照という項目があります。この参照に **GraphLibrary** が追加されています。もし、参照に含まれていない場合は、参照の追加で同ライブラリを加えてください。

2 つ目は、**Form1.cs** の始めのほうに `using Pleiades;` という文が追加されています。この 2 点が設定されれば **GraphLibrary** を使用する準備が整いました。

プログラムを実行してみてください。フォーム上にリサージュ図形が現れたはずです。では、どのように動作しているのでしょうか。

フォームのペイントイベントでこのプログラムは動き始めます。

18 行目の `sw` は 1 になっていますので `case 1` にある

```
PlotMain(this.CreateGraphics(), this.ClientSize);
```

が実行されます。

第一引数は、`Graphics` オブジェクト、第二引数はそのサイズです。何のことかピンときませんか。要するに、図形を描くためのキャンバスとその大きさを引数にしたイメージです。

ここでは、`this` を指定していますのでフォームの `Graphics` オブジェクトとフォームの大きさが渡されます。

39 行目の `PlotMain` 関数の中を見ていきましょう。

```
Pleiades.GraphLibrary gl = new Pleiades.GraphLibrary();
```

で、`GraphLibrary` の使用を宣言します。そしてまず実行するのが

```
gl.PlotBEGIN(g, s);
```

`Graphics` オブジェクトとそのサイズを保存し、ライブラリの準備をします。

そして重要なのが

```
gl.AxisInitialize(-1.5, -1.5, 1.5, 1.5);
```

です。ここで、ユーザー座標系の左下の `x`、`y` 座標と右上の `x`、`y` 座標を指定します。

この場合は `(-1.5, -1.5)` と `(1.5, 1.5)` の正方形のユーザー座標の使用を指定しました。

しかし `PlotBEGIN` 文の第二引数で渡されたキャンバスの大きさは正方形とは限りません。

そこで、ユーザー座標系とデバイス座標系の中心が一致し、かつ、アスペクト比（縦方向の高さ/横方向の幅）が同一となるように、ユーザー座標系を変更します。デバイス座標系が横長の場合はユーザー座標系の `X` 軸が、またデバイス座標系が縦長の場合はユーザー座標系の `Y` 軸が拡張されます。

こうすることにより、ユーザー座標系からデバイス座標系への変換は 1 組の簡単な一次式で可能になります。

これで、ユーザー座標系を使う準備ができました。

```
gl.Clear();
```

は、キャンバスを消去します。（白色で塗りつぶしているだけです）

次に実行しているのが、軸を描く `AxisX`、と `AxisY` です。これは `-1.0` から `1.0` まで `0.1` ステップで `X` 軸と `Y` 軸の座標軸を描いています。座標軸に関する関数は必要が予想される全ての場合を考えましたので沢山ありますが分かりやすい体系になっていると思います。

57 行目で `Plot_Lissajous` 関数を呼び出しています。

`Plot_Lissajous` 関数は描画する際に、使用するペンの設定と移動する `Move` 文しか使っていません。

まず、`SetPenColor` で赤のペンを選び、`SetPenWidth` で太さを指定します。

次に、`PenUp` の状態にして最初の地点へ移動させ、後は `PenDown` の状態にして描画を行います。

説明が前後しますが、`Move` 文はペンアップ状態の時は移動だけ、ペンドアウン状態の時は線を描きます。

XYプロッターをイメージして頂ければわかりやすいと思います。

ここで使われている計算式は、どこかで目にしたことがあるかもしれません。x、yの数値は数式の通りにC#の関数で記述しただけです。

ユーザー座標系を導入することによりこんなに簡単に数値関数を作図できてしまいました。色々な関数を当てはめれば面白い曲線が描けると思います。トロコイド曲線などは如何でしょうか。

さて、最後はPlotEND文でGraphLibraryの使用を終わります。

この様に

```
Pleiades.GraphLibrary gl = new Pleiades.GraphLibrary();
gl.PlotBEGIN(g, s);
gl.AxisInitialize(Xmin, Ymin, Xmax, Ymax);
GraphLibrary が提供する描画関数の実行
gl.PlotEND();
```

以上がGraphLibraryを使用する流れとなります。

PlotBEGIN文の実行の前にAxisInitialize文を実行した場合、警告メッセージを表示します。

デバイス座標系が決まる前にユーザー座標系を決めようがないからです。また、AxisInitialize文の実行の前に各描画関数を実行した場合は、その描画関数はすべて実行されません。ユーザー座標系が指定されていなければ描けないからです。また、PlotBEGIN文は重複して実行できないようにしています。プログラムを実行してエラーダイアログが表示された場合は、各文の実行順序を見直してください。各関数に依存関係があるため簡単なステートマシンを構成しています。

では、18行目のswを2にして実行すると、通常使うプリンタからフォームに表示したものと同じリサージュ図形が1枚印刷されます。

印刷する場合、PlotMain関数内のプログラムを変更する必要は一切ありません。

26行目のDoPrint(true)が実行されます。引数はtrueの時、ランドスケープ即ち横長、falseの時はポートレート即ち縦長に印刷します。

76行目からがDoPrint関数の本体です。

印刷するためにまず、PrintDocument pdでPrintDocumentの使用を宣言しています。すると、現在接続されているプリンタの情報にアクセスできます。

まず、印刷可能エリアを取得します。ランドスケープで印字する場合はプリンタをランドスケープモードに設定します。また、paperSizeに取得した印字領域を保存しています。

ポートレートの場合も同様です。

次に、PrintDocumentのPrintPageイベントにPd_PrintPage関数を登録しています。そして、Print文で印刷を実行します。

さて、このPd_PrintPage関数ですが、これはPrint文を実行し印刷の準備が整ったらWindowsから

呼び出されるコールバック関数です。

その第二引数で示される、`e.Graphics` にプリンタの `Graphics` オブジェクトがあります。

`Graphics` オブジェクトと `paparSize` の組み合わせ。どこかで見覚えがありませんか？

そうです、`PlotMain` 関数の引数と同じです。そこで

```
PlotMain(e.Graphics, paparSize);
```

を実行すれば、`PlotMain` 関数内の描画関数が実行され、プリンタのキャンバス上に描画される次第です。

でも待てよ、用紙の大きさはフォームの時の大きさと異なっているはずなのに？

大丈夫です。`AxisInitialize` 文はユーザー座標系を設定すると説明しました。`PlotMain` 関数の第二引数で渡されたプリンタのキャンバスの大きさと `AxisInitialize` 文で指定したユーザー座標値から計算し、用紙の大きさに合わせたユーザー座標系が設定されるからです。

これで印刷はもう終わりです。簡単ですね。

では `case 3` を見てみましょう

これは `Bitmap` に描画する例です。見ての通りなのですが、`Graphics.FromImage` 関数で `Graphics` オブジェクトを取得し、それとそのサイズを引数にして `PlotMain` 関数を呼んでいるだけです。

ここでは、その `Bitmap` をカレントディレクトリに `test.bmp` として保存しています。

ですから描画結果を他のアプリケーションで利用できます。

以上がこのサンプルプログラムの処理の流れでした。

ユーザー座標系を導入することにより描画が簡単にでき、印刷も手軽にでき、また、描画出力の保存もできることを体感して頂けましたでしょうか。

VBLissajousFigure について

これは `Visual Basic` での使用例です。

処理内容は `C#` の `CSLissajousFigure` プログラムの場合と全く同一です。

`C#` と同様に、ライブラリの参照の追加と `Import Pleiades` 文の追加で `GraphLibrary` の利用が可能になります。

CSLissajousCurve について

このサンプルプログラムはフォーム上のパネルを使った例です。

フォーム上に 1 つのパネルと 3 つのテキストボックス、3 つのスライダーそして 1 つのボタンを配置しています。

フォームがロードされたときに、各スライダーの設定とスライダーが動いた時のイベントハンドラを設定しています。また、スライダーのイベントとテキストボックスのイベントを定義しています。

`PlotMain` 関数は、パネルのペイントイベントが発生した時とテキストボックスの変更があった場合に `Panel1` の `Graphics` オブジェクトを引数として起動されます。

この例でも 48 行目付近の関数を変えることにより様々な応用が可能と思います。

DrawLine について

このプログラムは私が **GraphLibrary** を作成した際に使用したテストプログラムの一部です。実装したすべての関数を検証しましたので描画関数の使用例として提供します。描画結果をこの手引きの表紙に使用しました。

あ、説明を忘れていましたが 2 点ほど

まず 1 点は 22 行目付近です。フォームはその **ClientSize** を変更することで表示サイズを変えられます。ここでは A4 用紙のアスペクト比と同一になるように設定しています。こうするとフォームの上に描画したイメージと印刷したイメージが一致し便利です。

2 点目は 50 行目のコメントアウトしてあるピクチャーボックスに描く例です。ピクチャーボックスはコンテナではないので、直接描いても画像を保持できません。そのため **Bitmap** に作画し転送するほかないようです。

その他の描画関数について

基本的には、**.NET Framework** が提供する描画関数を参考にユーザー座標系に描画する関数を実装しています。詳細は **GraphLibrary** コマンド一覧、または、ヘルプファイルをご覧ください。

あとがき

今から 50 年近く前、自動車販売店のパンフレットにロータリーエンジンシーリングの計算式が掲載されていました。なんとか自分で描けないかと思い、三角関数表とソロバンを使い計算し、三日三晩かけ描くことができました。そんな複雑な計算式で構成されるエンジンを完成させた東洋工業（現マツダ）の技術力に驚嘆した次第です。

昔のことを思い出し自分のパソコンでロータリーエンジンのエピトロコイド曲線を表示、印刷できないかと思い作成したのがこの **GraphLibrary** です。

Ver1 は画面表示だけの基本的な構成を作り、**Ver2** で表示系は **Graphics** オブジェクトを使用し、印刷系は良い方法が思いつかず中間ファイルを使用する方法で完成させました。

ところが、完成したプログラムの **Graphics** オブジェクトを扱う表示系にプリンタの **e.Graphics** オブジェクトを渡して実行させたところ、思いがけず問題なく動作してしまいました。はじめは、なぜ処理が上手くできたのか理解できませんでした。

そこで、プログラム構成を大幅に見直しリファクタリングして完成させたのが **Ver3** です。

描画や印刷に使いやすいユーザー座標系を提供するライブラリとして、多くの方々に使っていただきたく思い、私の 63 回目の誕生日を記念して公開することとしました。

学生の方々の実験レポートの作成や、情報処理を業務とする方々の仕事に役立ってくれればと思っています。

ご意見、ご質問がございましたら時間の許す限りメールにてお答えしたいと思います。なお、メールの件名には必ず **GraphLibrary** の文字を入れてくださるようお願いいたします。

皆様方のパソコンライフに少しでも役立てばと願っております。

メールアドレス jad03444@nifty.com

松島 久男

平成 30 年 10 月 茅ヶ崎にて