C++による画像処理

著　OIDUS・David・Dekuron

[　改訂第二版　]

Hohama

―目次―

Mission1　…　画像とはそもそも何か？

Mission2　…　C++とは何か？

Mission3　…　画像処理とは？

Mission4　…　画像を数値化（符号化）する

Mission5　…　実践①　モノクロ化

Mission6　…　実践②　ノイズカット

Mission7　…　実践③　回転

―この記事を書くにあたって―

この記事は、僕が高校時代に書いたものです。それを、今回改訂し、第二版の公開となりました。第一版公開当時の記事に、こう書かれていました。

「　知らない時が（知ることの多い時）が最高の幸福である　」

今の僕は、そう思います。誤解しないでほしいですが、物を知らないときが、一番、勉学に意欲の沸くときなのではないでしょうか。今の僕には、知識と引き換えに、勉学に励む精神を失ってしまったと思います。今回の記事を書くうえで、その事を再認知しました。

今回の記事で、「　知る楽しさ　」と「　知識の豊富さ　」の両立を考えてもらえたら冥利に尽きるというものです。

今でも、そう思います。僕自身、日々の開発の時には、いつもこの事を念頭に置いています。改めて、この記事を読む皆さんにはその事をじっくり考えてもらえれば、嬉しい限りです。

【Mission1：画像とはそもそも何か？】

それでは、Mission1：画像の説明を始めましょう。まず知っておいて欲しい事が一つあります。それは、

「 画像とは何なのか 」

と言う事です。皆さんがお持ちの写真、そうそう、それです。それを手にとってまず見てください。きれいに山や川、ビルなどが写っていますね。しかし、それを虫眼鏡で見てください。ビルの窓に見えたのが、実は点々が集まっているだけだというのがわかりますね。近くで見ると点々の集まりだったのが、遠くから見るとビルに見えるわけです。

[遠くから見た図]

[近くで見た図]

この事は、今回の記事のようなコンピュータを使った画像処理に使う、デジタルカメラの写真においても、同じ事がいえます。デジタルカメラの写真は、右上の図のように、正方形がびっしりと並んだ構造になっています。とりあえず、「デジタル」の場合、点々の最小単位は「正方形」だと覚えてください。

画像とは、本来の意味では「 絵すがた 」です。しかし、今回はコンピュータを使って解説をします。ですので、とりあえず、「 画像は正方形の集まりだ 」と思ってください。詳しくは後で解説します。

【Mission2：C++とは何か？】

画像の事を簡単に解説しましたので、次に表題の通り、「C++」という物を簡単に解説しましょう。

皆さんが今回の記事を読んでいる時、コンピュータの前にいると思います。コンピュータは、皆さんの前に僕の記事を表示しているわけですが、コンピュータはどうやって文字や絵やらを表示しているのでしょうか？

コンピュータは、それを「プログラム」という物で行っています。プログラムとは、簡単に言うと、「作業」です。皆さんがアルバイトなどをする時、「牛乳を並べて」、「レジは任せた」などと、上司や雇い主に命令されることがあるでしょう。

コンピュータも、使う人が命令したことを、順番にこなしていきます。これを「実行」と言います。これは、皆さんと同じです。しかし、コンピュータは、出来ることが限られています。例えば、「足し算をする」、「掛け算をする」などです。これでは、特定のことしか出来ません。しかし、出来ないなら組み合わせればいいのです。やって欲しい事を順番に、「作業」として行わせればいいのです。皆さんも、いっぺんにあれこれ命令されても、一度には出来ないでしょう。でも、順番に一つずつこなしていけば、最後にはやりたい事を実行できます。例えば、「企画書を書いて FAX で送っておいて」といわれても、いっぺんには出来ません。しかし、まず企画書を書いて、それから FAX 送ればいいのです。

コンピュータも、一度に出来ることは原則一つで、出来ることも限られていますが、皆さんと同じように「順番に」処理すればいいのです。何も難しいことはありません。ただやりたい事を順番にわけて、最後に出来ればいいのです。大切なのは、その「やりたいこと」です。これが決まっていなければ、意味がありません。

ただ、簡単に「やればいい」といいましたが、「プログラミング」という名前は知っているが、どうやるのか知らないと言う人は多いと思います。皆さんは、プログラミングというものが、たいそう難しいものだと思っているでしょう。そうでしょう。それは、やはりというか、あの英語だけの文字がずらずらしている画面が原

因だと思います。ですが、考えてみれば、それが日本語で、それも文章で書かれていたらどうですか？まるで箇条書きされたスケジュールのように、プログラムというものを手にとるように理解できるでしょう。例えば、

```
PRINT   “ こんにちは ”
INPUT   “ あなたの名前は？ ”:NAMAE
PRINT    NAMAE ; “ でいいですね？ ”
```

と言うものも、

「こんにちは」と画面に表示しなさい。
「あなたの名前は」と言って、名前を入力させなさい。
名前があっているか、その人の名前を表示して「いいですね」と聞きなさい。

などと書かれていたら、プログラミングの言葉を知らない人でも、何をやるかが簡単に分かり、まるで自分がコンピュータの形の箱をかぶって使う人の目の前に座っているように思えてくるでしょう。

それが、ただ英語になっただけです。いやむしろ、昔は日本語でもプログラミング出来た時代がありました。ただ、日本語は日本だけでしか使われていません。アメリカなどでは「インパクト」として、日本語が使われていたりするくらいです。日本語は一般性が無いのです。

ならば、英語を使えばどうか。英語なら、いろいろな国などで使われていますから一般性もありますし、何も日本語しか使えない訳ではありません。知らないなら覚えるまでです。

少し遅れましたが、「プログラミング」は作業を決めていく、「作業日程表」の作成だと思ってください。そして、その表を作る際には、いろいろな言葉で作製できます。この「言葉」を「プログラミング言語」と言います。そして、そのうちの一つが「C++」なのです。今回は、C++の説明はしません。ですが、「興味」が「原動力」なのは変わりません。皆さん自身で勉強する、その道を開けたと思います。頑張って、まずはC++を覚えてください。

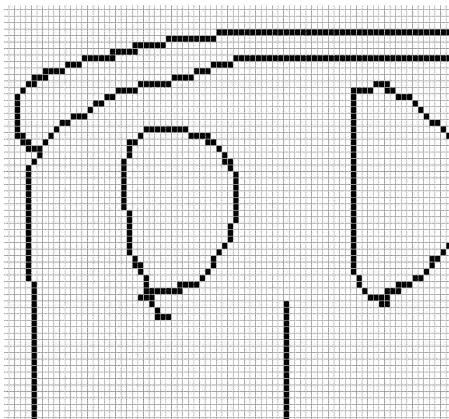
【Mission3：画像処理とは？】

画像処理とは、画像（点の集まり）にいろいろ手を加えることです。例えば、皆さんも一度は触ったことがある「 ぬり絵 」もその例です。皆さんはクレヨンを使って（パステルを使う人もいますね）、仮面ライダーの絵に色を塗ったり、消しゴムで消したりしますね。しかし、その時に、仮面ライダーが、実は点々の集まりだったと言う人はいますか？いないでしょう。つまり、皆さんは、虫眼鏡で見たときのような、「点々」でぬり絵をしないのです。ぐちゃぐちゃと塗っていくでしょう。

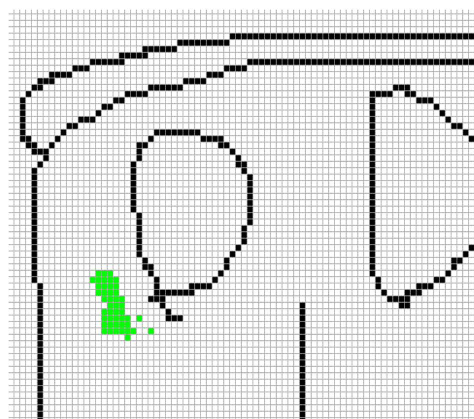
[元の絵の図]

[色を塗った後]

しかし、コンピュータでは、仮面ライダーを点々の集まりでぬり絵をします。なぜかというと、コンピュータは「 デジタル 」だからです。



[デジタルの仮面ライダー]



[色を塗るとき]

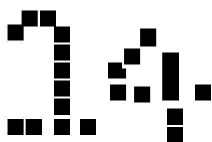
少し見にくいかもしれませんが、正方形がびっしりと並んでいるのが分かるでしょう。デジタルの画像は Mission：1 で説明したとおり、正方形の集まりなのです。前ページ右下の図のように、色を塗るときに正方形一個一個に色を塗っていきます。デジタルの画像処理は、このように、正方形一個一個に色を加えたりしていきます。

さて、デジタルは、「 2 進数 」という意味です。0100101110100010110100　なんて言うのがその例です。「 0 か 1 」しかないのです。2354 や、56778 も、010101010101001 なんて感じで、表されます。（ちなみに、8 進数は「 オクタル 」、16 進数は「 ヘキサデシマル 」です。）

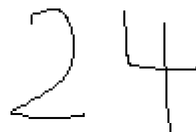
コンピュータは、いろいろなことを、この 0 と 1 でやっています。これは、コンピュータが「 ON 」と「 OFF 」しか使えないからです。皆さん、テレビの電源スイッチに「 ON でも OFF でもないスイッチです 」等と書かれているのを見たことがある人はいないでしょう。コンピュータには ON か OFF しかないのです。

しかし、コンピュータには一番の問題があります。それは、0 か 1（赤と青のボールだと思ってください）をしまっておく場所（倉庫）が無限に広くないため、何億個のボール(データ)になってくるとしまう場所が足りなくなってくることです。

つまり、しまう場所に「 限りがある 」のです。そのため、画像（点々）を 0 と 1 で表現するデジタルの場合、その点々の個数にも「 限りがある 」のです。これが、デジタルの画像が正方形（点々）の集まりだという理由です。限りがなければ、「 点々 」と考えても、その点々自体の小ささに限りがないため、点々でないとみなせます。これは「アナログ」です。



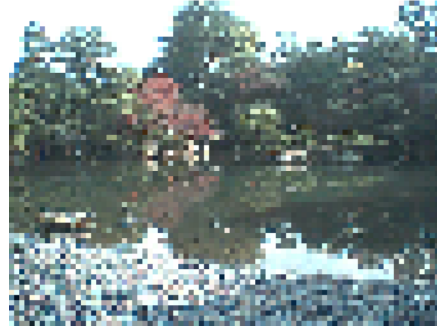
[デジタル]



[アナログ]

これは、画像を「 大体の形 」で表現しているに過ぎません。つまり、数十万個の点の集まりを、遠くから見ると「 絵 」に見えるわけです。これは、さっきの虫眼鏡のときと同じです。そして、この点々の数の多さを「 解像度 」といいます。

[解像度が高い]


[解像度が低い]

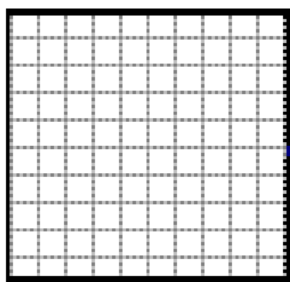
コンピュータを使った画像処理は、点々の集まりを 0 と 1 で表現した（数字で表した）ものにいろいろ手を加えることなのです。

これ以下の説明は、C++で紹介します。知らない人は、まず C++を覚えてください。でも、とりあえず、コンピュータを使って何が出来るのかは、分かるように説明していますので、わからないところは読み飛ばして、興味を持ってくれると嬉しいです。ぜひ最後まで見てください。

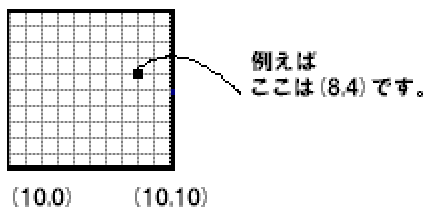
【Mission4：画像を数値化（符号化）する】

画像が点々の集まりだと言う事は既に紹介しました。今度は、画像をコンピュータが扱えるように、数字の集まりにします。

まず、縦と横の点の数（解像度）を決めます。下の例では、縦 10 個、横 10 個にしています。



次に、点々のある、縦と横の位置（座標）を（X, Y）で表してみましょう。上の例に当てはめた図が、下の絵です。



そして、その座標の点の色を数値にします。

★これを、C++で記述してみましょう。★

座標は、(X, Y)ですので、

```
struct CPOINT {
  int X;
  int Y;
};
```

とします。コンピュータには理論上、整数の座標しかなく、また、マイナスを扱うこともあるので、int で宣言しています。点（1 ドット）の物理的なサイズ（モニタのドット一つのサイズなど）は存在しますので、そう言ったものを考慮するのなら、double 型で宣言してもいいでしょう。

つぎに、色です。光の三原色の原理により、あらゆる光（色）は

「 青と緑と赤 」

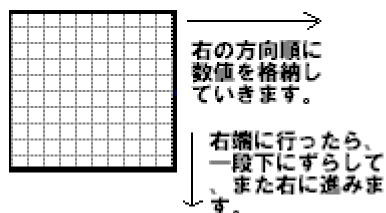
で、表現できます。なので、赤＝R、緑＝G、青=B とすると、

色は、

```
strct RGB {
 double R;
 double G;
 double B;
};
```

にします。（色の再現は整数でいいのですが、画像処理時に演算をするので、double 型で宣言をして、精度を高めています。）

最期に、点の集まりを以下のようにします。



これを、

```
RGB point_color[x][y] ;
```

とします。

これで、縦×横の座標の色を数値に出来ました。

さて、次のステップのために、これをクラスとして定義しましょう。

★クラスは、C++のオブジェクト指向の３本柱の１つです。クラスは、簡単に言うと、C の構造体の概念に、「 メンバ関数 」という概念を加えたようなものです。これは、クラスを「 もの 」として考える、つまりは、プログラミング作業を「 もの作り 」と考える際に、非常に効率のよい考え方です。

更に、「 派生クラス 」を導入することで、プログラミングの効率と手間を大幅に改善できます。

＜Cpicture.h＞

```
#ifndef CPFILE_IO
#define CPFILE_IO
#include "FILE_IO.h"   // ファイルの入出力ヘッダ
#endif

#ifndef CPICTURE
#include "Cpicture.cpp"
#endif

struct RGB   // RGB 情報
{
   double R;
   double G;
   double B;
};

typedef RGB   24COLOR;   // 24COLOR 型の導入

struct BMP_24   //  ビットマップ構造
{
            BITMAPFILEHEADER BFH;
            BITMAPINFOHEADER BIH;
            24COLOR * p_color;
};

class CPicture
{
private:
            BMP_24 o_Bitmap; //ビットマップ
public:
            inline BMP_24* GetBitmapPointer(){ return &o_Bitmap; }   //ビットマップ取得

            int Height;
            int Width;
//::::::::::::::::::::::::::()Draw:::::::::::::::::::::::::::::://
            int Point(int X,int Y);
            int PSet(int X,int Y,COLOR SetColor);
//::::::::::::::::::::::::::()SaveToFile:::::::::::::::::::::::::::://
            virtual int SaveToFile(const char * FileName,const char* FileType);
            virtual int ReadFromFile(const char * FileName);
//::::::::::::::::::::::::::()Create:::::::::::::::::::::::::::://
            virtual int CreatePicture();
};//********************************************************//
```

<Cpicture.cpp>

```
#define CPICTURE
#include "Cpicture.h"
#endif

CPicture::PSet(int X,int Y,COLOR SetColor)
{
                    if( (Y>=Height) && (X>=Width) && (X<=0) && (Y<=0) )
                    {
                              return -1;// PixelValueError
                    }

                    o_Bitmap.p_color[Y*Height+X]=SetColor ;

                    return 0 ;

}

//::::::::::::::::::::::::::::Point:::::::::::::::::::::::::::::://

CPicture::Point(int X,int Y)
{
                    if( (Y>=Height) && (X>=Width) && (X<=0) && (Y<=0) )
                    {
                              return -1;// PixelValueError
                    }

                    return (int)&o_Bitmap.p_color[Y*Height+X] ;
}

//:::::::::::::::::::::::::::Create::::::::::::::::::::::::::::://

CPicture::CreatePicture()
{
          if ( (Height>=1) && (Width>=1) )
          {
                    try
                    {
                              o_Bitmap.p_color=new COLOR [Width*Height];
                              /* throw CreateMemError; */
                    }
                    catch(...)
                    {
                              return -1 ; // MemoryCreateError
                    }
                    return Height*Width*3; // Byte
          }
          return -2;
}
//::::::::::::::::::::::::::SaveToFile:::::::::::::::::::::::::::://

CPicture::SaveToFile(const char* FileName,const char* FileType)
{

          if (FileType=="BITMAP")
          {
                    FILE_IO FIO;
                    FIO.SetDataPointer(&o_Bitmap);
                    FIO.SetFileName(FileName);

                    if (FIO.WriteFile( Height*Width*3+14+40 )==-1)return -1;
```

```
                return 0;
        }

        return -2;

}

//::::::::::::::::::::::::::ReadFromFile:::::::::::::::::::::::::::::://

CPicture::ReadFromFile(const char* FileName)
{
        FILE_IO FIO;
        FIO.SetFileName(FileName);
        if (FIO.ReadFile()==(-1))return -1;
        return 0;

}
```

＜FILE_IO.ｈ＞

```
/::::::::::::::::::::::::::::::::::::#include:::::::::::::::::::::::::::::::::::/

#include <fcntl.h>
#include <io.h>

/::::::::::::::::::::::::::::::::::::#struct::::::::::::::::::::::::::::::::::::/

/*struct s_FILE
{
		const char * FileName;
		void* p_Data;
};*/

/*******************class*********************/

class FILE_IO
{
private:
/::::::::::::::::::::::::::::::::::::$FILE:::::::::::::::::::::::::::::::::::::/
		const char * FileName;
		void* p_Data;
public:
/:::::::::::::::::::::::::::::::::::()SetPointer:::::::::::::::::::::::::::::::::::::/
		/*inline*/ virtual void SetDataPointer(void* Data)
		{
				p_Data=Data;
		}
/:::::::::::::::::::::::::::::::::::()ReadFile:::::::::::::::::::::::::::::::::::::/
		/*inline*/ virtual int ReadFromFile()
		{
				int hundle=open(FileName,O_RDONLY|O_BINARY);

						if ( hundle == -1 ) return -1;

				int FileSize=filelength(hundle);

						read( hundle,p_Data,FileSize ) ;

						close(hundle);

						return FileSize;

		}

/:::::::::::::::::::::::::::::::::::()WriteFile:::::::::::::::::::::::::::::::::::::/
		virtual int WriteFile(int BufSize)
		{

		int hundle=open(FileName,O_WRONLY|O_BINARY|O_CREAT);

				if ( hundle == -1 ) return -1;

				write( hundle,p_Data,BufSize ) ;
				close(hundle);
				return 0;
		}
/:::::::::::::::::::::::::::::::::::()SetFileName:::::::::::::::::::::::::::::::::::::/
		void SetFileName(const char* FileName)
		{
				FILE_IO::FileName=FileName;
		}
```

【Mission5：実践① モノクロ化】

画像をモノクロ（グレースケール＝輝度の階調）にしてみましょう。そして、それの応用として、セピア色にしてみます。

光（ここでは、信号と表しましょう。）には、簡単に言うと、2 種類の信号が含まれています。

①：輝度信号
②：色信号

です。

①の輝度信号は、光の「 明るさ 」が含まれています。明るい・暗いと言ったものは、この信号によって決まります。

②の色信号は、光の「 色 」が含まれています。赤っぽい・青っぽいと言ったものは、この信号によって決まります。

さて、モノクロ化は、②のすべての色の色信号を、輝度信号で置き換える（平均を取る）ことで、実現できます。

つまり、「 （R+G+B）/3 」を出して、元の数値と置き換えればいいのです。これを C++で記述すると、

```
const double AVERAGE = (R+G+B) / 3;

R = AVERAGE ;
G = AVERAGE ;
B = AVERAGE ;
```

となります。

これを、Cpicture を引数に（参照渡し）して記述してみましょう。

```
int Mono( &Cpicture toDO )
{
	BMP_24* TEMP=toDO.GetBitmapPointer();

	if ( !(	((TEMP->Height)>=0)&&((TEMP->Width)>=0)	) )
	{
		return –1; // SizeError
	}
	register i , j ;

	unsigned char AVR;

	const int HEIGHT = toDO.Height ;
	const int WIDTH = toDO.Width ;

	for (i=0;i<=HEIGHT;i++)
	{
		for (u=0;u<=WIDTH;u++)
		{

			AVR= (
				TEMP->p_color[u*WIDTH+i].R+
				TEMP->p_color[u*WIDTH+i].G+
				TEMP->p_color[u*WIDTH+i].B
				) / 3;

		TEMP->p_color[u*WIDTH+i].R=AVR;
		TEMP->p_color[u*WIDTH+i].G=AVR;
		TEMP->p_color[u*WIDTH+i].B=AVR;

		// (int)&TEMP->p_color[u*WIDTH+i] =
			(int)&TEMP->p_color[u*WIDTH+i] & SetColor;-①

		}
	}
	return 0;
}
```

これで、モノクロになりました。①を加えることで、セピアカラーのような色を加えることが出来ます。（SetColor との論理積です。）

ループ制御用の変数を register で宣言しているのは、処理が早くなるからです。アセンブルしたときに、外部メモリの読み込み用のコードを必要としないので、実際に、10000000 回の空ループをしたら、格段に早くなりました。

【Mission6：実践② ノイズカット】

　次は、ノイズカットです。皆さんが夢中のデジタルカメラ、ばしばし撮っていますよね。ですが、デジタルカメラで写真を撮るのに気になること、それが「 ノイズ 」です。空のところににじみが出来たり、色が極端におかしい部分が出来たり、と、写真家の宿敵なのであります。で、それは、理論上、

　「 完全な修復（カット）は不可能 」

　です。なぜなら、データはすでにノイズの乗ったものだからです。しかし、「 画像は点である 」ことを考えると、ノイズカットは「 幾分かは 」可能なのです。

　理論から紹介しましょう。

★メディアンフィルタ★

「 メディアンフィルタ 」とよばれる、ノイズカットの方法の 1 つです。この方法は、ノイズカットの手法として、かなりメジャーな方法です。興味のある人は、詳しい文献を調べてみると楽しいですよ。

　メディアンとは、「 中央値 」の意味で、ある数値の塊の中で、中央の大きさを持った値のことです。例えば、

1　4　5　8　12　3　6

があるとします。順に並べると、

1　3　4　5　6　8　12

ですね。

　この中で、（5+6）/2 が、メディアンになるわけです。

さて、このメディアンが、なぜノイズカットに役立つのかと言うと、

まず、図を見てください。

人間は、同じ色の点のつながった部分を「 形 」と見ています。これは、遠くから画像を見たときに、大体が形に見えるからです。

そして、ノイズとは、周りの点と極端に色の違う点の集まりです。

つまり、ノイズのある点を、周りの色と似せてしまえば、見かけ上、ノイズに見えなくなるのです。しかし、どうやって似せるのでしょうか？

人間が見れば、形として見えると言いましたが、コンピュータには、ただの数値の集まりにしか見えません。「 周りと違う 」のが、「 どうして 」違うのかは、数値だけを見てもわからないのです。どうしたものか…そこで！メディアンが登場するわけです。

ある点の周りの色どうし（8 方の点）が似ているなら、その真中の点も周りといっしょである可能性が高いです。

しかし、上半分 4 つと下半分 4 つとが色違いの場合は、どうでしょう。たまたま、その個所だけその色かもしれません。

その座標の数値を、回りの座標との平均値にしても良いのですが、そうすると、図のような場合、画像が変になってしまいます。

そこで、極端な値は端に行き、中央値としての値が必然と選ばれる、つまり、どんな場合にも対応できるメディアンフィルタが、ノイズカットの方法になっているわけです。

メディアンフィルタと平均フィルタ（移動平均フィルタ）を関数にしてみましょう。

```
int median(int* p_value,const int Number)
{
        register i;
        register u;
        int p_TMP[Number];

        int MEDI=(-1);

        for (i=0;i<= Number;i++)
        {
                p_TMP[i]=0;

                for (u=i;u<= Number;u++)
                {
                        if (p_value[i]>=p_value[u])
                        {
                                p_TMP[i]++;
                        }
                        else
                        {
                                p_TMP[u]++;
                        }
                }

                if (p_TMP[i]==Number/2)
                {
                        MEDI=p_TMP[i];
                }

        }
                        return p_value[MEDI];
}
```

＜main.cpp＞

```
#include "hhmath.h"

int MEDI[8];

register i,u;
register r,t;

int NOW;

int RR,GG,BB;

int ADC;

for (u=0;u<=HEIGHT-1;u++)
{

        for (i=0;i<=WIDTH;i++)
         {

         ADC=(-1);

         Pos.x=i-2;

         Pos.y=u-2;

                    for (t=(-1);t<=1;t++)
                    {

                    Pos.y++;

                    Pos.x=i-2;

                              for (r=(-1);r<=1;r++)
                              {

                                        Pos.x++;

                                        ADC++;

                                        MEDI[ADC]=0;

                                        NOW=GetPos(Pos);

                                        if (NOW!=(-1))
                                        {
```

```
/************移動平均の場合
                              RR+=COL[NOW].R;
                              GG+=COL[NOW].G;
                              BB+=COL[NOW].B;
*************/

/************メディアンの場合
                              MEDIR[ADC]=COL[NOW].R;
                              MEDIG[ADC]=COL[NOW].G;
                              MEDIB[ADC]=COL[NOW].B;
***************/

      MEDI[ADC]=(   ( COL[NOW].B+((COL[NOW].G)<<8) )+((COL[NOW].R)<<16)
);
                              }
                        }
                  }

if (ADC>0)
{
***************移動平均の場合
        COL[NOW].R＝ RR;
        COL[NOW].G＝ GG;
        COL[NOW].B＝ BB;
***************/

/************メディアンの場合
        COL[NOW].R=(median8(MEDIR,ADC)>>16);
        COL[NOW].G=(median8(MEDIR,ADC)>>8);
        COL[NOW].B=(median8(MEDIR,ADC));
***************/
}


        }
}
```

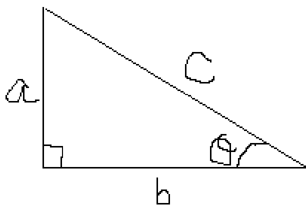
【Mission7：実践③ 回転】

次は、画像の回転です。画像の回転とは、文字通り、画像を回転することです。

画像を回転する前に、数学の話をしなければなりません。

中学生以下の皆さんは、ご存知無いかもしれませんが、数学の関数に、
「三角関数」と言うものがあります。

これは、簡単に言うと、「 角度 」と、「 ３角形の辺 」との関係を表したもので、
次の様に定義されます。



SINθ　＝ A ÷ C
COSθ ＝ B ÷ C
TANθ ＝ A ÷ B

SIN・COS・TAN は、それぞれ三角関数の関数です。

さて、ある点 A（X1、Y1）を、θずらした点 B（X2,Y2）との関係を式に表すと、B が、回転後の座標になります。

これを使い、１ドットずつ回転後の座標を出し、回転前の座標と置き換えることで、画像全体の回転が可能になります。

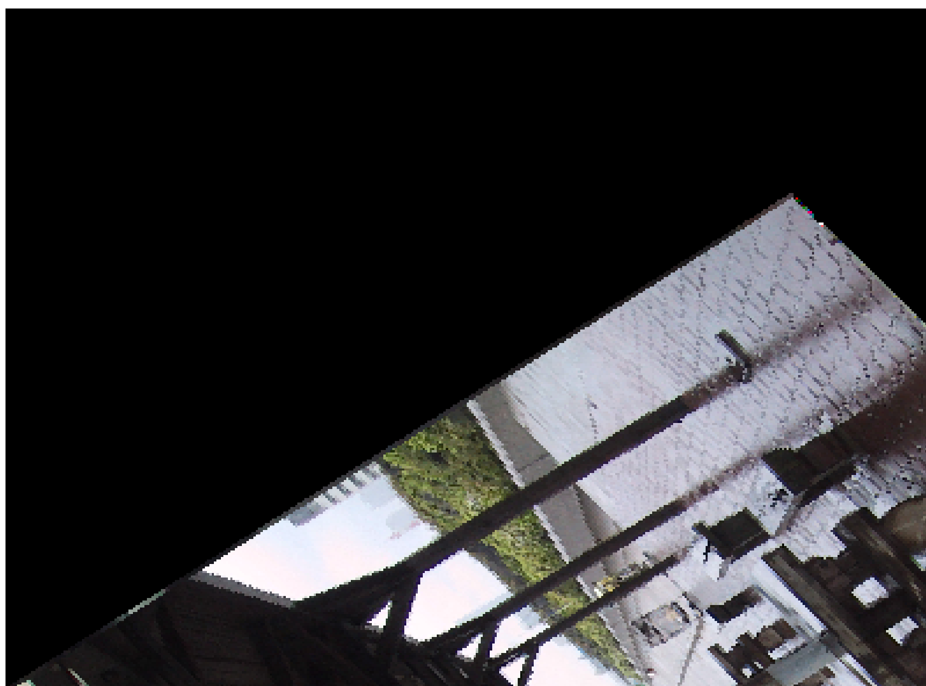
1 ドットずつ、次のページの「 GetRotate 」を呼び出せば、返り値に回転後の座標の、メモリアドレスを返しますから、そのアドレスに、回転前の座標の値を代入すれば、回転が出来るわけです。

注意：あらかじめ、データを読み込んだのと同じサイズの CPicture を作っておき、そこに代入してください。これをしないと、回転後の座標によっては、上書きされ、きちんと回転しない場合があります。

```
struct XY
{
        int Height;
        int Width;
        int x;
        int y;
};

int GetRotate(XY POS,double SHITA)
{
                int XX,YY;
                int RAD=((SHITA/360)*(3.1415926535*2));

                const double COS=cos(RAD);
                const double SIN=sin(RAD);

                XX=POS.x*COS-POS.y*SIN;
                YY=POS.y*COS+POS.x*SIN;

        if (    (XX<=POS.Width)&&(XX>=0)&&(YY<=POS.Height)&&(YY>=0)    )
        {
                return (POS.Height*POS.Width)-(YY*POS.Width+XX);
        }
        else
        {
                return -1;
        }
}
```

<出力結果>

【↓：－300 度回転】



2004年9月16日　初版発行

著者　　オイダス・ケイ
発行人　オイダス・ケイ
発行所　オイダス出版
　　　　http://www.oidus.com/oidus-print/　・・・予定