[OpenGL 2D 2018第03回]

画像を動かす

# タイトル画面を作る

## タイトルロゴを配置する

今回は「タイトル画面」を追加し、よりゲームらしくしていきましょう。  
新しいじゃんけんゲームのapplication関数には、次のようなプログラムが書き加えられています。

27 // タイトル画面を表示  
28 if (title\_flag) {  
29 title\_flag = false;  
30 /\*  
31 \* この下にタイトル画面のプログラムを追加します.  
32 \*/  
33 }

みなさんには「この下にタイトル画面のプログラムを追加します」と書かれている部分に、タイトル画面のプログラムを作ってもらいます。

タイトル画面といえばまずはタイトルロゴです。  
タイトルロゴはjanken\_title.pngという画像ファイルで用意してあります。まずはこれを表示してみましょう。画面に画像を配置するにはset\_image関数を使うのでしたね。  
それでは、次のように32行目の下にプログラムを追加してください。

30 /\*  
 31 \* この下にタイトル画面のプログラムを追加します.  
 32 \*/  
**+**33 set\_image(1, 0, 80, "janken\_title.png");  
 34 }

行頭に「+」記号が付いているのが追加する行です。

書けたら早速実行してみましょう！  
…うーん、何も表示されませんね。何がいけないんでしょう？

## 入力を待つ

さきほどのプログラムの続きを見てみると、普通にじゃんけんゲームが開始されています。  
つまり、「コンピューターはタイトルロゴを表示しようとした。けれど、すぐにゲームが始まってしまい、タイトルロゴは飛ばされてしまった」ということが原因のようです。  
ということは「すぐにはゲームが始まらない」ようにできれば、タイトルロゴを見ることができそうですね。

新しいCommand.hには、そんなときにぴったりの関数を追加してあります。  
その関数の宣言は次のようなものです。

/\*\*  
\* 何かキーが入力されるまで待つ.  
\*/  
void wait\_any\_key();

wait\_any\_key関数は、コメントでも説明されているように、何らかのキー入力があるまで待ちつづけます。早速この関数を使ってみましょう。次のように33行目の下にプログラムを追加してください。

30 /\*  
 31 \* この下にタイトル画面のプログラムを追加します.  
 32 \*/  
 33 set\_image(1, 0, 80, "janken\_title.png");  
**+**34 wait\_any\_key();  
 35 }

書けたら実行してみてください。  
今度こそタイトルロゴが表示されたでしょうか。

## メッセージを入れる

wait\_any\_key関数を使うことによって、タイトルロゴが表示された状態から、何かキーを押すとじゃんけんゲームが始まるようになりました。「これでタイトル画面は完成！」としたくなりますが、ちょっと待ってください。もし、みなさん以外の人にじゃんけんゲームを遊んでもらうとして、その人はタイトル画面でどうしたらゲームが始まるのかすぐに分かるでしょうか？

プログラムを書いたみなさんは「wait\_any\_key関数を使っているから何かキーを押せばゲームが始まる」ということを知っています。でも、他の人は知るはずがありません。そこで、画面にメッセージを表示して、何をしたらいいのかが伝わるようにしてみましょう。次のように33行目の下にプログラムを追加してください。

30 /\*  
 31 \* この下にタイトル画面のプログラムを追加します.  
 32 \*/  
 33 set\_image(1, 0, 80, "janken\_title.png");  
**+**34 set\_text(-192, -100, "何かキーを押してください");  
 35 wait\_any\_key();  
 36 }

書けたら実行してみてください。  
メッセージが表示されているのを確認したら、キーを押してゲームを始めてみましょう。  
…おや、メッセージが表示されたままになっています。

## 文字を消す

1.3節では、ゲームを始めるためのメッセージを表示しました。  
それはよかったのですが、ゲームが始まってもメッセージは画面に表示されたままです。  
このままではちょっと格好が悪いので、メッセージを消してみましょう。文字を消すにはreset\_all\_text関数を使うのがよさそうです。次のように33行目の下にプログラムを追加してください。

30 /\*  
 31 \* この下にタイトル画面のプログラムを追加します.  
 32 \*/  
 33 set\_image(1, 0, 80, "janken\_title.png");  
 34 set\_text(-192, -100, "何かキーを押してください");  
 35 wait\_any\_key();  
**+**36 reset\_all\_text();‌  
 37 }

書けたら実行してみてください。  
キーを押してゲームを開始したときにメッセージが消えていれば成功です。

# move\_image関数

画像が配置されているだけ、というのは味気ないので、タイトルロゴに動きを付けてみましょう。  
画像を移動させるにはmove\_imageという関数を使います。

/\*\*  
\* 画像を移動する.  
\*  
\* @param no 画像の管理番号.  
\* @param x 移動先のX座標.  
\* @param y 移動先のY座標.  
\* @param easing 補間方法:  
\* 0 等速で移動.  
\* 1 加速しながら移動.  
\* 2 減速しながら移動.  
\* 3 加速しながら移動を始めて、減速しながら停止する.  
\* 4 1に似ているが、少し行き過ぎてから戻ってきて停止する.  
\* 5 1に似ているが、移動先座標で何度か弾んでから停止する.  
\* @param seconds 動作時間(秒).  
\*  
\* 位置はウィンドウの中心を原点とし、左右がX軸(右がプラス方向)、上下がY軸(上がプラス方向)の座標系  
\* で指定する.  
\* ウィンドウの大きさは横800ドット、縦600ドットである.  
\* ここで指定する座標は画像の中心を指す.  
\*/  
void move\_image(int no, float x, float y, int easing, float seconds);

説明がいろいろ書いてありますが、要約すると「管理番号、X座標、Y座標、移動方法、移動時間」の5種類のデータを設定すれば画像が動きます、ということです。

試しに「タイトルロゴが左から真ん中まで移動する」という動きを作ってみましょう。次のように33行目を変更し、その下にプログラムを1行追加してください。

30 /\*  
 31 \* この下にタイトル画面のプログラムを追加します.  
 32 \*/  
**!**33 set\_image(1, -800, 80, "janken\_title.png");  
**+**34 move\_image(1, 0, 80, 0, 2);  
 36 set\_text(-192, -100, "何かキーを押してください");  
 37 wait\_any\_key();  
 38 reset\_all\_text();‌  
 39 }

行頭に「+」記号が付いているのは追加する行、「!」記号が付いているのが変更する行です。  
なお、「!」記号の行については、どこが変更されているかを注意深く見るようにしてください。

プログラムを書き終わったら実行してみてください。  
タイトルロゴが左から画面に入ってきて中央まで移動したら成功です。

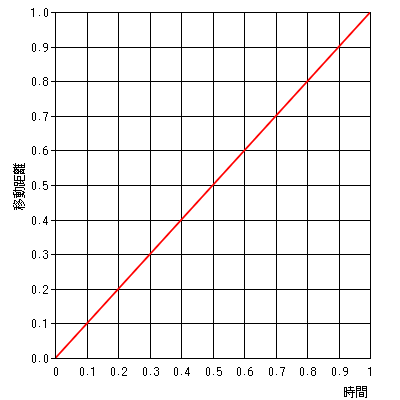
# イージング

2章で作ったタイトルロゴの動きは速度の変化などが全くなく、あまり面白みがありません。  
例えば慣性が働いているような動きをつけるとリアリティが増して格好よく見えます。このような動きをつけることを「イージング(easing,は「緩和する」という意味)」といいます。

実はmove\_image関数には既にイージング機能が備わっています。4番めの引数がそれです。  
この引数に0～5の整数を設定することで、6種類の動きをつけることができるようになっているのです。どの数値がどんな動きになるかを見ていきます。

## 0: リニア(linear)

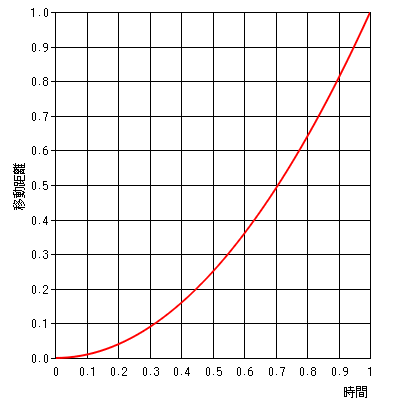
第4引数に「0」を設定したときの動きは最初から最後まで一定の速度で移動する、いわゆる「等速直線運動」的な動きです。グラフにするとこう。



これは移動距離も時間も0～1の場合ですが、言いたいことは分かってもらえると思います。

## 1: イーズ・イン(ease-in)

第4引数に「1」を指定したときの動きは、最初はゆっくりで徐々に加速しながら移動する動きです。グラフにするとこう。



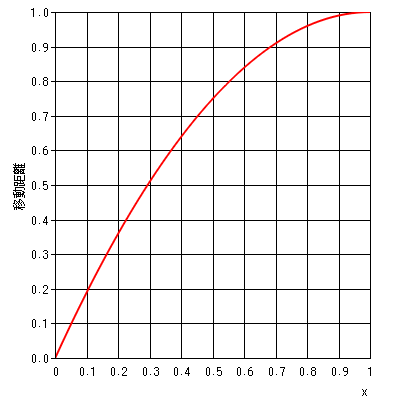
これがどのような動きになるのか、34行目のmove\_image関数の第4引数を0から1に変更して試してみましょう。

30 /\*  
 31 \* この下にタイトル画面のプログラムを追加します.  
 32 \*/  
 33 set\_image(1, -800, 80, "janken\_title.png");  
**!**34 move\_image(1, 0, 80, 1, 2);  
 36 set\_text(-192, -100, "何かキーを押してください");  
 37 wait\_any\_key();  
 38 reset\_all\_text();‌  
 39 }

プログラムを変更したら実行してみてください。分かりにくいですが、最初はゆっくりで、徐々に速く移動していると思います。

## 2: イーズ・アウト(ease-out)

第4引数に「2」を指定したときの動きは、最初は速く、徐々に減速しながら移動する動きです。図にするとこう。



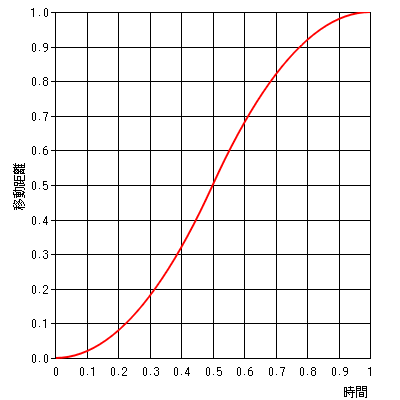
イーズ・インを180度回転させたようなグラフになっていますね。これがどのような動きになるのか、34行目のmove\_image関数の第4引数を2に変更して試してみましょう。

30 /\*  
 31 \* この下にタイトル画面のプログラムを追加します.  
 32 \*/  
 33 set\_image(1, -800, 80, "janken\_title.png");  
**!**34 move\_image(1, 0, 80, 2, 2);  
 36 set\_text(-192, -100, "何かキーを押してください");  
 37 wait\_any\_key();  
 38 reset\_all\_text();‌  
 39 }

プログラムを変更したら実行してみてください。これも分かりにくいですが、最初は速く、徐々に遅く移動していると思います。

## 3: イーズ・イン・アウト(ease-in-out)

第4引数に「3」を指定したときの動きは、最初はゆっくり、徐々に加速し、最後には減速しながら移動する動きです。図にするとこう。



よく見ると、イーズ・インとイーズ・アウトを0.5倍してくっつけたグラフになっています。これがどのような動きになるのか、34行目のmove\_image関数の第4引数を3に変更して試してみましょう。

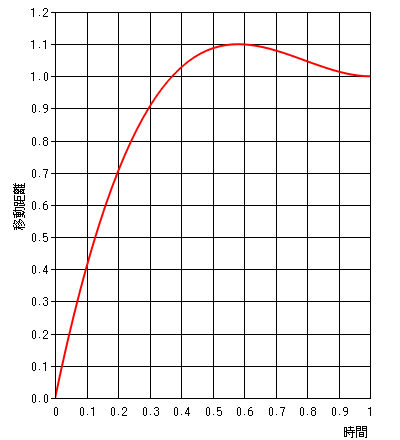
30 /\*  
 31 \* この下にタイトル画面のプログラムを追加します.  
 32 \*/  
 33 set\_image(1, -800, 80, "janken\_title.png");  
**!**34 move\_image(1, 0, 80, 3, 2);  
 36 set\_text(-192, -100, "何かキーを押してください");  
 37 wait\_any\_key();  
 38 reset\_all\_text();‌  
 39 }

プログラムを変更したら実行してみてください。これもかなり分かりにくいですが、最初はゆっくり、徐々に加速し、また減速して終わるような動きをしていると思います。

イーズ・イン・アウトは現実の物体の挙動に近い動きになります。リアリティのある動きをつけたい場合はまずこれを使ってみるとよいでしょう。

## 4: イーズ・アウト・バック(ease-out-back)

第4引数に「4」を指定したときの動きは、最初は速く、徐々に減速し、少し行き過ぎてから戻ってくるような動きです。図にするとこう。



イーズ・アウト・バックの大きな特徴は移動距離に1.0を越える部分があることです。このため、加速が付きすぎて行き過ぎてから戻るような動きに見えます。実際にどのような動きになるのか、34行目のmove\_image関数の第4引数を4に変更して試してみましょう。

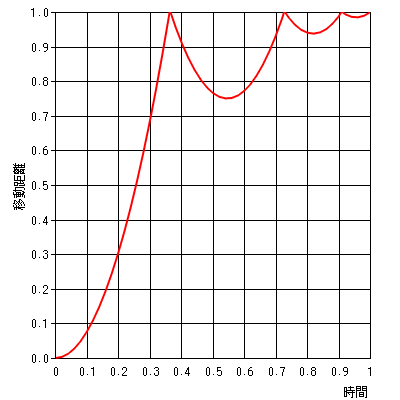
30 /\*  
 31 \* この下にタイトル画面のプログラムを追加します.  
 32 \*/  
 33 set\_image(1, -800, 80, "janken\_title.png");  
**!**34 move\_image(1, 0, 80, 4, 2);  
 36 set\_text(-192, -100, "何かキーを押してください");  
 37 wait\_any\_key();  
 38 reset\_all\_text();‌  
 39 }

プログラムを変更したら実行してみてください。イーズ・アウト・バックは「行き過ぎる」という特徴的な動きをするので、比較的違いが分かりやすと思います。

イーズ・アウト・バックは比較的軽い物体の動きを表現するのに向いています。

## 5: イーズ・アウト・バウンス(ease-out-bounce)

第4引数に「5」を指定したときの動きは、ボールを落としたときに何度か跳ねて停止するような動きです。図にするとこう。



他のグラフとの関係で上側で跳ね返っていますが、上下逆さまにして見れば、ボールを投げたときの動きに似ていることが分かると思います。実際にはどのような動きになるのか、34行目のmove\_image関数の第4引数を5に変更して試してみましょう。

30 /\*  
 31 \* この下にタイトル画面のプログラムを追加します.  
 32 \*/  
 33 set\_image(1, -800, 80, "janken\_title.png");  
**!**34 move\_image(1, 0, 80, 5, 2);  
 36 set\_text(-192, -100, "何かキーを押してください");  
 37 wait\_any\_key();  
 38 reset\_all\_text();‌  
 39 }

プログラムを変更したら実行してみてください。イーズ・アウト・バウンスは「数回跳ね返る」という特徴的な動きをするので、比較的違いが分かりやすと思います。

イーズ・アウト・バウンスを使うと、サッカーボールのように弾力のある物体が硬い壁にぶつかったり、その逆に硬い物体が弾力のある壁などにぶつかったときの動きを表現することができます。ただし、移動に要する時間が短すぎる(1秒程度)だと、跳ねるというよりすばやく振動しているだけに見えますので注意してください。

# scale\_image関数

画像は移動させるだけでなく、大きさを変えることもできます。  
画像の大きさを変えるにはscale\_imageという関数を使います。

/\*\*  
\* 画像を拡大・縮小する.  
\*  
\* @param no 画像の管理番号.  
\* @param x 拡大・縮小後のX方向の大きさ. マイナス値を設定すると左右が反転する.  
\* @param y 拡大・縮小後のY方向の大きさ. マイナス値を設定すると上下が反転する.  
\* @param easing 補間方法:  
\* 0 等速で拡大・縮小.  
\* 1 加速しながら拡大・縮小.  
\* 2 減速しながら拡大・縮小.  
\* 3 加速しながら拡大・縮小を始めて、減速しながら停止する.  
\* 4 1に似ているが、少し拡大・縮小し過ぎてから戻ってきて停止する.  
\* 5 1に似ているが、何度か弾むように拡大・縮小しなから停止する.  
\* @param seconds 動作時間(秒).  
\*/  
void scale\_image(int no, float x, float y, int easing, float seconds);

この関数の宣言にも説明がいろいろ書いてありますが、「管理番号、X方向の大きさ、Y方向の大きさ、大きさの変化方法、動作時間」の5種類のデータを設定すれば画像を拡大・縮小できます、ということです。画像の大きさは縦と横で別々に決められます。move\_image関数と同様に、4つめの引数によってイージングの種類を設定できます。

とりあえず使ってみましょう。次のように、34行目の下に、2つのscale\_image関数を追加してください。

30 /\*  
 31 \* この下にタイトル画面のプログラムを追加します.  
 32 \*/  
 33 set\_image(1, -800, 80, "janken\_title.png");  
 34 move\_image(1, 0, 80, 5, 2);  
**+**35 scale\_image(1, 0, 0, 0, 0);  
**+**36 scale\_image(1, 1, 1, 0, 2);  
 37 set\_text(-192, -100, "何かキーを押してください");  
 38 wait\_any\_key();  
 39 reset\_all\_text();‌  
 40 }

2行ありますが、まず35行目のscale\_imageは大きさを0にするためのものです。そして36行目で2秒かけて大きさを元に戻すように設定しています。

プログラムを追加したら実行してみてください。  
タイトルロゴが徐々に大きくなりながら登場していたら成功です。

# rotate\_image関数

画像を移動させたり大きさを変えるだけでなく、回転させることもできます。  
画像を回転させるにはrotate\_imageという関数を使います。

/\*\*  
\* 画像を回転する.  
\*  
\* @param no 画像の管理番号.  
\* @param degree 回転させる角度(度数法). マイナス値を設定すると回転方向が逆になる.  
\* @param easing 補間方法:  
\* 0 等速で回転.  
\* 1 加速しながら回転.  
\* 2 減速しながら回転.  
\* 3 加速しながら回転を始めて、減速しながら停止する.  
\* 4 1に似ているが、少し回転し過ぎてから戻ってきて停止する.  
\* 5 1に似ているが、何度か弾みなから停止する.  
\* @param seconds 動作時間(秒).  
\*/  
void rotate\_image(int no, float degree, int easing, float seconds);

この関数の宣言にも説明がいろいろ書いてありますが、「管理番号、回転角度、回転速度の変化方法、動作時間」の4種類のデータを設定すれば画像を回転できます、ということです。また、3つめの引数によってイージングの種類を設定できます。

とりあえず使ってみましょう。次のように36行目の下に、rotate\_image関数を追加してください。

30 /\*  
 31 \* この下にタイトル画面のプログラムを追加します.  
 32 \*/  
 33 set\_image(1, -800, 80, "janken\_title.png");  
 34 move\_image(1, 0, 80, 5, 2);  
35 scale\_image(1, 0, 0, 0, 0);  
36 scale\_image(1, 1, 1, 0, 2);  
**+**37 rotate\_image(1, 360, 0, 2);  
 38 set\_text(-192, -100, "何かキーを押してください");  
 39 wait\_any\_key();  
 40 reset\_all\_text();‌  
 41 }

書けたら実行してみてください。  
タイトルロゴが回転しながら登場したら成功です。

# color\_blend\_image関数

画像の色合いや透明度を変更したいこともあるでしょう。そんなときにはcolor\_blend\_imageという関数が使えます。

/\*\*  
\* 画像と色を合成する.  
\*  
\* @param no 画像の管理番号.  
\* @param red 合成する色の赤成分(0.0～1.0).  
\* @param green 合成する色の緑成分(0.0～1.0).  
\* @param blue 合成する色の青成分(0.0～1.0).  
\* @param alpha 合成する色の透明度(0.0～1.0).  
\* @param mod 合成方法:  
\* 0 乗算  
\* 1 加算  
\* 2 減算  
\* @param easing 補間方法:  
\* 0 等速で色を合成.  
\* 1 加速しながら色を合成.  
\* 2 減速しながら色を合成.  
\* 3 加速しながら合成を始めて、減速しながら停止する.  
\* 4 1に似ているが、少し色を出し過ぎてから戻ってきて停止する.  
\* 5 1に似ているが、何度か弾むように合成しなから停止する.  
\* @param seconds 動作時間(秒).  
\*/  
void color\_blend\_image(  
 int no,  
 float red, float green, float blue, float alpha,  
 int mode, int easing, float seconds);

コンピューターは色を光の三原色(赤、緑、青)で管理しています。コンピューターグラフィックスではさらに不透明さを表現するパラメータ(「アルファ値」といいます)を加えて、合計4つの値で色を表現します。

これもとりあえず使ってみましょう。次のように37行目の下に、２つのcolor\_blend\_image関数を追加してください。

30 /\*  
 31 \* この下にタイトル画面のプログラムを追加します.  
 32 \*/  
 33 set\_image(1, -800, 80, "janken\_title.png");  
 34 move\_image(1, 0, 80, 5, 2);  
35 scale\_image(1, 0, 0, 0, 0);  
36 scale\_image(1, 1, 1, 0, 2);  
37 rotate\_image(1, 360, 0, 2);  
**+**38 color\_blend\_image(1, 1, 1, 1, 0, 0, 0);  
**+**39 color\_blend\_image(1, 1, 1, 1, 1, 0, 2);  
 40 set\_text(-192, -100, "何かキーを押してください");  
 41 wait\_any\_key();  
 42 reset\_all\_text();‌  
 43 }

scale\_image関数のときと同様に2行追加しています。最初(38行目)のcolor\_blend\_imageは不透明さを0に(つまり完全な透明に)するためのプログラムです。そして次の39行目で、2秒かけて不透明に戻すように設定しています。

書けたら実行してみてください。  
タイトルロゴが徐々に不透明になりながら登場したら成功です。

# fade\_out関数とfade\_in関数

タイトル画面からゲームを開始するときのように、場面を切り替える時、場面が一瞬で切り替わってしまうと唐突に感じたり、味気なかったりします。  
場面転換としてまず思いつくのは、現在の場面が徐々に真っ暗になり、次の場面が徐々に明るくなりながら表示される、というものです。このような手法は「フェード・アウト(fade out)」、「フェード・イン(fade in)」と呼ばれています。そして、これを行う関数がfade\_out関数とfade\_in関数です。

この2つの関数は、Command.hの中で次のように宣言されています。

/\*\*  
\* 画面をフェードアウトする.  
\*  
\* @param red フェードアウトフィルターの赤成分(0.0～1.0).  
\* @param green フェードアウトフィルターの緑成分(0.0～1.0).  
\* @param blue フェードアウトフィルターの青成分(0.0～1.0).  
\* @param seconds 動作時間(秒).  
\*/  
void fade\_out(float red, float green, float blue, float seconds);  
  
/\*\*  
\* 画面をフェードインする.  
\*  
\* @param seconds 動作時間(秒).  
\*/  
void fade\_in(float seconds);

2つの関数のうちfade\_out関数だけが色を設定できるようになっています。なぜfade\_in関数には色を設定する機能がないのかというと、fade\_in関数の仕事は「fade\_out関数で設定した色から徐々に通常の画面へと戻す」ことだからです。色の設定はfade\_out関数で済ましているので、fade\_in関数では設定する必要がないわけです。

タイトル画面のゲームを開始する前の部分でフェード・アウトとフェード・インを使ってみましょう。  
以下のように、41行目の下付近に3行のプログラムを追加してください。

30 /\*  
 31 \* この下にタイトル画面のプログラムを追加します.  
 32 \*/  
 33 set\_image(1, -800, 80, "janken\_title.png");  
 34 move\_image(1, 0, 80, 5, 2);  
35 scale\_image(1, 0, 0, 0, 0);  
36 scale\_image(1, 1, 1, 0, 2);  
37 rotate\_image(1, 360, 0, 2);  
 38 color\_blend\_image(1, 1, 1, 1, 0, 0, 0);  
39 color\_blend\_image(1, 1, 1, 1, 1, 0, 2);  
 40 set\_text(-192, -100, "何かキーを押してください");  
 41 wait\_any\_key();  
**+**42 fade\_out(0, 0, 0, 1);  
 43 reset\_all\_text();‌  
**+**44 reset\_image(1);  
**+**45 fade\_in(1);  
 46 }

追加できたら実行してみてください。  
タイトル画面でなにかキーを押してゲームを開始した時、一度画面が真っ暗になり、再び戻ってきときに文字とタイトルロゴが消えていたら成功です。

# まとめ

今回見てきたように、プログラムによって画像にさまざまな演出を加えることが可能になります。  
また、移動や拡大・縮小、回転、色合いの変更といった関数は同時に使うことができます。これらを上手く組み合わせて、みなさん自身の手でオリジナルの演出を作り出してください。