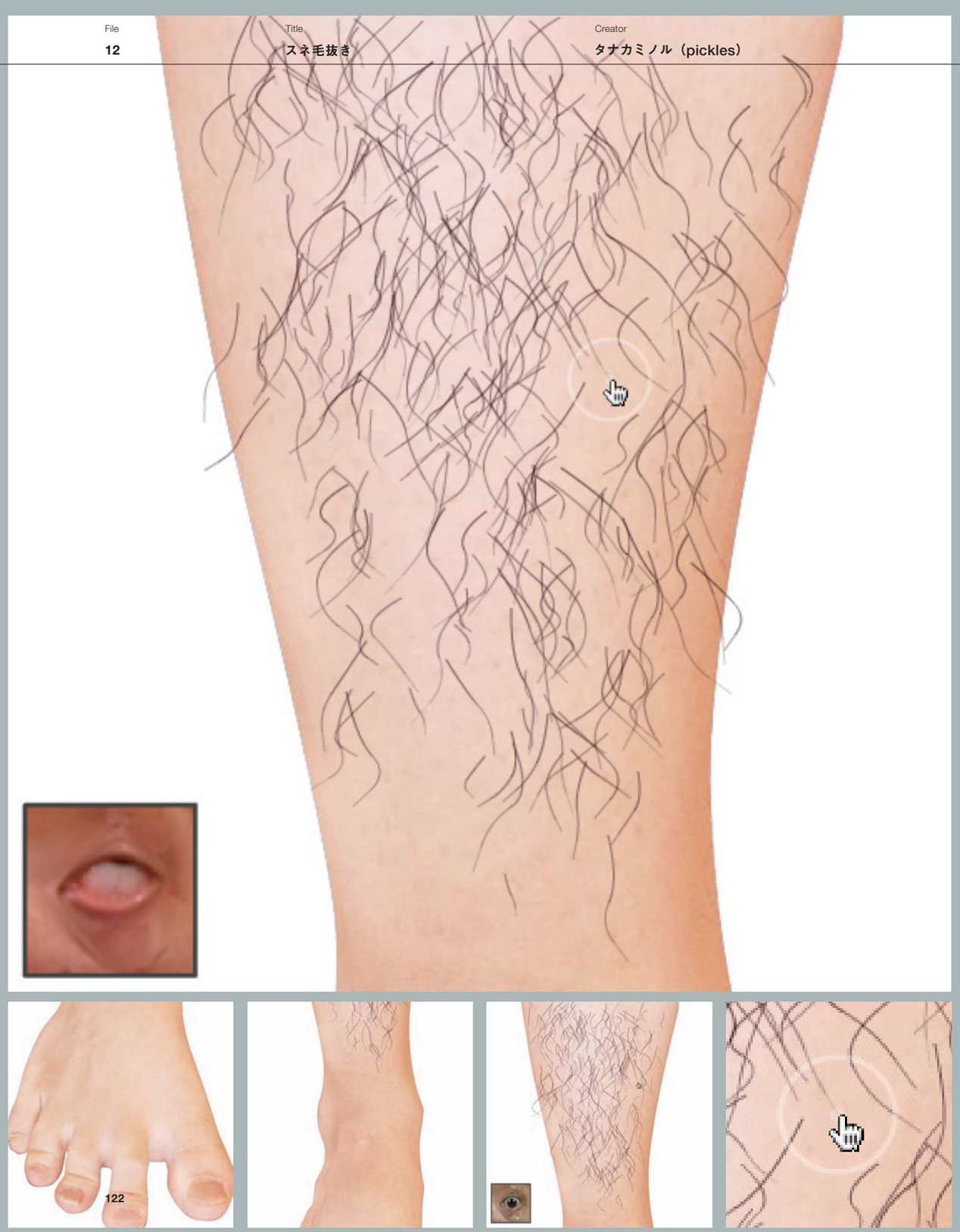


スネ毛抜き

Creator
タナカミノル（pickles）

「ひっぱると伸びる」線の動き

Flashによるインバースキネマティックスの実現。
ポイントは、2点間からcurveToの終点を求めること。

Title
スネ毛抜き

Sample URL
http://www.pickles.tv/dotfla/04/

Archive
sunegenuki.zip

File
12

スクリプト
ActionScript 1.0

対応プレーヤー
Flash Player 8以上

制作アプリケーション
Flash 8

発案～デザイン
「痛いから泣く」を表現する

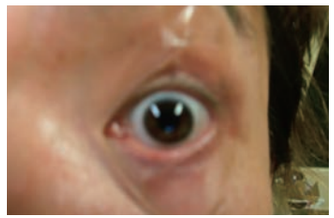
僕はかなり泣き虫でして、よく布団にくるまって泣いています。そんな布団の中で、「この泣くというのを端的に表現できないかな」と思ったのが発端です。

なんでもわかりやすくしてしまおうという気質なので、「痛いから泣く」となりまして、身近なのだけど、誰も表現したことがない「痛い」を考えました。それでいきついたのが、スネ毛抜きです（実際抜いたことあるのですが、かなり痛いです…）。ただ、それだけだと表現として弱いため、このスネ毛抜きと「目のアップ」を同時に表示し、引っ張られると痛がって泣くという演出にすることにしました。

○ グラフィック作り

目の部分は動画でやろうかなとも思ったのですが、普段使っているビデオカメラがマクロ撮影ができなかったので、静止画にすることにしました。ただ目を開いているだけではおもしろくないので、まぶたにテープを貼って、すごく見開いている感じを出しています。撮影したカットは「デフォルト」「痛がり」「ナミダ」の３種類です。

デフォルト



痛がり

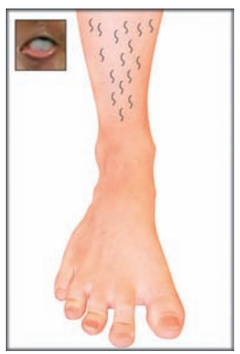
ナミダ

スネ毛自身はスクリプトで再現しようと思っていたので、スネ毛を生やすためにきれいなスネが必要でした。そのため、イヤイヤ片方だけスネ毛を剃り、美脚にして撮影しました。

これらの素材を組み合わせたグラフィックイメージが右図になります。

美脚にして撮影

グラフィックイメージ

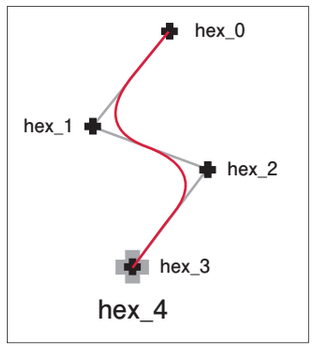


スクリプト
「ひっぱるとのびる」毛作り

今回のサンプルは、この毛作りに集約されています。毛を表現するのに、Flash MX から追加されたlineTo() とcurveTo() で描画しようと考えました。

ただ毛を抜く際にあらゆる方向に引っ張られても、その引っ張りにより毛が動的に変化しなければなりません。もちろん、引っ張るのを途中で止めたら、元の縮れた状態に戻ります。

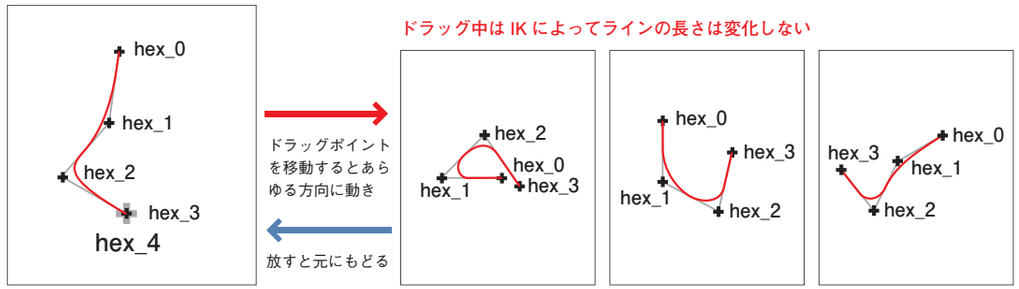
骨組みを操作する仕組み



骨組みの間接部分をランダムに設定する（hex_0 ～ hex_3）
↓
ドラッグできるポイントを設定する（hex_4）
↓
hex_4 の位置を元に、IK ルーチンを使い hex_1 ～ hex_3 の位置を動的に変化させる
↓
hex_0 ～ hex_3 の位置を元に曲線を描画（赤いラインが毛の部分）

そこで使えそうだなと思ったのが、「インバースキネマティックス」（IK）です。「Poser」などの 3D ソフトを使用したことのある人はわかると思いますが、骨組みを操作する仕組みと思ってください。

見えない骨組みを作り、その間接部分を線のアンカーポイントとして使用すれば、あらゆる動きに対応可能な線が描画できると考えました。



○ 初期設定

最初に骨組み用の空の MC を作ります。

SOURCE 骨組み用の空の MC の作成

```
this.createEmptyMovieClip("chichige", 0);
chichige._x = 300;// 出現 x 位置
chichige._y = 200;// 出現 y 位置
```

インバースキネマティックス用の間接部分であるアンカーポイントを配列として作ります。

SOURCE 配列としてアンカーポイントを作成

```
var point = [[0, 0, 0, 0]];//「長さ、角度、x位置、y位置」の配列を設定
var hoge = 3;// ラインの本数
var nagasa = 60;// 長さを設定
for (var i = 0; i<hoge; ++i) {
    var kakudo = 30+Math.floor(Math.random()*150);// 角度をランダムに設定
    var rad = kakudo*Math.PI/180;// 角度をラジアンに変更
    // 前の配列の位置とラジアンから次の位置を設定
    var xpos = point[i][2]+Math.floor(nagasa*Math.cos(rad));
    var ypos = point[i][3]+Math.floor(nagasa*Math.sin(rad));
    point.push([nagasa, kakudo, xpos, ypos]);// 設定した情報を配列に追加
}
// 最後位置をドラッグポイント位置として配列に追加
point.push([point[hoge][0], point[hoge][1], point[hoge][2], point[hoge][3]]);
```

A

上記設定した配列から各アンカーポイントを配置します。

SOURCE 配列から各アンカーポイントを配置

```
for (var i = 0; i<hoge+1; ++i) {
    var newname = "hex_"+i;// インスタンス名を設定
    var syoki = {_x:point[i][2], _y:point[i][3]};// 初期位置
    chichige.attachMovie("hex", newname, i, syoki);// 生成
}
```

ドラッグポイントを配置し、動作を設定します。

SOURCE ドラッグポイントの配置と設定

```
// 配置
var dp_name = "hex_"+(hoge+1);// インスタンス名を設定
var dp_syoki = {_x:point[hoge+1][2], _y:point[hoge+1][3]};// 初期位置
chichige.attachMovie("hex2", dp_name, 10, dp_syoki);// 生成
//
// プレス
chichige[dp_name].onPress = function() {
    this.startDrag();// ドラッグスタート
    ikmodoki();// インバースキネマティックスもどき関数動作
    linemove();// ライン描画関数動作
    this.onEnterFrame = function() {
        var ke_nobi = 0;//1 本のラインの伸び
        var ke_nobi_max = 20;//1 本のラインの最大伸び
        // 原点とドラッグポイントの距離を平方根から出す
        var dp_nagasa = Math.round(Math.sqrt(this._x*this._x+this._y*this._y));
        var line_total = nagasa*hoge;// 元のラインの長さ
        var sa_nagasa = dp_nagasa-line_total;// ドラッグポイントの距離と元のラインの長さの差
        var sa_nagasa_max = ke_nobi_max*hoge;// 全てのラインの伸びの合計
```

次のページに続く


```
        if (0<sa_nagasa && sa_nagasa<sa_nagasa_max) {
//2 つの差の長さが、0 を超え 60 未満だったら
            ke_nobi = Math.round(sa_nagasa/hoge);// トータルの差の長さから 1 本のラインの伸びを出す
            for (i=1; i<=hoge; i++) {
                // アンカーポイントの配列の長さに代入
                this._parent._parent.point[i][0] = nagasa+ke_nobi;
            }
        }
    };
};
//
// リリース（放すと元の位置に戻る）
chichige[dp_name].onRelease = function() {
    this.stopDrag();// ドラッグ停止
    this._visible = false;// 戻ってる最中は非表示
    for (i=1; i<=hoge; i++) {
        this._parent._parent.point[i][0] = nagasa;// アンカーポイントの配列の長さを元に戻す
    }
    this.onEnterFrame = function() {
        // 最初の位置に戻す
        this._x += (point[hoge+1][2]-this._x)/10;
        this._y += (point[hoge+1][3]-this._y)/10;
        var xhoge = Math.abs(point[hoge+1][2]-this._x);
        var yhoge = Math.abs(point[hoge+1][3]-this._y);
        if (xhoge<=1 && yhoge<=1) {
            this._x = point[hoge+1][2];
            this._y = point[hoge+1][3];
            this._visible = true;// 戻り終わったら表示
            delete this._parent.onEnterFrame;// インバースキネマティックスもどき関数停止
            delete this._parent._parent.onEnterFrame;// ライン描画関数停止
            delete this.onEnterFrame;
        }
    };
};
```

ここでのポイントは、プレス時の動作になります。毛はある程度は伸び縮みするので、「原点とドラッグポイントの距離」と「最初の長さ」を比較し、「最初の長さ」より長ければ、その分伸びるようになっています。

○インバースキネマティックスもどきルーチン

このスクリプトでは３点のみ求めるようになっています。

SOURCE インバースキネマティックスもどきルーチン

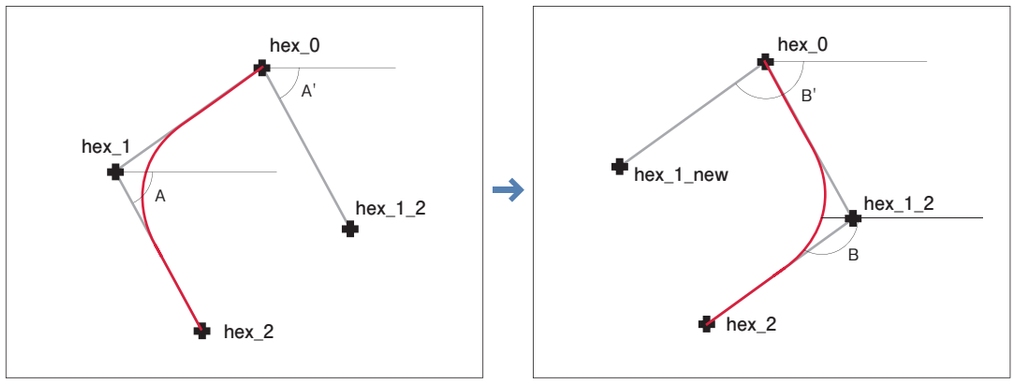
```
function ikmodoki() {
    chichige.onEnterFrame = function() {
        for (var i2 = 1; i2<=hoge; i2++) {
            // 現在の hex_4 の位置から新しい hex_3 の位置を求める
```

次のページに続く

```
            var xpos = this.hex_4._x-this.hex_2._x;//2 点間の X 距離
            var ypos = this.hex_4._y-this.hex_2._y;//2 点間の Y 距離
            // アークタンジェント 2 でラジアン（角度）を出し、hex_3 の位置を決める
            var rad = Math.atan2(ypos, xpos);
            this.hex_3._x = this.hex_2._x+this._parent.point[3][0]*Math.cos(rad);
            this.hex_3._y = this.hex_2._y+this._parent.point[3][0]*Math.sin(rad);
            this._parent.point[3][2] = this.hex_3._x;// 配列の X に代入
            this._parent.point[3][3] = this.hex_3._y;// 配列の Y に代入
            // 上記を hex_2 に対して繰り返し
            var xpos = this.hex_3._x-this.hex_2._x;
            var ypos = this.hex_3._y-this.hex_2._y;
            var rad = Math.atan2(ypos, xpos);
            this.hex_2._x = this.hex_1._x+this._parent.point[2][0]*Math.cos(rad);
            this.hex_2._y = this.hex_1._y+this._parent.point[2][0]*Math.sin(rad);
            this._parent.point[2][2] = this.hex_2._x;
            this._parent.point[2][3] = this.hex_2._y;
            // 上記を hex_1 に対して繰り返し
            var xpos = this.hex_2._x-this.hex_1._x;
            var ypos = this.hex_2._y-this.hex_1._y;
            var rad = Math.atan2(ypos, xpos);
            this.hex_1._x = this.hex_0._x+this._parent.point[1][0]*Math.cos(rad);
            this.hex_1._y = this.hex_0._y+this._parent.point[1][0]*Math.sin(rad);
            this._parent.point[1][2] = this.hex_1._x;
            this._parent.point[1][3] = this.hex_1._y;
        }
    };
}
```

各アンカーポイントの位置の求め方は、三角関数の Math.atan2() メソッドを使用して求めています（atan2 は 2 点間の角度を出すメソッドです）。今回は、図のように変わった求め方をしています。

移動先の求め方



1. hex_1 と hex_2 の位置から A の角度を求める
2. A と A' の角度は同一なので hex_0 を起点として A の角度と配列の距離から hex_1_2 の位置を求める
3. 今度は hex_2 と hex_1_2 の位置から B の角度を求める
4. B と B' の角度は同一なので hex_0 を起点として B の角度と配列の距離から hex_1_new の位置を求める

一度の計算だと前ページの図の左側のみの処理になり逆の位置を求めるだけになってしまいますので、同一の計算を繰り返すように for 文を使用して求めています。繰り返す回数はライン数と同数になります。もし 6 本のラインであれば 6 回繰り返して正確な移動先を求めるようになっています。

試しに、2<=hoge の hoge とフレームレートを 1 にしてみると、「なぜ繰り返しているか？」がわかりやすいと思います。

TIPS 角度と位置の関係

角度と位置の関係についての説明は、以下に詳しく載っています。興味を持たれた方はこちらを読み進めてください。

URL 角度と座標の計算—Flash の三角関数を使う
http://www.macromedia.com/jp/support/flash/ts/documents/fl0189.html

なお、本文で“インバースキネマティックスもどき”と書いている理由は、試行錯誤して無理矢理やった方法で、この求め方はかなりエセなやり方らしいからです。まともなインバースキネマティックス（IK）ルーチンを作りたい方は、構造力学などを勉強することをお勧めします。

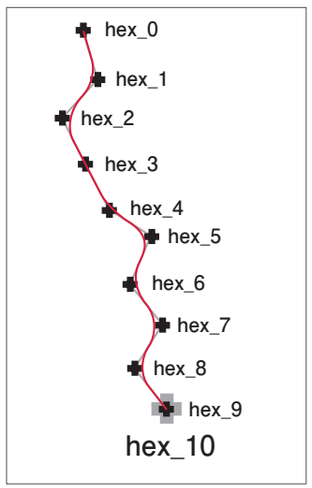
上記ルーチンは途中から繰り返しになっています。for 文を使って記述すると、以下のようになります。

SOURCE IK ルーチン（最終版）

```
function ikmodoki() {
    chichige.onEnterFrame = function() {
        for (var i2 = 1; i2<=hoge; i2++) {
            for (var i = 1; i<=hoge; i++) {
                // 各インスタンス名を設定
                var name_4 = "hex_"+((hoge+2)-i);
                var name_3 = "hex_"+((hoge+1)-i);
                var name_2 = "hex_"+(hoge-i);
                if(i==1){
                    // 一回目だけこっちでやる
                    var xpos = this[name_4]._x-this[name_2]._x;
                    var ypos = this[name_4]._y-this[name_2]._y;
                } else {
                    var xpos = this[name_4]._x-this[name_3]._x;
                    var ypos = this[name_4]._y-this[name_3]._y;
                }
                var rad = Math.atan2(ypos, xpos);
                this[name_3]._x = this[name_2]._x+this._parent.point[(hoge+1)-
                                                                    i][0]*Math.cos(rad);
                this[name_3]._y = this[name_2]._y+this._parent.point[(hoge+1)-
                                                                    i][0]*Math.sin(rad);
                this._parent.point[(hoge+1)-i][2] = this[name_3]._x;
                this._parent.point[(hoge+1)-i][3] = this[name_3]._y;
            }
            */
        }
    };
}
```

この「var hoge = 3;// ラインの本数」の 3（126 ページのⒶの部分）を変えると、下の図のように複数の関節を持たせることができます。最大のライン数は「9」です。

複数の関節を持たせることも可能



ラインの描画

IK ルーチンによってアンカーポイントの値を求めたら、それをもとにラインを描画します。

SOURCE ラインの描画

```
function linemove() {
    this.onEnterFrame = function() {
        line();
    };
}
function line() {
    this.createEmptyMovieClip("chichige2", 1);// 空の MC を生成
    this.chichige2._x = 300;// 出現 x 位置
    this.chichige2._y = 200;// 出現 y 位置
    this.chichige2.lineStyle(1, 0x333333, 70);// 線の内容を設定
    this.chichige2.moveTo(point[0][2], point[0][3]);// 線の開始
    // アンカーポイント同士を直線で結ぶ
    for (var i2 = 0; i2<hoge; i2++) {
        this.chichige2.lineTo(point[i2+1][2], point[i2+1][3]);
    }
    this.chichige2.lineStyle(1, 0xFF0000, 70);// 線の色を変更
    this.chichige2.moveTo(point[0][2], point[0][3]);// 線の開始
    // アンカーポイントをコントロールポイントとし、アンカー同士の中心点を終点として曲線を引く
    for (var i = 0; i<hoge; i++) {
        var point2_x = point[i][2]-((point[i][2]-point[i+1][2])/2);
        var point2_y = point[i][3]-((point[i][3]-point[i+1][3])/2);
        this.chichige2.curveTo(point[i][2], point[i][3], point2_x, point2_y);
    }
    this.chichige2.lineTo(point[hoge][2], point[hoge][3]);// 最後にちょこっと残った部分に直線を引く
}
```


このスクリプトのポイントは、2 点間から curveTo() での終点を求めているところにあります。もともと IK を使って毛を表現しようと思った理由の 1 つに、Flash のベジェが二次ベジェ曲線だった点があげられます。二次ベジェは三次ベジェと違い複雑な曲線は描画しにくいのですが、コントロールポイントが少ないので、なめらかで違和感のない曲線を描画するのに適しているといえます。

フィニッシュ

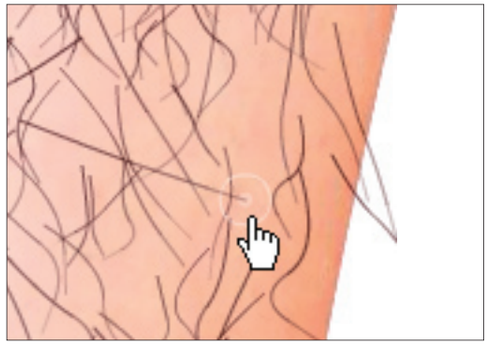
sunege_02.fla がフィニッシュファイルなので中身を見てもらえればわかると思うのですが、フィニッシュでは、上記スクリプトをもとにいろいろ追加しているだけです。

追加した点は、おおまかに以下のようになります。

- 上記を 1 つの MC にまとめてリンケージし、200 個ほどスネの回りに配置する
- 毛が抜けたら、その MC ごと落下して消えるようにする
- 目のサムネールを作って、毛の状態に合わせて画像が変化するようにする
- 毛が徐々に細くなるようにする

メインのスクリプトは 2 カ月くらいダラダラいじってなんとか制作し、それからフィニッシュまでは 12 時間ほどでできたので、さほど難しいことは追加していません。

マウスでつかんでひっぱると伸びる



抜けた毛は落下して消滅

めったに使われない lineTo() と curveTo() ですが、動的な線の描画ができるので、これからはもっとよく使われるのではないかと思っています。