scratch

Creator
セトウナオ（STAC STAR）

ビットマップ画像を使って「削る」動きを表現

αチャンネル対応のビットマップデータと組み合わせ、
αに抜くことで削っているように見せる。

Title
scratch

Sample URL
http://stacstar.jp/scratch/

Archive
scratch.zip

File
13

スクリプト
ActionScript 2.0

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
大きなスクラッチを削る爽快さを表現

このコンテンツは、現れるコインをつかみスクラッチをひたすら削るだけです。削りカスやコインの動き、スクラッチ全体の動きなど登場するアイテムをリアルに表現することで爽快感を演出しています。また、背景の絵を容易に変えられるように外部の素材を読み込ませているので、スクラッチ本来の何が出るのかという楽しみ方もあります。

○ コンセプト

このコンテンツのコンセプトは、誰もが削ったことのあるスクラッチをステージいっぱいに配置し、その大きなスクラッチを削ることの爽快さを表現することでした。

ここに現れるアイテムは

- スクラッチ
- コイン
- スクラッチのカス
- スクラッチの下の絵

の４つだけです。

爽快さを表現するためには、その１つ１つにリアリティを持たせることが重要になります。短い時間で各アイテムのリアリティをユーザーに認知させるためには、実際とは異なるデフォルメした表現も必要です。

また、Flash ８から加わった「エフェクト」を使用し、いままでの Flash のベクターを中心とした表現とは違った表現を演出しています。

○ コインの表現

コインはユーザーが操作するアイテムになります。

スクラッチを削るときコインの状態は

- 置いてある状態
- 立っている状態
- 動いている状態

の３つになります。

この状態をユーザーのマウス操作でスムーズにつなぎ、表現することでリアルな操作感を演出させています。

コインの状態をユーザーの操作に置き換える

置いてある状態 →	立っている状態	コインを「つかむ」動作
立っている状態 →	置いてある状態	コインを「離す」動作
動いている状態		コインを「動かす」動作

※コインの３つの状態をつなぐことで削る動作を表現

「つかむ･離す」に関しては、数枚の写真を実際に傾斜を付けて撮影を行い、アニメーションさせました。また、スピードを付けてコインを離したときには、普通に離したときのアニメーションとは違い、回転しながらカタカタと小刻みに揺れながら落ち着くというコインらしい動きを付けました。実際、スクラッチをしていてこのような動きをすることはないと思います。

しかし、ここでは「コイン」であるという表情を強く訴えたくて、あえてこの表現を加えました。この傾きながら落ち着くアニメーションでは、実際に撮影したコインの写真を使っていません。それは、アニメーションさせたときに光の反射の部分のつなぎが困難だったからです。そこで、正面から撮った画像に画像編集ソフトで遠近を付けて、それをうまくつなげました。

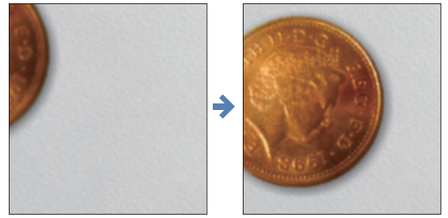
○ スクラッチの表現

スクラッチの画像は質感の似た反射の少ないパネルを撮影し、それに画像編集ソフトで色合いや質感を調整していきました。しかし、いくら繊細に表現しようが、日常それほど目にすることのないスクラッチを画像１枚で認識させるのはなかなか困難です。

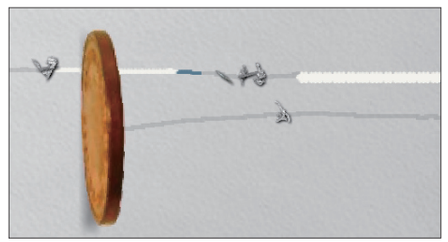
スクラッチはコインと合わせることで初めて「スクラッチ」であると認識できるでしょう。しかし、それをあえて利用してコインの登場を遅らせました。一度ユーザーに「これはなんだろう？」と思わせることにより興味を惹くためです。

スクラッチを削ったときに削れるのはもちろんですが、実際に削ってみると弱く削ったときやコインが滑ってしまうときは削れずにその部分だけキズが付きます。そういった細かな演出も取り入れています。

ムービー再生後、コインはすぐに登場しない



細かなキズの表情



○ カスの表現

特にスクラッチの表現を豊かにするのがカスの存在です。通常、スクラッチを使うコンテンツでは下に出てくる「アタリ」や「ハズレ」という「結果」が重要になりますが、今回作成したものは下の面に絵があるだけで「結果」はありません（もちろんランダムな絵を用意してアタリ／ハズレを用意することもできます）。結果が目的である

コンテンツでは、カスが邪魔な存在にさえなってしまいます。

しかし、このコンテンツでは「カス」の存在こそがリアリティを演出する大きな役割となっています。カスに関しては、多くのスクラッチを削って実際に出てきたカスをいくつも撮影しました。1つのカスに対して、6カット程度用意して画像編集ソフトで切り抜き、色彩調整を行いました。その画像を用いFlash上でアニメーションになるようにコマをつないでいます。そして、スクラッチを削っている速さ（強さ）に合わせて、カスの大きさや角度を変えています。

実際には、削りカスはコインにくっついてしまって、このコンテンツのように細かなカスがたくさん出てくることはありません。しかし、実際に起こることを忠実に表現するよりも、あえて頭の中にあるイメージとおりにカスを強調して表現してあります。削ったところに細かなカスを実際よりもたくさん出現させることで「削っている」という爽快感を強調しています。

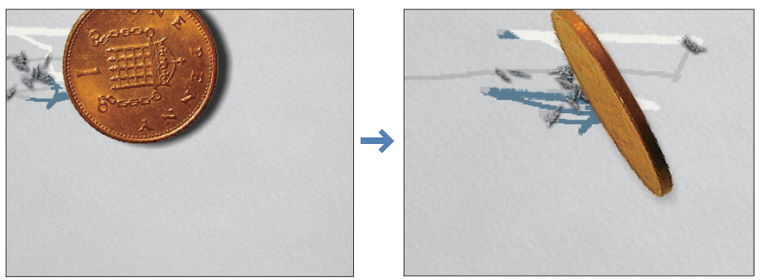
カスの表情

カメラの動き

はじめに作ったバージョンでは、スクラッチ全体をズームさせて動かすことはしていませんでしたが、実際に操作してみると、何か迫力が足りません。そこで、削っている部分をズームアップにしてカメラをアップしているような表現を加えました。

Flashには3Dソフトのように「カメラ」の設定はありません。しかし、対象のオブジェクトを動かすことで視線を変えるような表現が可能です。

削っている部分をズームアップ



実際にスクラッチを削っているときに視線を近付けることはあまりないとは思いますが、少なくとも削っている部分に集中しているでしょう。そこで、削っている部分にズームアップするという表現を加えました。これにより、自分が削っているというユーザーの視線がだいぶ強調され、気持ちのよいコンテンツに仕上がったと思います。

スクリプト
スクラッチを削る動作のスクリプト化

このスクリプトでは、以下の各クラスを使ってそれぞれの動作を表現します。

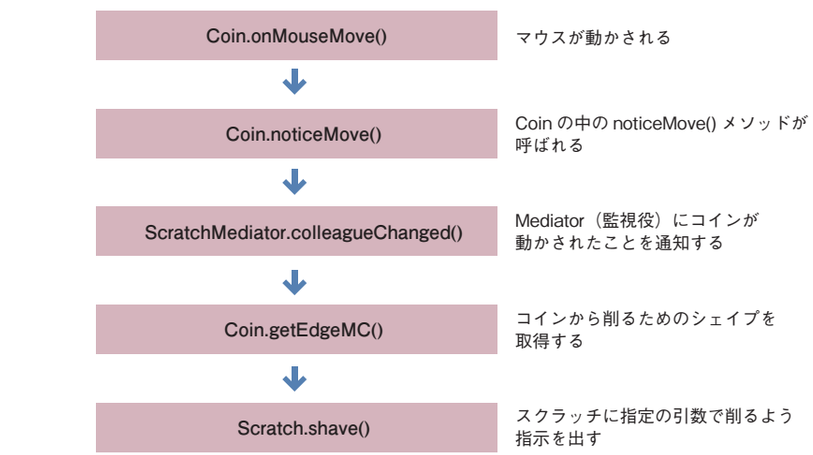
スクリプトで使用する各クラス

クラス名	概要
BitsOfScratch	スクラッチのカスを表すクラス
BitsOfScratchEdge	スクラッチを鋭角に削った際に出るカスを表すクラス
Coin	コインを表すクラス。ドラッグ可能でコインを移動した際にScratchMediatorへ通知する
Scratch	スクラッチを表すクラス。削ることができ、削った際に自動的にカスを生成する
ScratchContents	このスクラッチ全体を表すクラス。全体を拡縮したり移動したりして削っている部分へズームアップする機能やバックグラウンドの画像を読み込む機能を持つ
ScratchMediator	このコンテンツ全体の流れを制御するクラス。各アイテムに変更があった際にどうするかをコントロールする。主にコインがドラッグされた際にスクラッチへ削るよう命令を出す
ScratchShape	スクラッチの削る形を制御するクラス。削る形をシェイプとして保持し、コインを動かした際のスピード（強さ）から削るサイズを調整する

まずは、削るための大まかな流れを見ていきます。

次の図のような流れになります。

「削る」処理の流れ



コインを動かしたとき、onMouseMove() イベントハンドラが呼び出されます。このイベントハンドラはCoin.noticeMove() メソッドで上書きされていて、ScratchMediator.colleagueChanged() へコインが動いたことを通知しています。ScratchMediator はコインが動かされたときにどうするかという判断をコントロールします。ここではもちろんスクラッチを削る処理になります。

ScratchMediator は、Coin.getEdgeMC() メソッドからコインがスクラッチと接しているエリアとなる領域を示す MC を取得し、Scratch.shave へ必要な引数とともに呼び出します。これが、スクラッチを削るための大きな流れです。

このスクラッチを削る際のメインの処理が書いてあるのが「Scratch.shave() メソッド」です。この処理にフォーカスを当てて、スクリプトを見ていきます。

今回、スクラッチを削る表現にはマスクではなく Flash 8 からの新機能であるビットマップデータを加工する手法を取りました。はじめはマスクとシェイプ描画 API を使うことで制作しようと考えましたが、削り口が単調な形状となってしまい応用が利かないと感じて、この方法はやめました。

形状を自由にしたいと思うと、特定の形状のシェイプを持ったMC をマスクに複数attachMovie() していくことになります。しかし、こうなると処理速度に不安を感じます。そこで今回は、その両方を解決できそうなビットマップデータを加工する方法を取りました。

スクラッチを削る

Scratch.shave メソッドのメインとなるスクリプトを見ていきます。

SOURCE スクラッチを削る

```
public function shave(prevPoint:Point, vector:Vector, edge:MovieClip):Void {
     var size = vector.size();
     var currentPoint = new Point(prevPoint.x + vector.x, prevPoint.y + vector.y);


     // 削り口のサイズを変更
     this.shapeWrapped_mc.changeSize(size);

     var eraseBitmap = this.shapeWrapped_mc.getShapeBitmap();


     // 削るのに失敗して灰色になったとき
     var missMaxSize = 65;       // 弱い力での削り方として認知する移動距離
     var randomMissMaxSize = 150;  // これ以上の移動距離がある場合は灰色の削りにはしないしきい値
     var missRand = 15;          // missMaxSize 以上の移動距離でも灰色にする確率
     var missFlag =      this.isScratchPoint(prevPoint) and
                         (vector.size() < missMaxSize or (size < randomMissMaxSize and
                                                 MathUtil.randomInt(missRand) == 0));

     for (var i = 0; i <= this.trimSpan; i++) {
          var offset = new Point(prevPoint.x + (vector.x / this.trimSpan) * i,
                                 prevPoint.y + (vector.y / this.trimSpan) * i);
```

次のページに続く


```
          var rectangle = new Rectangle(offset.x, offset.y, eraseBitmap.width,
                                                                  eraseBitmap.height);

          // 削る場合
          if (!missFlag)
this.scratchBitmap.copyPixels(this.scratchBitmap, rectangle, offset, eraseBitmap, new
                                                                  Point(0, 0), false);
// 削るのに失敗してすこし色が濃くなるとき
          else
               this.scratchBitmap.copyPixels(this.darkScratchBitmap, rectangle, offset);
     }
```

Scratch.shave メソッドは、引数を 3 つ取ります。

スクリプトで使用する引数

引数	概要
prevPoint	Point クラスのインスタンスで削りはじめる始点を表す
vector	Vector クラスのインスタンスで始点の位置からどの方向にどのくらい削るかを表す
edge	MovieClip クラスのインスタンスで削っているときにカスとの接触を調べ、接触した場合にはカスを転がす

ここでの Vector、Point クラスは Flash.geom パッケージではなく、org.graffiti.web.dotfla.data.type パッケージを使用しています。

最初のほうで行っている部分がまさにスクラッチを削る処理です。いっけん、このメソッドは長いスクリプトに見えますが、多くは削るための領域を取得したり、カスなどの演出のための処理であって、実際にスクラッチのビットマップを削っているのは次の 1 行です。

```
this.scratchBitmap.copyPixels(this.scratchBitmap, rectangle, offset, eraseBitmap, new
                                                                  Point(0, 0), false);
```

BitmapData.copyPixels は、指定の BitmapData に別の BitmapData の任意の矩形領域をコピーする機能です。オプションで α チャンネルを持つBitmapData を指定し、その部分を α で抜く（削る）ことができます。ここでは、this.scratchBitmap（スクラッチとなっているビットマップ）に対して、第 1 引数にスクラッチ自身の BitmapData である this.scratchBitmap を指定し、そして第 2、第 3 引数にて置き換える部分の矩形領域を指定します。

ここまでは、置き換え元と置き換える素材が同じビットマップで同じ指定をしているので、何も変化が起きません。オプションである第 4 引数に α チャンネルの eraseBitmap を指定し、スクラッチの BitmapData の任意の矩形部分を α チャンネルに置き換えています。

この矩形領域を α チャンネルで置き換えるという作業を 1 フレームの間に始点から終点まで指定の回数（Scratch.TRIM_SPAN）繰り返すことによって、あたかも 1 フレーム間を直線で削っているように見せています。

● スクラッチにキズを付ける

スクラッチは、毎回削れるとは限りません。弱く削ったときや滑ってしまうときにはスクラッチにキズを付けるようにしています。キズ自体は、元のスクラッチ画像よりも明度の低いビットマップを用意して、スクラッチを削るときと同じようにスクラッチのビットマップの矩形領域を明度の低いビットマップで置き換えています。以下の部分です。

```
this.scratchBitmap.copyPixels(this.darkScratchBitmap, rectangle, offset);
```

この場合は、オプションのαチャンネルを使用せずに単純に明度の低いビットマップで置き換えているだけです。

○ 削りカスとコインの当たり判定

スクラッチを削っているときに、コインがカスと接触した際にカスが転がるように演出しています。その部分を見てみましょう。

SOURCE 削りカスとコインの当たり判定

```
        // 削りカスの当たり判定
        for (var i in this.arrBitsOfScratch) {
             if (this.arrBitsOfScratch[i].hitTest(edge))
                 this.arrBitsOfScratch[i].move(vector);
        }


        // 削りカスの角の当たり判定
        for (var i in this.arrBitsOfScratchEdge) {
             if (this.arrBitsOfScratchEdge[i].hitTest(edge)) {
                 this.arrBitsOfScratchEdge[i].removeMovieClip();
                 this.arrBitsOfScratchEdge.splice(i, 1);
             }
        }
```

第３引数で取られるMCが、コインとスクラッチの接触面を定義したMCになります。その接触面であるMCとスクラッチのカスを格納した配列Scratch.arrBitsOfScratchを、１つ１つMovieClip.hitTest()メソッドで衝突しているかを確認し、衝突していれば、そのカス（BitsOfScratch）のBitsOfScratch.move()メソッドにより指定のベクトル方向へ転がしています。

鋭角をなした際に生成された削りカスに関しては、転がすのではなく、MovieClip.removeMovieClip()によりステージ上から消しています。

○ 削りカスの生成

コインが動かされるたびに生成したのではステージはカスだらけになってしまうので、削りカスを生成する条件は一定の確率で生成しています。

SOURCE 削りカスの生成

```
        // 削りカスを生成
        var bitsRand = 5; // 削りカスを生成する頻度（大きいほど確率が下がる）

        if (isScratchPoint(currentPoint)) {
             if (MathUtil.randomInt(bitsRand) == 0)
                  this.createBitsOfScratch(prevPoint, vector);
        }


        // 鋭角をなした場合の処理
        var thresholdDeg = 150; // 折り返したとみなす角度
        var edgeBitsRand = 8; // 角を生成する確率

        if (this.prevVector != null and vector.getDegree(this.prevVector) >= thresholdDeg) {
             if (MathUtil.randomInt(edgeBitsRand) == 0)
                  this.createBitsOfScratchEdge(prevPoint, vector);
        }
```

また、コインを削っているときに一定の角度以上で折り返したときには、そのエッジの部分にも折り返したとき用のカスを生成しています。ここでは、角度がthresholdDeg以上の角度をなしたときに折り返しとみなし、また、その折り返しの中でもedgeBitsRandに１回だけ生成するようにしています。

こういったスクラッチを削る以外の演出を多く取り入れることで、コンテンツのおもしろさを深めています。