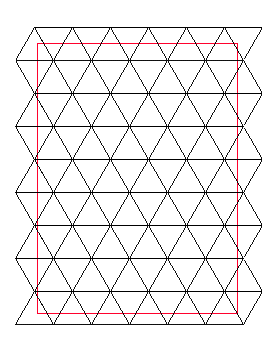
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 仕様書 | | 前作スペルカード背景 | | |
| 種類 | プログラミング資料 | | 作成者 | たかなし |
| 日付 | 2014/8/25 | |

**1　内容**

　前作スペルカード背景で使用した、ポリゴンを使ったスペルカード背景について述べる。

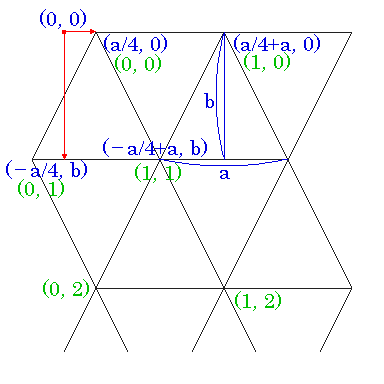


**2　原理**

2. 1　三角形の集合と位置・UV座標

　スペルカード背景を描画する際は、2Dポリゴンの三角形の集合を使ってそのテクスチャに画像を割り当て、UV座標を変化させることで変形する背景を描画している。

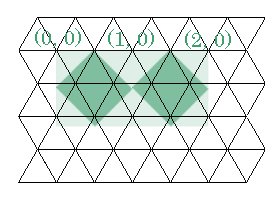
　まず、三角形の集合とは右図のようなもの。赤枠はゲーム画面。三角形の集合は画面を覆うように配置している。

　基本となる三角形の大きさをあらかじめ決めておけば、三角形の各点の座標は自動で決まる（下図）。ここで、三角形の大きさは正三角形でも良いが、√の計算が入ってきてややこしいので、底辺と高さの値をそれぞれ決めた二等辺三角形とした。また、座標の計算式が割とシンプルになるように、下図のような位置を原点とした。

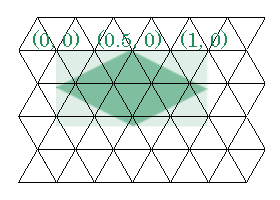
　前回は、aをゲーム画面の幅を20で割った値、bは高さを28で割った値にした。多分おおむね正三角形になるように紙面上で計算した結果だと思うが、正三角形に近い必要はあまりない。細かい模様の時に歪む可能性が出てくるくらい。

　この三角形の集合の各点は、最初に座標を決めて座標は固定し、その後適宜UV座標を割り当てる。このUV座標はエフェクトの種類や時間によって変化する。

　最も基本になるのが、三角形の座標とUV座標をそのまま対応させるものである。すなわち、テクスチャ画像のサイズを256として、画面にそのままテクスチャを敷き詰めたいときは、ある点の座標が(x, y)であれば、UV座標を(x/256, y/256)とすれば良い。

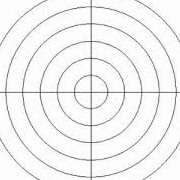
2. 2　簡単な変形

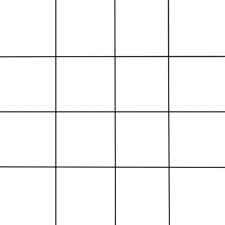
　次に、画像を拡大縮小することを考えてみると、例えば画像を横方向に2倍に引き延ばしたい時、UV座標の値をどうすれば良いかというと、1/2にすれば良い。そうすると、本来UV座標が2になる位置で初めてUV座標が1になるので、結果的に引き延ばされたようになる（右図）。

　このように、UV座標を経由して画像を変形させる場合、「逆の変形を加える（上の例なら、2倍にしたいけど、UV座標は1/2にする）」と正しく変形される感じになる。

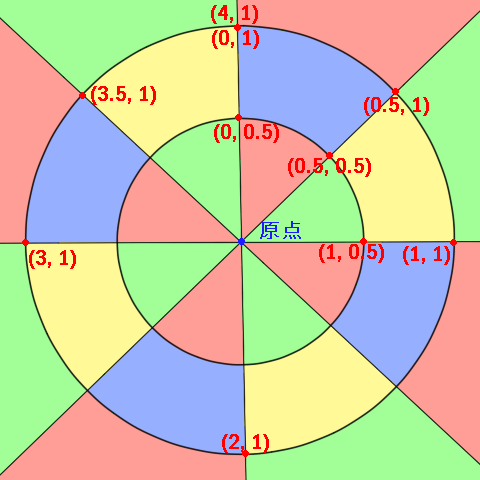
　同様に、例えば画像を時間により平行移動させたい場合、例えば画像を1フレームに2ピクセル、x軸正方向に動かしたければ、UV座標を1フレームに2ピクセル分、負の方向にずらしていけばよい。

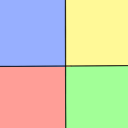
　画像を波のように変形させたい場合は、例えば横方向にずれるような波を作る場合、UV座標のU座標を、三角形の座標のy値に沿っていくらかずらしてやれば良い。パラメータを調整することで、時間変化する波にしたり、周期が長い・短い波を作ったり出来る。

2. 3　極座標変換

　よく使うけどもちょっと難しい変換に、極座標変換がある。これは、画像をある位置を中心にした円状に変形させる方法である。（右図）

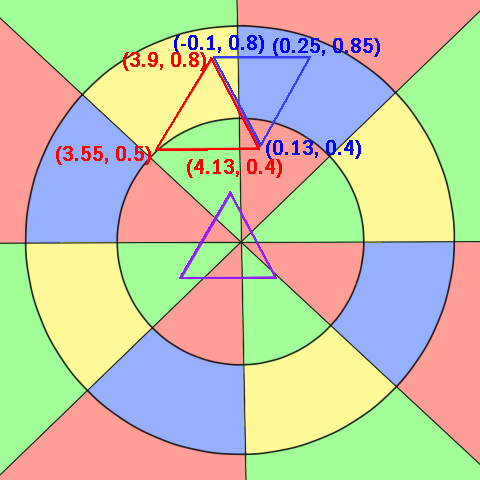
　これは、中心点からの距離と角度を元に、UV座標を計算して割り当てる方法である。





　同心円状に画像を割り当てることを考えてみると、図のようにUV座標を割り当てると円のような模様にすることが可能である。

　すなわち、中心点から、ある位置を見た時の角度をU座標に、中心からの距離をV座標にする。

　ここで注意したいのが、角度0近辺のU座標である。図の上の点を見ると、この点のUV座標は、(0, 1)でもあり、一方で円を回ってくると(4, 1)でもある。同じ位置でも、考えられるUV座標が複数あるのである。

　そのため、この背景システムでは、同じ位置にある点でも別のUV座標が取れるように全て三角形で分け、このような場合には周囲の点の座標に合わせてUV座標を決めるように補正している。

　すなわち、他の点とのU座標の差が半周分以上にならないように、適宜一周分の値を足したり引いたりしている。

　また、紫色の三角形のような、原点付近にまたがる三角形では、UV座標をうまく考えることが出来ず、たいてい変に引き延ばされた感じの三角形になる。これは、三角形の座標を固定している以上避けられないと思われる。

　そこで、この極座標変換を使う場合には、中心は別途黒く塗るなどして見えないようにしたり、中心を画面の外に置くようにしている。

**3.　前作のスペルカード背景に関するクラス・関数**

3.1　ImageMorphクラス

　あまり考えずに、グローバル変数として宣言している。

　ImageMorph\_Itemクラスの配列をメンバに持つ。

　ImageMorphクラスを経由してImageMorph\_Itemクラスにアクセスするようになっているが、このクラスに特別な機能があるわけではない。

3.2　ImageMorph\_Itemクラス

　このクラスが実際の画像変形を扱うクラスと考えて良い。

　CreateFace関数

　　最初に実行される関数。三角形の集合体の面を作成する。

　　ここでは位置を決めるだけで、UV座標にはノータッチ。

　AttachImage関数

　　画像を割り付ける時に実行する関数。

　　画像を、サイズを指定して割り付ける。

　　UV座標にはとりあえず変形なしの時の値を代入する。

　Grid関数

　　三角形の座標は一次元配列になっており、

　　ある格子状の点をその配列の要素番号に変換する。特に使う必要はない。

　GetGridX, GetGridY関数

　　ある格子状の点の平面上の位置を返す。割と使う。

　UpdateMorph関数

　　この関数が一番複雑な関数。

　　やることはつまり何かというと、三角形の座標に対して、

　　変形をほどこした上で、UV座標を割り当てる。

　　今までの説明では、画像を変形させたいときは、「UV座標を変えて変形させる」と

　　説明していたが、このスペルカード背景システムでは、

　　便宜上「三角形の座標を、実際の座標とは別の位置にあるように考える」

　　ことによって変形している。例えば、画像のサイズが256, 256で、

　　三角形のある点の座標（UV座標でなく、原点を基準とした座標）が

　　(256, 128)であるとする。この時、変形しなければUV座標は(1, 0.5)である。

　　ここで、画像を横方向に2倍に伸ばしたいとき、先程までの説明では

　　その点のUV座標を本来の半分、つまり(0.5, 0.5)にすれば良い、と書いていた。

　　だが、実際にはこのゲームでは、この三角形の点の座標の方をずらすことで変形する。

　　すなわち、実際には三角形の座標は(256, 128)だが、

　　変形によって三角形の座標が(128, 128)(横方向に半分)になったと考える。

　　その上でUV座標を考えると、画像サイズが256, 256に対して座標が(128, 128)なので、

　　UV座標は(0.5, 0.5)であり、結果的には同じ値となる。

　　なぜこんな風に考えるかというと、「変形を複数組み合わせる」ケースが

　　あるからである。

　　変形と同時に座標をUV座標に変換してしまうと、

　　そこからさらに座標を変換しようとしたときに、

　　UV座標になっているためうまくいかない。そのため、

　　変形はあくまで座標のままで行った上で、最後にUV座標に変換する

　　という方法で変形を行う。

　　この、座標から座標への変換を行う部分が

　　rslt = morphFunc(&val);

　　である。val、rsltはそれぞれMORPHPARAM構造体の変数で、

　　（三角形の）座標、時間t、あと変形に用いるパラメータ(MORPHOPTION)を含む。

　　座標からUV座標への変換を行っているのが、その後の

　　 tmpU = rslt.a/uvWid;

pVertex[(j\*xDiv+i)\*6+n].u = lastU = tmpU;

pVertex[(j\*xDiv+i)\*6+n].v = rslt.b/uvHei;

　　である。

3.3　MorphFuncに書かれている関数群

　さきほどmorphFuncが座標から座標への変換を行うと書いたが、この関数として使われるのは、MorphFunc.hとMorphFunc.cppに書かれた関数である。実際にどう書くかは、MorphFunc.cppの内容を参照。result.aとresult.b（変形後のx座標、y座標）に、tの値や変形のパラメータoptの値を使ってしかるべき座標を代入している。