報告書

ＩＶＲＣ２００１　Ｂ部門
インタラクティブアート作品「ＣＯＳＭＯＳ３」

Interactive Art “COSMOS3”

九州芸術工科大学　画像設計学科

松永　康佑

matsunaga　kosuke

企画名	：	**ＣＯＳＭＯＳ３**
大学名	：	**九州芸術工科大学**
チーム名	：	**なし**
チーム構成員	：	**松永康佑**
使用機材	：	VAIO NOTE PCG-N505A/BP

１　制作方法

・１－１概要

はじめにイメージのモティーフとなるものを決める。原点となるモティーフは万華鏡の無限に続く色とりどりの世界である。無限に広がっていく空間を表現する際に有効と思われる表現が３次元処理を施した世界がよいと思われたので、グラフィックスライブラリである、ＯｐｅｎＧＬを使用することにした。開発の容易さと、その後のプログラムの実行可能な環境などを総合的に考え、WINDOWS　OS での使用を想定し、その中で最大限の性能を引き出せるように努力した。そのため、最後に多くの WINDOWS パソコンにおいてチェックを行った。

なお、一般的にターゲットのコンピュータを限定すればするほど、そのコンピュータの本来の性能を引き出すことが可能であるが、性能と引き換えにプログラムの実行環境が限定されたものとなってしまう。逆に多種の OS での実行を求めるらならば、コンピュータの性能を引き出すことは難しく表現の幅は狭まってしまう傾向にある。.

・１－２制作環境

開発に際して使用したハードウェア及びプラットフォームを以下に列記する。

SONY　VAIO　PCG-N505A/BP

Microsoft　Windows98SE

Microsoft　Visual　C++　6.0　Professional

OpenGL　Version　1.1

また、開発に際し動作チェックを行ったハードウェア及びプラットフォームを以下に列記する。

COMPAQ　PROFESSIONAL　WORKSTATION　SP750

WINDOWS　NT　4.0

自作パソコン 3 台

WINDOWS　98

WINDOWS　Me

WINDOWS　2000

・１－３アルゴリズム

まず基本形状の説明からします。基本形状は

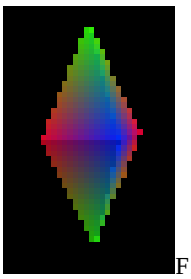


Fig.1

このような形をしている。四角錐を上下につなげた形になっており 8 面体の構造をしている。各頂点には OpenGL による頂点カラーを割り振ってあり、そのカラーは反対側に位置するカラーと同じ色になっていて、シンメトリな構造になっている。各頂点カラーは R、G、B 独立で各々のレベルが SIN 関数によってコントロールされており、パラメータも R、G、B 各々が所有している。

パラメータの変化のさせかたによって、ランダムで周期的な変化の中で滑らかなグラデーション表現が可能となる。

基本形状の動きの説明。

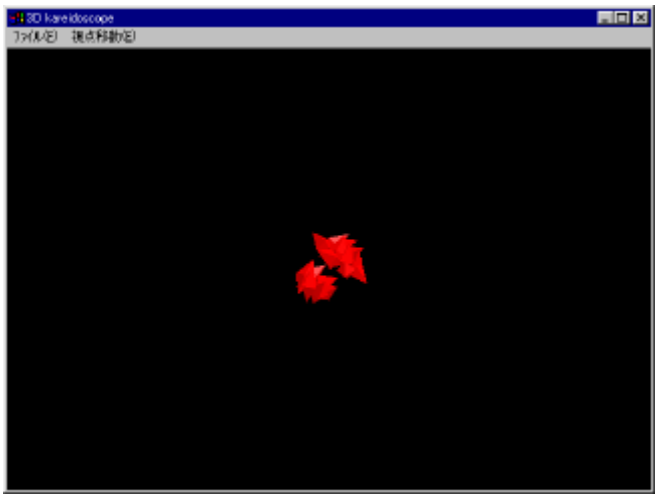


Fig.2

この状態が初期状態となる。真っ赤に塗られた基本形状が６個×３段、計 18 個が並んでいる。これらは中心を原点として振幅運動をしている。生命の心臓の鼓動をイメージしてある。

18 個の基本形状の位置はマウスの座標に対応しており、

マウスを動かすと連動して赤い物体の配置も変わる。マウスが中心にあるときに赤い物体は寄り集まって、マウスが画面の端にあればあるほど赤い物体はお互いに離れるような動きをする。

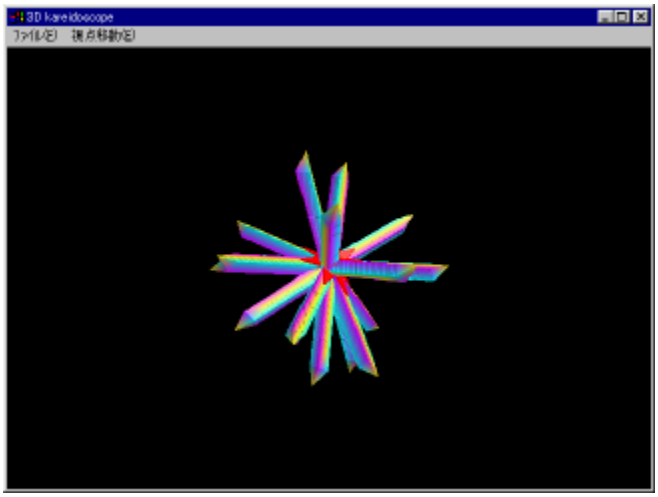


Fig.3
マウスの左ボタンを押しっぱなしにした状態の画像である。中心から頂点カラーで塗られた基本形状が連続的に放出する。その際、飛び出す方向ベクトルは赤い物体の方向に飛び出るようになっている。

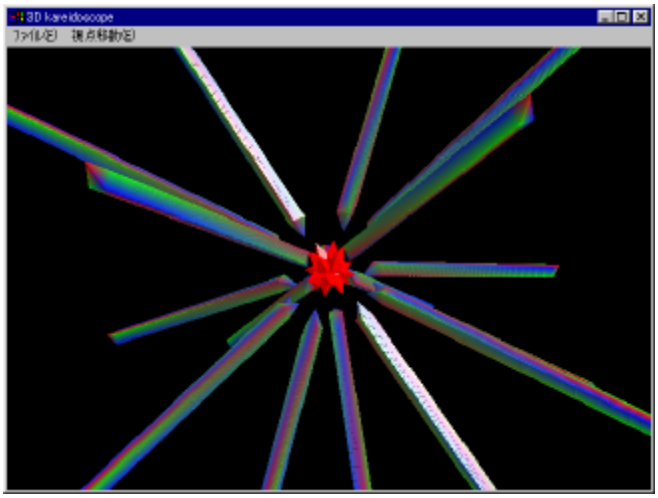


Fig.4
しばらく左ボタンを押しっぱなしにしていると、直線的に放出されマウスボタンを離すと放出の動作は終了する。

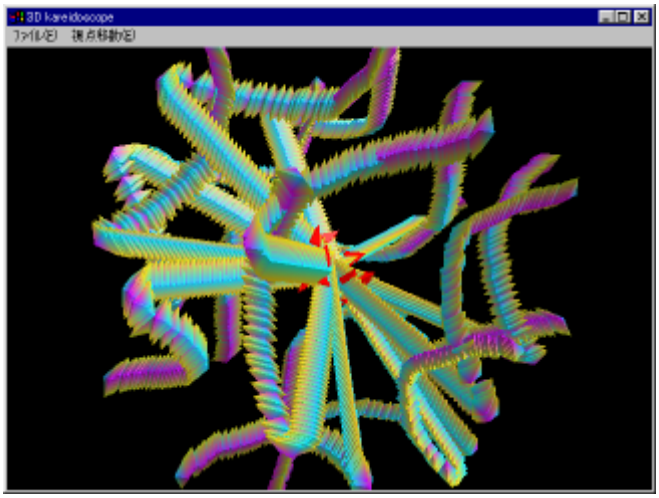


Fig.5
　放出しつづけている状態でマウスを移動させると、マウスの動きは赤い物体に関連付けられていて、とびだしてくる物体の飛ぶ方向は赤い物体によって関連付けられているので、マウスの連続的な移動操作は最終的にこのような図形的変化となって表現される。

・１－４ソースの抜粋

以下に、プログラムの中心的となる部分のソースの抜粋と説明を記す。

OPENGL **の命令セットによる部分。**

glPushMatrix();**と** glPushMatrix();**によって、はさまれている部分である。**

基本的には、６角形の頂点に配置した基本形状を３段に重ねて表示し、

合計１８の基本形状を一度に放射状に放出している。

```
        DrawALL();
glRotatef( 60 , 0.0 , 0.0 , 1.0 );
        DrawALL();
glRotatef( 60 , 0.0 , 0.0 , 1.0 );
        DrawALL();
glRotatef( 60 , 0.0 , 0.0 , 1.0 );
        DrawALL();
glRotatef( 60 , 0.0 , 0.0 , 1.0 );
        DrawALL();
glRotatef( 60 , 0.0 , 0.0 , 1.0 );
        DrawALL();
glRotatef( 60 , 0.0 , 0.0 , 1.0 );
```

ここまでの処理において DrawALL **メソッドは基本形状の１つを表示する。**

基本形状とは本作品において、頂点カラーを持った８面体や４面体などのオブジェクトを指す。

DrawALL **と** glRotate **は対となっていて６０度の角度ごとに基本形状を** GL **スタックに入れている。**

この段階で中央に６つの基本形状を平面的に六角形に配置したことになる。これが１段目である。

```
glRotatef( 30 , 0.0 , 0.0 , 1.0 );
glRotatef( 60 , 1.0 , 0.0 , 0.0 );
        DrawALL();
glRotatef(-60 , 1.0 , 0.0 , 0.0 );
glRotatef( 60 , 0.0 , 0.0 , 1.0 );
glRotatef( 60 , 1.0 , 0.0 , 0.0 );
        DrawALL();
glRotatef(-60 , 1.0 , 0.0 , 0.0 );
glRotatef( 60 , 0.0 , 0.0 , 1.0 );
glRotatef( 60 , 1.0 , 0.0 , 0.0 );
        DrawALL();
glRotatef(-60 , 1.0 , 0.0 , 0.0 );
glRotatef( 60 , 0.0 , 0.0 , 1.0 );
```

```
glRotatef( 60 , 1.0 , 0.0 , 0.0 );
        DrawALL();
glRotatef(-60 , 1.0 , 0.0 , 0.0 );
glRotatef( 60 , 0.0 , 0.0 , 1.0 );
glRotatef( 60 , 1.0 , 0.0 , 0.0 );
        DrawALL();
glRotatef(-60 , 1.0 , 0.0 , 0.0 );
glRotatef( 60 , 0.0 , 0.0 , 1.0 );
glRotatef( 60 , 1.0 , 0.0 , 0.0 );
        DrawALL();
glRotatef(-60 , 1.0 , 0.0 , 0.0 );
glRotatef( 30 , 0.0 , 0.0 , 1.0 );
```

ここまでの処理において glRotatef、DrawALL、glRotatef **の組み合わせが１セットとなっていて、その１セットを別の** glRotate **によって６０度ずつ回転させて** GL **スタックに入れている。また始めと終わりに３０度回転させて、始めの６つの基本形状と段違いになるようにしている。**

これが２段目となる。

```
glRotatef( 180 , 0.0 , 1.0 , 0.0 );
```

３段目に入る前に、ここで上下を反転しておく。

```
glRotatef( 30 , 0.0 , 0.0 , 1.0 );
glRotatef( 60 , 1.0 , 0.0 , 0.0 );
        DrawALL();
glRotatef(-60 , 1.0 , 0.0 , 0.0 );
glRotatef( 60 , 0.0 , 0.0 , 1.0 );
glRotatef( 60 , 1.0 , 0.0 , 0.0 );
        DrawALL();
glRotatef(-60 , 1.0 , 0.0 , 0.0 );
glRotatef( 60 , 0.0 , 0.0 , 1.0 );
glRotatef( 60 , 1.0 , 0.0 , 0.0 );
        DrawALL();
glRotatef(-60 , 1.0 , 0.0 , 0.0 );
glRotatef( 60 , 0.0 , 0.0 , 1.0 );
glRotatef( 60 , 1.0 , 0.0 , 0.0 );
        DrawALL();
```

```
glRotatef(-60 , 1.0 , 0.0 , 0.0 );
glRotatef( 60 , 0.0 , 0.0 , 1.0 );
glRotatef( 60 , 1.0 , 0.0 , 0.0 );
        DrawALL();
glRotatef(-60 , 1.0 , 0.0 , 0.0 );
glRotatef( 60 , 0.0 , 0.0 , 1.0 );
glRotatef( 60 , 1.0 , 0.0 , 0.0 );
        DrawALL();
glRotatef(-60 , 1.0 , 0.0 , 0.0 );
glRotatef( 30 , 0.0 , 0.0 , 1.0 );
```

２段目と同様の手順により３段目を GL スタックに入れる。

これで６×３＝１８個の基本形状を表示する準備が整う。

次に、１８個ひとかたまりのオブジェクトはマウスボタンを押している間中放出されつづけるので、随時新しいオブジェクトを生成し、追加表示していかなくてはならない。

以下が放出速度を決定する式である。

```
x_locate[i] = x_locate[i] + add_x_locate[i];
y_locate[i] = y_locate[i] + add_y_locate[i];
z_locate[i] = z_locate[i] + add_z_locate[i];
x_length = x_locate[i];
y_length = y_locate[i];
z_length = z_locate[i];
```

これは毎フレームごとに更新されるので add_?_locate[]の距離分だけ中心から離れていく。

Add_?_locate[]は次式により定義される。

```
add_x_locate[obj_num_last] =    (MouseXp - OldRect.right/2) / 300.0;
add_y_locate[obj_num_last] = 0.4;
add_z_locate[obj_num_last] = -(MouseYp - OldRect.bottom/2) / 300.0;
```

ここで obj_num_last は、最後の放出のみ放出ベクトルを決定するための目印になっている。

MouseXp - OldRect.right/2 はモニタ中心からのマウスの X 方向の距離である。

GL での Y 座標の課産地は 0.4 に固定してある。

何度かの加算の結果、ある一定の距離まで達したものは表示しないものとする。

これは、シーン全体のポリゴン数にある程度の限界を設定するためである。

```
if (y_locate[i] > 1200.0){
        draw_or_not[i] = FALSE;
}
```

上記の変数?_length は DrawALL メソッドの中でポリゴンの頂点座標を指定する際に、次のように使用されている。

```
glVertex3f( px[index][cnt]+x_length, py[index][cnt]+y_length, pz[index][cnt]+z_length);
```

以上がオブジェクトの位置を指定するソースである。
次に色情報を指定する。

```
color1 = cos(((y_locate[i]+th)*0.11) * 3.1415926535/180.0) / 2.0 + 0.5;
color2 = sin(((y_locate[i]+th)*0.04) * 3.1415926535/180.0) / 2.0 + 0.5;
color3 = 1.0-color1-color2;//cos(((y_locate[i]+th)*1.85) * 3.1415926535/180.0) / 2.0 + 0.5;
```

ここで color1,2,3 は RGB 情報として使われるが、R、G、B に対応させる際に、１２３、１３２、２３１、３２１、３１２、２１３のように対応させる順番をコントロールしつつ使用する。これらの 3 つの色情報を使い、基本形状である、8 面体もしくは 4 面体を表示する。その際に相対する頂点は同じ色になるようにする。
Color1,2 ともに GL 空間における Y 座標の位置をパラメータとして SIN,COS 関数として定義されている。SIN,COS 関数は－１～１の値をとるので、計算結果は０～１の値をとるようになっている。これは GL のパラメータにあわせている。 Color3 も同様に関数によって定義すると RBG 値が０，０，０になってしまう恐れがあるため、適当な式を用いて補正する。

最後に GL の座標系全体を回転させて、全体に動きをつけている。
次の命令によってコントロールされる。

```
glKM_RotateX(    0.25 );
glKM_RotateY(    0.50 );
glKM_RotateZ(    0.75 );
```

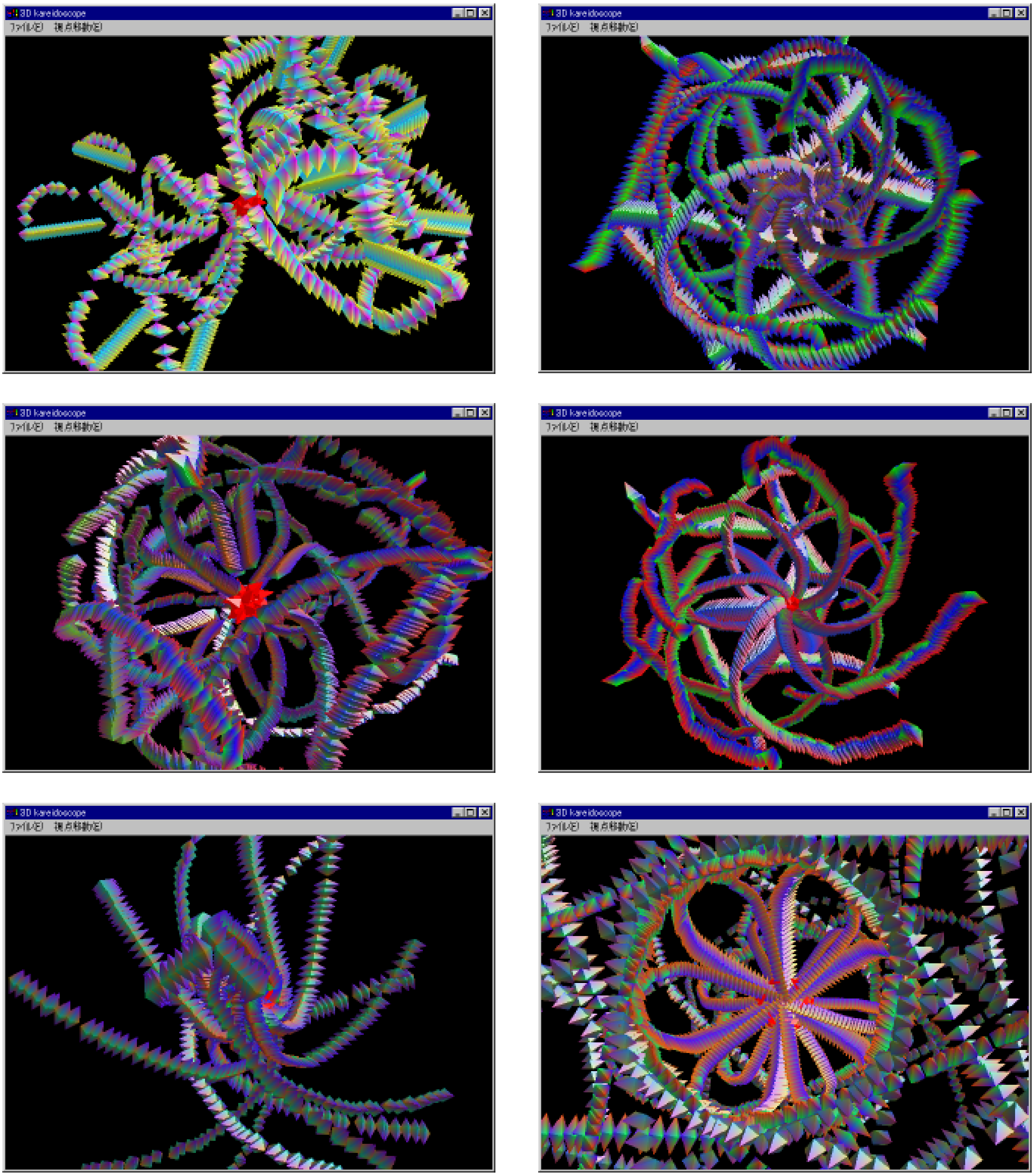
この計算は作品の、うねるような動きの生成にはまったく関係ない。
マウスを動かしたときのうねるような動きの元となるのは前ページの

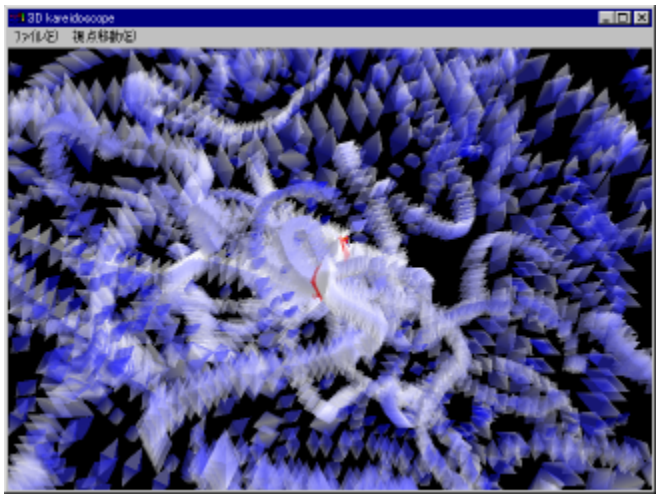
```
add_x_locate[obj_num_last] =    (MouseXp - OldRect.right/2) / 300.0;
add_y_locate[obj_num_last] = 0.4;
add_z_locate[obj_num_last] = -(MouseYp - OldRect.bottom/2) / 300.0;
```

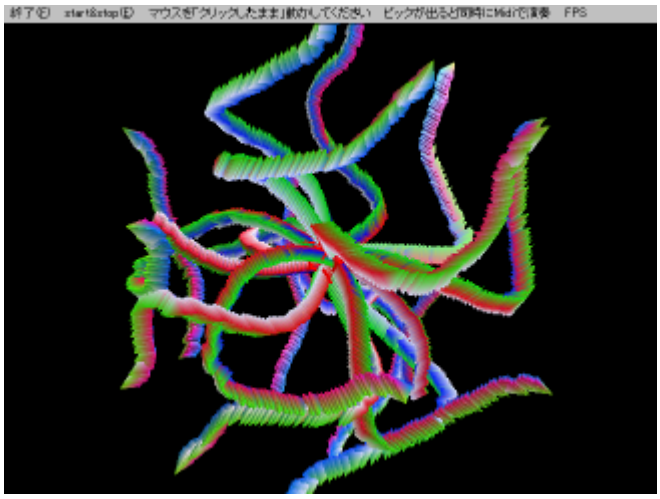
の部分のマウスの操作による初速度のベクトルによるもので、各々の基本形状が初速度を持つためにこのような現象が生じる。例えて言うなら、水が出ているホースを揺らすと水がうねるように動くのと同じで、水粒子は一定の方向に重力に従い放物運動をしているだけだが全体としては、違った印象を与える動きに見えるのである。

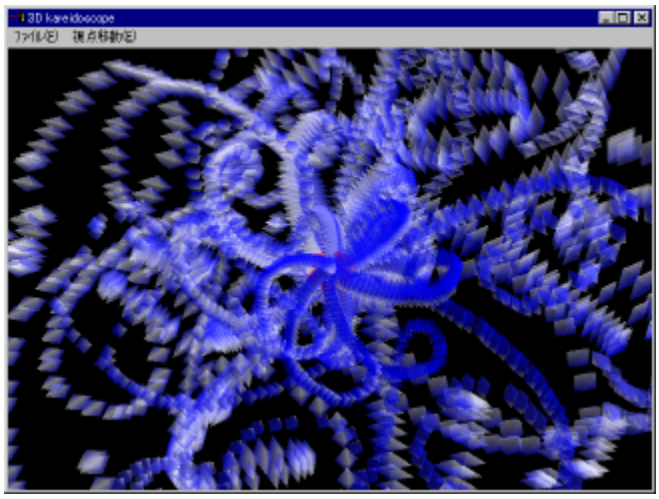
2　結果

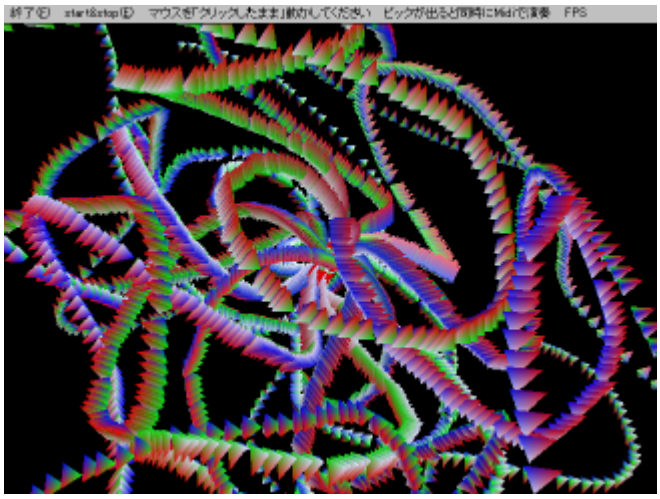
作品「COSMOS３」の操作中のスクリーンショット一覧

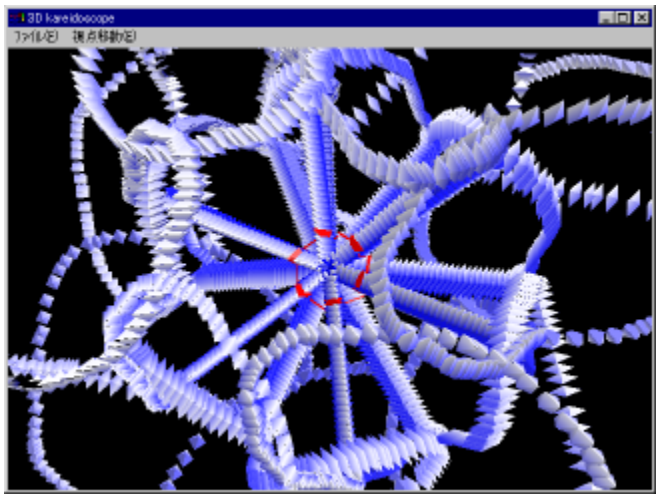


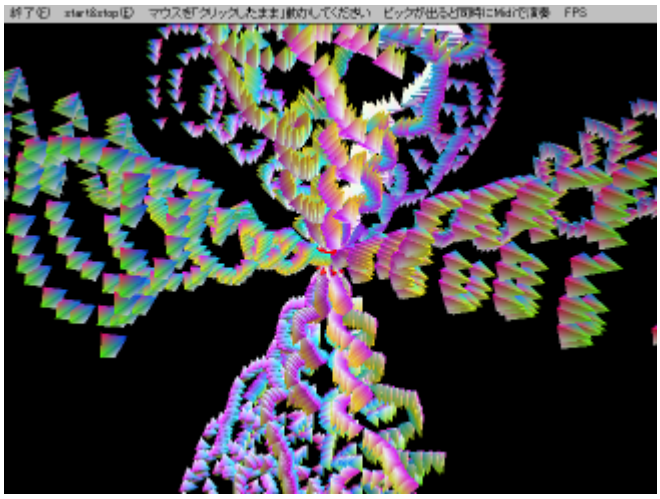

3D kaleidoscope
ファイル(F)　視点移動(E)


終了(E)　start&stop(E)　マウスを「クリックしたまま」動かしてください　ビックが出ると同時にMidiで演奏　FPS


3D kaleidoscope
ファイル(F)　視点移動(E)


終了(E)　start&stop(E)　マウスを「クリックしたまま」動かしてください　ビックが出ると同時にMidiで演奏　FPS


3D kaleidoscope
ファイル(F)　視点移動(E)


終了(E)　start&stop(E)　マウスを「クリックしたまま」動かしてください　ビックが出ると同時にMidiで演奏　FPS

３　考察及び将来性

コンピュータに自動的に計算させる CG にマウスによるインタラクティブ性を取り入れて対話型の映像表現を試みた結果、既存ソフトでは表現できなかった表現を可能とすることが出来た。参加者の反応では、音声を伴うインタラクティブな作品としたことで映像との一体感が生まれ、写実的な CG 映像とはまた違った仮想空間への没入感を生じさせた。色彩や形態の変化する様は、人に心地よさ、緊張感、目まいに似た未知の感情を与え、安らぎやトリップ感覚を覚えた等、さまざまな印象を参加者に与えた。

今回の制作において開発環境は速度的には満足できないコンピュータで行ったが、最終的に高速なコンピュータでの再生を行うことが出来たため満足な結果を得るにいたった。私の求めていたインタラクティブ作品では、画像の生成をリアルタイムで行う必要があるため、必然的にコンピュータの能力に左右されてしまう作品となる。よって、現在のコンピュータの平均的能力では少々能力不足である。しかし、日進月歩のこの世界では、1 年も経てば標準的なコンピュータで再生可能なレベルに達する。また、開発時に速度の遅いコンピュータで制作したため、プログラムの高速化に対する必要性があり、その分最終的な結果に反映させることができた。

マウスという標準デバイスによる表現として、満足の行く作品に仕上がったが、インタラクティブな作品を今後追求していくためにはマウスというデバイスから離れなければならない。マウスはコンピュータを扱う人間にとっては身近な存在ではあるが、一般人からしてみれば特別な装置に他ならない。もっと人間の直感的に操ることの出来る入力インターフェイスを作りだし、人間からの入力と映像などの出力とのインタラクティブ性を追求する必要がある。

また、この作品ではマウスの位置による簡単な音階を表現することが出来る。つまり、マウス操作に伴い、ピアノの鍵盤をかき鳴らすような音が出る。しかし、あまり美しい音声ではないので、操作に対する音声面でのインタラクティブ性も今後追求する必要がある。

４　参考文献

秀和システム
OpenGL リアルタイム３D プログラミング
新藤義昭 阿部正平　著