ピラミンクスの群論的考察と OpenGL による実装

熊丸　温子　　(指導教員：金子　晃)

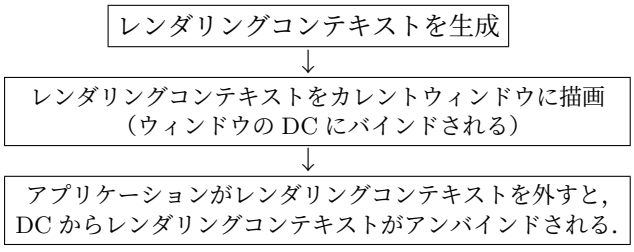
1　はじめに

WindowsNT3.5 より標準ライブラリに含まれている OpenGL は，ほんの少しの労力で，洗練された３次元アプリケーションを描くことを可能にした．OpenGL は広範囲に渡るライブラリで，３次元のシーンを作成したり，そのシーンをいろいろな角度から見るだけでなく，照明やテクスチャマッピング・混合アンチエイリアシング・その他の特殊効果を適用して，３次元アプリケーションに現実味を与える．ここでは，OpenGL を使って３次元オブジェクトを描画する方法と，それらのオブジェクトを３次元ワールド内で動かすための移動・スケーリング・回転を適用する方法について復習した後，それをピラミンクスというゲームの表現に適用してみる．

2　OpenGL とは

OpenGL グラフィックス・システムは，ウインドウ・システムに存在しないグラフィックス・ハードウェア用高機能ソフトウェア・インタフェースである．OpenGL は約 350 のグラフィックスルーチンを使用し，高品質のカラー・イメージを効率的に生成できる．また３ D グラフィックスのインタラクティブレンダリングに最適であり，２ D グラフィックスの画像処理に利用することもできる．そしてグラフィックスハードウェアとアプリケーションプログラムとの間に抽象層を設け，アプリケーションプログラミングインタフェース (API) を形成する OpenGL のルーチンを使って，グラフィックス・プリミティブ（点・線・多角形・ビットマップイメージ）をフレームバッファレンダリングすることができる．プリミティブを利用し，レンダリング制御を操作すれば，３ D オブジェクトの高品質カラー・イメージ生成とアニメーションが容易にできる．

3　OpenGL プログラム



4　glut によるプログラミング手順

glut を用いた OpenGL プログラムは，まず，init ルーチンでクリアカラーや描画色，光源の設定等をし，画像を描く場所が「空の状態」になるようにカラーバッファをクリアする．次いで基本的な display 関数を準備して基本的な描画を記述する．オブジェクトの描画は，glBegin() と glEnd() 関数の対の間に画像の頂点や法線方向，面の素材などを定義することで行う．glBegin() の引数は図形の種類を指定する．その他，ウインドウ変形処理やキー・マウスの割り込み処理，アイドル状態での処理を記述し，最後に main 関数で描画用ウインドウを準備し，init をコール後，上記のサブルーチンを glut の対応する関数の引数に入れて設定した後，イベント待ち状態に入る．

5　ピラミンクスとは

ピラミンクス (Pyraminx) は，ルービックキューブのテトラ (正４面体) 版で，ドイツの数学者 Uwe Meffert により発明された．下図 1 のように面に番号を振ると，ピラミンクスの動きは S_{36} の部分群である置換群とみなせる．実際に動きを細かく見ると，

(1) 頂点の４個のテトラ
(2) 辺の中心にある６個のテトラ
(3) １面しか見えない１２個の正３角形

に分類されるが，最後の正３角形は，実は３個ずつペアになって正８面体の一部を成して一緒に移動することが分かる．

この変換群 G は，以下のような，いずれも位数が３の基本置換より生成される．正成元がすべて偶置換なので A_{36} の部分群となる．ここで，$A1$, $A2$, $A3$ は，それぞれ頂点 A から見て，１段目，２段目，３段目を図 1 の方向に 120° 回転させる操作に対応する置換を表す．他の頂点についても同様である．各々は，位数３の独立な巡回置換のいくつかの積から成っている．

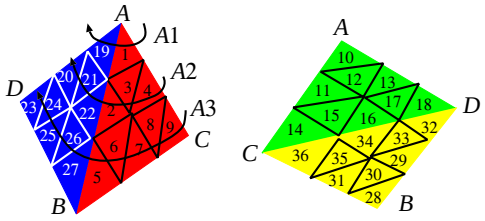


図 1

$A1 = (10, 1, 19)$
$A2 = (11, 2, 20)(12, 3, 21)(13, 4, 22)$
$A3 = (14, 5, 23)(15, 6, 24)(16, 7, 25)(17, 8, 26)$
$\quad (18, 9, 27)(32, 36, 28)(33, 35, 30)(34, 31, 29)$
$B1 = (27, 5, 28)$
$B2 = (25, 2, 31)(26, 6, 30)(22, 7, 29)$
$B3 = (23, 1, 36)(24, 3, 35)(20, 4, 34)(21, 8, 33)$
$\quad (19, 9, 32)(10, 14, 18)(13, 11, 16)(12, 15, 17)$
$C1 = (9, 14, 36)$
$C2 = (7, 11, 34)(8, 15, 35)(4, 16, 31)$
$C3 = (5, 10, 32)(6, 12, 33)(2, 13, 29)(3, 17, 30)$
$\quad (1, 18, 28)(19, 23, 27)(21, 24, 26)(20, 25, 22)$
$D1 = (23, 32, 18)$
$D2 = (20, 29, 16)(24, 33, 17)(25, 34, 13)$
$D3 = (19, 28, 14)(21, 30, 15)(22, 31, 11)(26, 35, 12)$
$\quad (27, 36, 10)(1, 5, 9)(3, 6, 8)(2, 7, 4)$

これを GAP に代入し計算してみた．GAP は Scotland の University of St. Andrews の School of Math. and Comp. Sci. で開発・配布されている，有限群論を中心とする代数計算のフリーソフトである．GAP に上記のデータを代入すると，ピラミンクスの置換群が生成元 12 個で位数 9 億 699 万 2640 を持つことが分かる．この計算を GAP はほぼ一瞬でやってくれる．

ピラミンクスの解法を発見するには，任意に与えられたこの置換群の元を，$A1 \sim D3$ の積に分解しなければならない． この分解は，これらの基本的な置換が，置換群の生成元としてはあまり基本的でないため，人間が目の子で探すのはそう簡単ではない．

もう少し GAP での様子を調べてみると，1 型と 2 型の回転は正規部分群 H_1, H_2 を作るが，3 型の回転 H_3 は正規部分群ではないことが分かる．また，最初の二つはピラミンクスを動かしてみれば当然予想されるように，この置換群全体の中で直積 $S = H_1 \times H_2$ を成している．この部分群による剰余類群は位数がたった 12 で，しかもその構造は A_4 に等しいことが分かる．これは，テトラの 4 頂点の (向きを変えない) 置換の群と一致する．しかも，明らかに H_3 の元により各面の三角形は同じ面に留まる．

以上の考察で，一つの解法の方針として，まず $A3, B3, C3, D3$ を適当に用いて，頂点のテトラを正しい位置に直し，次に $A1, B1, C1, D1$ を適当に用いて，頂点のテトラの向きを合わせると，残るは H_2 の元による置換だけとなる．よってこのゲームの核心は部分群 H_2 の解析である．

ちなみに，群構造の問い合わせを直接部分群に行ってみると，

```
gap> StructureDescription(h1);
"C3 x C3 x C3 x C3"
gap> StructureDescription(h2);
"C3 x C3 x C3 x C3 x
     ((C2 x C2 x C2 x C2 x C2) : A6)"
```

ここで，x は直積を，: は半直積を表している．最後の結果は AMD Athlon64 2GHz，主記憶 1GB の機械で 2 分くらいの計算時間がかかったので，$H3$ や G は 1 日では終わらないだろうと想像される．実際にやってみると，コンピュータは 30 分ほどでスタックオーバーフローとなり，途中で計算を放棄した．しかし，今までの考察から

$$G = (H_1 \times H_2) \ltimes A_4 = \{C_3^8 \times (C_2^5 \ltimes A_6)\} \ltimes A_4$$

という構造をしていることが推論できる．GAP には，離散数学で普通に学ぶ左剰余類を与える関数が存在しないので，右剰余類を尋ねると 1 2 個の右剰余類の代表元 $x1 \sim xb$ が得られる．これらは見にくいので具体形を略すが，代表元を取り直すと

```
gap> y1:=CanonicalRightCosetElement(s,x1);
(5,9,18)(6,8,17)(14,23,28)(15,24,30)(26,35,33)
(27,36,32)
gap> y2:=CanonicalRightCosetElement(s,x2);
(5,18,9)(6,17,8)(14,28,23)(15,30,24)(26,33,35)
(27,32,36)
gap> y3:=CanonicalRightCosetElement(s,x3);
(1,5,18)(3,6,17)(10,27,32)(12,26,33)(19,28,23)
(21,30,24)
gap> y4:=CanonicalRightCosetElement(s,x4);
(1,5,9)(3,6,8)(10,27,36)(12,26,35)(14,19,28)
(15,21,30)
gap> y5:=CanonicalRightCosetElement(s,x5);
(1,5)(3,6)(8,17)(9,18)(10,27)(12,26)(14,23)
(15,24)(19,28)(21,30)(32,36)(33,35)
gap> y6:=CanonicalRightCosetElement(s,x6);
(1,9,5)(3,8,6)(10,36,27)(12,35,26)(14,28,19)
(15,30,21)
gap> y7:=CanonicalRightCosetElement(s,x7);
(1,9,18)(3,8,17)(10,36,32)(12,35,33)(14,23,19)
(15,24,21)
gap> y8:=CanonicalRightCosetElement(s,x8);
(1,9)(3,8)(5,18)(6,17)(10,36)(12,35)(14,19)
(15,21)(23,28)(24,30)(26,33)(27,32)
gap> y9:=CanonicalRightCosetElement(s,x9);
(1,18,9)(3,17,8)(10,32,36)(12,33,35)(14,19,23)
(15,21,24)
gap> ya:=CanonicalRightCosetElement(s,xa);
(1,18,5)(3,17,6)(10,32,27)(12,33,26)(19,23,28)
(21,24,30)
gap> yb:=CanonicalRightCosetElement(s,xb);
(1,18)(3,17)(5,9)(6,8)(10,32)(12,33)(14,28)
(15,30)(19,23)(21,24)(26,35)(27,36)
```

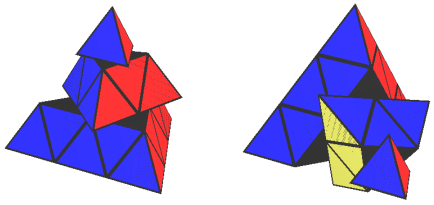
となって，かなり見通しがよい．取り替えられたものの中から，例えば，y1, y2, y3 が G/S を生成することが容易に確かめられる．

最後に H_2 の元の面白い例を示す．

```
M1=A2*B2*A2^-1*B2^-1=(2,25,11)(4,22,29)
M2=A2*C2*A2^-1*C2^-1=(4,31,13)(7,20,11)
M1*M2=(2,25,7,20,11)(4,22,29,31,13) (位数 5)
M3=B2*C2*B2^-1*C2^-1=(2,31,16)(7,34,22)
M1*M2*M3=(2,25,34,22,29,16)(4,7,20,11,31,13)
M1^3=(2,22)(4,11)(7,31)(13,20)(16,34)(25,29)
```

6 まとめと今後の課題

ピラミンクスを群論的に考察し，OpenGL で実装してみた．今後はより単純な置換を発見し，テトラの分かりやすい解法に結びつけたい．



A2, B2 の実行途中の画面スナップ

参考文献

[1] Win32OpenGL プログラミング, クレイトン・ウォルナム（松田晃一訳），ピアソンエデュケーション，2000.

[2] OpenGL プログラミングガイド (第 2 版)，アジソン・ウェスレイ，星雲社，1997.

[3] GAP Reference Manual, The GAP Group, http://www.gap-system.org