情報工学実験III
unix実験
2D/3D Grahpics, GUI

チーム名：Watering KissMint
実験メンバー
(リーダー)045753B：宮城　　健
045702H：池原　　匠
045721D：金城　巨樹
045737A：渡口　誠太
045742G：中西　洋貴

1　実験目的

UNIX 上で動作する GUI(Graphical User Interface) の代表的なシステムの プログラムを行う。

- 一番低レベルなビットマップ (Frame Buffer) レベル
- X Window System レベル
- ツールキットレベル
- スクリプト言語レベル
- OpenGL ライブラリ

これらを用いて、コンピュータのインターフェースの設計に関する理解を深め、経験を積む事が出来る。また、GUI はイベント駆動型のプログラムとオブジェクト指向計算が必須であり、これらの理解を深める事ができる。

2　基本事項

2.1　X-Window System

UNIX 系 OS で利用されるグラフィカルユーザインターフェース (GUI) 環 境。元来 UNIX は文字ベースの操作環境しか利用できなかったが、マサチュー セッツ工科大学 (MIT) の Athena Widget Project が中心となって X Window System が開発された。現在ほとんどの UNIX 系システムで標準的に採 用されている。X Window Sytem には、クライアント/サーバモデル、ネッ トワーク透過性などの特徴がある。

2.2　GUI(Graphical User Interface)

ユーザに対する情報の表示にグラフィックを多用し、大半の基礎的な操作をマウス などのポインティングデバイスによって行うことができるユーザインターフェース のこと。最近では GUI を利用するための基本的なプログラムを OS が提供するこ とにより、アプリケーションソフトの操作感の統一や、開発負担の軽減などが図れ ている。GUI を実装した OS には、Microsoft 社の Windows シリーズや Apple 社の MacOS などがある。UNIX 系の OS は、OS 自体は GUI 機能をもっていないことが 多いが、X Window System というソフトウェアを組み込んで GUI 機能を追加する 場合がある。

2.3 Open/GL

OpenGL とは、2D および 3D グラフィックス処理のに関するプログラミングインターフェースのことである。OpenGL は Silicon Graphics 社が中心となって開発された。

OpenGL は、三次元図形にイメージを貼り付けるテクスチャマッピング機能や Z バッファ法、グローシェーディングなどの機能を持っており、映画の特殊効果を出す際などに幅広く利用されている。OpenGL の基となっているのは、従来から Silicon Graphics で提供されていたグラフィックスワークステーション向けに開発された GL（グラフィックスライブラリ）で、これをハードウェアや OS に依存しない形に改良したものが OpenGL にあたる。

3 level 1 : ビットマップレベル

カラー画像の一点の色は、RGB と呼ばれるように、三原色で表現することが出来る。 現在のコンピュータでは、8bit ずつ 24bit の RGB でカラーを表現することが一般的であ る。画像自体は、これらの一点を二次元配列を津化って構成することが出来る。 適当なプログラム言語を用いて、

```
struct bitmap {
unsigned char r, g, b;
} frame_buffer{x}{y};
```

のような構造を定義せよ。これらの上に、 RGB の入れ換え 輝度変換 ガンマカーブ変換拡大縮小、回転、平行移動などのアフィン変換 などの変換を施すプログラムを作成せよ。

3.1 プログラムの概要

このプログラムは、RGB 変換、輝度変換、ガンマカーブ変換、アフィン変換を施すプログラムである。

3.2 関数の説明

level1.c

XWD 画像データの読み込み、XImage 構造体の生成、画像表示のためのウィンドウの生成などをする。また、実際の画像処理 (アフィン変換、RBG 変換等) の手続きが記述してある。

affine.c

各変数宣言の後に、画像回転用、拡大縮小用、平行移動用の行列が定義してある。その後、行列の要素を求める処理、回転後の縦横の幅を求める処理、メモリの確保と実際のアフィン変換処理の定義が書いてある。

確保したメモリ内で、画像が存在する記憶領域と画像が存在しない記憶領域を区別し、画像が存在している場合はマスク処理を行うようになっている。

アフィン変換とは

ユークリッド幾何学的な線型変換と平行移動の組み合わせによる図形や形状の移動、変形方式。4×4 の行列演算で表現できる移動、回転、左右反転、拡大、縮小、シアーの座標変換。アフィン変換は元の図形で直線上に並ぶ点は変換後も直線上に並び、平行線は変換後も平行線であるなど、幾何学的性質が保たれる変換方式。

convert.c

RGB 変換を定義する chrgb や実際の変換を行う chrgb _n などの関数を定義している。また、輝度変換やガンマ変換、モノクロ変換を行う関数も定義してある。(bright、 chgamma、chmono)

ディスプレイが 16 ビットモードの時でも対応できるように処理を分けるよう記述されている。

ガンマ補正とは画像などの色のデータと、それが実際に出力される際の信号の相対関係を調節して、より自然に近い表示を得るための補正操作。γ (ガンマ) 値とは、画像の明るさの変化に対する電圧換算値の変化の比で、これが 1 に近づくのが理想だが、素子の特性により機器によってそれぞれ異なった値となる。このため、元データに忠実な表示を再現したければ、これらの誤差を修正する必要がある。これがガンマ補正である。

3.3 実行結果

図 1: Level1 の実行結果

3.4 実行時間測定

```
[j04042@pw042 ~/level12]% time ./level1 bg.xwd
0.080u 0.030s 0:06.05 4.3% 0+0k 0+0io 210pf+0w
```

./level1 にかかった CPU 時間: 0.080u
./level1 にかかったカーネル時間:0.03 s
経過時間: 6 秒 05

3.5 考察

今回の実験で分かったことは、まず第一に色の表現方法についてだった。RGB（赤、緑、青）の三原色を使い、それぞれの強度を決めて色を表現しているということが分かった。
また、アフィン変換やガンマ変換についても理解が深まったように思う。今後の課題としては、出力した画像をそれぞれ保存できるようになれば良かったと思う。

参考文献

@nifty:デジタル用語辞典
http://www.nifty.com/webapp/digitalword/word/057/05763.htm—

4　level 2 : 直線の描画

frame_buffer をクリアする関数、および、任意の色の直線を描画する関数を作成せよ。

三角函数を使わずに描画する方法が速度的、誤差的に有利である。

4.1　プログラム

Web を参照。

4.2　プログラムの考察

サーバとの接続、カラーマップ ID の所得、ルートウィンドウのウィンドウ ID を所得する。

ウィンドウの生成やイベントマスクの設定をする。

printf で直線の属性や色の設定を出力する。

Start1 で直線の値入力。

Start2 で直線の色入力。

while 文：

case ButtonPress :

ボタンの設定。

```
if(event.xany.window == sw1)
 XClearWindow (d, w);
```

出力されるウィンドウの白紙化。
つまり描画がクリアされる。

```
if(event.xany.window == sw2)
goto Start1;
```

```
if(event.xany.window == sw3)
goto Start2;
```

```
if(event.xany.window == sw4)
goto FINISH;
```

それぞれ Start1,Start2,FINISH にいく。

4.3 前のプログラムとの比較

前のプログラムでは実行後、クリア、直線の属性変更、ウィンドウを閉じる
ボタンがあった。
今回は直線の属性変更を、直線のパターンの変更と色の変更に分けた。

4.4 実行結果

```
nw0453:~/level2 miyagiken$ time ./level2
tyokusen=1
tensen1=2
tensen2=3
Blue   Button = Clear
Green  Button = Change Design
Parple Button = Exit

Line Pattern  ----> 1
Color    ----> red
```

```
Color    ----> blue
Color    ----> yellow
Color    ----> purple

Line Pattern  ----> 2
Color    ----> red
Color    ----> red

Line Pattern  ----> 2
Color    ----> blue

Line Pattern  ----> 2
Color    ----> blue

Line Pattern  ----> 2
Color    ----> yellow

Line Pattern  ----> 2
Color    ----> purple

real    3m58.206s
user    0m0.020s
sys     0m0.030s
```

図 2: 実行結果

4.5 参考文献

http://www.ie.u-ryukyu.ac.jp/~j02039/unix4/UNIX4.html
http://xjman.dsl.gr.jp/

5 level 3 : ワイヤフレーム３次元画像表示

上記の frame_buffer の絵を、各面に持つ立方体の３次元画像表示を作成したい。とりあえず、立方体の３次元画像表示を格納する frame_buffer を作成する。表示する立方体の角度や位置を指定するパラメータとしては、座標変換行列を用いるのが簡単だと思われる。
座標変換行列は、単位行列を回転や拡大縮小などのアフィン変換を利用して作成する。3x3 を用いるのが簡単だが、平行移動を入れた 4x4 の行列を用いる方法もある。
これらの変換行列を使って、立方体の頂点を作り、直線描画を使って、ワイ

ヤフレーム画像を frame_buffer 内に描画し、表示してみよ。

5.1 プログラムの説明

このプログラムは、dat ファイルを用いて３次元の立方体画像を表示させるものである。dat ファイルは描画する頂点と連結する頂点のリストを書く必要がある。dat ファイルを読み込んだら任意の変換値を受け取り、座標を変換して、直線で立方体を表示する。
表示した画像をキーボード、マウスからの入力での回転と終了処理ができるようになっている。
キーボードは'2'、'4'、'6'、'8' にそれぞれ上下左右に一定角度回転するようにした。
また、'q'、'5' で終了するようになっている。
マウスは右、左、中央のボタンでそれぞれドラッグしたときにドラッグしている間上左右に回転する。
回転の表示は一度静止画像を表示して、マウスやキーボードから入力があった場合、画像の回転角度を格納する theta をわずかに加算、減算して描画し直してずらして表示している。

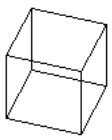

図 3: 実行結果 (cube1.dat)

```
ib09:~/report/source j04002$ ./lv3 xlogo.dat
```

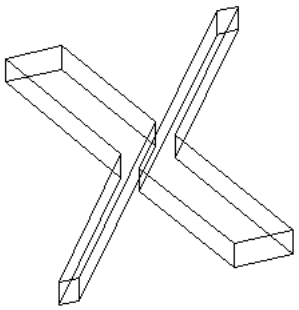

図 4: 実行結果 (xlogo.dat)

参考文献

[1] X11 による画像処理基礎プログラミング、技術評論社、柴山守

[2] X Japanese Documentation Project、http://xjman.dsl.gr.jp/

6 Level 4 : ポリゴン表示

立方体を描画する際に、各面に別な frame_buffer を表示したい。これはテクスチャ・マッピングと呼ばれる。各面は、テクスチャ画像を、アフィン変換したものになる。長方形は、平行四辺形 (遠近による縮小を考慮する場合は、4 辺形) となる。これらのアフィン変換を順番に適用し、立方体のポリゴン表示を行え。適当な数の立方体の表示を行い、時間を測定せよ。プログラムを高速化するにはどうしたら良いか? ハードウェアで実現する場合を考慮して、考察し、実装せよ。

6.1 追加機能

2004 年度前期の P 歩班のプログラムをもとにして追加機能を付けた。

もとのプログラムは x 軸、y 軸、z 軸それぞれの方向で角度を変えて再描写する機能があったが、y 軸方向と z 軸方向はある角度を超えると、エラーになってプログラムが終了してしまう。よって、再描写する方向は x 軸方向だけに限定した。

また、このプログラムの y 軸の回転角度を 0 度から 90 度の間に限定し、そのとき立方体の 6 面全部を描写せず、前面にある 3 面だけを描写するように

した。

6.2 使用した変数や関数の説明

- plane_buffer[][]
  構造体はテクスチャマッピング用のバッファ構造体。

- makeframe()
  関数はフレームバッファを作る関数。このフレームバッファに各面の RGB 値を格納する。

- vertex[][]
  構造体は立方体の頂点を格納する構造体である。

- edge[][]
  構造体は立方体の辺間の位置を格納する構造体である。

- afin()
  関数は各軸にアフィン変換を行う関数である。

- refresh()
  関数は frame_buffer 構造体を初期化する。

- plerefresh()
  関数はテクスチャマッピング用の plane_buffer に RGB 値を格納する。

- drline()
  関数は点による直線の描画関数。

- makecube()
  関数はポリゴンの作成をテクスチャマッピングを行う関数。

- allpaint()
  窓全体の再描画関数。上で求めた各頂点の座標の 最大値、最小値を元に描画開始位置・終了位置を決め、描画をする。

- redraw()
  Expose イベント、ポリゴンを回転させた後の再描写のための関数。

6.3 実行結果

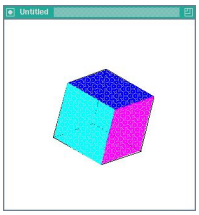

図 5: 実行結果 1

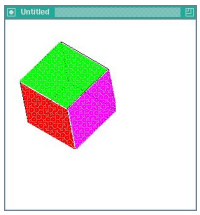

図 6: 実行結果 2

6.4 実行時間、処理能力の測定

```
[mw0437:~/Desktop/level4] j04037% time ./lv4
0.230u 0.340s 0:05.08 11.2%     0+0k 0+0io 0pf+0w
```

実行時間がかなり遅い。

6.5 考察

ワイヤーフレームを作成する level3 と比べ、ポリゴン表示を行うこの level4 のプログラムは実行処理時間がかなりかかった。これは、level3 では XDraw-Point でワイヤーフレームのみを描写していたが、level4 では面全体を描写しているため、level4 のプログラムの描写する点の数が非常に多いことが要因であると考える。処理時間を短くする一つの方法は、描写するときに本来は見えない裏面を描写せず、前面だけを描写することなどが考えられる。

また、元のプログラムは、回転を続けると表示がおかしくなった。これはポリゴンを書く順序に原因がある。ポリゴンは前に書いた面が後に書いた面に上書きされて表示される。よって、本来裏面にあって見えないはずの面が後の方に書くようにプログラムされていると、前面にある本来見えるべき面に上書きして表示される。これを解決するには立方体の前面、裏面を判断してポリゴンを書く順序を変更するなどのプログラムを書く必要がある。今回、この機能を追加しようとしたが失敗した。かわりに、y 軸の回転角度を 0 度から 90 度までの間に限定した条件の下で、立方体の 6 面全部を描写せず前面にある 3 面だけを描写するようなプログラムにした。

6.6 今後の課題

まず、元のプログラムにおいて、y 軸と z 軸の回転角度がある値を超えるとなぜエラーになるのかが分からなかったのでこれを解明する必要がある。

また、変更したプログラムのクリッピングの方法は、y 軸の回転角度を限定した立方体を描写する場合にのみ有効である。この機能を一般化するには、z 軸の値により書くポリゴンが前面か裏面かを判断し、それによりクリッピング操作を行うなどの方法を検討する必要があると考える。

6.7 参考文献

参考文献

[1] X11 による画像処理基礎プログラミング ； 技術評論社

[2] 2004 年度前期 2D/3D Graphics, GUI 「P 歩班」
http://www.ie.u-ryukyu.ac.jp/ j02005/Unix/X11/index.html

[3] 2004 年度前期 2D/3D Graphics, GUI 「ポロチラリッコロ」
http://www.ie.u-ryukyu.ac.jp/ j02023/UNIX/X11/index.html

7 level 6

7.1 課題

任意のツールキットを用いて、簡単な 2D ドローツールを作成せよ。

直線、曲線、四角形、円、楕円

の描画ができること。

また、書いた内容を、SVG フォーマットなどに類似した XML で格納する機能を含めること。

7.2 プログラムの解説

マウスポインタからの始点の座標、終点の座標を取得し、ツールウィンドウからラインサイズと任意のカラーを設定し、直線、四角形、円、楕円、塗りつぶした円、楕円そしてマウスからの任意の線を描画するプログラム。
また、保存形式を jpg, png の 2 種類から選ぶことができる。

7.3　関数の解説

JAVA では,draw,fill, というメソッドが用意されている。そこで Line,Rect, などのオブジェクトを与えることで簡単に表示することができ、また、class Frame に機能拡張した DrawWindow2 を使うことにより classToolbox からのデータの受け渡し行っている。

addMouseMotionListener － このコンポーネントからマウスモーションイベントを受け取るために、指定されたマウスモーションリスナーを追加する。

他に、MouseEvent,MouseMotionListener,removeMouseMotionListener などでマウスのベントの処理を行う部分のプログラムを作成していく。

7.4　実行結果

実際に作成したドローツールを使用していろいろ描画してみる．以下が実行結果である．

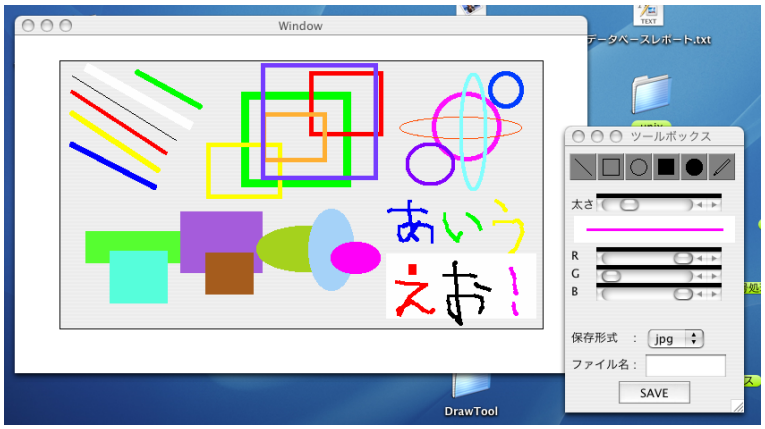

図 7: プログラムの実行結果

7.5　考察

このようにドローツールを java によって実現することが出来た。しかし、XML で格納する機能が実現できなかった。というか機能をもっとふやしたかった。

7.6 参考文献

- 独習 Java　　ジョゼフ・オニール著
- 改訂 新 Java 言語入門 ビギナー編　　林晴比古著