ストップウォッチ 設計書

目次

~設計書の書き方~

・図のリンクは、図番号と図の下段に図名を示しています。
・実際のソースコードからの抜粋はグレーの蛍光ペンで示しています。
　例:#include <stdio.h>

1. 概要

OpenGL のグラフィック描画を用いて、ストップウォッチを作成する。

アナログ時計を再現することで、視覚的にも感覚的にも計測時間が分かるように描画しました。

1-1,グラフィック設定

時計盤を XZ 平面に描画し、視点を Y 軸の正から負の方向へ見るように設定しました。(図 1:グラフィック設定)

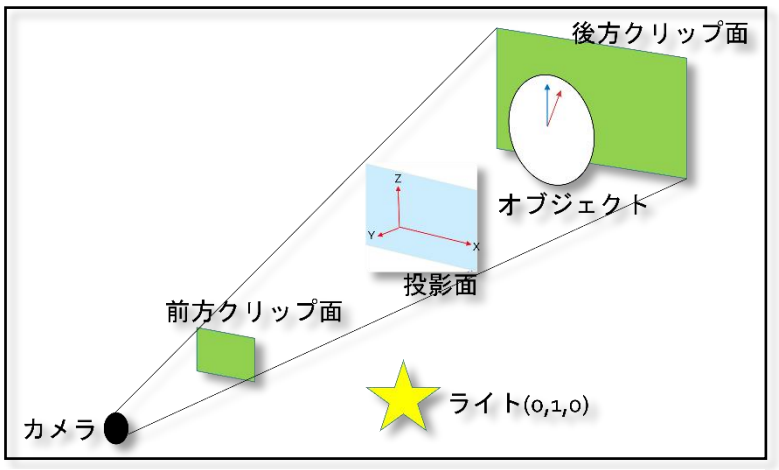

図 1:グラフィック設定

2. 画面構成について

描画した要素は以下の通りです。

・時計盤

・秒針、分針

・目盛り

画面中心を原点とし、時計盤とアナログ時計のような

秒針を赤い針、分針を青い針で時間を表示する。

また、操作ごとにコマンドプロンプトでデジタル時刻

完成イメージ図は以下の通りです(図 2:イメージ図)

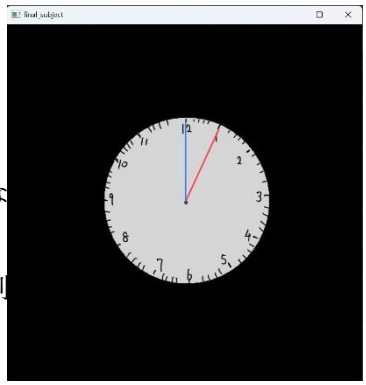

図 2:時計盤のイメージ

各要素の詳細な設定について記述します。

2-1,時計盤(図 3:時計盤のイメージ)

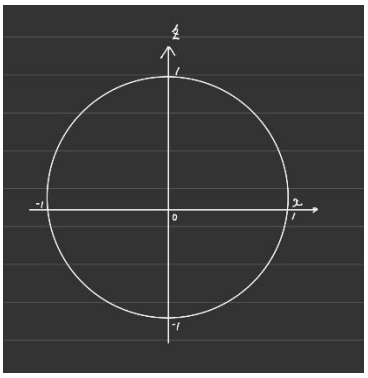

図 3:時計盤のイメージ

点を 50 個打ち、GL_POLYGON により、円を近似しています。

半径は 1 に設定しています。

要素の光の当たり方については、以下のパラメータを用意しました。

```
float diffuse1[] = { 1.0, 1.0, 1.0, 1.0 };
float specular1[] = { 0.5, 0.5, 0.5, 1.0 };
float ambient1[] = { 0.2, 0.2, 1.0, 1.0 };
float shininess1 = 60.0;
```

法線ベクトルは glNormal3f(0.0,1.0,0.0);と設定しています。

2-2,秒針、分針の描画(図 4:分針、秒針のイメージ)

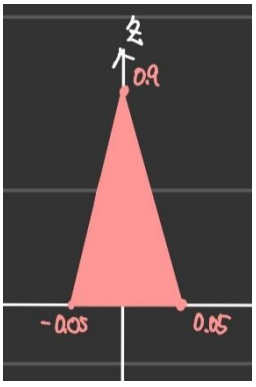

図 4:分針、秒針のイメージ

GL_POLYGON により、三角形を生成します。

回転では、glRotatef(-theta ,0.0 ,1.0 ,0.0);を使用し、原点中心に、y 軸を軸として theta 角で回ります。また、-theta にした理由としては、時計回りに回転させるためです。

秒針の光の当たり方は、以下のように設定しました。

```
float diffuse2[] = { 0.7, 0.1, 0.0, 1.0 };
float specular2[] = { 0.8, 0.0, 0.0, 1.0 };
float ambient2[] = { 0.1, 0.1, 0.1, 1.0 };
float shininess2 = 100.0;
```

法線ベクトルは glNormal3f(0.0,1.0,0.0);と設定しています。

分針の光の当たり方は、以下の通りです。

```
float diffuse3[] = { 0.0, 0.4, 7.0, 1.0 };
float specular3[] = { 0, 0, 0, 1.0 };
float ambient3[] = { 0.1, 0.1, 0.1, 1.0 };
float shininess3 = 128.0;
```

法線ベクトルは glNormal3f(0.0,1.0,0.0);と設定しています。

2-3,小さな目盛り(図 5:目盛り 6°間隔_イメージ)

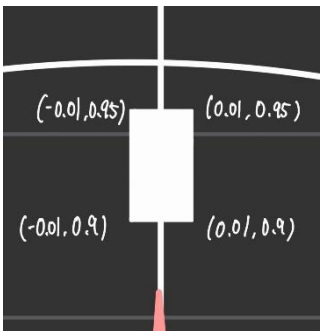

図 5:目盛り 6°間隔_イメージ

GL__POLYGON によって、小さな四角形を生成します。
60 目盛り必要なため、for 文で角度を調整し、描画する。
1 目盛りは 6°の間隔であるため、
glRotatef(i*6 ,0.0 ,1.0 ,0.0);で回転
目盛りの光の当たり方は、以下のように設定しました。

```
float diffuse4[] = { 0.0, 0.0, 0.0, 1.0 };
float specular4[] = { 0, 0, 0, 1.0 };
float ambient4[] = { 0.1, 0.1, 0.1, 1.0 };
float shininess4 = 128.0;
```

法線ベクトルは glNormal3f(0.0,1.0,0.0);と設定しています。

2-4,大きな目盛り(図 5:目盛り 30°間隔_イメージ)

また、5 分間隔で目盛りを大きくし、視認性を上げるため、生成する四角形を大きくします。
for 文で 30°毎目盛りを 12 個描画します。
glRotatef(i*30 ,0.0 ,1.0 ,0.0);で回転しています

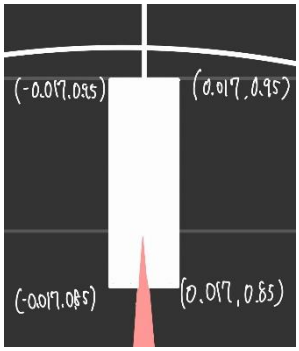

図 5:目盛り 30°間隔_イメージ

目盛りの光の当たり方は、以下のように設定しました。

```
float diffuse4[] = { 0.0, 0.0, 0.0, 1.0 };
float specular4[] = { 0, 0, 0, 1.0 };
float ambient4[] = { 0.1, 0.1, 0.1, 1.0 };
float shininess4 = 128.0;
```

法線ベクトルは glNormal3f(0.0,1.0,0.0);と設定しています。

3. システム構造について

ストップウォッチの状態として、「初期状態」「動作状態」「一時停止状態」の 3 状態を用意します。(図 6:状態遷移図)

3-1,初期状態

プログラム開始後の初期画面状態のことを指します。時間は 00:00 を示しています。「space」を押すことで、動作状態へ遷移します。

3-2,動作状態

ストップウォッチが動作中の状態を指します。針が 1 秒ごとに動き、経過時間を表示してくれます。以下の操作をすることで、異なる状態に遷移します。

- Space キー　:一時停止状態へ遷移、停止時間が表示されます。
- R キー　:初期状態へ強制的に遷移、停止時間もリセットされます。

3-3,一時停止状態

ストップウォッチの一時停止状態です。この状態は、停止された経過時間の情報を保持したまま、停止しています。以下の操作で、異なる遷移をします。

- Space キー　:動作状態へ遷移、針も再び動作します。
- R キー　:初期状態へ強制的に遷移、停止時間もリセットされます

3-4,変数の説明

count_min : 分を数える変数です。一時停止状態の際にコマンドプロンプトに何分か表示する際にも使用します。

count_sec : 秒を数える変数です。一時停止状態の際にコマンドプロンプトに何分か表示する際にも使用します。

System_key : 針や時間の経過を切り替える変数です。0,1 で使い分け、以下に説明します。

- 0…針や経過時間のカウントを行わない状態を表します。
- 1…針や経過時間のカウントを行う状態を表します。

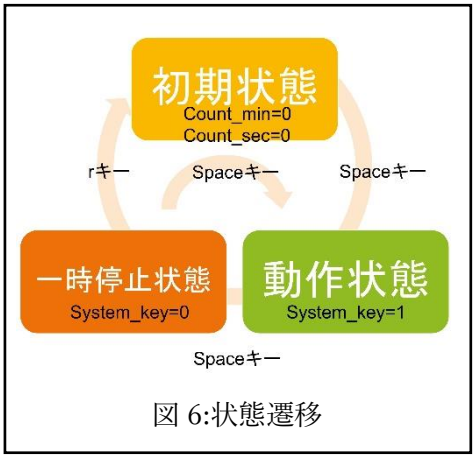

図 6:状態遷移

4. 関数について

関数について以下のように設定しました。

- display 関数
- idle 関数
- mykbd 関数
- myInit 関数
- myReshape 関数
- polarview 関数
- resetview 関数
- main 関数

4-1,display 関数

この関数は、描画を行う処理について記述しています。以下に内容を記載します。

- Z Buffer の反映
- ライトの設定
- 秒針の記述
- 分針の記述
- 時計盤の記述
- 目盛りの記述

4-2,idle 関数

この関数は、イベントによる割り込みが発生しない限り、常に実行される関数です。キーボード入力が行われると、入力に対し変数の値を変更し、画面に反映します。mykbd 関数によって変更された変数に従い、状態を遷移します。詳しくは(図 6:状態遷移図)での遷移をご覧ください。

4-3,mykbd 関数

キーボード入力に関する入力感知関数です。変数 Key に入力されたキーの情報が入り、switch 関数で分岐して処理をおこないます。詳しくは(図 6:状態遷移図)での遷移をご覧ください。また、ESC キーで、実行画面が終了します。

引数 : unsigned char key, int x, int y

4-4,myInit 関数

画面表示用ウインドウの設定や入力を行う関数です。また、0 番のライトを使うことも明記されています。

引数 ：char *progname

4-5,myReshape 関数

画像サイズに関する処理を行います。画面サイズを変更した場合でも、比率を調整します。

引数 ：int width , int height

4-6,polarview 関数

物体を中心に視覚移動を行う際に設定する関数です。この関数は、開発段階で利用した関数であり、保守、運用で行います。

4-7,resetview 関数

各パラメータを初期状態に戻す関数です。polarview 関数同様、開発段階に使用した関数であり、初期化を行っています。保守、運用で行います。

4-8,main 関数

このプログラムのメイン関数です。各種関数を呼び出します。

5. 1秒を表すアルゴリズム

今回のプログラムでは、カウント変数を用いることで、時間の計測を行い、針の再描画を行いました。この章では、この再描画を1秒間隔のタイミングで再現するアルゴリズムについて解説したいと思います。

idle関数内(図8:idle関数)にて、基準時刻とUTCを基準としての現在時刻との差が1秒を超えた瞬間にカウントを1増やします。また、カウントを増やすと同時に基準時刻を1秒増やし、繰り返します。右の図では、00.00秒を表しています。(図7:1秒のイメージ)

図7:1秒のイメージ

基準時刻をprev_time 、現在時刻をcurrent_timeとしています。

if文で比較することで、Trueの場合、count_secに1加算されます。

その後の関数内では、count_secが60を超えた際にcount_minに1加算すると同時に、count_secを0に変更します。

最後に、prev_timeに現在時刻を代入し、次の1秒のカウントのための基準時刻に変更しています。

```
void idle(void)
{
    //system_key=1:ストップウオッチの動作中
    if (system_key == 1)
    {
        // ストップウォッチが動作中の処理
        //現在時刻の取得⇒スペース押された時刻
        time_t current_time=time(NULL);

        //1秒間のタイミングを生成 & 角度の調整
        if (current_time!=prev_time)
        {
            count_sec++;
            if(count_sec>59)
            {
                count_sec=0;
                count_min++;
            }
            //秒数を表示する
            // printf("%2d:%2d\n",count_min,count_sec);
            prev_time=current_time;
        }
    }
    theta=count_sec*6;
    theta_min=count_min*6;
    glutPostRedisplay();
}
```

図8:idle関数

6. まとめ

今回の OpenGL を用いた描画処理を終え、以下の点で評価できると思います。

・モデリング(対象物)

針や時計盤の描画をブレンダー等を用いずに、数学的に描画を行いシンプルに仕上げることができました。また、初期では、ビットマップ画像を時計盤に貼り付け、表現しようとしました。しかし、針の指す角度が目盛りとずれてしまう可能性があったため、目盛りも小さな四角形で表現しました。

・インタラクション(操作)

実世界のストップウォッチは、スタートストップのボタンが、同じであることから、このプログラミでも再現しようとしました。Space キーを押すことでスタートストップを操作することができます。奇数回目では、スタートし操作状態へ、偶数回目では、ストップし一時停止状態を表現しています。

リセットについては、Reset の頭文字をとり、r キーで行うことが来出ます。

直観的な操作を可能とした部分では、大きく評価できると思います。

・アニメーション(動き)

針の動きの部分が該当すると思います。針が、1 秒ごとに動作し、ストップウォッチの動作を明確に表現できているのではないかなと思います。より作業ができるのであれば、滑らかな針の描画ができればよかったかなと思います。

今後の改善点として、1 秒の表現の部分についてこんな意見がありました。

システム時間を取得し、1 秒から実行時間を引いた差を sleep させる方法で、1 秒を正確に表すことができるという意見です。この方法は、既存の実装している計測よりもより正確に 1 秒を表現できるのではないかとおもいました。

また、既存のストップウォッチは、ラップ機能があり、その機能を盛り込むことができればなと思います。さらに、アナログの時計盤を使用しましたが、画面の端にデジタルで経過時間を追加で描画することで、より詳しく時間経過を確認することができるのではないかと思いました。

まとめとして、基本的機能は実装することができましたが、まだまだ工夫や改善できる部分は大いにあり、今後もアップデートをくりかえしたいと思いました。

URL : https://github.com/ShotaArima/OpenGL-stopWatch