プログラミング演習

—アナログ時計—

学籍番号：16426
4 年 電子情報工学科 23 番
福澤 大地

提出日：2019 年 12 月 9 日

1　目的

OpenGL (Open Graphics Library) に準拠した C 言語のライブラリ “GLUT (OpenGL Utility Toolkit)” を用いてアナログ時計のプログラムを作成することで、グラフィカルなウィンドウアプリケーションを作成できるようになる。また、コールバック関数を用いたイベント駆動型プログラミングを行うことで、インタラクティブなユーザーインターフェースを実現する。

2　作成したアナログ時計

内部の歯車や振り子が透けて見えるような振り子時計を作成した。歯車や振り子などのパーツは GLUT の関数を用いて描画し、時間変化に応じて現実の時計と同じような仕組みで動作する。

また、ウィンドウのサイズを変更したときに自動的に時計の大きさが変わり、ウィンドウの領域を最大限に利用して表示されるような機能も実装した。

3　開発環境

プログラムの開発、実行を行った環境を表 1 に示す。

表 1　開発環境

CPU	Intel Core i5-7400 @ 3.0GHz
メモリ	8GB
OS	Microsoft Windows 10 Home
システム	64bit
実行環境	Cygwin 3.0.7
コンパイラ	GCC 7.4.0
OpenGL ライブラリ	GLUT 3.7

4　OpenGL と GLUT [1]

OpenGL とは、2 次元/3 次元のコンピュータグラフィックライブラリである。幅広い処理系に対応しており、汎用性が高いため、広く普及している。

一方で、簡素な機能しか用意されていない分複雑な処理は向いていないため、それを補うために登場したのが GLU (OpenGL Utility Library) である。GLU は OpenGL の補助的なライブラリであり、円柱などの複雑な図形や、テクスチャの処理といった機能を提供する。GLU の機能に加え、ウィンドウ作成やイベント処理などの機能を提供し、より高度なグラフィックを作成することを可能にしたものが今回使用する GLUT である。

5　振り子時計の動作原理 [2]

振り子時計を構成するパーツは図 1 のようになっている。

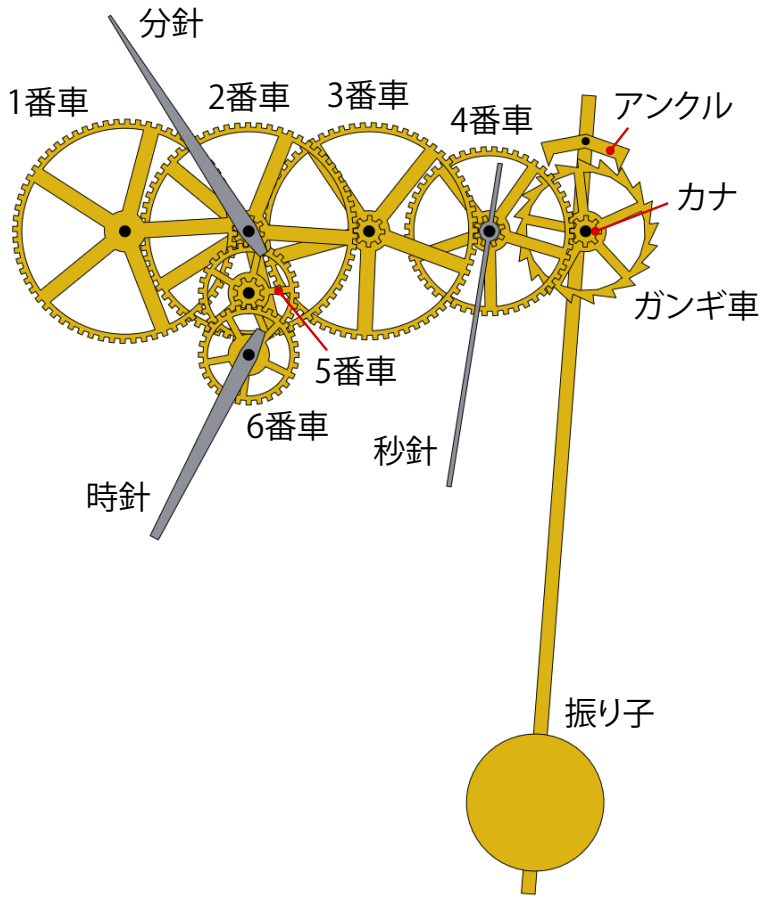

図 1　振り子時計のパーツ

図 1 を見ると、1〜6 番車とガンギ車、アンクル、振り子で構成されていることが分かる。

2, 4, 6 番車には、それぞれ分針、秒針、時針が取り付けられており、3, 5 番車でそれぞれの回転速度の変換を行っている。各歯車と、その中心にあるカナという歯車の歯数の比が、回転速度の比になる。1 番車は動力源となる歯車で、多くの場合は錘やモーター等によって動かされる。

しかしこれらのパーツだけでは、不安定な速度で周ってしまう。そこで、速度を一定に保つために振り子を利用する。振り子には、振れ幅が十分小さければ、振れ幅に関係なく周期が一定になるという性質がある。振り子の運動をガンギ車とアンクルで回転運動に変換することによって、精度良く回転速度を一定にすることができる。

なお、今回作成したプログラムでは、見栄えを良くするためにダミーの歯車を 1 つ追加した。これを 0 番車と呼ぶことにする。

6　プログラムリスト

プログラムのソースコードを、リスト 1–4 に示す。

リスト 1　clock.h

```
1 #ifndef CLOCK_H
2 #define CLOCK_H
3
4 void Display(void);
5 void Reshape(int w, int h);
```

```
6  void Timer(int value);
7
8  extern long double t;
9  extern GLuint dial_img;
10 extern pngInfo dial_info;
11
12 #endif
```

リスト 2　clock.c

```
1  #include <stdio.h>
2  #include <math.h>
3
4  #include <time.h>
5  #include <sys/time.h>
6
7  #include <GL/glut.h>
8  #include <GL/glpng.h>
9
10 #include "clock.h"
11 #include "shape.h"
12
13 #define FPS    60   // フレームレート
14 #define ASPECT 0.75 // アスペクト比(幅/高さ)
15
16 // プロトタイプ宣言
17 void Display(void);
18 void Reshape(int, int);
19 void Timer(int);
20
21 // 現在の時刻
22 long double t;
23
24 // 画像
25 GLuint dial_img;
26 pngInfo dial_info;
27
28 int main(int argc, char *argv[]) {
29     // 初期化
30     glutInit(&argc, argv);
31     glutInitWindowSize(540, 720);
32     glutCreateWindow("Mechanical Clock");
33     glutInitDisplayMode(GLUT_RGBA);
34     glClearColor(0.9, 0.9, 0.9, 1.0);
35
36     // 混合処理を有効化
37     glEnable(GL_BLEND);
38     glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
39     glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
40
41     // 線のアンチエイリアスを有効化
42     glEnable(GL_LINE_SMOOTH);
43     glHint(GL_LINE_SMOOTH_HINT, GL_NICEST);
44
45     // 画像読み込み
46     dial_img = pngBind("dial.png", PNG_NOMIPMAP, PNG_ALPHA, &dial_info, GL_CLAMP,
```

```
    GL_NEAREST, GL_NEAREST);

    // コールバック関数登録
    glutDisplayFunc(Display);
    glutReshapeFunc(Reshape);
    glutTimerFunc(1000.0 / FPS + 0.5, Timer, 0);

    // メインループ開始
    glutMainLoop();

    return 0;
}

// 画面描画
void Display(void) {
    // 振り子の周期 (Beats/Second)
    double bps = 2;

    // 振り子の運動に合わせて時刻が変化するように数値変換
    time_t t_sec = (time_t)t;
    long double t_usec = t - t_sec;
    double map = t_usec * bps;
    double tt = t_sec + (pow(map - (int)map, 6) + (int)map) / bps;

    // 時・秒・分を取得
    struct tm *t_st = localtime(&t_sec);
    double sec = t_st->tm_sec + tt - t_sec;
    double min = t_st->tm_min + sec / 60;
    double hour = t_st->tm_hour + min / 60;

    // 時計の中心座標
    double x0 = 0;
    double y0 = 0.32;

    // 針の長さ
    double lh = 0.45;
    double lm = 0.56;
    double ls = 0.565;

    // 針の角度
    double ah = -2 * M_PI * hour / 12 + M_PI_2;
    double am = -2 * M_PI * min / 60 + M_PI_2;
    double as = -2 * M_PI * sec / 60 + M_PI_2;

    // 歯の長さ
    double l = 0.01;  // 普通の歯車
    double lg = 0.02; // ガンギ車

    // カナを1とした時のギア比
    double p0 = 9;    // 0番車
    double p1 = 8;    // 1番車
    double p2 = 7.75; // 2番車
    double p3 = 7.75; // 3番車
    double p4 = 6;    // 4番車
    double p5 = 3.5;  // 5番車
    double p6 = 3.5;  // 6番車
```

```
102     double pg = 5;    // ガンギ車
103
104     // 半径
105     double rk = 0.03;    // カナ
106     double r0 = rk * p0; // 0番車
107     double r1 = rk * p1; // 1番車
108     double r2 = rk * p2; // 2番車
109     double r3 = rk * p3; // 3番車
110     double r4 = rk * p4; // 4番車
111     double r5 = rk * p5; // 5番車
112     double r6 = rk * p6; // 6番車
113     double rg = rk * pg; // ガンギ車
114     double rp = 1.45;    // 振り子
115
116     // 歯数
117     int nk = 8;             // カナ
118     int n0 = nk * p0 + 0.5; // 0番車
119     int n1 = nk * p1 + 0.5; // 1番車
120     int n2 = nk * p2 + 0.5; // 2番車
121     int n3 = nk * p3 + 0.5; // 3番車
122     int n4 = nk * p4 + 0.5; // 4番車
123     int n5 = nk * p5 + 0.5; // 5番車
124     int n6 = nk * p6 + 0.5; // 6番車
125     int ng = 20;            // ガンギ車
126
127     // 角速度
128     double wp = M_PI * bps;  // 振り子
129     double wg = wp / ng * 2; // ガンギ車
130     double w4 = wg / -p4;    // 4番車
131     double w3 = w4 / -p3;    // 3番車
132     double w2 = w3 / -p2;    // 2番車
133     double w1 = w2 / -p1;    // 1番車
134     double w0 = w2 / -p0;    // 0番車
135     double w5 = w2 / -p5;    // 5番車
136     double w6 = w5 / -p6;    // 6番車
137
138     // 位相
139     double a0 = M_PI / n0; // 0番車
140     double a1 = M_PI / n1; // 1番車
141     double a2 = 0;         // 2番車
142     double a3 = 0;         // 3番車
143     double a4 = 0;         // 4番車
144     double a5 = M_PI / 4;  // 5番車
145     double a6 = M_PI / 4;  // 6番車
146     double ag = M_PI / 8;  // ガンギ車
147     double ap = 0.5;       // 振り子
148
149     // 色
150     GLubyte backColor[3] = {10, 10, 40};       // 時計の背景
151     GLubyte gearColor[3] = {219, 180, 20};     // 歯車
152     GLubyte pendulumColor[3] = {219, 180, 20}; // 振り子
153     GLubyte handColor[3] = {142, 145, 154};    // 針
154     GLubyte axisColor[3] = {0, 0, 0};          // 軸
155
156     // 背景をクリア
157     glClear(GL_COLOR_BUFFER_BIT);
```

```
158
159     // 時計の背景
160     glColor3ubv(backColor);
161     drawPolygon(x0, y0, 0, 0.55, 6);
162
163     // 振り子
164     glColor3ubv(pendulumColor);
165     drawPendulum(x0, y0 + r4 + rg + rk + 0.047, -M_PI_2 + 0.08 * sin(t * wp + ap), rp);
166     glColor3ubv(axisColor);
167     drawPolygon(x0, y0 + r4 + rg + rk + 0.047, 0, 0.008, 20);
168
169     // ガンギ車
170     glColor3ubv(gearColor);
171     drawGangi(x0, y0 + r4 + rk, tt * wg + ag, rg, lg, ng);
172     drawKana(x0, y0 + r4 + rk, tt * wg + ag, rk, l, nk);
173     glColor3ubv(axisColor);
174     drawPolygon(x0, y0 + r4 + rk, 0, 0.012, 20);
175
176     // 4番車
177     glColor3ubv(gearColor);
178     drawGear(x0, y0, tt * w4 + a4, r4, l, n4);
179     drawKana(x0, y0, tt * w4 + a4, rk, l, nk);
180
181     // 0番車
182     glColor3ubv(gearColor);
183     drawGear(x0 + r0 + rk, y0, tt * w0 + a0, r0, l, n0);
184     drawKana(x0 + r0 + rk, y0, tt * w0 + a0, rk, l, nk);
185     glColor3ubv(axisColor);
186     drawPolygon(x0 + r0 + rk, y0, 0, 0.012, 20);
187
188     // 3番車
189     glColor3ubv(gearColor);
190     drawGear(x0, y0 - r2 - rk, tt * w3 + a3, r3, l, n3);
191     drawKana(x0, y0 - r2 - rk, tt * w3 + a3, rk, l, nk);
192     glColor3ubv(axisColor);
193     drawPolygon(x0, y0 - r2 - rk, 0, 0.012, 20);
194
195     // 2番車
196     glColor3ubv(gearColor);
197     drawGear(x0, y0, tt * w2 + a2, r2, l, n2);
198     drawKana(x0, y0, tt * w2 + a2, rk, l, nk);
199
200     // 1番車
201     glColor3ubv(gearColor);
202     drawGear(x0 - r1 - rk, y0, tt * w1 + a1, r1, l, n1);
203     drawKana(x0 - r1 - rk, y0, tt * w1 + a1, rk, l, nk);
204     glColor3ubv(axisColor);
205     drawPolygon(x0 - r1 - rk, y0, 0, 0.012, 20);
206
207     // 5番車
208     glColor3ubv(gearColor);
209     drawGear(x0 + r5 + rk, y0, tt * w5 + a5, r5, l, n5);
210     drawKana(x0 + r5 + rk, y0, tt * w5 + a5, rk, l, nk);
211     glColor3ubv(axisColor);
212     drawPolygon(x0 + r5 + rk, y0, 0, 0.012, 20);
213
```

```
214     // 6番車
215     glColor3ubv(gearColor);
216     drawGear(x0, y0, tt * w6 + a6, r6, l, n6);
217     drawKana(x0, y0, tt * w6 + a6, rk, l, nk);
218 
219     // 文字盤
220     putSprite(dial_img, x0 - 0.6, y0 - 0.6, 1.2, 1.2, &dial_info);
221 
222     // 針
223     glColor3ubv(handColor);
224     drawHour(x0, y0, ah, lh);
225     drawMinute(x0, y0, am, lm);
226     drawSecond(x0, y0, as, ls);
227     glColor3ubv(gearColor);
228     drawPolygon(x0, y0, 0, 0.012, 20);
229 
230     glFlush();
231 }
232 
233 // ウィンドウサイズ変更
234 void Reshape(int w, int h) {
235     // ウィンドウの縦横比
236     double ratio = (double)w / h;
237 
238     // 座標系再設定
239     glViewport(0, 0, w, h);
240     glLoadIdentity();
241 
242     if (ratio > ASPECT)
243         gluOrtho2D(-ratio, ratio, -1, 1);
244     else
245         gluOrtho2D(-ASPECT, ASPECT, -ASPECT / ratio, ASPECT / ratio);
246 }
247 
248 // タイマー
249 void Timer(int value) {
250     struct timeval tv;
251 
252     // 次のタイマーを登録
253     glutTimerFunc(1000.0 / FPS + 0.5, Timer, 0);
254 
255     // 時刻を取得
256     gettimeofday(&tv, NULL);
257     t = tv.tv_sec + tv.tv_usec * 1e-6;
258 
259     Display();
260 }
```

リスト 3　shape.h

```
1 #ifndef SHAPE_H
2 #define SHAPE_H
3 
4 void drawPolygon(double x, double y, double a, double r, int n);
5 void drawGear(double x, double y, double a, double r, double l, int n);
6 void drawKana(double x, double y, double a, double r, double l, int n);
```

```
7  void drawGangi(double x, double y, double a, double r, double l, int n);
8  void drawHour(double x, double y, double a, double r);
9  void drawMinute(double x, double y, double a, double r);
10 void drawSecond(double x, double y, double a, double r);
11 void drawPendulum(double x, double y, double a, double r);
12 void putSprite(GLuint num, double x, double y, double w, double h, pngInfo *info);
13
14 #endif
```

リスト 4　shape.c

```
1  #include <math.h>
2
3  #include <GL/glut.h>
4  #include <GL/glpng.h>
5
6  #include "shape.h"
7
8  // 正多角形を描画
9  // x, y : 座標
10 // a : 角度
11 // r : 半径
12 // n : 角数
13 void drawPolygon(double x, double y, double a, double r, int n) {
14     int i;
15     GLdouble color[4];
16
17     // 1ループあたりの角度
18     double w = 2 * M_PI / n;
19
20     // 現在の色を保存
21     glGetDoublev(GL_CURRENT_COLOR, color);
22
23     // 塗りつぶし
24     glBegin(GL_POLYGON);
25     for (i = 0; i < n; i++)
26         glVertex2d(x + r * sin(i * w + a), y + r * cos(i * w + a));
27     glEnd();
28
29     // 縁取り
30     glColor3ub(0, 0, 0);
31     glBegin(GL_LINE_LOOP);
32     for (i = 0; i < n; i++)
33         glVertex2d(x + r * sin(i * w + a), y + r * cos(i * w + a));
34     glEnd();
35
36     // 色を元に戻す
37     glColor4dv(color);
38 }
39
40 // 歯車を描画
41 // x, y : 座標
42 // a : 角度
43 // r : 半径
44 // l : 歯の長さ
45 // n : 歯数
```

```
46 void drawGear(double x, double y, double a, double r, double l, int n) {
47     int i;
48     GLdouble color[4];
49 
50     double rs = 0.045;    // 中心の円の半径
51     double hf = r * 0.07; // 骨組みの幅
52 
53     // 1ループあたりの角度
54     double w = M_PI / n;
55     double wf = 2 * M_PI / 5;
56 
57     // 現在の色を保存
58     glGetDoublev(GL_CURRENT_COLOR, color);
59 
60     // 歯の内側の輪
61     glBegin(GL_QUAD_STRIP);
62     for (i = 0; i <= n * 2; i++) {
63         glVertex2d(x + (r - l * 1.5) * cos(i * w + a), y + (r - l * 1.5) * sin(i * w + a));
64         glVertex2d(x + (r - l * 0.5) * cos(i * w + a), y + (r - l * 0.5) * sin(i * w + a));
65     }
66     glEnd();
67 
68     // 縁取り
69     glColor4ub(0, 0, 0, 0);
70     drawPolygon(x, y, a, r - l * 1.5, n * 2);
71 
72     // 中心の円
73     glColor4dv(color);
74     drawPolygon(x, y, a, rs, n * 2);
75 
76     // 骨組み
77     glBegin(GL_QUADS);
78     for (i = 0; i < 5; i++) {
79         glVertex2d(x + hf * sin(i * wf + a), y - hf * cos(i * wf + a));
80         glVertex2d(x - hf * sin(i * wf + a), y + hf * cos(i * wf + a));
81         glVertex2d(x + (r - l) * cos(i * wf + a) - hf * sin(i * wf + a), y + (r - l) * sin(
    i * wf + a) + hf * cos(i * wf + a));
82         glVertex2d(x + (r - l) * cos(i * wf + a) + hf * sin(i * wf + a), y + (r - l) * sin(
    i * wf + a) - hf * cos(i * wf + a));
83     }
84     glEnd();
85 
86     // 縁取り
87     glColor3ub(0, 0, 0);
88     glBegin(GL_LINES);
89     for (i = 0; i < 5; i++) {
90         glVertex2d(x + rs * cos(i * wf + a - asin(hf / rs)), y + rs * sin(i * wf + a - asin
    (hf / rs)));
91         glVertex2d(x + (r - l * 1.5) * cos(i * wf + a - asin(hf / (r - l * 1.5))), y + (r -
     l * 1.5) * sin(i * wf + a - asin(hf / (r - l * 1.5))));
92         glVertex2d(x + rs * cos(i * wf + a + asin(hf / rs)), y + rs * sin(i * wf + a + asin
    (hf / rs)));
93         glVertex2d(x + (r - l * 1.5) * cos(i * wf + a + asin(hf / (r - l * 1.5))), y + (r -
     l * 1.5) * sin(i * wf + a + asin(hf / (r - l * 1.5))));
94     }
95     glEnd();
```

```
96
97      // 歯
98      glColor4dv(color);
99      glBegin(GL_QUADS);
100     for (i = 0; i < n * 2; i += 2) {
101         glVertex2d(x + (r - l * 0.5) * cos((i - 0.5) * w + a), y + (r - l * 0.5) * sin((i -
          0.5) * w + a));
102         glVertex2d(x + (r - l * 0.5) * cos((i + 0.5) * w + a), y + (r - l * 0.5) * sin((i +
          0.5) * w + a));
103         glVertex2d(x + (r - l * 0.5) * cos((i + 0.5) * w + a) + l * cos(i * w + a), y + (r
        - l * 0.5) * sin((i + 0.5) * w + a) + l * sin(i * w + a));
104         glVertex2d(x + (r - l * 0.5) * cos((i - 0.5) * w + a) + l * cos(i * w + a), y + (r
        - l * 0.5) * sin((i - 0.5) * w + a) + l * sin(i * w + a));
105     }
106     glEnd();
107
108     // 縁取り
109     glColor3ub(0, 0, 0);
110     glBegin(GL_LINE_LOOP);
111     for (i = 0; i < 2 * n; i++) {
112         if (i % 2) {
113             glVertex2d(x + (r - l * 0.5) * cos((i - 0.5) * w + a), y + (r - l * 0.5) * sin
        ((i - 0.5) * w + a));
114             glVertex2d(x + (r - l * 0.5) * cos((i + 0.5) * w + a), y + (r - l * 0.5) * sin
        ((i + 0.5) * w + a));
115         } else {
116             glVertex2d(x + (r - l * 0.5) * cos((i - 0.5) * w + a) + l * cos(i * w + a), y +
          (r - l * 0.5) * sin((i - 0.5) * w + a) + l * sin(i * w + a));
117             glVertex2d(x + (r - l * 0.5) * cos((i + 0.5) * w + a) + l * cos(i * w + a), y +
          (r - l * 0.5) * sin((i + 0.5) * w + a) + l * sin(i * w + a));
118         }
119     }
120     glEnd();
121
122     glColor4dv(color);
123 }
124
125 // カナを描画
126 // x, y : 座標
127 // a : 角度
128 // r : 半径
129 // l : 歯の長さ
130 // n : 歯数
131 void drawKana(double x, double y, double a, double r, double l, int n) {
132     int i;
133     GLdouble color[4];
134
135     // 1ループあたりの角度
136     double w = M_PI / n;
137
138     // 現在の色を保存
139     glGetDoublev(GL_CURRENT_COLOR, color);
140
141     // 円
142     glBegin(GL_POLYGON);
143     for (i = 0; i < n * 2; i++)
```

```
144         glVertex2d(x + (r - l * 0.5) * sin(i * w + a), y + (r - l * 0.5) * cos(i * w + a));
145     glEnd();
146 
147     // 歯
148     glColor4dv(color);
149     glBegin(GL_QUADS);
150     for (i = 0; i < n * 2; i += 2) {
151         glVertex2d(x + (r - l * 0.5) * cos((i - 0.5) * w + a), y + (r - l * 0.5) * sin((i -
        0.5) * w + a));
152         glVertex2d(x + (r - l * 0.5) * cos((i + 0.5) * w + a), y + (r - l * 0.5) * sin((i +
        0.5) * w + a));
153         glVertex2d(x + (r - l * 0.5) * cos((i + 0.5) * w + a) + l * cos((i) * w + a), y + (
    r - l * 0.5) * sin((i + 0.5) * w + a) + l * sin(i * w + a));
154         glVertex2d(x + (r - l * 0.5) * cos((i - 0.5) * w + a) + l * cos((i) * w + a), y + (
    r - l * 0.5) * sin((i - 0.5) * w + a) + l * sin(i * w + a));
155     }
156     glEnd();
157 
158     // 縁取り
159     glColor3ub(0, 0, 0);
160     glBegin(GL_LINE_LOOP);
161     for (i = 0; i < 2 * n; i++) {
162         if (i % 2) {
163             glVertex2d(x + (r - l * 0.5) * cos((i - 0.5) * w + a), y + (r - l * 0.5) * sin
    ((i - 0.5) * w + a));
164             glVertex2d(x + (r - l * 0.5) * cos((i + 0.5) * w + a), y + (r - l * 0.5) * sin
    ((i + 0.5) * w + a));
165         } else {
166             glVertex2d(x + (r - l * 0.5) * cos((i - 0.5) * w + a) + l * cos(i * w + a), y +
     (r - l * 0.5) * sin((i - 0.5) * w + a) + l * sin(i * w + a));
167             glVertex2d(x + (r - l * 0.5) * cos((i + 0.5) * w + a) + l * cos(i * w + a), y +
     (r - l * 0.5) * sin((i + 0.5) * w + a) + l * sin(i * w + a));
168         }
169     }
170     glEnd();
171 
172     glColor4dv(color);
173 }
174 
175 // ガンギ車を描画
176 // x, y : 座標
177 // a : 角度
178 // r : 半径
179 // l : 歯の長さ
180 // n : 歯数
181 void drawGangi(double x, double y, double a, double r, double l, int n) {
182     int i;
183     GLdouble color[4];
184 
185     double rs = 0.045;    // 中心の円の半径
186     double hf = r * 0.07; // 骨組みの幅
187 
188     // 1ループあたりの角度
189     double w = -2 * M_PI / n;
190     double wf = 2 * M_PI / 5;
191 
```

```
192     // アンクルと噛み合うように角度調整
193     a += 0.03;
194
195     // 現在の色を保存
196     glGetDoublev(GL_CURRENT_COLOR, color);
197
198     // 歯の内側の輪
199     glBegin(GL_QUAD_STRIP);
200     for (i = 0; i <= n; i++) {
201         glVertex2d(x + (r - l) * cos(i * w + a), y + (r - l) * sin(i * w + a));
202         glVertex2d(x + (r - l * 0.5) * cos(i * w + a), y + (r - l * 0.5) * sin(i * w + a));
203     }
204     glEnd();
205
206     // 縁取り
207     glColor4ub(0, 0, 0, 0);
208     drawPolygon(x, y, a, r - l, n);
209
210     // 中心の円
211     glColor4dv(color);
212     drawPolygon(x, y, a, rs, n);
213
214     // 骨組み
215     glBegin(GL_QUADS);
216     for (i = 0; i < 5; i++) {
217         glVertex2d(x + hf * sin(i * wf + a), y - hf * cos(i * wf + a));
218         glVertex2d(x - hf * sin(i * wf + a), y + hf * cos(i * wf + a));
219         glVertex2d(x + (r - l * 0.75) * cos(i * wf + a) - hf * sin(i * wf + a), y + (r - l
    * 0.75) * sin(i * wf + a) + hf * cos(i * wf + a));
220         glVertex2d(x + (r - l * 0.75) * cos(i * wf + a) + hf * sin(i * wf + a), y + (r - l
    * 0.75) * sin(i * wf + a) - hf * cos(i * wf + a));
221     }
222     glEnd();
223
224     // 縁取り
225     glColor3ub(0, 0, 0);
226     glBegin(GL_LINES);
227     for (i = 0; i < 5; i++) {
228         glVertex2d(x + rs * cos(i * wf + a - asin(hf / rs)), y + rs * sin(i * wf + a - asin
    (hf / rs)));
229         glVertex2d(x + (r - l) * cos(i * wf + a - asin(hf / (r - l))), y + (r - l) * sin(i
    * wf + a - asin(hf / (r - l))));
230         glVertex2d(x + rs * cos(i * wf + a + asin(hf / rs)), y + rs * sin(i * wf + a + asin
    (hf / rs)));
231         glVertex2d(x + (r - l) * cos(i * wf + a + asin(hf / (r - l))), y + (r - l) * sin(i
    * wf + a + asin(hf / (r - l))));
232     }
233     glEnd();
234
235     // 歯
236     glColor4dv(color);
237     glBegin(GL_TRIANGLES);
238     for (i = 0; i < n; i++) {
239         glVertex2d(x + (r - l * 0.5) * cos(i * w + a), y + (r - l * 0.5) * sin(i * w + a));
240         glVertex2d(x + (r + l * 0.5) * cos(i * w + a), y + (r + l * 0.5) * sin(i * w + a));
241         glVertex2d(x + (r - l * 0.5) * cos((i + 1) * w + a), y + (r - l * 0.5) * sin((i +
```

```
        1) * w + a));
242     }
243     glEnd();
244
245     // 縁取り
246     glColor3ub(0, 0, 0);
247     glBegin(GL_LINE_LOOP);
248     for (i = 0; i < n; i++) {
249         glVertex2d(x + (r - l * 0.5) * cos(i * w + a), y + (r - l * 0.5) * sin(i * w + a));
250         glVertex2d(x + (r + l * 0.5) * cos(i * w + a), y + (r + l * 0.5) * sin(i * w + a));
251     }
252     glEnd();
253
254     glColor4dv(color);
255 }
256
257 // 時針を描画
258 // x, y : 座標
259 // a : 角度
260 // r : 半径
261 void drawHour(double x, double y, double a, double r) {
262     GLdouble color[4];
263
264     double rr = 0.06; // 根本の半径
265     double hr = 0.01; // 根本の幅
266     double hc = 0.02; // 中心の幅
267     double ht = 0.01; // 先端の幅
268
269     // 現在の色を保存
270     glGetDoublev(GL_CURRENT_COLOR, color);
271
272     // 塗りつぶし
273     glBegin(GL_POLYGON);
274     glVertex2d(x - rr * cos(a) - hr * sin(a), y - rr * sin(a) + hr * cos(a));
275     glVertex2d(x - rr * cos(a) + hr * sin(a), y - rr * sin(a) - hr * cos(a));
276     glVertex2d(x + hc * sin(a), y - hc * cos(a));
277     glVertex2d(x + r * cos(a) + ht * sin(a), y + r * sin(a) - ht * cos(a));
278     glVertex2d(x + r * cos(a) - ht * sin(a), y + r * sin(a) + ht * cos(a));
279     glVertex2d(x - hc * sin(a), y + hc * cos(a));
280     glEnd();
281
282     // 縁取り
283     glColor3ub(0, 0, 0);
284     glBegin(GL_LINE_LOOP);
285     glVertex2d(x - rr * cos(a) - hr * sin(a), y - rr * sin(a) + hr * cos(a));
286     glVertex2d(x - rr * cos(a) + hr * sin(a), y - rr * sin(a) - hr * cos(a));
287     glVertex2d(x + hc * sin(a), y - hc * cos(a));
288     glVertex2d(x + r * cos(a) + ht * sin(a), y + r * sin(a) - ht * cos(a));
289     glVertex2d(x + r * cos(a) - ht * sin(a), y + r * sin(a) + ht * cos(a));
290     glVertex2d(x - hc * sin(a), y + hc * cos(a));
291     glEnd();
292
293     glColor4dv(color);
294 }
295
296 // 分針を描画
```

```
297 // x, y : 座標
298 // a : 角度
299 // r : 半径
300 void drawMinute(double x, double y, double a, double r) {
301     GLdouble color[4];
302 
303     double rr = 0.06; // 根本の半径
304     double hr = 0.01; // 根本の幅
305     double hc = 0.02; // 中心の幅
306     double ht = 0.003; // 先端の幅
307 
308     // 現在の色を保存
309     glGetDoublev(GL_CURRENT_COLOR, color);
310 
311     // 塗りつぶし
312     glBegin(GL_POLYGON);
313     glVertex2d(x - rr * cos(a) - hr * sin(a), y - rr * sin(a) + hr * cos(a));
314     glVertex2d(x - rr * cos(a) + hr * sin(a), y - rr * sin(a) - hr * cos(a));
315     glVertex2d(x + hc * sin(a), y - hc * cos(a));
316     glVertex2d(x + r * cos(a) + ht * sin(a), y + r * sin(a) - ht * cos(a));
317     glVertex2d(x + r * cos(a) - ht * sin(a), y + r * sin(a) + ht * cos(a));
318     glVertex2d(x - hc * sin(a), y + hc * cos(a));
319     glEnd();
320 
321     // 縁取り
322     glColor3ub(0, 0, 0);
323     glBegin(GL_LINE_LOOP);
324     glVertex2d(x - rr * cos(a) - hr * sin(a), y - rr * sin(a) + hr * cos(a));
325     glVertex2d(x - rr * cos(a) + hr * sin(a), y - rr * sin(a) - hr * cos(a));
326     glVertex2d(x + hc * sin(a), y - hc * cos(a));
327     glVertex2d(x + r * cos(a) + ht * sin(a), y + r * sin(a) - ht * cos(a));
328     glVertex2d(x + r * cos(a) - ht * sin(a), y + r * sin(a) + ht * cos(a));
329     glVertex2d(x - hc * sin(a), y + hc * cos(a));
330     glEnd();
331 
332     glColor4dv(color);
333 }
334 
335 // 秒針を描画
336 // x, y : 座標
337 // a : 角度
338 // r : 半径
339 void drawSecond(double x, double y, double a, double r) {
340     GLdouble color[4];
341 
342     double rr = 0.15;  // 根本の半径
343     double rc = 0.022; // 中心の円の半径
344     double h = 0.005;  // 針の幅
345 
346     // 現在の色を保存
347     glGetDoublev(GL_CURRENT_COLOR, color);
348 
349     // 中心の円
350     drawPolygon(x, y, a, rc, 20);
351 
352     // 塗りつぶし
```

```
353     glBegin(GL_QUADS);
354     glVertex2d(x - rr * cos(a) - h * sin(a), y - rr * sin(a) + h * cos(a));
355     glVertex2d(x - rr * cos(a) + h * sin(a), y - rr * sin(a) - h * cos(a));
356     glVertex2d(x + r * cos(a) + h * sin(a), y + r * sin(a) - h * cos(a));
357     glVertex2d(x + r * cos(a) - h * sin(a), y + r * sin(a) + h * cos(a));
358     glEnd();
359 
360     // 縁取り
361     glColor3ub(0, 0, 0);
362     glBegin(GL_LINE_STRIP);
363     glVertex2d(x - rc * cos(a - asin(h / rc)), y - rc * sin(a - asin(h / rc)));
364     glVertex2d(x - rr * cos(a) - h * sin(a), y - rr * sin(a) + h * cos(a));
365     glVertex2d(x - rr * cos(a) + h * sin(a), y - rr * sin(a) - h * cos(a));
366     glVertex2d(x - rc * cos(a + asin(h / rc)), y - rc * sin(a + asin(h / rc)));
367     glEnd();
368 
369     glBegin(GL_LINE_STRIP);
370     glVertex2d(x + rc * cos(a - asin(h / rc)), y + rc * sin(a - asin(h / rc)));
371     glVertex2d(x + r * cos(a) + h * sin(a), y + r * sin(a) - h * cos(a));
372     glVertex2d(x + r * cos(a) - h * sin(a), y + r * sin(a) + h * cos(a));
373     glVertex2d(x + rc * cos(a + asin(h / rc)), y + rc * sin(a + asin(h / rc)));
374     glEnd();
375 
376     glColor4dv(color);
377 }
378 
379 // 振り子を描画
380 // x, y : 座標
381 // a : 角度
382 // r : 半径
383 void drawPendulum(double x, double y, double a, double r) {
384     GLdouble color[4];
385 
386     double rw = 0.15;       // 錘の半
387     double rr = 0.1;        // 根本の半径
388     double rs = r + 0.2;    // 棒の半径
389     double hs = 0.015;      // 棒の幅
390     double aa = M_PI / 12; // アンクルの角度
391     double ha = 0.015;      // アンクルの幅
392     double lt = 0.04;       // アンクルの先端の長さ
393     double lu = 0.10;       // アンクル上部の長さ
394     double lc = 0.079;      // アンクル中部の長さ
395     double ld = 0.07;       // アンクル下部の長さ
396 
397     // 現在の色を保存
398     glGetDoublev(GL_CURRENT_COLOR, color);
399 
400     // 棒
401     glBegin(GL_QUADS);
402     glVertex2d(x - rr * cos(a) - hs * sin(a), y - rr * sin(a) + hs * cos(a));
403     glVertex2d(x - rr * cos(a) + hs * sin(a), y - rr * sin(a) - hs * cos(a));
404     glVertex2d(x + rs * cos(a) + hs * sin(a), y + rs * sin(a) - hs * cos(a));
405     glVertex2d(x + rs * cos(a) - hs * sin(a), y + rs * sin(a) + hs * cos(a));
406     glEnd();
407 
408     // 縁取り
```

```
409     glColor3ub(0, 0, 0);
410     glBegin(GL_LINE_LOOP);
411     glVertex2d(x - rr * cos(a) - hs * sin(a), y - rr * sin(a) + hs * cos(a));
412     glVertex2d(x - rr * cos(a) + hs * sin(a), y - rr * sin(a) - hs * cos(a));
413     glVertex2d(x + rs * cos(a) + hs * sin(a), y + rs * sin(a) - hs * cos(a));
414     glVertex2d(x + rs * cos(a) - hs * sin(a), y + rs * sin(a) + hs * cos(a));
415     glEnd();
416
417     // 重り
418     glColor4dv(color);
419     drawPolygon(x + r * cos(a), y + r * sin(a), -a, rw, 50);
420
421     // アンクル
422     glBegin(GL_TRIANGLES);
423     glVertex2d(x + lt * cos(a) + lc * sin(a + aa), y + lt * sin(a) - lc * cos(a + aa));
424     glVertex2d(x + ha * cos(a) + ld * sin(a + aa), y + ha * sin(a) - ld * cos(a + aa));
425     glVertex2d(x - ha * cos(a) + lu * sin(a + aa), y - ha * sin(a) - lu * cos(a + aa));
426     glVertex2d(x + lt * cos(a) - lc * sin(a - aa), y + lt * sin(a) + lc * cos(a - aa));
427     glVertex2d(x + ha * cos(a) - ld * sin(a - aa), y + ha * sin(a) + ld * cos(a - aa));
428     glVertex2d(x - ha * cos(a) - lu * sin(a - aa), y - ha * sin(a) + lu * cos(a - aa));
429     glEnd();
430
431     glBegin(GL_QUAD_STRIP);
432     glVertex2d(x + ha * cos(a) + ld * sin(a + aa), y + ha * sin(a) - ld * cos(a + aa));
433     glVertex2d(x - ha * cos(a) + lu * sin(a + aa), y - ha * sin(a) - lu * cos(a + aa));
434     glVertex2d(x + ha * cos(a), y + ha * sin(a));
435     glVertex2d(x - ha * cos(a), y - ha * sin(a));
436     glVertex2d(x + ha * cos(a) - ld * sin(a - aa), y + ha * sin(a) + ld * cos(a - aa));
437     glVertex2d(x - ha * cos(a) - lu * sin(a - aa), y - ha * sin(a) + lu * cos(a - aa));
438     glEnd();
439
440     // 縁取り
441     glColor3ub(0, 0, 0);
442     glBegin(GL_LINE_LOOP);
443     glVertex2d(x + lt * cos(a) + lc * sin(a + aa), y + lt * sin(a) - lc * cos(a + aa));
444     glVertex2d(x + ha * cos(a) + ld * sin(a + aa), y + ha * sin(a) - ld * cos(a + aa));
445     glVertex2d(x + ha * cos(a), y + ha * sin(a));
446     glVertex2d(x + ha * cos(a) - ld * sin(a - aa), y + ha * sin(a) + ld * cos(a - aa));
447     glVertex2d(x + lt * cos(a) - lc * sin(a - aa), y + lt * sin(a) + lc * cos(a - aa));
448     glVertex2d(x - ha * cos(a) - lu * sin(a - aa), y - ha * sin(a) + lu * cos(a - aa));
449     glVertex2d(x - ha * cos(a), y - ha * sin(a));
450     glVertex2d(x - ha * cos(a) + lu * sin(a + aa), y - ha * sin(a) - lu * cos(a + aa));
451     glEnd();
452
453     glColor4dv(color);
454 }
455
456 // 画像を表示
457 // num : 画像の番号
458 // x, y : 座標
459 // w, h : サイズ
460 // pngInfo : 画像情報
461 void putSprite(GLuint num, double x, double y, double w, double h, pngInfo *info) {
462     float color[4];
463
464     // 現在の色を保存
```

```
    glGetFloatv(GL_CURRENT_COLOR, color);

    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, num);
    glColor4ub(255, 255, 255, 255);

    glBegin(GL_QUADS);

    glTexCoord2i(0, 0);
    glVertex2d(x, y + h);

    glTexCoord2i(0, 1);
    glVertex2d(x, y);

    glTexCoord2i(1, 1);
    glVertex2d(x + w, y);

    glTexCoord2i(1, 0);
    glVertex2d(x + w, y + h);

    glEnd();
    glColor3fv(color);
    glDisable(GL_TEXTURE_2D);
}
```

リスト 5　Makefile

```
TARGET = clock.exe

SRCS = clock.c shape.c
ICON = icon.ico

ICON_RC = ${ICON:.ico=.rc}
ICON_OBJ = ${ICON:.ico=.o}
OBJS = ${SRCS:.c=.o} $(ICON_OBJ)

HEADERS = clock.h shape.h

CC = gcc
CCFLAGS = -Wall -I/usr/include/opengl
LD = gcc
LDFLAGS =
LIBS = -lm -lglpng -lglut32 -lglu32 -lopengl32

$(TARGET): $(OBJS)
    $(LD) $(OBJS) $(LDFLAGS) -o $(TARGET) $(LIBS)

.SUFFIXES: .o .c
.c.o:
    $(CC) $(CCFLAGS) -c $<

$(OBJS): $(HEADERS) Makefile

$(ICON_OBJ): $(ICON) $(ICON_RC)
    windres -i $(ICON_RC) -o $(ICON_OBJ)

```

```
30 .PHONY: clean
31 clean:
32     rm -f $(TARGET) $(OBJS) core *-
```

7　リソース

プログラム中で用いられる文字盤の画像 “dial.png” を図 2, アプリケーションのアイコン “icon.ico” を図 3 に示す。

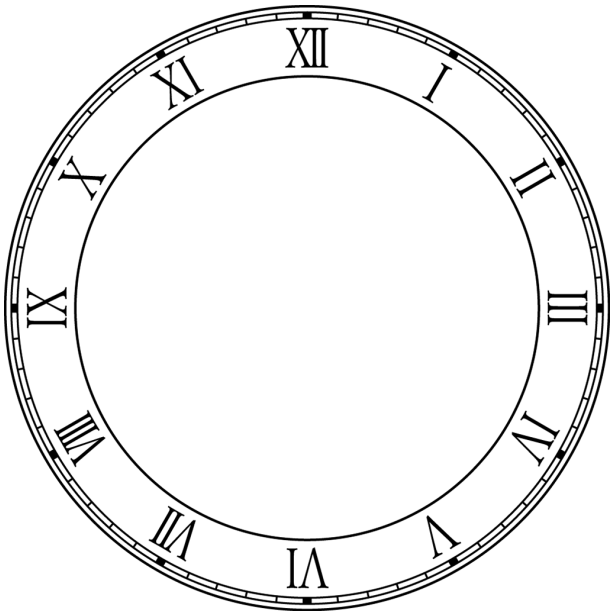

図 2　dial.png

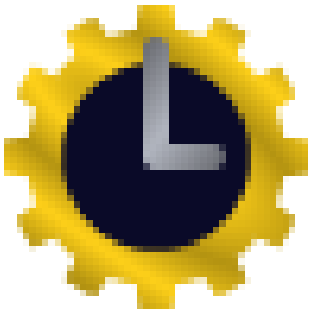

図 3　icon.ico

8　ビルド方法

GLUT と glpng が導入されている Cygwin 上で、次のコマンドを入力することでビルド、実行することができる。

```
$ make
$ ./j16426.exe
```

9　実行結果

本プログラムを実行した結果を図 4 に示す。図 4 を見ると、現在時刻が表示されており、文字盤の中には歯車と振り子が描画されていることが分かる。また、振り子の動きに連動して各歯車と針が動いている。

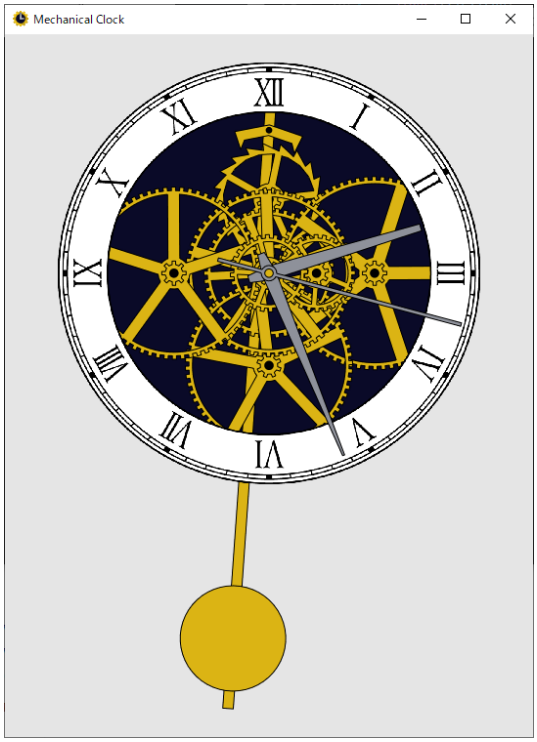

図 4　実行結果

9.1　アニメーション

アニメーションは、`glutTimerFunc` 関数を用いて、短い間隔で描画関数を呼び出すことによって実現している。タイマーのコールバック関数である `Timer` の中でさらにタイマーを登録することで、連続的なアニメーションを表現することが出来る。

また、`Timer` 関数では現在時刻の取得も行う。描画関数である `Display` の中で時刻の取得を行わないのは、機能ごとに関数を分けることで保守性や移植性などを高めるためである。

9.2　座標系

ウィンドウを横長になるように広げた状態を図 5, 縦長になるように縮めた状態を図 6 に示す。図 5 を見ると、ウィンドウサイズに合わせて時計が拡大され、ウィンドウの中心に表示されている。図 6 を見ると、ウィンドウサイズを小さくしても見切れることなく、ウィンドウの横幅いっぱいに表示されていることが分かる。

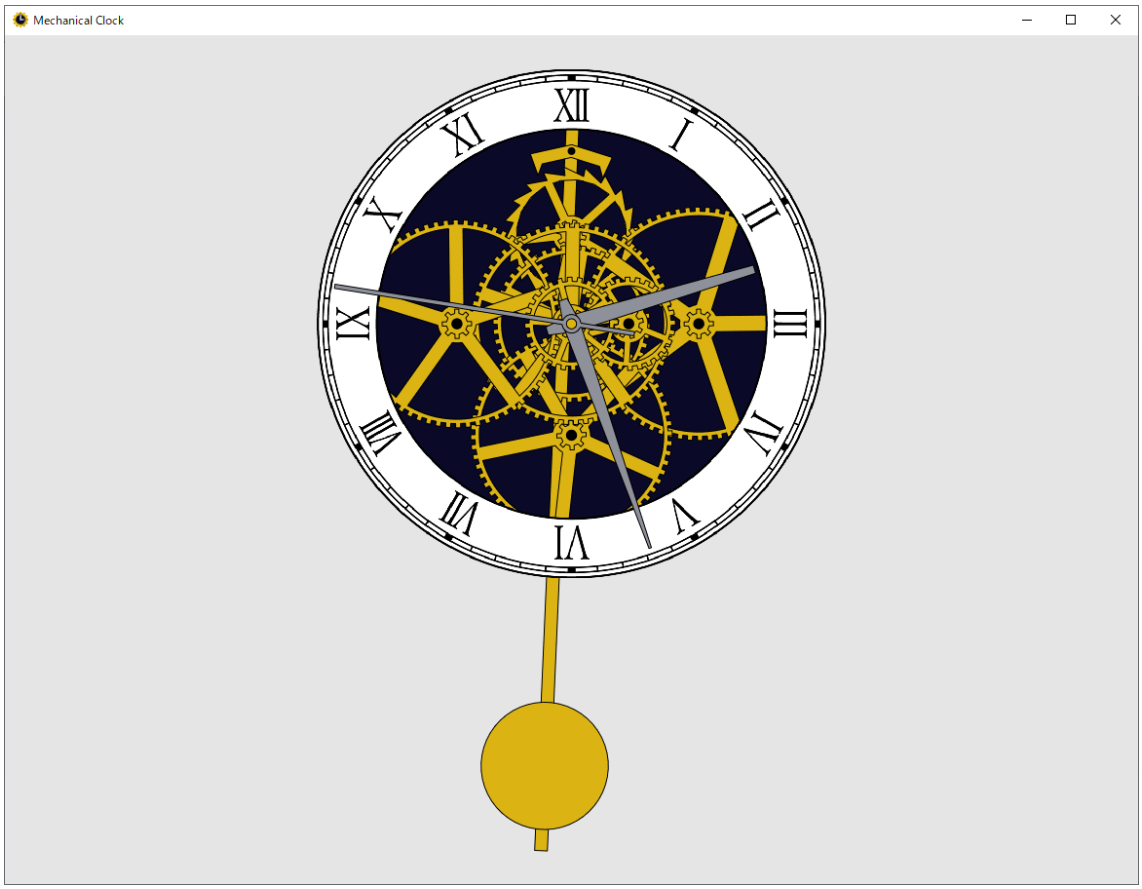

図 5　横に長いウィンドウ

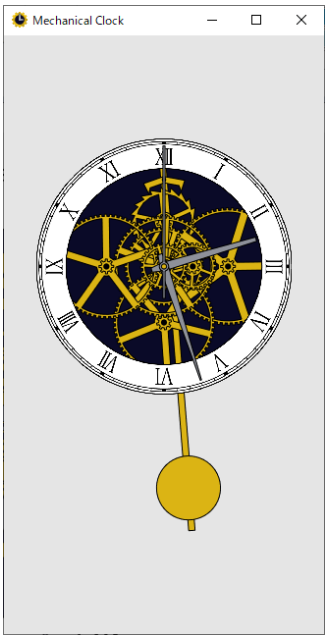

図 6　縦に長いウィンドウ

これは、ウィンドウサイズが変更された際に逐次座標系を変更しているためである。ウィンドウが横に長い際の座標系は図 7, 縦に長い際は図 8 のようになる。

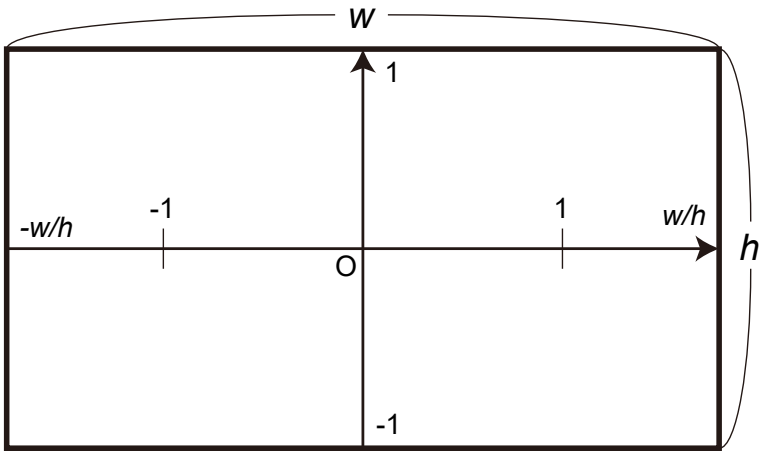

図 7　横に長い座標系

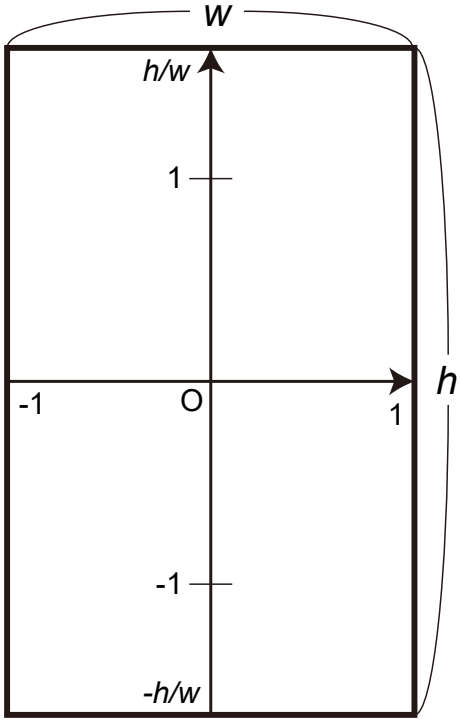

図 8　縦に長い座標系

この設定を行っているのが Reshape 関数である。ウィンドウサイズが変更された際に呼び出されるコールバック関数として glutReshapeFunc 関数で登録されている。gluOrtho2D という関数で画面端に対する座標を指定することができる。これを図 7, 8 のように設定することで、画面サイズに対する比で座標をしていすることになるので、ウィンドウサイズが変更されてもそれに合わせて拡大縮小されるようになる。また、x 軸方向と y 軸方向の座標の比を 1:1 になるようにしているため、横長、縦長にしても時計が潰れることなく、正円が保たれる。

9.3 図形描画

shape.c には、歯車や振り子を描画する関数が用意されている。その中で、図 9 のような歯車を描画する方法を説明する。

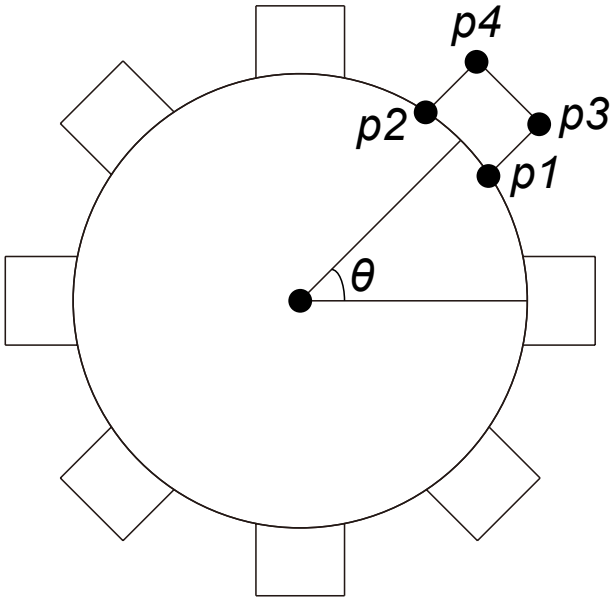

図 9　歯車の描画

ここで、歯の中心までの角度を θ, 歯車の半径を r, 歯の長さを l, 歯数を n とすると、p_1〜p_4 の座標は、それぞれ式 1〜4 のようになる。

$$p_1 = \left(r * \cos\left(\theta - \frac{2\pi}{n}\right), r * \sin\left(\theta - \frac{2\pi}{n}\right)\right) \tag{1}$$

$$p_2 = \left(r * \cos\left(\theta + \frac{2\pi}{n}\right), r * \sin\left(\theta + \frac{2\pi}{n}\right)\right) \tag{2}$$

$$p_3 = \left(r * \cos\left(\theta - \frac{2\pi}{n}\right) + l\cos(\theta), r * \sin\left(\theta - \frac{2\pi}{n}\right) + l\sin(\theta)\right) \tag{3}$$

$$p_4 = \left(r * \cos\left(\theta + \frac{2\pi}{n}\right) + l\cos(\theta), r * \sin\left(\theta + \frac{2\pi}{n}\right) + l\sin(\theta)\right) \tag{4}$$

このような計算を `drawGear`, `drawKana`, `drawGangi`, `drawPendulum` 内で行うことにより、歯車を描画している。

9.4 歯車の回転

見かけ上はそれぞれの歯車が噛み合っているように見えるが、内部的にはそれぞれが独立して回転している。それぞれの歯車の半径、ギア比、角速度などは、`Display` 関数内で設定している。これらの数値を調整することで、それぞれのパーツが連動して動作しているように見える。

9.5 画像表示

画像表示は、shape.c 内の `putSprite` 関数で行っている。glpng の `pngBind` 関数で読み込んだ画像を引数に渡すことにより、指定した座標に任意のサイズで画像を表示することができる。今回は、文字盤を画像として表示した。

9.6 make について

make とは、ビルド作業を自動化するプログラムである。Makefile 内にファイルの依存関係などビルドのルールを記述しておくことで、`make` コマンドを実行するだけでビルドすることができる。今回は、ソースファイルの変更に加え、アイコンの画像ファイルとリソースファイルが書き換えられた際も再ビルドが行われるように依存関係を記述した。

10 感想

OpenGL のライブラリである GLUT を用いて、アナログ時計を描画するプログラムを作成した。図形を描画する処理を汎用的な関数にまとめることで、完結にコードを記述できた。また、図形だけでなく、glpng を用いて png 画像を読み込み表示することにより、リッチなインターフェイスを実現できた。

参考文献

[1] 伊藤祥一, “Springs of C 楽しく身につくプログラミング”, 森北出版株式会社, 2017, pp. 109–110.
[2] “Pendulum clock”, Wikipedia, `https://en.wikipedia.org/wiki/Pendulum_clock`, 参照 2019/12/5.