計算機プログラミング'95 ＃6

久野　靖*

1995.10.11

1　ウィンドウプログラミング＋α

1.1　復習: イベントドリブン

まず、計算機科学基礎で取り上げた X-Window の例題プログラムを再掲する (ちょっとだけ改良してあるが)。

```
/* t61.c --- very simple X client */

#include <X11/Xlib.h>

main() {
  Display *disp = XOpenDisplay(NULL);
  Screen *scr = DefaultScreenOfDisplay(disp);
  Window root = DefaultRootWindow(disp);
  unsigned long black = BlackPixelOfScreen(scr);
  unsigned long white = WhitePixelOfScreen(scr);
  Window mw = XCreateSimpleWindow(disp,root,100,100,400,200,2,black,whi
te);
  XSelectInput(disp, mw, ButtonPressMask|KeyPressMask|ExposureMask);
  XMapWindow(disp, mw);
  while(1) {
    XEvent ev;
    XNextEvent(disp, &ev);
    if(ev.type == KeyPress)
      exit(0);
    else if(ev.type == Expose)
      draw(disp, mw, 20, 20);
    else if(ev.type == ButtonPress)
      draw(disp, mw, ev.xbutton.x, ev.xbutton.y);
  }
}

draw(Display *disp, Window mw, int x, int y) {
  GC dgc = DefaultGC(disp, 0);
```

*筑波大学大学院経営システム科学専攻

```
    XFillArc(disp, mw, dgc, x-20, y-20, 40, 40, 0, 360*64);
  }
```

なお、これを動かすには次のようにする。

```
  % export LD_LIBRARY_PATH=/usr/local/X11R6/lib  ← 1 回やればよい
  % gcc t61.c -I/usr/local/X11R6/include -lX11 -lnsl -lsocket
```

で、計算機科学基礎の練習問題では「上に別の窓を重ねても黒丸が復活できるようにせよ」ということだったが…やりましたか?

練習 1 t61.c をコピーしてきて動かせ。

練習 2 同じ練習問題を再度やってみよ。

1.2 黒丸を移動可能にする

次に、画面上の「もの」を直接操作 (direct manipulation) するための基本技術を見てみよう。基本的な要件は次の通り。

1. ボタンを押した状態でのマウスの移動と、ボタン離しとを通知してもらうようにする。
2. ボタンを押したときに、「どの黒丸の上で押したか」を調べる (picking)。
3. マウス移動イベントが来るごとに、画面をマウスに追従して更新。
4. ボタンを離したらそこでおしまい。

1 は、単に XSelectInput でマスクを書き足せばよい。2 は、順番に探せばよい。見つかったらそれを変数に覚える。一番むずかしいのは 3 の画面更新 (毎回クリアして描き直しているとちらちらして汚い)。ここではラバーバンドと呼ばれる技法を使う。基本的なアイデアは、反転演算で画面に描くこと。反転だと 2 回同じ操作をすると元に戻るので、描いたものを消して新しい位置に描くのが簡単。これを実現するには、GC(graphic context、描画操作の特性をまとめた「ペン先」) としてラバーバンド用のものをあらかじめ用意しておく。4 の「おしまい」では、本当に元の黒丸を消して移動するので、画面をクリアして再描画している。

```
  /* t63.c --- make the circle movable */

  #include <X11/Xlib.h>

  struct obj {
    int x, y; } objs[1000];
  int ouse = 0;

  GC rubbergc;

  main() {
    int moving = -1;
    Display *disp = XOpenDisplay(NULL);
    Screen *scr = DefaultScreenOfDisplay(disp);
    Window root = DefaultRootWindow(disp);
    unsigned long black = BlackPixelOfScreen(scr);
```

```
    unsigned long white = WhitePixelOfScreen(scr);
    Window mw = XCreateSimpleWindow(disp,root,100,100,400,200,2,black,whi
te);
    { XGCValues gcv; gcv.function = GXinvert; gcv.line_width = 3;
      rubbergc = XCreateGC(disp, mw, GCFunction|GCLineWidth, &gcv); }
    XSelectInput(disp, mw, ButtonPressMask|KeyPressMask|ExposureMask
      |ButtonMotionMask|ButtonReleaseMask);
    XMapWindow(disp, mw);
    while(1) {
      XEvent ev;
      XNextEvent(disp, &ev);
      if(ev.type == KeyPress)
        exit(0);
      else if(ev.type == Expose) {
        drawall(disp, mw);
      }
      else if(ev.type == ButtonPress) {
        if(ev.xbutton.button == Button1) {
          objs[ouse].x = ev.xbutton.x; objs[ouse].y = ev.xbutton.y; ++ous
e;
          draw(disp, mw, ev.xbutton.x, ev.xbutton.y); }
        else if(ev.xbutton.button == Button2) {
          int i = moving = lookup(ev.xbutton.x, ev.xbutton.y);
          if(i >= 0) drawrubber(disp, mw, objs[i].x, objs[i].y); }
      }
      else if(ev.type == MotionNotify) {
        if(moving >= 0) {
          int i = moving;
          drawrubber(disp, mw, objs[i].x, objs[i].y);
          objs[i].x = ev.xbutton.x; objs[i].y = ev.xbutton.y;
          drawrubber(disp, mw, objs[i].x, objs[i].y); }
      }
      else if(ev.type == ButtonRelease) {
        if(moving >= 0) {
          XClearWindow(disp, mw); drawall(disp, mw); moving = -1; }
      }
    }
  }

  lookup(int x, int y) {
    int i;
    for(i = 0; i < ouse; ++i)
      if(near(objs[i].x, objs[i].y, x, y, 20)) return i;
    return -1;
  }
```

```
near(int x0, int y0, int x1, int y1, int d) {
  return (x0-x1)*(x0-x1) + (y0-y1)*(y0-y1) <= d*d;
}

drawall(Display *disp, Window mw) {
  int i;
  for(i = 0; i < ouse; ++i)
    draw(disp, mw, objs[i].x, objs[i].y);
}

draw(Display *disp, Window mw, int x, int y) {
  GC dgc = DefaultGC(disp, 0);
  XFillArc(disp, mw, dgc, x-20, y-20, 40, 40, 0, 360*64);
}

drawrubber(Display *disp, Window mw, int x, int y) {
  XDrawArc(disp, mw, rubbergc, x-20, y-20, 40, 40, 0, 360*64);
}
```

練習 3 このまま動かせ。

練習 4 動かし初めのところが少し変だがわかるか? これを「なめらかに」するにはどうしたらいいか? 直してみよ。

練習 5 ラバーバンドをやめて毎回クリア+描画にしてみよ。

練習 6 データ構造に「半径」を増やして、新しくできるものほど 1 ドットずつ半径が増すようにしてみよ。

練習 7 右ボタンを押して動かした場合、黒丸の半径が変化させられるようにしてみよ。もちろん、ラバーバンドを使う。

1.3 受動的なデータ構造から抽象データ型へ

まず、練習 7 の回答例から見てみる。

```
/* t64.c --- make the circle movable and resizable */

#include <X11/Xlib.h>
#include <math.h>

struct obj {
  int x, y, r; } objs[1000];
int ouse = 0;

GC rubbergc;

main() {
  int moving = -1;
  int resizing = -1;
  int radius = 20;
  Display *disp = XOpenDisplay(NULL);
  Screen *scr = DefaultScreenOfDisplay(disp);
  Window root = DefaultRootWindow(disp);
  unsigned long black = BlackPixelOfScreen(scr);
  unsigned long white = WhitePixelOfScreen(scr);
  Window mw = XCreateSimpleWindow(disp,root,100,100,400,200,2,black,whi
te);
  { XGCValues gcv; gcv.function = GXinvert; gcv.line_width = 3;
    rubbergc = XCreateGC(disp, mw, GCFunction|GCLineWidth, &gcv); }
  XSelectInput(disp, mw, ButtonPressMask|KeyPressMask|ExposureMask
    |ButtonMotionMask|ButtonReleaseMask);
  XMapWindow(disp, mw);
  while(1) {
    XEvent ev;
    XNextEvent(disp, &ev);
    if(ev.type == KeyPress)
      exit(0);
    else if(ev.type == Expose) {
      drawall(disp, mw);
    }
    else if(ev.type == ButtonPress) {
      if(ev.xbutton.button == Button1) {
        objs[ouse].x = ev.xbutton.x; objs[ouse].y = ev.xbutton.y;
        objs[ouse].r = ++radius; ++ouse;
        draw(disp, mw, ev.xbutton.x, ev.xbutton.y, radius); }
      else if(ev.xbutton.button == Button2) {
        int i = moving = lookup(ev.xbutton.x, ev.xbutton.y);
        if(i >= 0) drawrubber(disp, mw, objs[i].x, objs[i].y, objs[i].r
```

```
); }
        else if(ev.xbutton.button == Button3) {
          int i = resizing = lookup(ev.xbutton.x, ev.xbutton.y);
          if(i >= 0) {
            objs[i].r = distance(objs[i].x, objs[i].y, ev.xbutton.x, ev.x
button.y);
            drawrubber(disp, mw, objs[i].x, objs[i].y, objs[i].r); }
        }
      }
      else if(ev.type == MotionNotify) {
        if(moving >= 0) {
          int i = moving;
          drawrubber(disp, mw, objs[i].x, objs[i].y, objs[i].r);
          objs[i].x = ev.xbutton.x; objs[i].y = ev.xbutton.y;
          drawrubber(disp, mw, objs[i].x, objs[i].y, objs[i].r); }
        if(resizing >= 0) {
          int i = resizing;
          drawrubber(disp, mw, objs[i].x, objs[i].y, objs[i].r);
          objs[i].r = distance(objs[i].x, objs[i].y, ev.xbutton.x, ev.xbu
tton.y);
          drawrubber(disp, mw, objs[i].x, objs[i].y, objs[i].r); }
      }
      else if(ev.type == ButtonRelease) {
        if(moving >= 0 || resizing >= 0) {
          XClearWindow(disp, mw); drawall(disp, mw); moving = resizing =
-1; }
      }
    }
  }

  lookup(int x, int y) {
    int i;
    for(i = 0; i < ouse; ++i)
      if(near(objs[i].x, objs[i].y, x, y, objs[i].r)) return i;
    return -1;
  }

  near(int x0, int y0, int x1, int y1, int d) {
    return (x0-x1)*(x0-x1) + (y0-y1)*(y0-y1) <= d*d;
  }

  distance(int x0, int y0, int x1, int y1) {
    return (int)sqrt((x0-x1)*(x0-x1) + (y0-y1)*(y0-y1));
  }

  drawall(Display *disp, Window mw) {
```

```
    int i;
    for(i = 0; i < ouse; ++i)
      draw(disp, mw, objs[i].x, objs[i].y, objs[i].r);
  }

  draw(Display *disp, Window mw, int x, int y, int r) {
    GC dgc = DefaultGC(disp, 0);
    XFillArc(disp, mw, dgc, x-r, y-r, 2*r, 2*r, 0, 360*64);
  }

  drawrubber(Display *disp, Window mw, int x, int y, int r) {
    XDrawArc(disp, mw, rubbergc, x-r, y-r, 2*r, 2*r, 0, 360*64);
  }
```

かなり「限界」という感じがしませんか? このように、受動的なデータ構造を使っている場合、機能を増やしていくとデータ構造に関する「約束ごと」がどんどん増え、それをアクセスする手続き側で約束を破らないようにするのがたいへんになってくる。

このような問題に対する回答として、「抽象データ型」(Abstract Data Types、ADT) がある。ADT は前にやったモジュールによく似ているが、モジュールはあくまでもデータが 1 式だったのに対し、ADT ではデータ (インスタンスとか実体とも呼ぶ) が N 個あってよい。ここでは「1 つの黒丸」を 1 つの ADT としてみる。

```
  /* t65.c --- ADT circle */

  #include <X11/Xlib.h>
  #include <math.h>

  typedef struct obj {
    Display *d; Window w; int x, y, r; } *obj_t;
  obj_t objs[1000], obj_create(), lookup();
  int ouse = 0;

  GC rubbergc;

  main() {
    obj_t moving = 0, resizing = 0;
    int radius = 20;
    Display *disp = XOpenDisplay(NULL);
    Screen *scr = DefaultScreenOfDisplay(disp);
    Window root = DefaultRootWindow(disp);
    unsigned long black = BlackPixelOfScreen(scr);
    unsigned long white = WhitePixelOfScreen(scr);
    Window mw = XCreateSimpleWindow(disp,root,100,100,400,200,2,black,whi
te);
    { XGCValues gcv; gcv.function = GXinvert; gcv.line_width = 3;
      rubbergc = XCreateGC(disp, mw, GCFunction|GCLineWidth, &gcv); }
    XSelectInput(disp, mw, ButtonPressMask|KeyPressMask|ExposureMask
```

```
      |ButtonMotionMask|ButtonReleaseMask);
    XMapWindow(disp, mw);
    while(1) {
      XEvent ev;
      XNextEvent(disp, &ev);
      if(ev.type == KeyPress)
        exit(0);
      else if(ev.type == Expose) {
        drawall();
      }
      else if(ev.type == ButtonPress) {
        if(ev.xbutton.button == Button1) {
          objs[ouse] = obj_create(disp, mw, ev.xbutton.x, ev.xbutton.y, +
+radius);
          obj_draw(objs[ouse]); ++ouse; }
        else if(ev.xbutton.button == Button2) {
          moving = lookup(ev.xbutton.x, ev.xbutton.y);
          if(moving) obj_drawrubber(moving); }
        else if(ev.xbutton.button == Button3) {
          resizing = lookup(ev.xbutton.x, ev.xbutton.y);
          if(resizing) {
            obj_resize(resizing, ev.xbutton.x, ev.xbutton.y);
            obj_drawrubber(resizing); }
        }
      }
      else if(ev.type == MotionNotify) {
        if(moving) {
          obj_drawrubber(moving);
          obj_move(moving, ev.xbutton.x, ev.xbutton.y);
          obj_drawrubber(moving); }
        if(resizing) {
          obj_drawrubber(resizing);
          obj_resize(resizing, ev.xbutton.x, ev.xbutton.y);
          obj_drawrubber(resizing); }
      }
      else if(ev.type == ButtonRelease) {
        if(moving || resizing) {
          XClearWindow(disp, mw); drawall(); moving = resizing = 0; }
      }
    }
  }

  obj_t lookup(int x, int y) {
    int i;
    for(i = 0; i < ouse; ++i)
      if(obj_near(objs[i], x, y)) return objs[i];
```

```
  return 0;
}

near(int x0, int y0, int x1, int y1, int d) {
  return (x0-x1)*(x0-x1) + (y0-y1)*(y0-y1) <= d*d;
}

distance(int x0, int y0, int x1, int y1) {
  return (int)sqrt((x0-x1)*(x0-x1) + (y0-y1)*(y0-y1));
}

drawall() {
  int i;
  for(i = 0; i < ouse; ++i) obj_draw(objs[i]);
}

obj_t obj_create(Display *disp, Window mw, int x, int y, int r) {
  obj_t o = (struct obj*)malloc(sizeof(struct obj));
  o->d = disp; o->w = mw; o->x = x; o->y = y; o->r = r; return o;
}

obj_draw(obj_t o) {
  int x = o->x, y = o->y, r = o->r;
  GC dgc = DefaultGC(o->d, 0);
  XFillArc(o->d, o->w, dgc, x-r, y-r, 2*r, 2*r, 0, 360*64);
}

obj_drawrubber(obj_t o) {
  int x = o->x, y = o->y, r = o->r;
  XDrawArc(o->d, o->w, rubbergc, x-r, y-r, 2*r, 2*r, 0, 360*64);
}

obj_near(obj_t o, int x, int y) {
  return near(o->x, o->y, x, y, o->r);
}

obj_move(obj_t o, int x, int y) {
  o->x = x; o->y = y;
}

obj_resize(obj_t o, int x, int y) {
  o->r = distance(o->x, o->y, x, y);
}
```

練習 8 そのまま動かせ。

練習 9 動かし初めのずれをなくすには、どういう ADT インタフェースがあればいいか。設計し、

実装してみよ。

練習 10 形を黒丸でなく黒四角に変更してみよ。

練習 11 黒丸と黒四角の両方が使えるようにしてみよ。たとえばシフトキーを押しながらクリックすると四角ができる、というふうに。

なお、四角の場合には次の描画ルーチンを使えばよい。

```
XFillRectangle(disp, win, GC, x, y, width, height);
XDrawRectangle(disp, win, GC, x, y, width, height);
```

また、練習 11 のためには、キーを押したらすぐ終わりでは困るので、キーイベントの処理部分を次のようにする。

```
char buf[20];
if(XLookupString(&ev, buf, 20, 0, 0) == 1 && buf[0] == 'q') exit(0);
```

つまり、XLookupString でイベントを文字列に変換して、長さ 1 の「q」の場合だけおしまいにする。あとは、マウスボタンイベントのところで Shift が押されているかどうか調べるには

```
ev.xbutton.state&ShiftMask
```

で見ることができる。