インターフェイスの街角 (34) – ピクセルの限界を超えて

増井 俊之

ビットマップ・ディスプレイでは、画面上に表示可能なピクセル（ドット）の数が決まっており、それより多くのピクセルは表示できません。640 × 480 ピクセルのディスプレイをもつ計算機を例に考えてみましょう。たとえば、このディスプレイの画面上に縦線を引く場合には、一般的な方法で白と黒の線を交互に描画すると最大でも 320 本の線しか描けないことになります。また、GUI 部品でスクロールバーを動かすときも、ノブを表示できる位置は 480 カ所しかないので、それより細かい制御は難しくなります。

一般にひろく使われている GUI 部品では、このような制約はあって当然と考えられているようです。しかし、表示装置や入力装置のハードウェアの解像度が決まっていても、ソフトウェア側で工夫すれば擬似的に解像度を高められる場合もあり、そのための各種の手法が考案されています。今回は、これらの手法のいくつかを紹介します。

最近のデスクトップ計算機で利用するディスプレイは高解像度のものが多いので、この問題はそれほど重要とはいえないかもしれません。しかし、PDA などの携帯型計算機では、その特性から解像度を飛躍的に高めるのは難しく、このような手法を探求する意義はあると思います。

表示装置の解像度と入力装置の解像度は、かならずしも同じとはかぎりません。しかし、GUI では入力装置の操作のフィードバックを画面上に表示する処理が基本になっているため、両者の解像度はおおむね同等に設定されていることが多いようです。マウスの解像度がいくら細かくても、マウスの動きが画面上のカーソルの動きなどに反映されなければユーザーが解像度を認識することはできないからです。

図 1　ジャギーが目立つ斜線

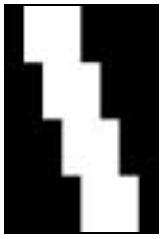
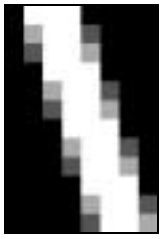


図 2　アンチ・エイリアシング

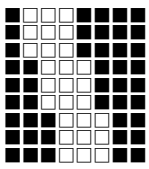


表示解像度の向上

解像度の低い表示装置上に斜線を表示しようとすると、線の縁がギザギザになって見栄えが悪くなることがあります。また、解像度の低い表示装置上でゆっくり動く物体を表示すると、動きがぎくしゃくして滑らかに表示できません。解像度の低さに起因するこのような表示上の問題を**エイリアス**と呼びます。前者のようなギザギザ（ジャギー）を除去するために、アンチ・エイリアシングという手法がよく使われています。

アンチ・エイリアシング

幅 3 ピクセル、傾きが −3 の白い直線をモノクロ・ディスプレイで表現するには、通常は次のようにピクセルを並べます。



解像度の低い表示装置でこのような緩い傾きの直線を表示すると、とくにジャギーが目立ちます（図 1）。

モノクロの表示装置でエイリアスを除去するのは困難ですが、ピクセルごとに輝度レベル（階調）の設定が可能な表示装置であれば、図 2 のように中間階調のピクセルを使ってジャギーの度合いを減らせます。このように、中間階調をもつピクセルを用いてジャギーを減らす手法が**アンチ・エイリアシング**です。

最近の CG の分野ではアンチ・エイリアシングはほぼ常識といってもよく、安価なゲーム機でもハードウェアで実装されているものがあります。CG 以外の一般的なアプリケーションでも、Adobe の Acrobat や Microsoft の PowerPoint、あるいは T-Time のような電子ブックリーダーなどではアンチ・エイリアシングを用いたフォントが使われ、画面には読みやすい文字が表示されます。

アンチ・エイリアシングは、画面の解像度を高めるわけではありません。しかし、小さなサイズのフォントでも比較的きれいに見えるため、より多くの文字を表示できるという特徴があります。

Information Mural

アンチ・エイリアシングの手法を情報視覚化に応用すれば、より多くの情報を画面に表示できます。

たとえば 100 ドットのピクセル幅をもつ領域に棒グラフを表示する場合、各ピクセル幅にデータの 1 エントリを対応させると 100 エントリぶんのグラフしか表示できません。しかし、複数のエントリを 1 ピクセル幅に描画すれば、より多くの情報が得られることもあります。Information Mural[3] は、アンチ・エイリアシングの手法を用いて小さな領域に大量の情報を提示するための手法です。

図 3 は、Information Mural を用いてオブジェクト間のメッセージ通信を視覚化したものです[1]。このグラフでは X 軸が時間に、Y 軸がクラスに対応しており、縦線がクラス間のメッセージを表現しています。この例では、合計で 50,000 個のメッセージが 500 ピクセル幅の領域上に重ね書きされています。個々のメッセージの内容は分かりませんが、どのクラス間でどのようにメッセージ通信がおこなわれているかの概要は把握できるでしょう。

重ね書きを使わず、すべてのメッセージを白い線で描画すると、図 4 のようにほぼすべての領域が真っ白に塗り潰されてしまい、有用な情報は得られません。

図 3　Information Mural によるメッセージの視覚化

図 4　Information Mural を使用しない場合

アンチ・エイリアシング手法は、CG の分野では画像を美しく表示するために使われます。その同じ手法が、かならずしも画像の美しさを追求していない情報視覚化の分野に応用されている点が興味深いところです。UNIX 上のアプリケーションでは、この手法を使えるものはまだ少数のようですが、可能であればつねに使う価値のある技術といえます。

ClearType とサブピクセル法

Microsoft は、1998 年末に「液晶ディスプレイに表示する文字の解像度を 3 倍 (300%) に向上させる、まったく新しい ClearType 技術を開発した」と発表しました[2]。現在のところ、ClearType の詳細は明らかにされていませんが、発表資料をもとにその内容を類推した Web ページがいくつかあり[3]、ネットワーク上でも議論がさかんにおこなわれています。

これらの情報をもとに推測すると、ClearType は、カラー液晶のハードウェアの特徴と人間の視覚特性を利用して液晶表示装置の見かけ上の解像度を高める技術のようです。現時点では、「3 倍に向上させる」という表現から判断して、カラー液晶のピクセルの RGB 領域を別々に使って高解像度を得る技術だろうという説が有力です。

カラー液晶のピクセルは、図 5 のように RGB の縦長の矩形サブピクセルの並びで構成されています。事実、液晶画面をルーペで拡大するとこのような並びが見えます。ただし、ピクセルが十分に小さいため、普通に見ているか

1 http://www.cc.gatech.edu/gvu/softviz/infoviz/information_mural.html

2 http://www.microsoft.com/typography/cleartype/
http://www.zdnet.co.jp/news/9811/16/gates.html

3 http://grc.com/cleartype.htm
http://www.hirax.net/dekirukana/cleartype/
など。前者のページでは、液晶画面上でフォントを美しく表示するためのデモプログラムも公開されています。

図 5　液晶ディスプレイのサブピクセルの並び

```
RGBRGBRGBRGBRGBRGBRGBRGB
RGBRGBRGBRGBRGBRGBRGBRGB
RGBRGBRGBRGBRGBRGBRGBRGB
RGBRGBRGBRGBRGBRGBRGBRGB
RGBRGBRGBRGBRGBRGBRGBRGB
```

ぎりでは RGB を別々に認識することはできません。

通常の用途では、RGB のサブピクセルの組を 1 ピクセルとして扱っているわけですが、これを 3 個のサブピクセルとして独立に扱えれば解像度が 3 倍になるかもしれません。

図 1 の斜線を図 5 と同様なサブピクセル表現で記述すると、

```
RGBRGBRGB
RGBRGBRGB
RGBRGBRGB
   RGBRGBRGB
   RGBRGBRGB
   RGBRGBRGB
      RGBRGBRGB
      RGBRGBRGB
      RGBRGBRGB
```

のようになります。これを PPM (Portable PixMap) 形式で表現すると以下のようになります。

```
# line1.ppm
P3
8 12
9
0 0 0  9 9 9  9 9 9  9 9 9  0 0 0  0 0 0  0 0 0  0 0 0
0 0 0  9 9 9  9 9 9  9 9 9  0 0 0  0 0 0  0 0 0  0 0 0
0 0 0  9 9 9  9 9 9  9 9 9  0 0 0  0 0 0  0 0 0  0 0 0
0 0 0  0 0 0  9 9 9  9 9 9  9 9 9  0 0 0  0 0 0  0 0 0
0 0 0  0 0 0  9 9 9  9 9 9  9 9 9  0 0 0  0 0 0  0 0 0
0 0 0  0 0 0  9 9 9  9 9 9  9 9 9  0 0 0  0 0 0  0 0 0
0 0 0  0 0 0  0 0 0  9 9 9  9 9 9  9 9 9  0 0 0  0 0 0
0 0 0  0 0 0  0 0 0  9 9 9  9 9 9  9 9 9  0 0 0  0 0 0
0 0 0  0 0 0  0 0 0  9 9 9  9 9 9  9 9 9  0 0 0  0 0 0
0 0 0  0 0 0  0 0 0  0 0 0  9 9 9  9 9 9  9 9 9  0 0 0
0 0 0  0 0 0  0 0 0  0 0 0  9 9 9  9 9 9  9 9 9  0 0 0
```

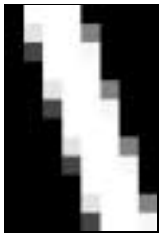
一方、図 2 のようにアンチ・エイリアシングを施した斜線は以下のように表現できるでしょう（輝度に応じて R、r、. と表記しています）。

```
RGBRGBRGB
rgbRGBRGB...
...RGBRGBrgb
   RGBRGBRGB
   rgbRGBRGB...
   ...RGBRGBrgb
      RGBRGBRGB
      rgbRGBRGB...
      ...RGBRGBrgb
```

これを PPM 形式で表現すると以下のようになります。

```
# line2.ppm
P3
8 12
9
0 0 0  9 9 9  9 9 9  9 9 9  0 0 0  0 0 0  0 0 0  0 0 0
0 0 0  6 6 6  9 9 9  9 9 9  3 3 3  0 0 0  0 0 0  0 0 0
```

図 6　サブピクセル法による直線



```
0 0 0  3 3 3  9 9 9  9 9 9  6 6 6  0 0 0  0 0 0  0 0 0
0 0 0  0 0 0  9 9 9  9 9 9  9 9 9  0 0 0  0 0 0  0 0 0
0 0 0  0 0 0  6 6 6  9 9 9  9 9 9  3 3 3  0 0 0  0 0 0
0 0 0  0 0 0  3 3 3  9 9 9  9 9 9  6 6 6  0 0 0  0 0 0
0 0 0  0 0 0  0 0 0  9 9 9  9 9 9  9 9 9  0 0 0  0 0 0
0 0 0  0 0 0  0 0 0  6 6 6  9 9 9  9 9 9  3 3 3  0 0 0
0 0 0  0 0 0  0 0 0  3 3 3  9 9 9  9 9 9  6 6 6  0 0 0
0 0 0  0 0 0  0 0 0  0 0 0  9 9 9  9 9 9  9 9 9  0 0 0
0 0 0  0 0 0  0 0 0  0 0 0  6 6 6  9 9 9  9 9 9  3 3 3
0 0 0  0 0 0  0 0 0  0 0 0  3 3 3  9 9 9  9 9 9  6 6 6
```

サブピクセル法は、上に示したような通常のアンチ・エイリアシング手法を使用する代わりに、各ピクセル内の RGB のサブピクセルを用いて解像度を高める技術です。アンチ・エイリアシングではピクセル単位の輝度を制御しますが、サブピクセル法では以下のようにサブピクセル単位で輝度を制御します。

```
RGBRGBRGB
 GBRGBRGBR
  BRGBRGBRG
   RGBRGBRGB
    GBRGBRGBR
     BRGBRGBRG
      RGBRGBRGB
       GBRGBRGBR
        BRGBRGBRG
```

上から 3 行目の左端のピクセルは青 (B) になっています。白色の直線を表現するために青のピクセルを使うのはおかしいと思うかもしれませんが、実際にこのような色指定をおこなうと縁が滑らかできれいな白い斜線が得られます（図 6）。この画像を PPM 形式で表現すると以下のようになります。

```
# line3.ppm
P3
8 12
9
#R G B  R G B  R G B  R G B  R G B  R G B  R G B  R G B
 0 0 0  9 9 9  9 9 9  9 9 9  0 0 0  0 0 0  0 0 0  0 0 0
 0 0 0  0 9 9  9 9 9  9 9 9  9 0 0  0 0 0  0 0 0  0 0 0
 0 0 0  0 0 9  9 9 9  9 9 9  9 9 0  0 0 0  0 0 0  0 0 0
 0 0 0  0 0 0  9 9 9  9 9 9  9 9 9  0 0 0  0 0 0  0 0 0
 0 0 0  0 0 0  0 9 9  9 9 9  9 9 9  9 0 0  0 0 0  0 0 0
 0 0 0  0 0 0  0 0 9  9 9 9  9 9 9  9 9 0  0 0 0  0 0 0
 0 0 0  0 0 0  0 0 0  9 9 9  9 9 9  9 9 9  0 0 0  0 0 0
 0 0 0  0 0 0  0 0 0  0 9 9  9 9 9  9 9 9  9 0 0  0 0 0
 0 0 0  0 0 0  0 0 0  0 0 9  9 9 9  9 9 9  9 9 0  0 0 0
 0 0 0  0 0 0  0 0 0  0 0 0  9 9 9  9 9 9  9 9 9  0 0 0
 0 0 0  0 0 0  0 0 0  0 0 0  0 9 9  9 9 9  9 9 9  9 0 0
 0 0 0  0 0 0  0 0 0  0 0 0  0 0 9  9 9 9  9 9 9  9 9 0
```

白黒の誌面では比較は不可能ですが、液晶ディスプレイ上で図 1 と図 2、図 6 の 3 者を比較すると、明らかに図 6 が一番きれいな直線に見えます。

サブピクセル法の限界

サブピクセル法は興味深い技術ですが、どんな場面でも有効なわけではありません。前例の場合はたしかに効果がありますが、以下の問題があります。

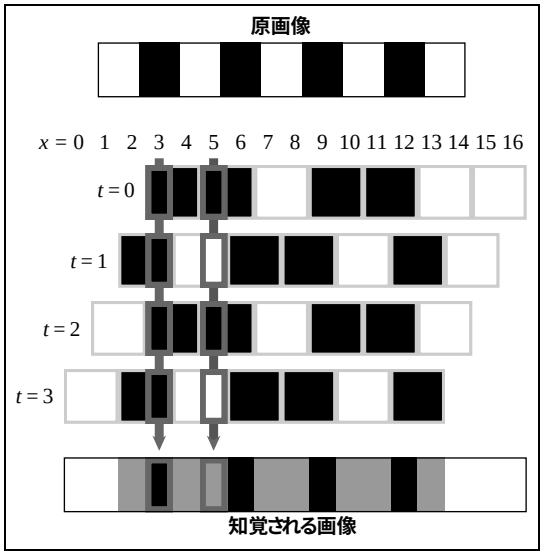
- カラー画像には適用できない
  モノクロ画像の表示には効果がありますが、各色に対応した画素の数は変わらないので、単色や暗い色の画像では効果がありません。
- 細い線の両端がにじんで見える
  サブピクセル法で 4/3 ピクセルの幅をもつ白い縦線を引くと、以下のサブピクセルが ON になります。

```
GBRG
GBRG
GBRG
GBRG
```

  液晶ディスプレイ上でこの画像を確認すると、緑の要素がほかよりも多いため、左右の端が緑がかって見えてしまいます。輝度を左右に拡散させれば、このような望ましくない発色現象を低減することはできますが、画像はぼやけてしまいます。
- 水平方向の解像度にしか効果がない
  通常、液晶ディスプレイのサブピクセルは横方向に並んでいるため、水平方向の解像度しか向上しません。多くの漢字は横線の密度のほうが縦線よりも細かいので、サブピクセル法を用いて漢字を表示するときは液晶ディスプレイを横にしたほうが効果的です。
- 液晶ディスプレイ以外では効果がない
  サブピクセル法では、水平方向に並んだ RGB サブピクセルを正確に ON/OFF する必要があります。CRT はピクセル位置の正確さに欠けるため、目立った効果は得られません。

ClearType がサブピクセル法と類似した技術かどうかは不明ですが、この手法に似たものだとすると、たしかに有用ではあるものの、〝まったく新しい技術〟とか〝解像度を 3 倍にする〟というのはちょっと言いすぎかもしれません。

図 7　6 ピクセル幅で 4 本の線を表示



PixelDoubler

アンチ・エイリアシングやサブピクセル法のような技術を使うと、たしかに画面表示は美しくなりますが、ハードウェアの解像度を超える表示が可能なわけではありません。しかし、動画を利用すれば、ハードウェアの解像度以上の表示が得られる場合もあります。

テレビで動画がきれいに見えていても、画面を停止させるとノイズや解像度が気になることがあります。これは、画面が動いているとノイズがあまり気にならないということもありますが、動画では静止画より高い解像度が得られるのも一因です。

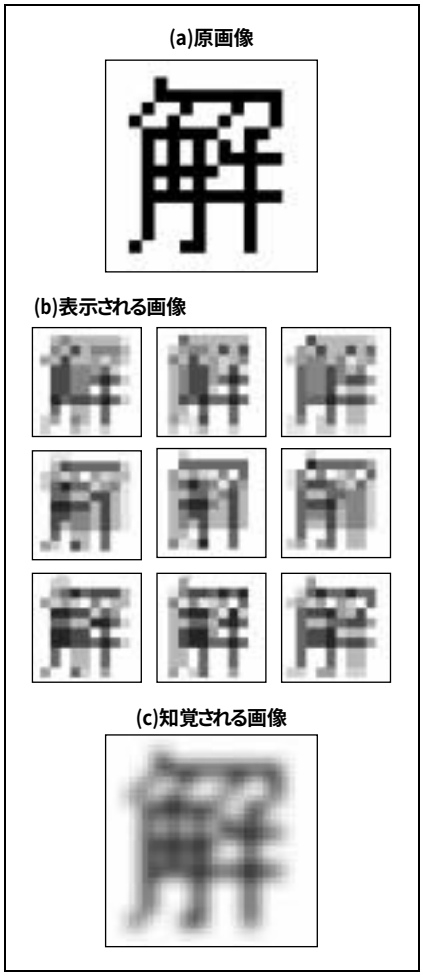
動画のほうが静止画より高い解像度が得られることを示すため、表示装置を移動させながら画面を変化させていき、6 ピクセル幅の領域に 4 本の線を表示する例をもとに説明します（図 7 もあわせて見てください）。

ディスプレイを 1 タイムスライスごとに半ドットずつ左に動かしながら、次のように表示していきます。

$t = 0$　■■□■■□□
$t = 1$　■□■■□■□
$t = 2$　□■■□■■□
　　……

このとき、$x = 3$ の位置では黒いピクセルが、$x = 5$ の位置では点滅するピクセルが表示されます。タイムスライスが十分に小さければ、ユーザーには $x = 5$ の位置には

図 8　ディスプレイを揺らすことによる高解像度表示

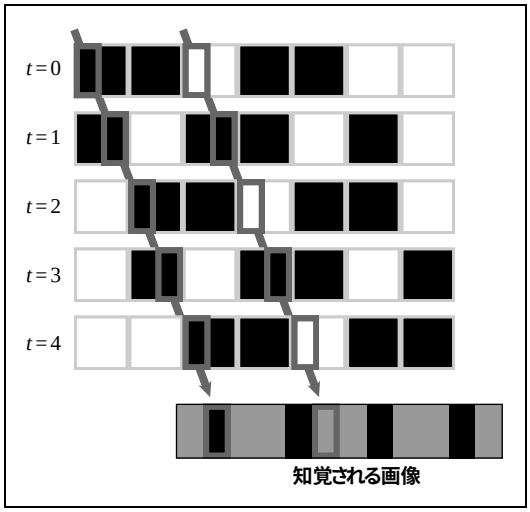


灰色のピクセルが表示されているように見えます。結果として、図 7 の最下行のように、ユーザーには 4 本の線が見えることになります。静止した 6 ピクセル幅の領域には 3 本しか線を表示できないので、ディスプレイを動かすことによって解像度が 33%向上したことになります。

同様の手法により、10 × 10 ピクセルの領域に 14 × 14 ドットのフォントを表示する手法を図 8 に示します。ここで、ディスプレイのピクセルは表示するフォントのピクセルより 50%大きいにもかかわらず、ディスプレイを揺らしながら表示を変化させていけば、フォントの細い線も図 8-c のように認識できます。

このように、ディスプレイを動かしたり揺らしたりすれば、装置の解像度以上の表示が得られます。

図 9　4 本の線が右に移動



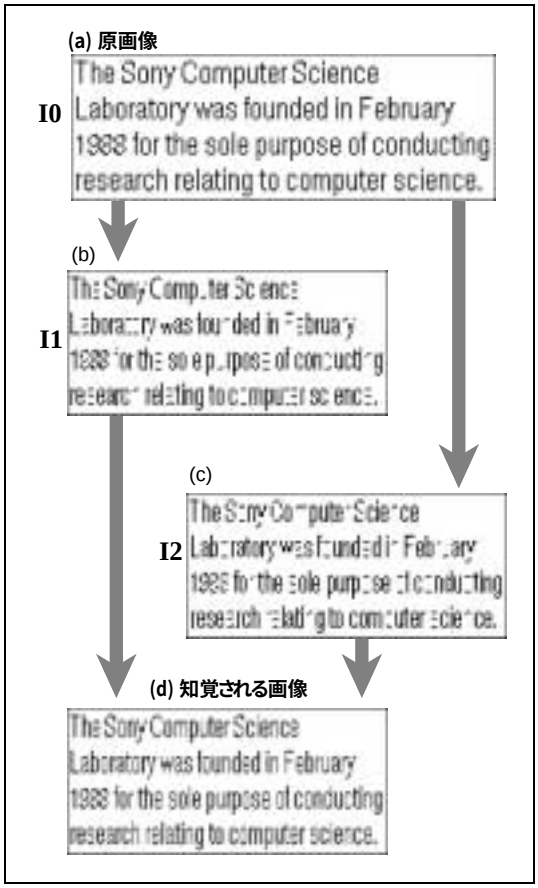
ディスプレイを動かさなくても、図 7 と同様の手法を使えば高解像度の表示が可能になります。図 7 で、表示装置を左に動かさずに静止させた場合の表示を図 9 に示します。ディスプレイが静止しているときは、パターンが変化しながら右に移動することになります。しかし、人間の目は移動に追随するため、ユーザーには図 7 と同様に 4 本の線が右に移動するように見えます。

この効果を利用して画像を移動させながら表示すれば、ディスプレイの解像度以上のピクセルをもつ画像を表示させることができます。私は、この手法を **PixelDoubler** と呼んでいます [5]。

図 10 は、ビットマップ画像の表示に PixelDoubler を応用した例です。2 種類の画像を交互にシフトさせながら表示させることで、実際の解像度より細かい画像が表示できます。

図 10-a が 226 × 76 ピクセルの原画像 I0 です。これをもとにして、2 種類の小さな（170 × 78 ピクセル）画像 I1（図 10-b）と I2（図 10-c）を用意します。PixelDoubler は、$t = 0$ において $(0, 0)$ の位置に I1 を表示し、$t = 1$ において $(0, 0)$ の位置に I2 を表示します。また $t = 2$ において $(-1, 0)$ の位置に I1 を表示し、$t = 3$ において $(-1, 0)$ の位置に I2 を表示します。このように、2 枚の画像をシフトしながら交互に表示し続けていくと、ユーザーには I0 と同じ解像度をもち、かつサイズが小さな画像が見えます（図 10-d）。I1 と I2 は I0 より少ない情報し

図 10　2 種類の画像を交互に表示させて高解像度を実現



かもっていませんが、これらを交互に表示していくとユーザーには I0 と同じ画像のように認識されます。

図 7 のように、画像 I1、I2 は原画像 I0 を低い解像度の穴から覗いて得られる画像に対応しています。I0 からの I1 と I2 の生成は、以下のアルゴリズムに従っておこなわれます（図 11）。

● 画像 I0（図 11-a）の各行について以下を実行する。

1. ピクセルを 6 倍に拡大する（図 11-b）。
2. 1 番目のギャップを始点とし、ピクセルを 8 ピクセルごとのブロックに分割する（図 11-c）。
3. 各ブロックについて■の数と□の数を比較し、その結果から I2 のピクセルの色を決定する。
4. 5 番目のギャップを始点とし、ピクセルを 8 ピクセルごとのブロックに分割する。（図 11-d）。
5. 各ブロックについて■の数と□の数を比較し、その結果から I1 のピクセルの色を決定する。

操作解像度の向上

画面の解像度が低くても、入力装置の解像度を上げることによって、より高度な操作がおこなえる可能性があります。たとえば、テキストを 20 行しか表示できないスクロール画面でも、入力装置が 10,000 行ぶんの解像度をもっていればスクロールしながら 10,000 行のテキストを自由に編集できるでしょう。この場合、表示装置の解像度の制約によってスクロールバーは入力装置の動きに追いつけないと思いますが、入力装置の状態はテキストに反映されるので、結果として制御が可能になります。

■ こういうことでしょうか？

通常、入力装置の解像度は表示装置のそれよりも高いのですが、一般的なウィンドウ・システムやツールキットでは入力と表示が 1 対 1 に対応することが前提となっています。したがって、両者の解像度が等しくなるように設定されているのが普通であり、その場合には画面の解像度を超える細かい制御はできません。たとえ入力装置の解像度が高くても、人間がマウスなどを正確に細かく操作するのは難しいため、あまり現実的ではありません。

■ 人間が操作するのが難しいということ？

ここで、表示装置も入力装置も 200 ドットの解像度しかない場合に 10,000 行のテキストを編集する状況を考えてみます[4]。このような装置では、画面に表示できるのはせいぜい 20 行程度です。画面上をドラッグして次画面を表示させるとすると、全体を眺めるには 500 回もの操作が必要になってしまいます。また、普通のスクロールバーによるページ移動では、スクロールバーを 1 ドット動かしただけでテキストが 50 行ぶんもスクロールしてしまうため、ひどく使いにくくなります。

デスクトップ計算機で一般的な GUI ツールはこういう状況には不向きで、微調整も可能な GUI ツールが必要になるでしょう。そこで、低解像度の入力装置でも細かい制御を可能にする、各種の**微調整インターフェイス**が考案されています。

操作の解像度の問題については、以下のような解決策が考えられます。

4 PalmPilot や PocketPC のように画面サイズに制限のある装置でも、メモリの増加によって大きなデータが扱えるようになりつつあります。したがって、今後はこういった状況が増えてくるでしょう。

図 11　原画像からの I1 と I2 の生成

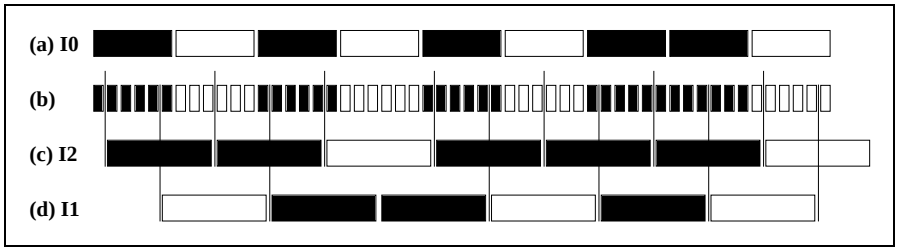
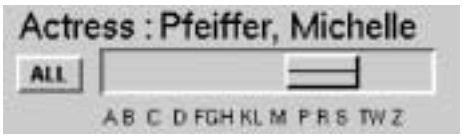


図 12　AlphaSlider



- 表示可能な量を制御する。
- 微調整と粗調整の両方の操作を用意する。
- 微調整用の新たな操作を利用する。
- 適応的な調整手法を利用する。

これらの手法について順番に解説します。

表示可能な量を制御する

一度に 10,000 行のデータ全体を扱うのではなく、ズーミングやフィルタリングによって表示量を制御する手法です。たとえば、なんらかの方法によってデータを 1/10 に絞れるのであれば、全体で 1,000 行だけ操作すればよいことになり、スクロールバーによる操作でも十分対応できます。

ズーミングやフィルタリングは大規模なデータを扱う場合の強力な武器になります。しかし、表示量の制御という操作は、ユーザーにはあまり馴染みがないために難しいと感じたり、どのレベルのデータを眺めているのかが分からなくなることがあります。

とはいえ、直感的で簡単なズーミング/フィルタリング操作の手法が開発されて普及すれば、この方式がもっとも有効かもしれません。

微調整/粗調整の両方の操作を用意する

これは、スクロールバーに複数のノブを用意しておきそれぞれに微調整と粗調整の機能を割り当てておく手法です。メリーランド大学で開発された AlphaSlider[1] は、この考えにもとづいた微調整スライダです。

一見したところ、AlphaSlider は普通のスライダと同様の外観をしていますが（図 12）、ノブの部分が上下に分かれており、それぞれが微調整/粗調整用になっています。粗調整部を操作する場合は、普通のスライダと同様にマウスの移動（＝カーソルの移動）にノブが追随します。一方、微調整部を操作するときはマウスの操作量とノブの移動量が一致せず、マウスを大きく動かしてもノブはわずかしか移動しません。結果として、エントリを 1 つずつ動かしていけるようになっています。

AlphaSlider は一般的なスライダを拡張した GUI ツールであり、既存の GUI しか使ったことのないユーザーでも抵抗なく利用できるでしょう。さらに、1 つのスライダで何万件もの項目が選択可能という点でも意義があります。

AlphaSlider の開発者の Christopher Ahlberg 氏は、これ以外にも興味深いインターフェイス・ツールをいろいろと開発しています。IVEE という会社[5]を設立し、AlphaSlider を含む各種の GUI ツールの販売をおこなっています。残念ながら、AlphaSlider はまだまだ一般的に知られていませんし、操作性にも若干難がありますが、微調整インターフェイスの 1 つとしてぜひ広まってほしいものです。

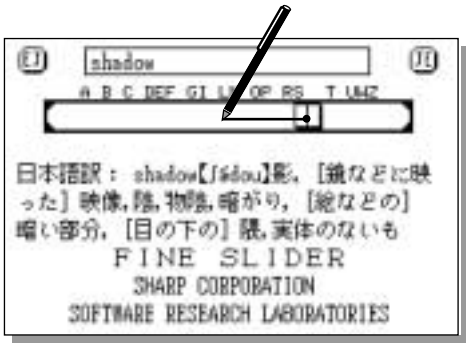
微調整用の新たな操作を利用

FineSlider

スライダやスクロールバーには、上下もしくは左右に微調整用の矢印ボタンが付いているものがあります。たいていは、これらのボタンを押すと表示が 1 エントリぶんだけ移動するようになっていますが、これではデータが大量にある場合は不便です。

5 http://www.ivee.com/

図 13　FineSlider による辞書検索の例



FineSlider[4] は、スクロールバーのなかのノブ以外の部分を使って画面スクロールをおこなう GUI 部品です。一般的なスクロールバーでは、ノブ以外の部分をクリックすると、ノブがその位置にジャンプしたり、一定量だけノブが移動したりします。一方、FineSlider でスクロールバーのノブ以外の部分をクリックすると、クリック位置とノブのあいだに〝ゴムひも〟のような表示が現れ、クリック位置からノブを引っ張るようにしてノブの位置を動かすことができます。ゴムひもは長さによって引っ張る強さが変わるので、ノブとマウスカーソルの位置が近い場合はノブはゆっくり移動し、遠ければ高速に移動します。ノブ上でマウスをクリックすると、通常のスクロールバーと同様にノブはマウスの動きに応じて移動します。

Popup Vernier

Popup Vernier[2][6] も、モード切替えによってスライダの微調整がおこなえる GUI 部品です。AlphaSlider や FineSlider では微調整モードでもスライダノブの表示は変わりませんが、Popup Vernier ではノブの周辺にバーニヤダイヤルのような目盛りが表示され、どのようにノブの微調整がおこなわれているかを認識できるようになっています（図 14）。

適応的な調整手法

以上のような手法では、微調整と粗調整をユーザーが明示的に切り替える必要がありますが、これらがユーザーの意図どおりに自動的に切り替わるようになっていればさらに便利でしょう。たとえば、ユーザーがノブを大きく動かしたときは粗調整、小さく動かしたときは微調整といったふうに切り替える方式が考えられます。

6 http://www-masuda.is.s.u-tokyo.ac.jp/~aya/works/pv-j.html

図 14　Popup Vernier

おわりに

今回は、ピクセルの限界を超えるための各種の技法について紹介しました。十分な解像度がある入出力装置を使うのであれば、これらの手法の多くは不要かもしれません。しかし、初めにも述べたように PDA やウェアラブル計算機など、小さくて解像度の低い計算機はまだまだ増えそうですから、しばらくはこういった工夫が報われる時代が続きそうです。

（ますい・としゆき　ソニー CSL）

[参考文献]

[1] Christopher Ahlberg and Ben Shneiderman, AlphaSlider: A compact and rapid selector, In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'94)*, pp.365–371, Addison-Wesley, April 1994

[2] Yuji Ayatsuka, Jun Rekimoto and Satoshi Matsuoka, Popup vernier: A tool for sub-pixel-pitch dragging with smooth mode transition, In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'98)*, pp.39–48, ACM Press, November 1998

[3] Dean F. Jerding and John T. Stasko, The information mural: A technique for displaying and navigating large information spaces, In *IEEE Transactions on Visualization and Computer Graphics*, Vol.4, No.3, pp.257–271, July 1998

[4] 柏木宏一、ボーデン・ジョージ、増井俊之「高精細スライダ "FineSlider"」、第 10 回ヒューマン・インタフェース・シンポジウム 論文集、pp.297–300、計測自動制御学会 ヒューマンインタフェース部会、1994 年 10 月

[5] 増井俊之「Pixeldoubler」、Interaction'99 講演論文集、pp.105–109, 1999 年 3 月