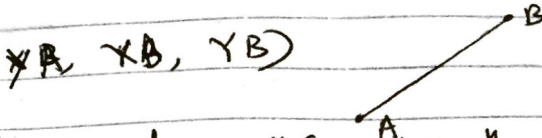
05/27/2016

Computer Graphics

1. Lines and Coordinates

In java, drawing a line can be done using the command

g.drawline (XA, YA, XB, YB)



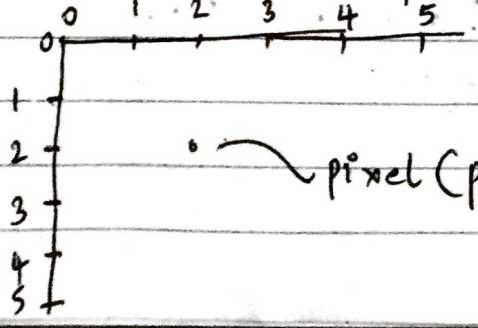
"drawline" is a method in class "Graphics"
"g" (Graphics context) is normally a parameter of "paint" method
The above is same as

g.drawline (XB, YB, XA, YA)

For example, we can draw largest possible rectangle on canvas:

```
Class CvRedRect extends Canvas {
    public void paint (Graphics g) {
        Dimension d = .getSize();
        int maxX = d.width - 1;
        int maxY = d.height - 1;
        g.drawString ("d.width = " + d.width, 10, 30);
        g.drawString ("d.height = " + d.height, 10, 60);
        g.setColor (Color.red);
        g.drawLine (0, 0, maxX, 0); // Top edge
        g.drawLine (maxX, 0, maxX, maxY); // Right edge
        g.drawLine (maxX, maxY, 0, maxY); // Bottom edge
        g.drawLine (0, maxY, 0, 0); // Left edge
    }
}
```

1.1 Device Co-ordinate System :-



pixel (picture element)

d.width = maxX + 1
d.height = maxY + 1

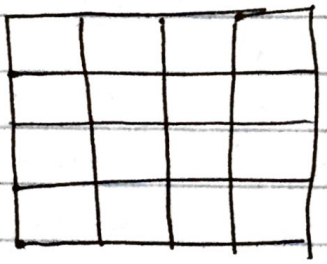
∴ The rectangle size drawn by g.drawRect (x, y, maxX, maxY) is :

Rectangle size drawn by
g.drawRect (0, 0, maxX, maxY) is
(maxX + 1) by (maxY + 1)

The smallest rectangle is a square 2×2
∵ using drawRect (X, Y, 1, 1)

To draw one dot : drawLine (x, y, x, y)

To fill a rectangle of size w × h
g. fillRect (x, y, w, h)
g. drawRect (x, y, w, h)



1.2 Logical Co-ordinate System :-

Suppose we want the origin at the left-bottom corner as in usual co-ordinate system, simply

$$y' = maxY - y$$

x-axis is unchanged, since it has the same direction as in math

	Convention	Data Type	Feature	Positive Y-axis
Logical	small case letters	float	continuous	upward
Device	Capital letters	integer	discrete	downward

1.3 Converting between logical and device co-ordinates:

① Rounding:

```
int iX (float x) {
        return Math.round(x);
}
float fx (int x) {
        return (float)x;
}
```

② Truncating:

```
int iX (float x) {
        return (int)x;
}                          // Logical to Device
float fX (int x) {
        return (float) X + 0.5 F;
}
```

① Eg: iX(2.8) = 2 ; fx(2) = 2.5

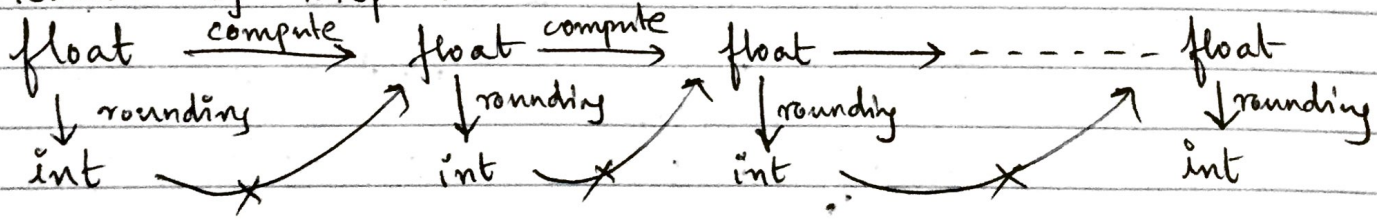
For both ① & ②

$|X - fX(iX(x))| \leq 0.5$ (max lost precision is 0.5)

Example program called "Triangles.java" in the book. It shows the advantage of using FP logical co ordinates.

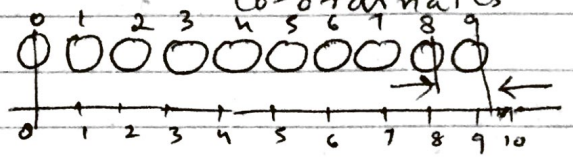
An important principle in the example:

Step i+1 computation is performed on FP results of Step i:



1.4 Mapping of Logical co-ordinates to Device co-ordinates :

Can we use `int iX (float x) { return Math.round(x); }` to map from logical co-ordinates 0.0 ~ (0.0 (float) to device co-ordinates 0 ~ 9 (int)?



Pixel width in terms of logical co-ordinates is :

$10/9 = 1.1$

The enhanced method is :

```
int iX (float) {
    return Math.round(x/pixelwidth);
}
```

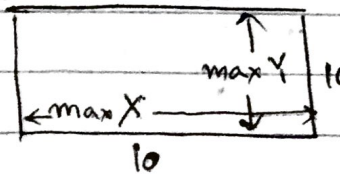
In this example, 9 = # of pixels (10) − 1,
10 = # of logical intervals.
(i.e $0 \le x \le$ rWidth)

It works similarly for vertical (y) co-ordinate (rHeight)

There are 2 mappings :

① Anisotropic mapping :-

pixel width ≠ pixel Height.



pixel Width = 10/maxX

pixel Height = 10/maxY

Pixel Height > pixel width (since maxX > maxY), So not suitable for shapes like squares & circles.

(2) Isotropic mapping :-

pixel width = pixel Height (pixel size)

Usually, we want the origin of the logical coordinate system to be in the center,

$-\frac{1}{2} r\text{width} \leq x < \frac{1}{2} r\text{width}$

$-\frac{1}{2} r\text{Height} \leq y < \frac{1}{2} r\text{Height}$

mapped to device co-ordinates on maxX and on maxY.

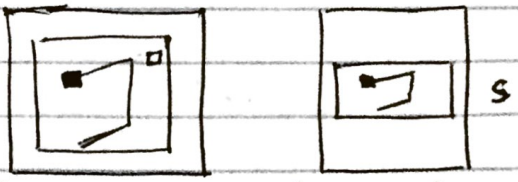
General Java Code :

```
Dimension d = getSize();
int maxX = d.width - 1, maxY = d.height - 1;
pixelSize = Math.max(rWidth/maxX,
                     rHeight/maxY);

CenterX = maxX/2; centerY = maxY/2;
int iX(float x) {
        return Math.round(CenterX + x/
                          pixelSize);
}
int iY(float y) {
        return Math.round(CenterY - y/
                          pixelSize);
}
float fx(int x) {
        return (x - centerX) * pixelSize;
}
float fY(int Y) {
        return (centerY - Y) * pixelSize;
}
```

Final Example :-

What are required?



→ The first vertex is drawn as a small rectangle.

→ If a later vertex is inside the small rectangle, the drawing of one polygon in complete. Back to ①

→ Only vertices in the drawing rectangle are drawn.

→ Change of window shape (Dimension) will not change drawing rectangle & polygon.

We will use isotropic mapping mode:

Algorithm for vDevPoly:

1. Activate mouse.
2. When mouse button pressed
   - 2.1 get x & Y co-ordinates at mouse click
   - 2.2 If its the first point
     Empty Vertex vector
   - 2.3 If its a later point that is inside the small rectangle,
     Finish the current polygon else not the last point.
     Store this point in vertex vector.
3. Draw all vertices in vertex vector

To draw all vertices (for paint in Java)

1. Obtain dimension of drawing rectangle based on logical co-ordinates.
2. Draw drawing rectangle.
3. Get first vertex from vertex vector.
4. Draw small rectangle
5. Draw a line between every two consecutive vertices.