1.4 Mapping of Logical Coordinates to Device Coordinates

Can we use:

```java
int ix (float x)
{
    return Math.round (x);
}
```

to map from logical coordinates 0.0 – 10.0 (reals) to device coordinates 0 – 9 (integer)?
1.4 Mapping of Logical Coordinates to Device Coordinates

- No.
- Why?
1.4 Mapping of Logical Coordinates to Device Coordinates

- Because the mapping between the device and logical coordinate systems is NOT 1-to-1

- Example:
  \[ i_x(10.0) = 10 \]

<table>
<thead>
<tr>
<th>Device: 0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical: 0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

(pixel width)
1.4 Mapping of Logical Coordinates to Device Coordinates

So, pixel width in terms of logical coordinates is \( \frac{10}{9} = 1.11 \)

The more accurate conversion method then is:

```java
int ix (float x)
{
    return Math.round (x/pixelWidth);
}
```
1.4 Mapping of Logical Coordinates to Device Coordinates

- With this example, 9 = # of pixels
  10 = # of logical intervals
  (i.e. 0 ≤ x ≤ rWidth)

- It works similarly for vertical (y) coordinates (rHeight)
1.4 Mapping of Logical Coordinates to Device Coordinates

- 1. Anisotropic mapping
  - pixelWidth ≠ pixelHeight
    - pixelWidth = 10/maxX
    - pixelHeight = 10/maxY
    - pixelWidth < pixelHeight (since maxX > maxY)
    - So, this is not suitable for shapes like squares and circles
1.4 Mapping of Logical Coordinates to Device Coordinates

- 2. Isotropic mapping
  - pixelWidth = pixelHeight
  - Usually we want the origin of the logical coordinate system to be in the center, so:
    
    \[-\frac{1}{2} \ rWidth \leq x < \frac{1}{2} \ rWidth\]
    \[-\frac{1}{2} \ rHeight \leq y < \frac{1}{2} \ rHeight\]

- Mapped to device coordinates 0 – maxX and 0 – maxY:
1.4 Mapping of Logical Coordinates to Device Coordinates

- Generalized Java code:

  ```java
  Dimension d = getSize();
  int maxX = d.width - 1;
  int maxY = d.height - 1; // why?

  pixelSize = Math.max(rWidth/maxX, rHeight/maxY); // get the same
  // scale factor (declared earlier)
  centerX = maxX/2; // declared earlier
  centerY = maxY/2; // declared earlier

  int ix (float x) // converting from logical to device
  {
      return Math.round(centerX + x/pixelSize);
  }

  int iy (float y) // converting from logical to device
  {
      return Math.round(centerY - y/pixelSize);
  }
  ```
1.4 Mapping of Logical Coordinates to Device Coordinates

- Generalized Java code:

```java
float fx (int x) // converting from device to logical
{
    return (x - centerX) * pixelSize;
}

float fy (int y) // converting from device to logical
{
    return (centerY - y) * pixelSize;
}
```
1.4 Mapping of Logical Coordinates to Device Coordinates

- Generalized Java code:

- Example:

```
(rWidth/2, rHeight/2)
(maxX/2,0)
(maxX,0)
```

```
(-rWidth/2, 0)
(0,maxY)
```

```
(-rWidth/2, -rHeight/2)
(0,maxY)
```

```
(rWidth/2, -rHeight/2)
(maxX,maxY)
```

```
(-rWidth/2, -rHeight/2)
(0,0)
```

```
(0, rHeight/2)
(maxX/2,0)
```

```
(maxX, maxY/2)
```

```
(maxX, maxY)
```

```
(maxX, maxY/2)
```

```
(maxX, maxY)
```

```
(maxX, maxY/2)
```

```
(maxX, maxY)
```

```
(maxX, maxY/2)
```

```
(maxX, maxY)
```

```
(maxX, maxY/2)
```

```
(maxX, maxY)
```

```
(maxX, maxY/2)
```

```
(maxX, maxY)
```

```
(maxX, maxY/2)
```

```
(maxX, maxY)
```

```
(maxX, maxY/2)
```

```
(maxX, maxY)
```

```
(maxX, maxY/2)
```

```
(maxX, maxY)
```

```
(maxX, maxY/2)
```

```
(maxX, maxY)
```

```
(maxX, maxY/2)
```

```
(maxX, maxY)
```

```
(maxX, maxY/2)
```

```
(maxX, maxY)
```

```
(maxX, maxY/2)
```

```
(maxX, maxY)
```

```
(maxX, maxY/2)
```

```
(maxX, maxY)
```

```
(maxX, maxY/2)
```

```
(maxX, maxY)
```

```
(maxX, maxY/2)
```

```
(maxX, maxY)
```

```
(maxX, maxY/2)
```

```
(maxX, maxY)
```

```
(maxX, maxY/2)
```

```
(maxX, maxY)
```

```
(maxX, maxY/2)
```

```
(maxX, maxY)
```

```
(maxX, maxY/2)
```

```
(maxX, maxY)
```

```
(maxX, maxY/2)
```

```
(maxX, maxY)
```

```
(maxX, maxY/2)
```

```
(maxX, maxY)
```
1.4 Mapping of Logical Coordinates to Device Coordinates

Example: drawing polygons using mouse

Lines and Coordinate Systems
What are required:

1. The first vertex is drawn as a small rectangle
2. If a later vertex is inside the small rectangle, the drawing of one polygon is complete. Back to (1)
3. Only vertices in the drawing rectangle (white box) are drawn
4. Change of window shape (Dimension) will not change the drawing rectangle and polygon

We will use isotropic mapping mode
Java algorithm for CvDefPoly:
  1. Activate mouse

  2. When mouse button pressed
     • 2.1. Get x and y coordinates at mouse click
     • 2.2. If it’s the first point
       Empty vertex vector
     • If it’s a later point that is inside the small rectangle
       Finish the current polygon
     • Else it’s 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) (#3 above)
  ◦ 1. Obtain dimension of drawing rectangle based on logical coordinates
  ◦ 2. Draw drawing rectangle
  ◦ 3. Get first vertex from vertex vector
  ◦ 4. Draw small rectangle
  ◦ 5. Draw a line between every two consecutive vertices
To make a point in 2D logical coordinates as an object

```java
class Point2D {
    float x;
    float y;

    Point2D (float x, float y) {
        this.x = x;
        this.y = y;
    }
}
```
Practice Exercises (1-1)

- How many pixels are drawn on the screen with:

  1. `g.drawLine (20, 40, 100, 60)`
  2. `g.drawRect (20, 40, 7, 9)`
  3. `g.fillRect (20, 40, 7, 9)`
Practice Exercises (1-1)

How many pixels are drawn on the screen with:

1. `g.drawLine (20, 40, 100, 60)`
   
   Answer: Since there is a pixel in each progression (along either the x or y directions), with both end-points, \((100 - 20) + 1 = 81\) pixels.

2. `g.drawRect (20, 40, 7, 9)`
   
   Answer: 2 horizontal edges and 2 vertical edges, so \(2 \times 7 + 2 \times 9 = 32\) pixels (Note: 7 and 9 above are the edge lengths, NOT the bottom right corner coordinates)
Practice Exercises (1-1)

- How many pixels are drawn on the screen with:

3. `g.fillRect (20, 40, 7, 9)`

Answer: All pixels in the rectangle of size 7 by 9, so $7 \times 9 = 63$ pixels (Note: 7 and 9 above are the edge lengths, NOT the bottom right corner coordinates)
Questions/comments/Ride to Marti Gras?