CS 355 Lab #3: Scrolling and Zooming

October 11, 2013

Overview

This lab builds on the previous ones by introducing scrolling and zooming. It also has you do the transformations you did last time by actually building matrices for affine transformations and applying them.

User Interface

All functionality from the first two labs should still work for this lab. Anything not working in prior labs that you get working for this final phase of this three-part program will receive up to half of the original credit (or half of whatever credit you did not receive earlier).

Here is the external functionality you should add to the program:

- You should add functionality to support the horizontal and vertical scroll bars in your window. This will include updating the view transform and initiating a refresh of the drawing area.
- You should also add functionality for the "Zoom In" and "Zoom Out" buttons, which should enlarge or reduce the size of the displayed objects by a factor of two up to a maximum of 400% and a minimum of 25% zoom. The maximum drawing area is what would be visible when zoomed out to 25% (i.e., the maximum viewable area). This will also include updating the view transform and initiating a refresh of the drawing area.
- All drawing in the viewport should be at the current scroll position and level of zooming, then converted to the world-space drawing area. That is, if you draw a shape at 50% zoom and then go back to 100%, the object should be twice as large as what you drew it at. Similarly, if you draw it in the center of the viewport but the viewport is scrolled over, the object should stay at that position in the world-space drawing area and move accordingly on the screen.
- When zooming in or out, the center of the viewport should stay at the current position in the world space unless doing so would take the viewed area outside the maximum drawable area.

Model

The model should stay unchanged from the previous assignment, and all viewing transformations should be handled in the controller and viewer.

Remember that all interactions in the model should already be in *world coordinate space* (i.e., in terms of a global drawing area), so no changes should be required. If you did mix some notions of what's happening on the screen into the model, you may make changes to correct that.

One exception: if you implemented the point-in-shape or point-near-line tests in the model in terms of world coordinates, the threshold for "near enough" to a line should scale with the viewing so that the on-screen tolerance stays constant regardless of the zoom level. That is why I suggested you pass the tolerance as a parameter to the "near enough to the line" test if you implemented it there. If you implemented the selection tests in the controller, that can all stay there.

Implementation Notes

This lab uses the same program shell (helper classes) as Labs #1 and #2.

Transformations

For this lab, you must implement all transformations as individual `AffineTransformation` objects, and you must calculate the matrix entries yourself. Each `AffineTransformation` should be constructed by passing it the six entries of the matrix, not by making individual calls to apply rotation, scaling, or translation to the transformations. *You may only create or apply transformations by passing in the entries for the matrix or by composition of transformations you've already built.*

Drawing

You should draw with an affine transformation that converts from object coordinates to viewport coordinates. This per-object drawing transformation $\mathbf{M}_i$ is simply a compositing of the respective object-to-world transformation and the world-to-viewport transformation:

$$\mathbf{M}_i = \mathbf{V}\ \mathbf{O}_i \tag{1}$$

You can compose matrix transformations using the `AffineTransformation` class's `concatenate` method.

Object to World

The object-to-world transformation is the same as what you did when drawing in the previous lab: rotate first, then translate to the object's position in the world:

$$\mathbf{O}_i = \mathbf{T}(\mathbf{c}_i)\ \mathbf{R}(\theta_i) \tag{2}$$

where $\mathbf{T}(\mathbf{c}_i)$ denotes translation by offset $\mathbf{c}_i$, and $\mathbf{R}(\theta_i)$ denotes rotation by angle θ_i.

You can use this transformation to convert from world to viewport coordinates:

$$\mathbf{p}_{\text{world}} = \mathbf{O}_i\ \mathbf{p}_{\text{object}} \tag{3}$$

World to View

You should similarly construct a viewing transformation $\mathbf{V}$ that converts between world space and the viewport space. It should scale the coordinates accordingly as the zoom level changes and translate the coordinates according to the scrolling bars.

$$\mathbf{V} = \mathbf{S}(f)\ \mathbf{T}(-\mathbf{p}) \tag{4}$$

where $\mathbf{S}(f)$ denotes scaling by the current scaling factor f, and $\mathbf{T}(-\mathbf{p})$ is a translation from the position $\mathbf{p}$ (in world coordinates) of the origin of the viewport's coordinate system to the origin of the world coordinate system.

You can use this transformation to convert from world to viewport coordinates using V:

$$\mathbf{p}_{\text{view}} = \mathbf{V}\ \mathbf{p}_{\text{world}} \tag{5}$$

Doing the Drawing

For drawing, you should pass this to the `setTransformation` method of the `Graphics2D` object you're drawing to. For example, if the `Graphics2D` object passed to your viewer is `g`, and the composite transformation $\mathbf{M}$ you have computed (Eq. 1) is `objectToScreen`, this would look like this:

```
g.setTransformation(objectToScreen);
```

Selecting

To select shapes, you will need to convert from screen coordinates to world coordinates, and from world coordinates to object coordinates. You can, of course, do this with a single transformation that is the inverse of the matrix $\mathbf{A}$ you already calculated.

You should draw with an affine transformation that converts from object coordinates to viewport coordinates. This per-object transformation $\mathbf{A}_i$ is simply a compositing of the object-to-world and world-to-viewport transformations:

$$\mathbf{M}_i^{-1} = \mathbf{O}_i^{-1}\ \mathbf{V}^{-1} \tag{6}$$

Remember that from Eq. 2,

$$\begin{aligned} \mathbf{O}_i^{-1} &= \mathbf{R}^{-1}(\theta_i)\ \mathbf{T}^{-1}(\mathbf{c}_i) \\ &= \mathbf{R}(-\theta_i)\ \mathbf{T}(-\mathbf{c}_i) \end{aligned}$$

and from Eq. 4,

$$\begin{aligned} \mathbf{V}^{-1} &= \mathbf{T}^{-1}(-\mathbf{p})\ \mathbf{S}^{-1}(f) \\ &= \mathbf{T}(\mathbf{p})\ \mathbf{S}(1/f) \end{aligned}$$

To transform points using the matrix $\mathbf{M}_i^{-1}$, use the `AffineTransformation` class's `transform` method. For example, if the transformation $\mathbf{M}_i^{-1}$ is stored in an object called `screenToObject`, and you want to transform the point `screenPoint` (in viewing coordinates) to the point `objectPoint` (in object coordinates):

```
screenToObject.transform(screenPoint,objectPoint);
```

Submitting Your Lab

To submit this lab, again zip up your src directory to create a single new `src.zip` file, then submit that through Learning Suite. If you need to add any special instructions, you can add them there in the notes when you submit it.

If you choose to upgrade anything from the earlier labs and want the TA to re-grade it, make sure to include that in your notes as well.

Rubric

This is tentative and may be adjusted up to or during grading, but it should give you a rough breakdown for partial credit.

- Correctly drawing while zoomed in or out (5 points)
- Correctly drawing while scrolled (5 points) — note that you can only scroll while zoomed in
- Correctly selecting while scaled (5 points)
- Correctly selecting while scrolled (5 points) — again, you can only scroll while zoomed in
- Correctly implementing all transformations ($\mathbf{M}_i$, $\mathbf{V}$, $\mathbf{O}_i$, $\mathbf{T}(\mathbf{c}_i)$, $\mathbf{R}(\theta_i)$, $\mathbf{S}(f)$, $\mathbf{T}(\mathbf{p})$, and their inverses) as `AffineTransformation` objects created either by *you* giving it the matrix elements or by composition (15 points)
- Otherwise generally correct behavior (5 points)

TOTAL: 40 points

Change Log

- October 3: initial version
- October 8:
  - revised notation to match in-class presentation
  - correct an error in the ordering of the elements of the viewing transformation
  - other minor revisions for clarity
- October 10: amended to include case where zooming in, scrolling, then zooming out might take the viewed area outside the drawable area