

CS 418 Program 2: Draw

(revised February 19, 2003)

out: Tuesday, February 11, 2003

due: Friday, February 21, 2003

but accepted without penalty until **Monday, February 24, 2003**

Your assignment is to write a drawing/drafting program in Java and OpenGL. It has features along the lines of Adobe Illustrator, the Microsoft Office drawing tools, and the 2D part of a CAD package like AutoCAD. The program will provide the user with a drawing area and a set of tools that can be used to create objects and then to select, edit, transform, and group them. The canvas is infinite in extent, and the user can smoothly pan and zoom to choose with part to work on.

Bear in mind that this program will form the basis for the following two assignments, which will build a 3D modeling system by allowing the user to create and manipulate 3D objects derived from the 2D outlines you're working with in this assignment.

In this revision to the assignment I have added clarifications to the original requirements in small sans serif type.

Requirements

To be specific, here is what your program needs to do:

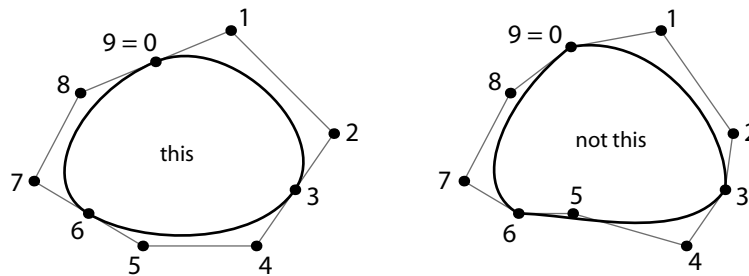
1. Display a drawing area and provide a means for selecting among a set of tools. (The framework already does this.)
2. Provide tools for creating rectangles, ellipses, closed polygons, and closed Bézier splines. The user interface should be the usual one for drawing programs: for rectangles and ellipses the user clicks at one corner, drags to the other, and releases; for drawing polygons and splines the user clicks the vertices or control points one at a time, clicking back on the first point to finish.

The presence of the word "closed" is meant to make things easier, not harder. If you want to support both open and closed shapes that is completely fine.

To clarify the expectations for a "usual" user interface:

- You should provide live feedback while objects are being drawn. A rectangle or ellipse should stretch out to follow the mouse as it moves. The already entered segments of a polygon or spline should be shown, with the one about to be created following the mouse. The general principle is that if the next event is going to create something, you should draw that thing on the screen while the mouse is moving.
- Whenever the user is supposed to click on something (in this case clicking on the starting point to close off a shape), you should provide a little leeway so that if the click lands within 2–3 pixels of its target it counts as a hit.
- For rectangles, ellipses, and polygons the semantics are pretty clear-cut: in the first two cases, the mouse-down and mouse-up events define the corners of the bounding box of the shape; for polygons the series of mouse-down events defines the vertices. For splines there are many reasonable ways to proceed. Bézier splines are defined in segments, with four control points influencing each segment. In order for the spline to be continuous at all, adjacent segments will share the same endpoint, so points 0, 1, 2, 3 will define the first segment, points 3, 4, 5, 6 will define the second, and so on. Once the spline is closed, the endpoint of the last segment is point 0.

In order for the spline to be smooth (which it should be), the points on either side of each endpoint need to end up collinear with the endpoint:



When the user creates a spline it should have this property. One way to map mouse events into points is Adobe's way: down \rightarrow 0, up \rightarrow 1, down \rightarrow 3, up \rightarrow 4 (and also generate 2), down \rightarrow 6, up \rightarrow 7 (and also generate 5), down (at point 0) \rightarrow 9 = 0 (and also generate 8). Another way that may be a bit simpler is: down \rightarrow 0, down \rightarrow 1, down \rightarrow 2, down \rightarrow 3 (and also generate 4), down \rightarrow 5, down \rightarrow 6 (and also generate 7), down (at point 0) \rightarrow 9 = 0 (and also generate 8).

3. Using the hand tool, allow the user to pan and zoom arbitrarily, with real-time feedback. That is, there should be no limits on the size of the drawing or how far the user may zoom in and out, other than the limits imposed by your floating-point representation. Provide a "home" mechanism that returns to the default view in case the user gets lost.

When the user drags with the hand tool, the drawing should move along with the mouse cursor, with the same part of the drawing remaining under the hand (the framework has this property as supplied).

In order to support panning and zooming with one mouse you'll have to have the user do something like holding down shift and dragging vertically to zoom, or using the left button to pan and the right button to zoom.

4. Provide a selection tool that allows the user to select single or multiple objects by clicking on them. The user interface should again be the usual one: clicking sets the

selection to the indicated object, and shift-clicking adds the indicated object to the selection or removes it if it is already selected.

Clicking in the interior of a filled shape should select it; shapes that are not filled must be selected by clicking the boundary. You should provide some margin of error so that shapes' boundaries can be hit by clicking within a pixel or two. Clicking on any shape in a group selects the group, not the object (that is, you can only select shapes at the top level of the hierarchy).

5. Provide a means for grouping the selected objects and for ungrouping groups. Groups can nest within groups up to a depth of at least 32.¹

Groups are organized strictly in a tree structure. Because objects within groups cannot be selected, there is no way for the user to try to put an object in more than one group, so you don't have to do anything special to enforce the tree structure.

6. Allow the user to apply 2D transformations to a selected object. These transformations may be limited to rotation, translation, and scaling. The reference point for the transformations should be the center of the object (for some reasonable definition of "center"). Selecting several objects and rotating one of them causes the objects to rotate about their own centers, while grouping those same objects and then rotating the group causes them all to rotate together about the group's center. The user interface for transformation should live entirely off the arrow tool used for selection. Here is a reasonable way to set up the interface:

- (a) Around each selected object draw 8 square handles on the corners and edges of the object's bounding box. (It's OK to use the bounding box of the control points for the spline.)
- (b) When the user drags the object itself, apply a translation to keep the object under the mouse cursor.
- (c) When the user clicks on a scaling handle and drags, apply a scale to keep the handle under the mouse cursor
- (d) When the user clicks on a rotation handle and drags, apply a rotation to keep the handle, the mouse cursor, and the center point collinear.

The center of a shape's bounding box is a reasonable definition of "center". The rotations must happen at the top level of the hierarchy—a rotated rectangle should still have square corners. The transformation handles should stay the same size on the screen even if you scale the shape.

7. Provide a means to select the outline and fill color for the selected object, including none as an option for each.
8. For splines and polygons provide a tool for editing them by dragging the corners or control points.

¹This limit is so that you may rely on the OpenGL matrix stack.

You could implement this as an additional tool or as some sort of mode associated with the arrow tool. You should draw handles so that the user can see where to click (especially for the spline!). When the user edits a spline control point, the other points should move as needed to maintain the spline's continuity.

Principles of operation

Because this program needs to evolve into a 3D modeling program over the next two assignments, we recommend that you follow the design outlined here fairly closely.

Your main data structure will be a tree, known as the *scene graph*, that contains all the shapes that exist. The leaves are the individual shapes, and the interior nodes are groups. Every shape has a 2D affine transformation (a 3x3 matrix) associated with it. The different types of shapes (circles, polygons, splines, groups) should be implemented by classes that extend an abstract shape class. You could make the scene graph a tree, with a group at the root, or a forest. Grouping is a very simple operation: make a new node and move the selected shapes below it. Ungrouping is a tiny bit more involved: when you delete a node you need to concatenate its transformation with those of what used to be its children.

To draw the window, simply set the view using scaling and translation transformations, then traverse the scene graph calling a drawing method on each object. Each object concatenates its own transformation with the current state before drawing itself; a group draws itself by drawing its members. To draw circles and splines you can approximate them using polygons. It is easy to implement ellipses by using transformed circles.

When you are drawing the interior of a filled shape you will need to use area primitives (triangles), and when you are drawing the outline you will use line primitives (lines, line loops). Note that there is a facility available in the GLU library to handle breaking non-convex polygons into triangles. This facility is discussed in the OpenGL Programming Manual, and the particular quirks of using it under GL4Java are discussed in the GL4Java FAQ.

Your program does not need to correctly fill self-intersecting polygons or splines (though it shouldn't crash or throw unhandled exceptions).

For selection, one of the easiest approaches is to use an object ID buffer. The idea is to draw all the objects, not in their real colors, but in colors that encode a unique identifier per object. Reading the framebuffer at the mouse location then gives you the identity of the object that was clicked on. You may want to draw outlines with a larger line width to help make it easier to hit things with the mouse.

You want to put this object ID map into the back buffer so that the user does not see it.

Up until now all drawing has been done under the `display` method of the `GLEventListener` interface. For selection (also called "picking") you will likely want to make some OpenGL calls outside this environment. To do so, you need to make sure that the current OpenGL context is the one that corresponds to your window (so that your OpenGL commands apply to your window rather

than applying to some other window or just being discarded). Under GL4Java, you can get the `GLContext` from the `GLCanvas` held by `Viewport2D`, and make it the current context using the `gljMakeCurrent` method. This method also acquires a lock to make things thread safe, and you need to release this lock when you are done by calling the `gljFree` method.

When the user applies a transformation, you will compute it based on the object they transformed and then apply that same transformation to all the selected shapes (adjusted so that the reference point is the center point for each).

Framework code

Like the previous assignment, this assignment comes with some code to build on. However, unlike the previous assignment, the code is only a simple starting point rather than a nearly complete program. In this assignment you will design and build all the fundamental data structures and algorithms yourself.

The framework code constructs a window with a menu bar, a toolbar full of appropriate buttons, and a large drawing area. It provides a simple `Transform2D` class with the ability send itself to OpenGL; a `Viewport2D` class that manages the `GLCanvas` where drawing occurs and sets up the view before drawing the scene; and a simple `Shape2D` class with subclasses for `Circle` and `Square`. The example code in `main` adds a circle and a square to the scene. The framework has no concept of grouping or hierarchy, and it does not implement any tools other than the hand tool for panning.

We expect you will build your assignment by editing the framework rather than by extending it as in the previous assignment.

Groups

This project is to be done in groups of two.

Extra credit

As always, we encourage you to add extra features for extra credit. The ground rules are that (a) a program that scores below 95/100 on the basic requirements cannot earn any extra credit and (b) no program can have a score greater than 115/100. The following are some suggestions that would be worth 2–5 points out of 100 if implemented well:

- Add line widths and line styles to the object properties.
- Draw filled self-intersecting polygons and splines correctly (only 1 point, but it's an easy point because GLU will do this for you; see the GLU 1.3 spec and the GL4Java

tessellation examples).

- Implement NURBS and unify the representation of polygons, circles, ellipses, and splines.
- Implement a user interface to the complete set of affine transformations (including shears and reflections).
- Output to PostScript or PDF (this is actually pretty easy, but requires learning about PostScript).
- Implement a snapping feature that makes it possible to place points exactly coincident, exactly on other primitives, and at the intersection of two primitives.