COMP6700/2140 Animation: Splotch and Rolling Wheel

Alexei B Khorev

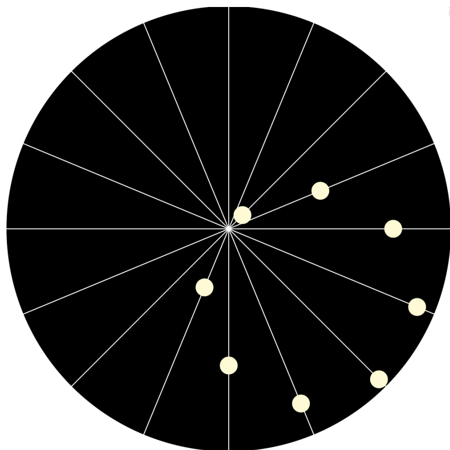
Research School of Computer Science, ANU

May 2017

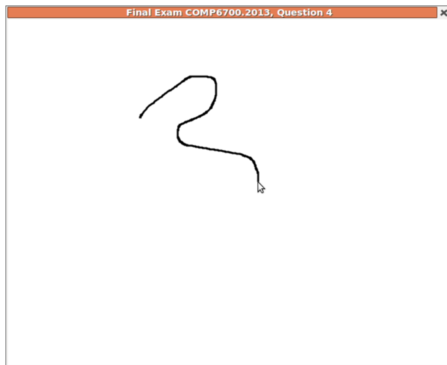
Splotch: an Advanced Animation

1. Animation: Timeline populated by KeyFrames
2. Rolling Wheel: a Simpler Case Study
3. Model and View: Separation is Essential
4. Model: Path and its rounding
5. Morphing the Splotch
6. A few Projects

Curious for Clueless: Rolling Wheel



Splotch



JavaFX Animation Machinery

1. KeyValues
2. KeyFrames
3. Timeline

Splotch's MVC architecture

This is taken from one of the yesteryears final exam (don't panic! — instead, pay attention!). There are three files which represent separation on *Model* (two files, one is a mere auxiliary) and *View*:

- Point.java (that small auxiliary)
- Smoother.java is the View (and the main-class)
- Splotch.java is the Model

You can download all three class files in a directory; compile it with

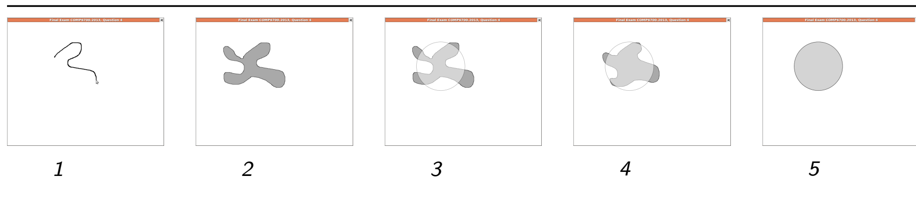
```
% javac -d . *.java
```

and run as follows

```
% java splotch.Smoother
```

Splotch in Action

What the program does:

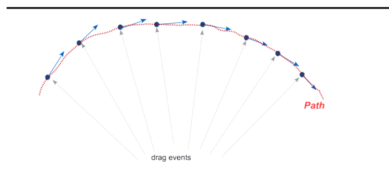


1. It opens a window in which you can draw by dragging the mouse
2. Once the dragging is over, the program closes the drawn path and fills the interior figure
3. The user presses “s” (for *smooth*) — a circle overlaying the figure is drawn; it has (at least when the figure is “simple”) the same area, and located in the same “centre of mass”
4. The user then presses “m” (for *morph*) — the original figure “gracefully” morphs into a circle
5. Next dragging event repeats the same with a new figure (the circle gets immediately erased)

Inside the View, Smoother.java

The dragging is a “continuous” sequence of events located at (“almost”) every pixel swept by the mouse. The pixels are assembled into a `javafx.scene.shape.Path` as the result of this *callback*:

```
// dragging creates lineTo element added to the path
scene.onMouseDraggedProperty().set(new EventHandler<MouseEvent>() {
    public void handle(MouseEvent event) {
        currentPoint = Point.makePoint(event.getX(), event.getY());
        points.add(currentPoint);
        onePath.getElements().add(new LineTo(currentPoint.x, currentPoint.y));
    }
});
```



- The *Path* object is initialised with `scene.onMousePressedProperty().set(...)`
- and closed (to represent a closed curve) with `scene.onMouseReleasedProperty().set(...)`

The Model, `Splotch.java`

The model describes the closed path mathematically as an ordered set of points (x_i, y_i) (represented by the `ArrayList<Point> points` field). The Model contains the most essential calculations of *smoothing*, ie, given a set of `points`, which represent an arbitrary closed path drawn by the user, calculate the smoothed shape — a circle, which is centred at the same point as the splotch, and has the same area (both the centre, and the area are calculated approximately using the coordinates (x_i, y_i)). The centre is calculated by `medX()` and `medY()` (using the notion of the *centre of mass* — both the original splotch and the smoothed circle have the same "mass" and "centre of mass").

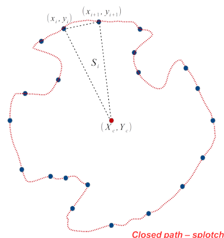
The splotch area could be computed as a sum of triangles S_i, but this approach is not possible if the splotch area is a non-convex region (its boundary, *ie* our path, has forward and retrograde "movements", relative to the centre). Therefore another formula is used:

$$Area = \frac{1}{2}\sum_{i=0}^{n} \text{abs}\left\{x_i \cdot y_{i+1} - y_i \cdot x_{i+1}\right\},\ (x_n, y_n) \text{ is the same point as } (x_0, y_0).$$

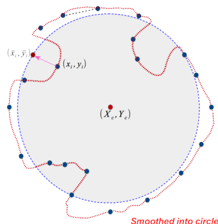
(which should be known to everyone familiar with the vector product of two planar vectors). The method `Splocth.area()` is nothing but the coded formula shown above.

Smoothing

Once we know area, we can calculate the radius of the circle (the method`Splotch.radius()`). Each point from the splotch must be mapped into a point on a circle (the circle must have the same number of points as the splotch, this is needed to define the animated transition `splotch --> circle`). The circle point coordinates are easily computed given their number, the radius and the centre (the method `Splotch.smoothSplotch()`, which returns a splotch object), if we make a natural choice to distribute points with equal density.



Closed path – splotch



Smoothed into circle

Animated transition of smoothing

Once the original splotch is smoothed (into a circle shaped splotch), one can use JavaFX's animation API (package `javafx.animation`), and define a `Timeline` object which will implement change of properties of the original object ("splotch"-path) into the final one ("circle"-path, which has the same number of path elements, `LineTo`, so the two paths are "homeomorphic").

The details are slightly more elaborate, but the essence is the same as in the "fake morphing" example, Morphism.java: The timeline is fitted with `KeyFrames` (one for each pair of path elements), which are in turn created to contain two `KeyValue`'s, each representing the coordinate properties of the "starting" shape and the "finishing" shape. The method `Smoother.makeTimeline(p1,p2)` creates this timeline object which "temporally" connects the two paths `p1` and `p2`. All intermediate properties (needed to animate the change which looks like morphing) are interpolated by the API.

Note: both paths — for "splotch" and for "circle" — exist from the beginning since they have to be part of the scene all along. But when we create a new splotch by dragging, the second path ("circle") is emptied (and is not seen); once the user chooses to `smoothPath()` (by pressing `'s'`), the second path is filled with elements which are created using the circle, which in turn is created by the Model as described above). Animation through timeline only involves manipulating the properties of the existing scene components.

The triggers to create the smoother splotch and to play the timeline transition are programmed as callbacks for `scene.onKeyPressedProperty().set(...)` event handler, `Smoother:108-135`.

That's how a less trivial animation can be programmed with `javafx.animation` API. "Easy!"

Voilà