COMP6700/2140 Effects

Alexei B Khorev and Josh Milthorpe

Research School of Computer Science, ANU

May 2017

Topics

1. Transitions
2. Morphing (simple)
3. Animation
4. Resources for further study

Transition and Animation Effects

Graphic elements (leaves and nodes) can undergo changes (*transitions*) including:

- Motions: *Translate* (incl. along an arbitrary *Path*) and *Rotate*
- Size and form changes — *Scale* and *Shear* (*Affine* can be used to compose a complex motion transition which consists of *Translate*, *Rotate*, *Scale* and *Shear*)
- Colour fading (opacity change), blending and other property changes (*FadeTransition* etc)
- Animations (morphisms) — by transforming one object into another (of different shape and size) over a prescribed time interval:
  - *Timeline* sets interpolation across a sequence of
  - *KeyFrame*s which define intermediate target states by using
  - *KeyValue*s (states of a node at selected intermediate reference points)
- Transitions can be executed sequentially (`SequentialTransitions`) and simultaneously (`ParallelTransition`). See example ComboTransitions.java

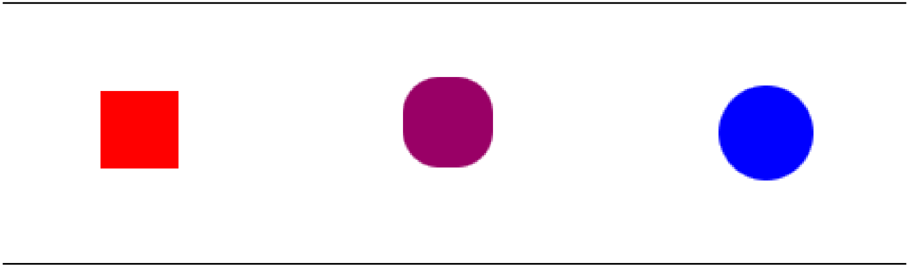
The transition and animation effects API are documented in `javafx.scene.transform` and `javafx.animation` packages.

The animation effects are generated by synchronising changing scene graph data with their visual representation. The JavaFX rendering engine called *Prism* (apparently not related to the NBA secret illegal global surveillance program) generates a special event called *pulse* (with up to 60 fps frequency) during animation run.

Morphing

An interesting version of animation effects is morphing — making an object to change its shape (with or without moving) and other characteristics (colour, opacity).

Morphing shape (rectangle-to-circle), moving it and changing its colour:



It is **not** very easy to achieve unless the objects (original "from" and final "to") are simple enough, like in the example Morphism.java.

One can use *Path* class and fill it to generate the shapes involved. The morphing would involve calculating the coordinates of path's points during the transition. A case study is *Splotch*, discussed in F5 lecture.

GUI (Rich Client) Programming Key Steps

1. Choose your interface (how many windows *etc*)
2. Choose layout of every container window, fill them with control and display elements
3. Define (set element properties, define and create event handler objects) what events will be detected and how application will respond
4. Decide on transition and animation effects
5. If UI is complex, use *SceneBuilder* tool to create it and load its `fxml`-description into application
6. Decide if elements need adornment and other "rich style" features, use CSS styling and applied them by using resources from `css`-file

Study JavaFX

The best way to learn GUI programming with JavaFX is to write a JavaFX GUI program. Despite there are already several books on the subject, one can get by by using only two sources:

Official JavaFX Tutorial A set of presentations supported by good examples about all major aspects of creating (and deploying) a GUI/RichClient application. It contains far more than we need in our course, so pay particular attention to these sections:

- Get Started with JavaFX
- Work with the Scene Graph
- Work with Properties and Bindings
- Events/Event Handlers
- Transitions/Animations

JavaFX API Documentation The standard and indispensable JavaFX API documentations (as usual, often with coding examples which facilitate understanding). Use it! (you can get a local copy to install on your computer).

The *Oracle's* Youtube channel has a number of conference presentations and educational videos, including those devoted to *JavaFX* (do the text search on “javafx” to speed up finding them).