Chapter 7

*Squeak: Object-oriented design with multimedia applications*
Story

- Text
  - TextMorphs, filling a figure, flowing
- Graphics
  - Different formats, 3-D
- Sound
  - Recording, editing, playback, MIDI
- Squeak-specific media: Movies and BookMorphs
Exploring TextMorphs

- TextMorphs
  - Multiple fonts and styles
  - Fill curved shapes
  - Wrap around complex shapes

Text morphs can be chained together, causing their contents to flow between containers as the contents or the containers change. If a TextMorph is embedded in another morph, the content as defined (under the create button) becomes a TextMorph. Text can be oriented in a curve, then you can ask to have the text follow that curve, as illustrated in this image. You can also ask to have the text avoid occlusions, in which case it will do its best to avoid siblings that are in front of it. Kerning (cmd+h, win + or -) can help with the awkward spacing that results from narrow margins. Other morphs can also be embedded in text as glyphs. Embedding placement is based on the top left corner of the morph’s bounding box.
How PWM-5 is Built

- Four TextMorphs
  - One in large CurveMorph on top which is filling owner’s shape
  - Second one in “pipe” CurveMorph is linked to top and is also filling owner’s shape
  - Third one in rectangle (GradientMorph) is also linked to pipe, and is set to avoid occlusions
  - Last one is in an EllipseMorph, also filling owner’s shape, which is setting on top of rectangle
Changing Fonts and Styles

- Halos provide access to:
  - Changing font size
  - Changing font
  - Changing style
- Command-keys can also do change fonts/styles:
  - See Help menu
  - Can also create hyperlinks
TextMorph Capabilities via Red-Halo Menu

• Fill owner’s shape
• Avoid occlusions
• Add predecessor/successor
• Copy Postscript or print PS to file
TextMorphs (all morphs!) can Render to Postscript

This is a test. I am writing some text here for generating PostScript

This is a test. I am writing some text here for generating PostScript
Even SketchMorphs can Generate PostScript

• EPSCanvas and DSCPostScriptCanvas
  – morphAsPostScript: “returns EPS or PS”
Generating Postscript

- Individual morphs generate EPS
- We can generate PS using a BookMorph
Creating Linked TextMorphs

1. Add successor
   - Just drop it somewhere

2. Resize original
   - Text flows into successor

This is a test. I am writing some text here for generating PostScript

This is a test. I am writing some text here for generating PostScript
Workspace code to do all this

- Create a TextMorph and embed into an Ellipse
- Make TextMorph fill the Ellipse
- Select text and change emphasis
- Create a second TextMorph to fill into
Example code

texta := TextMorph new openInWorld.
ellipse := EllipseMorph new openInWorld.
ellipse addMorph: texta. "Embed the text in the Ellipse"
texta fillingOnOff. "Make the text fill the Ellipse."
texta contentsWrapped: 'My first textMorph in which I
   explore
different kinds of flowing.'.
"Demonstrating of changing emphases"
texta editor selectFrom: 1 to: 2.
texta editor setEmphasis: #bold.
"Make a second text area"

textb := TextMorph new openInWorld. 
texta setSuccessor: textb. "A flows to B"
textb setPredecessor: texta. "B flows back to A"
texta recomposeChain. "Make the flow work."
Generating Postscript in same Workspace

b := BookMorph new openInWorld. "Books generate .ps"
b currentPage addMorph: ellipse. "Put the ellipse in the book"
ellipse topLeft: (b currentPage topLeft). "Position it properly"
f open.
f nextPutAll: (DSCPostscriptCanvas morphAsPostscript: b
    rotated: true).
f close.
Other TextMorph Messages

- occlusionsOnOff — Toggles whether occlusions are avoided
- fillingOnOff — Toggles between rectangular and filling owner’s shape
- editor — To get at TextMorph’s ParagraphEditor
  - changeAlignment, changeEmphasis, changeStyle
Graphics Tools

- From file list, open image in different formats
  - .bmp, .gif, .jpeg, .jpg

- Make New Drawing from New Morph menu
  - Creates a SketchMorph
Manipulating Graphics in Code

- Forms are the key to graphics in Squeak
  - Forms can read any of these formats via fromFileNamed:

- Can use SketchMorph or ImageMorphs as wrappers on Forms

(ImageMorph new image:
  (Form fromFileNamed: ‘myfile.gif’) openInWorld
(SketchMorph new form:
  (Form fromFileNamed: yourfile.gif’) openInWorld)
Manipulating Forms

■ Lots of low-level classes that can drawOn:
a Form
    ■ Pen, Rectangle, Quadrangle, Arc, Circle, Spline
■ Set colors with Color
    ■ Color transparent — allows a color to show color “behind”
Simple Animating

Class Morph knows how to do path-based animation

- Follow the Sensor to define a path, then replay the path later
- `definePath` fills an `OrderedCollection pathPoints` (property, via `MorphExtension`) with points from the Sensor
- `followPath` moves the Morph along the defined `pathPoints` path
BitBlt and WarpBlt

- BitBlt is key to animated graphics and dealing with transparency
  - Example: Display restoreAfter: [BitBlt alphaBlendDemo]

- WarpBlt allows us to do much more sophisticated animations and transformations, including rotations
Using WarpBlt to create Thumbnails of any Morph

makeThumbnail
| viewSize thumbnail |
"Make a thumbnail image of this image from the Display."
thumbnail := Form extent: viewSize depth: Display depth.
(WarpBlt toForm: thumbnail)
  sourceForm: Display;
  cellSize: 2; "installs a colormap"
  combinationRule: Form over;
  copyQuad: (self bounds) innerCorners
toRect: (0@0 extent: viewSize).
(ImageMorph new image: thumbnail) openInWorld.
Creating external graphics

- Forms can be written out with
  - writeOn: “Internal format”
  - writeBMPfileNamed:

- ImageReadWriter subclasses handle other formats
  - GIF, JPEG, PCX, and XBM
3-D Computer Graphics

- Andreas Raab’s Balloon (a portable plugin!) + Jeff Pierce’s Wonderland
History of Wonderland

- Alice: 3-D end-user programming environment for Windows
  - Developed by Randy Pausch’s group at Carnegie-Mellon U.
  - Used in videogame and movie classes
- Wonderland developed by Randy’s student Jeff Pierce on a Disney internship
Pieces of Wonderland

- WonderlandCameraMorph – displays the 3-D Wonderland world (can have many open at once)
- WonderlandEditor – where end-user scripts the world.
  - Can get rid of it safely.
- WonderlandActor – what all objects in the Wonderland are subclasses of
Creating and using a Wonderland

■ Wonderland new
■ Within Wonderland editor
  ▪ Wonderland is accessed through \( w \)
  ▪ Current camera is \( camera \)
  ▪ Camera window is \( cameraWindow \)
■ Wonderland can also be used from workspace
  ▪ \( w := \) Wonderland new.
  ▪ \( w \) getEditor hide. “\( w \) getEditor show.”
Making Wonderland Actors

- MDL is the Alice and Wonderland internal format
  - `w makeActorFrom: ‘myfile.mdl’`
- Can also read VRML and 3DS
  - `w makeActorFromVRML: ‘myfile.vrml’`
- Can also make actors by composition of other actors
  - `makeActor, makeActorNamed:, becomeChildOf:`
Manipulating Actors

- Actors and their components are known to the workspace (by namespace and accessors)
  - bunny head turn: right
  - Things like right, green, etc. are known to the editor workspace

- All changes are animated—they “morph” between states
  - bunny head setColor: green

- Methods are chosen so that simple things are simple, and complex things are possible
Animation in Wonderland

- Scripts return Animation objects
  - Viewing an animation as an object (Noun) makes undo in Wonderland easier to implement

- Allows for ease of scripting
  - `fd := snowman move: forward.
  - rt := snowman turn: right.
  - W doTogether: {fd. rt }`
Active Textures in Wonderland

- Any Wonderland object can be “textured” with 2-D Morphs—and they dynamically update!
  - Set camera window to be “open to drag and drop”
  - Create an object e.g., w makePlaneNamed: ‘myPlane’
  - Morphic-select the plane
  - Choose “enable active texture” from its menu
  - Choose “auto adjust to text” from its menu
  - Drop a morph onto the plane (even System Windows!)
Example of Active Textures
Useful Wonderland Tricks

- Control the camera directly
  - cameraWindow showCameraControls

- Release the objects from their world
  - ground hide.
  - camera turnBackgroundOff.
  - cameraWindow hideCameraControls.

- Make objects respond to mouse clicks (useful for games)
  - bunny respondWith: [:event | bunny head turn: left] to: leftMouseClick. “bunny stop to remove constraint”
More Useful Wonderland

Tricks

- Control the camera
  camera move: up.

- Point objects at one another
  bunny head pointAt: camera duration: eachFrame.

- Do something each frame
  bunny head doEachFrame: [ bunny head pointAt:
    (camera transformScreenPointToScenePoint:
      (Sensor mousePoint) using: bunny) duration:
    rightNow ].
Controlling Wonderland without the Editor

- Can hide the editor: in the Wonderland, or on a flap
  - Or from a workspace
    - `w := Wonderland new.`
    - `w getEditor hide. "w getEditor show"`
    - `w makeActorFrom: 'mycharacter.mdl'`
    - `(w getNamespace at: 'bunny') head turn: #left.`
Flash movies

- Vector-based (small), interactive animation format
- Flash movies (.fla, .swf) can be opened by Squeak
- Flash Movie Player
  - Supports stepping, viewing thumbnails, defining a subset of movie as a new movie
Flash Movie Characters are Morphs

- By making *everything* a Morph, can even manipulate Flash characters
Recording Sound

RecordingControlsMorph allows for recording sounds, trimming them, tiling them, or viewing and editing waveform.
SpectrumAnalyzerMorph

- Squeak can present FFT, signal, or songram views of sounds
Sound implementation

- Heart of Squeak’s sound support: 
  *AbstractSound*
  - Provides *play* and *viewSamples*
  - All sounds are a set of *samples* (A-to-D values)
  - *samplingRate* tells you the samples-per-second

- *SampledSound* is actually recorded

- Other sounds (like *FMSound* and *SequentialSound*) are synthesized into samples, then treated the same
  - *mixSampleCount:into:startingAt:leftVol:rightVol:* does the synthesis
Synthesized Instruments in Squeak

- Several synthesized instruments in Squeak
  - AbstractSound soundNames
  - Built out of Envelopes that define attack, sustain, and delay for sounds

- Can use them to play sounds in Squeak
  (AbstractSound noteSequenceOn:
   (FMSound soundNamed: 'brass1') from:
    #((c4 1.0 500) (d4 1.0 500) (e4 1.0 500))) play
  (Triplets are note, duration, and volume)
Importing/Exporting Sounds

- AIFF can be read and written
  - fromAIFFFileNamed: and storeAIFFOnFileNamed:
- WAV files read — fromWaveFileNamed:
- U-Law
  - uLawEncode: and uLawDecode:
- SoundCodec architecture for more sophisticated compression/decompression
MIDI Support

- MIDI Player, including ability to change orchestration, and Piano Roll to view score
Playing a MIDI file

f binary. “Open file for MIDI input”
“Read as Score.”
score := (MIDIFileReader new readMIDIFrom: f)
asScore.
f close.
“Open and play as ScorePlayer”
scorePlayer := ScorePlayer onScore: score.
scorePlayer reset; resumePlaying.
Squeak Movies: “SqueakTime”

- Not MPEG or QuickTime: Squeak-only format
- Requires frames to be input as Forms
  - For example, can dump movies to series of BMPs
- Eats up LOADS of disk space
  - But can be synchronized with MIDI
Synchronizing SqueakTime and MIDI

- Create thumbnails from Squeak Movies and drag into MIDI
BookMorphs

- Each page is a *PasteUpMorph*
  - Just like the World: Can hold anything

- Can be saved to disk or network
  - Understands file:, FTP, and HTTP URLs
    
    ```
    ```

- It’s an important step on the way to Dynabooks