Outline

Software layers
Graphical libraries
Window systems
User interface toolkits
Applications frameworks
Interface builders

Software layers

Application
Framework
User interface toolkit
Window system
Graphical library
Operating system
Device drivers

Output devices

Bitmap screens
CRT, LCD, Plasma, ...
Spatial resolution: about 100dpi
Color resolution (« color depth »):
B&W, grey levels, color table, direct color

Temporal resolution: 10 to 100 frames per second
Bandwidth:
25 img/s * 1000x1000 pixels * 3 bytes/pixel = 75 Mb/s
GPU: Graphics Processing Unit

(c) 2011, Michel Beaudouin-Lafon, mbl@lri.fr
Input devices

2D input devices
- Mouse, Tablet, Joystick, Trackball, Touch screen
- Type of user control: position, motion, force, …; linear, circular, …
- Mapping of input dimensions: position, speed, acceleration, transfer function (gain)
- Motor space vs. visual space: separate or identical

Other input devices
- Keyboards, Button boxes, Sliders
- 3D position and orientation sensors
- Simulated devices

Graphical libraries

Drawing model
- Direct drawing (painter’s algorithm)
- Structured drawing: scene graph
- Edit the data structure

Graphical objects are defined by:
- Their geometry
- Their graphical attributes: color, texture, gradient, transparency, lighting

Graphical libraries
- Direct drawing: Xlib, Java2D, OpenGL
- Structured drawing: Inventor (3D), SVG

Managing input in an interactive system

Query
- Blocking

Sampling
- Busy waiting

Events
- Event queue

Event-driven programming

while running do
  wait until event queue not empty // blocking
  ev := first event from queue // extract event
  target := findTarget(ev)
  if target ≠ NIL then target.handleEvent(ev)
end while

Very different from traditional algorithmic approach
Window systems

Organize display space in independent areas
Resource sharing

Window = autonomous area on the screen
- for display
- for input (event dispatching)

Window management
User interface: « window manager »
Application programming interface

Windowing models

Tiling

Overlapping

Hierarchical

Virtual screens

Window systems

Drawing model
Redraw hidden parts

Input management
Demultiplex event across applications
Concept of « focus »
New events
Window system:
request redraw, create/delete window
Input devices:
focus changes, cursor enters/leaves window

Client-server architecture

requests
multiplexing
display

events
demultiplexing
input

Client

Server

Client
User interface toolkits

Abstraction: the widget
Interactive object, component
Button, menu, scrollbar, dialog box, …

A widget = three facets
Presentation – Behavior – Application interface

Interface = widget tree
Nodes: containers (windows, menu bar, dialog box, …)
Leaves: simple widgets (buttons, scrollbars, …)

Widget layout

General rules
A widget is geometrically enclosed in its parent
The parent controls the layouts of its children

Layout algorithm
Natural size of each child
Final size and positions imposed by the parent
Constraints:
Grid, form, etc.

Dynamic layout

Facets of a widget

Presentation
Visual appearance
Configurable (« resources »)

Behavior
Reaction to user actions
Non configurable (or very limited)

Application interface
Notification of state changes

Application interface: callback functions

1. Registration of callback when widget is created

DoSave (...) { … }

2. Callback function is called when widget is activated

DoSave (...) { … }

Problem: « spaghetti » of callbacks
Sharing state among widgets and callbacks using global variables

DoSave () { SaveTo(filename) }

DoSave () { SaveTo(filename) }

global string filename;
DosetFile () { filename = …

(c) 2011, Michel Beaudouin-Lafon, mbl@lri.fr
**Application interface: active values**

Bi-directional link between a state variable of the widget and a variable of the application

- Problems: Limited to simple data types, Back link (widget to app) can be costly
- Advantages: Multiples views

**Application interface: message passing**

An object is associated to a widget, its methods are called when a state of the widget changes

- Problems: Multiples views
- Advantages: Better encapsulation

**User interface toolkits**

- Many available toolkits: Xt, Motif – historical (X Windows), Qt, GTK – Linux, AWT, Swing – Java, Tk/Tk – multi-plateformes [active values]
- Many limitations: Programming is cumbersome, Interaction limited to the interior of the widget, Limited extensibility: adding new widgets types is difficult

**Application frameworks**

- Application skeleton: Incomplete code: general structure of the application includes what is not supported by the toolkit, Global structure of the application, Global functions (history, copy-paste, …), Non-widget interaction (e.g., drag-and-drop)
- Shows the limitations of the programming language
- Example: MacApp (Apple, 1986): Concept of a document (content of a window), Concept of action (that can be done and undone)
Interface builders

Description (text or graphics) of part of the interface
Generation of a runnable version

Editor
Functional core
Run-time module of interface builder

Interactive application

Interface builders

Generating the final application

Compile
Functional core
Run-time module of interface builder

Interactive application

Interface Builder

NeXT, then Apple

Conclusion

Advantages of these tools
- Reduce development and maintenance costs
- Facilitate compliance with style guides

Limitations of these tools
- Interaction style based on widgets
- Limited extensibility
- Difficult to program non-standard interactions

Research issues
- Beyond the widget model
- Define better languages and environments