GUI Programming & Interaction

GUI Programming

- Assume the display is an image
- How do we write code for it?
- And how do we tie it into the GUI?
  - *Graphic User Interface*

GUI Concepts

- Modal vs. Non-Modal Interfaces
- Desktops, Windows & Widgets
- Events, Event Loops & Event Handling
- Mouse Interaction
- GUI Toolkits

An Old Text Interface

```
Enter First Number:
7
Enter Second Number:
what second number?
Badly formed input.
Enter Second Number:
Dammit! I wanted to square it!
Badly formed input.
^-C
> Process Terminated
```

Modal Interfaces

- This interface is modal
  - it traps you in a mode
  - driven by the computer
- It's like being interrogated
- Not a good model for interaction
- But an application is still a mode

Non-modal Interfaces

- Basis of most modern GUI's
- Put the human in control
- Have the computer respond to the user
- Commands or actions applied to objects
- Computer displays result
- Based on visual metaphors

Desktop Metaphors

- The screen is your desk
- Files represent documents
- Directories represent folders
- Hard drive is the filing cabinet
- Use mouse to move objects

Communication

- Human uses mouse & keyboard
  - mouse to select (pick up) objects
  - mouse (menus) / keys to act
- Computer uses screen to respond
  - programs have visual representations

Applications

- Actions are applied to objects
  - Print a document
  - Copy a document
  - Edit a document
- So a program is an application

Select & Operate

- Use mouse to select by clicking (once)
- The object the user wants to affect
- Highlight selection with visual change
- Apply command to target
- Display results of command
- Do commands always have a target?

Inside An Application

- Window is the visual representation
  - metaphor: window on an envelope
    - lets you look inside the document
  - documents contain smaller objects
    - paragraphs, words, letters, numbers
      - select & act metaphor continues

Windows & Widgets

- Window is visual representation of data
  - divided into widgets - visual entities
    - data displays - show data
    - user controls - manipulate data
- We will work *directly* with a window
  - i.e. we have only one widget

Sample Widgets

Window controls

Menu Bar

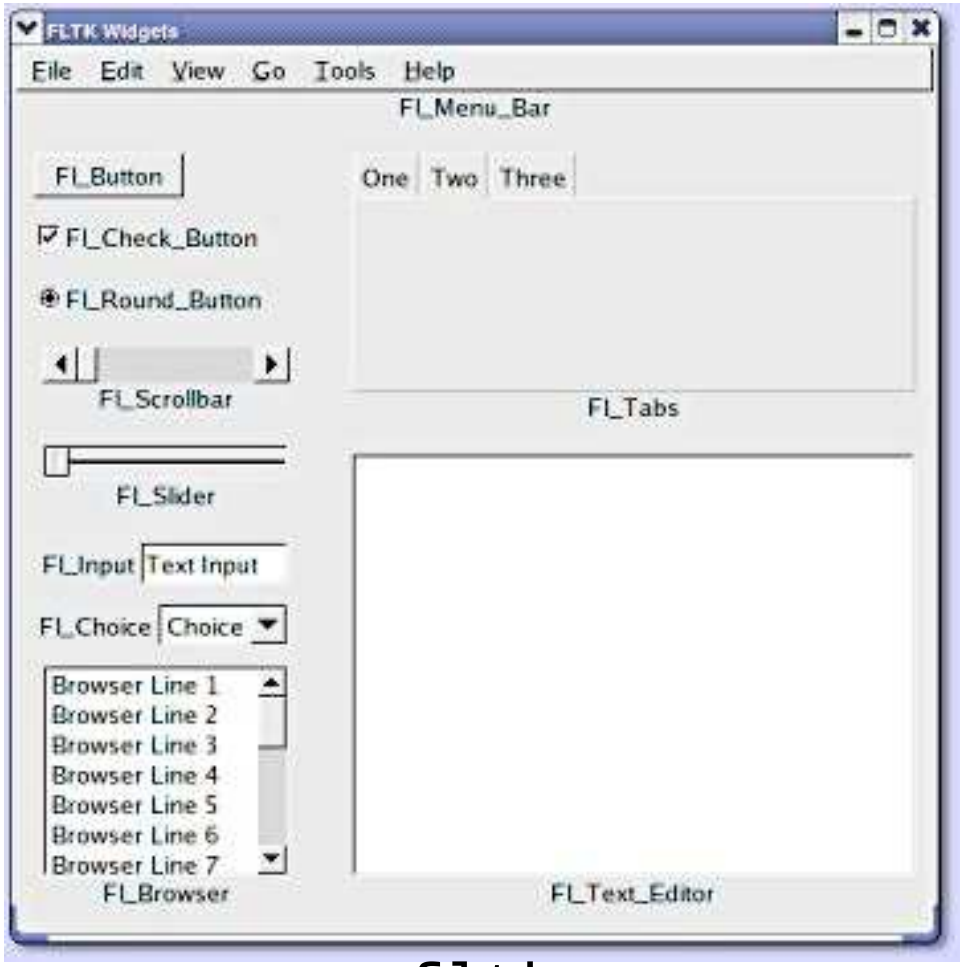
Button

Checkbox

Scrollbar

Slider

Text Input

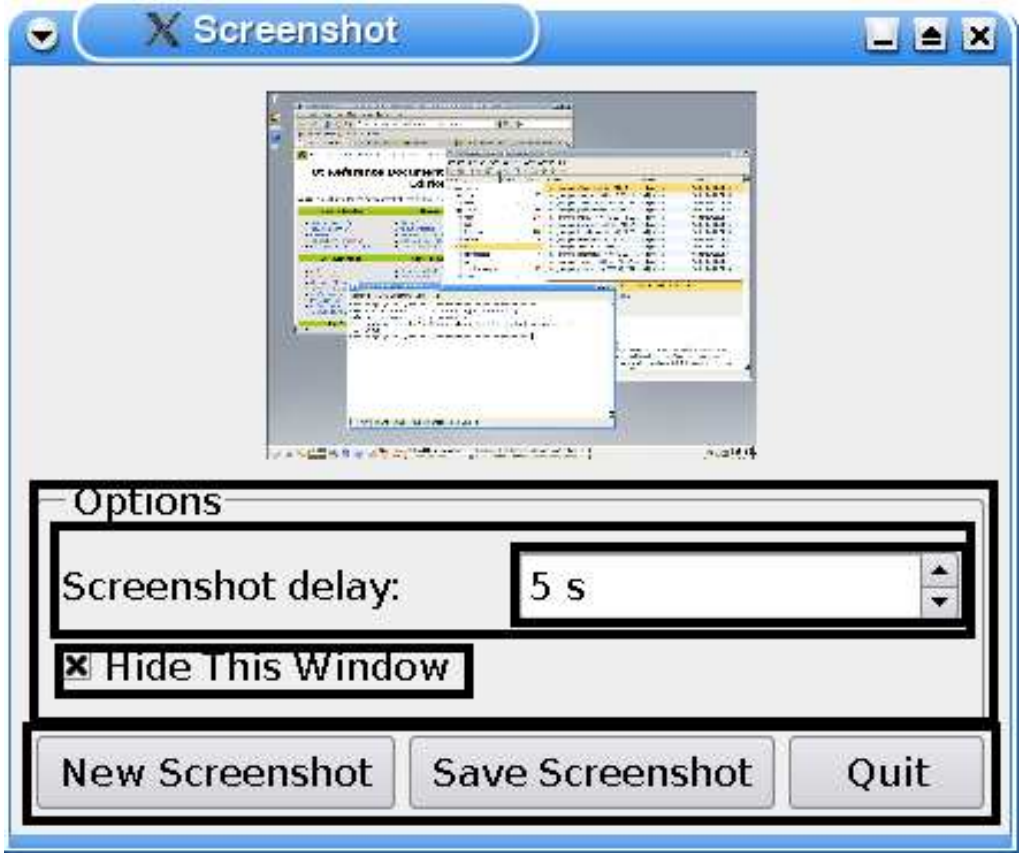


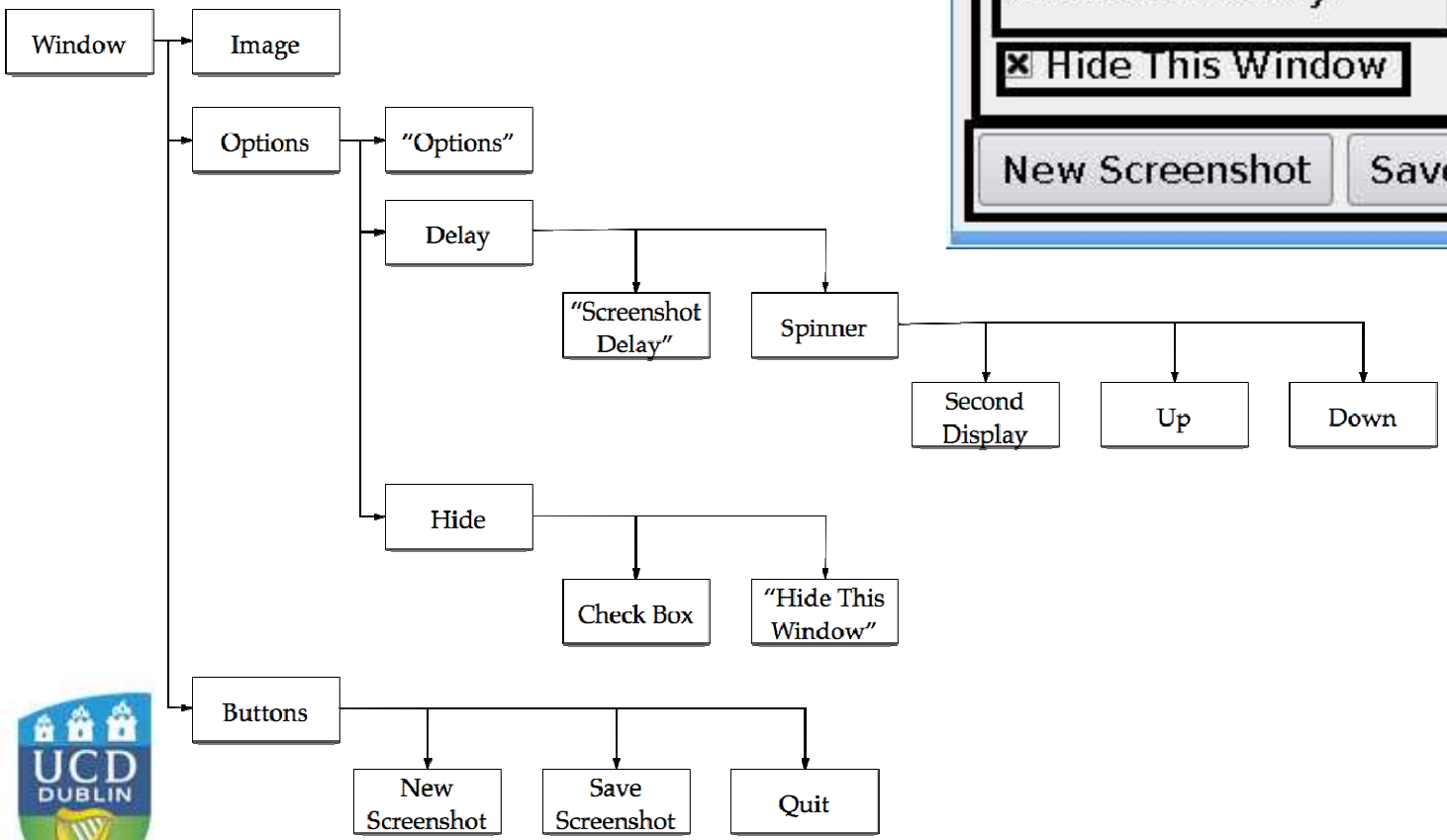
www.fltk.org

The Visual Hierarchy

- Widgets are inside each other
- They form a natural hierarchy
  - visual hierarchy
  - tree structure
  - the window is also a widget
- Often used for event dispatch

Visual Hierarchy





Graphics Contexts

- Want to render one widget at a time
- Inconvenient to draw directly
- Want independent coordinate systems
  - in fact, independent drawing state
  - called a graphics context
- So context switching costs apply

Events

- Based on non-modal commands
- Program waits for user input
  - then responds
- Program sits in event loop
  - processing events as they occur
  - user events, system events

Event Loop

- The event loop is a simple loop:

```
while (not done)
  {
  CheckForMouse();
  CheckForKeyboard();
  CheckSystemEvents();
  ProcessEvents();
  UpdateDisplay();
  }
```

```
while (not done)
  {
  event = system.GetEvent();
  ProcessEvent(event);
  UpdateDisplay();
  }
```

  - Just keeps checking user input
    - gets very complicated

Event Types

- Three fundamental types:
  - keyboard events
  - system events
  - mouse events
- Each representing user actions

Keyboard Events

- Simplest type of events
  - User presses one or more keys
- But the processing can be complex
  - Each key is a different command
  - Mnemonics are important

System Events

- Idle processing (periodic)
  - when no other events to process
- Incoming network events
  - &c., &c., &c.
- Usually treated as commands

Mouse Events

- Mouse buttons can be:
  - pressed (initially):
  - dragged (at multiple positions)
  - released (finally)
- Display should reflect events visually

Mouse Semantics

- Mouse down hits an object:
  - select it or activate it
- Mouse down misses all objects:
  - unselect current selection
- In either case, drag events then occur
- Drag events end with mouse up

Effects of Drag

- Mouse down initiates action by user
  - highlight effect of action (repeatedly)
  - update with each drag event
  - commit when done (on mouse up)
- This is called rubberbanding

Drawing a Line

- On mouse down, create temporary line
  - both ends at location of mouse
- On mouse drag, update one end
- On mouse up, commit the line to memory
  - i.e. add it permanently

GUI Toolkit Paradigms

- Lightweight Java Game Library
- Direct Event Processing (Mac OS 1)
- Callback Functions (X / GLUT / GLUI)
- Live Variables (GLUI)
- Widgets & Virtual Functions (Fltk, QT, AWT)
- Message Passing (Qt, MFC)
- Event Handlers (Java Swing)

Direct Processing

- Hand-coded event loop

```
Initialize();
while (not done)
  {
  CheckForMouse();
  CheckForKeyboard();
  CheckSystemEvents();
  ProcessEvents();
  if (noEvent)
     Idle();
  UpdateDisplay();
  }
CleanUp();
```

```
Initialize();
while (not done)
  {
  event = system.GetEvent();
  ProcessEvent(event);
  if (noEvent)
     Idle();
  UpdateDisplay();
  }
CleanUp();
```

- Present in *every* program
- So it *belongs* in a library

Event Loop Library

- Isolate common code in a library
  - retrieving events from the O/S
  - setting up graphics contexts
  - mapping windows to screens
  - event loop iteration
- Specialize code with *callback* functions

Callback Processing

- Could have standard functions to fill in
- But not very flexible
- Instead, have standard *type* of function
- Pass the function to the library
- Library remembers function
- *Calls* the function *back* when it's needed

Callback Implementation

- C uses *function pointers*
  - a pointer to a function in memory
  - called by dereference: (*myFunc)(x)
- Pass a *callback* function for each event
- C++ / Java can use *function objects*
  - an encapsulated function pointer

Callback Processing

(Lightweight Java Game Library example)

```
while (!Display.isCloseRequested()) {
    int delta = getDelta();
    update(delta);
    renderGL();
    Display.update();
    Display.sync(120);
}
```

LWJGL Structure

- Data usually stored as *globals*
- Callback functions passed to library
- Data processing embedded in callbacks
- Display callback not automatic
  - **Display.*update()*** requests on next loop

Scaling Up with OOP

- Widgets are naturals for OOP
- Each widget type is a class
- Shared behaviour in parent classes
  - *object hierarchy* - class design / compile
  - *visual hierarchy* - GUI design / load
  - *command hierarchy* - event dispatch / run

Object Hierarchy

- A window is also a widget
- But not all widgets are windows
  - although most are *subwindows*
  - *aka views, panes, canvasses, panels, frames*
- So Widget is the base class
  - Window is the derived class

Command Processing

- Commands are *processed* by:
  - widgets
  - controllers
  - event handlers
- depending on the toolkit

Widget Focus

- Each widget processes events differently
  - we need to decide *which widget*
  - based on *focus* - i.e. user's intention
  - *focus follows mouse* - X standard
  - *sticky focus* - Mac / Windows

Focus Follows Mouse

- All events happen under cursor
  - *including* keyboard events
  - disconcerting if mouse jiggles
  - arguably more powerful

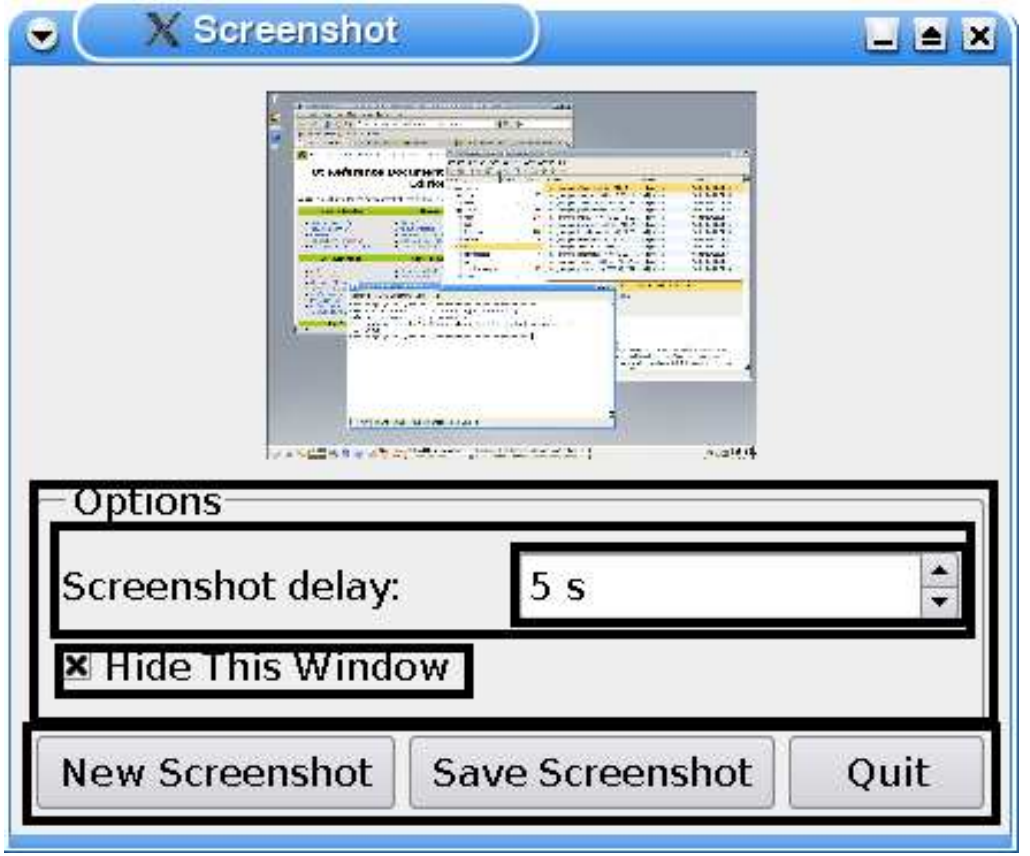
Sticky Focus

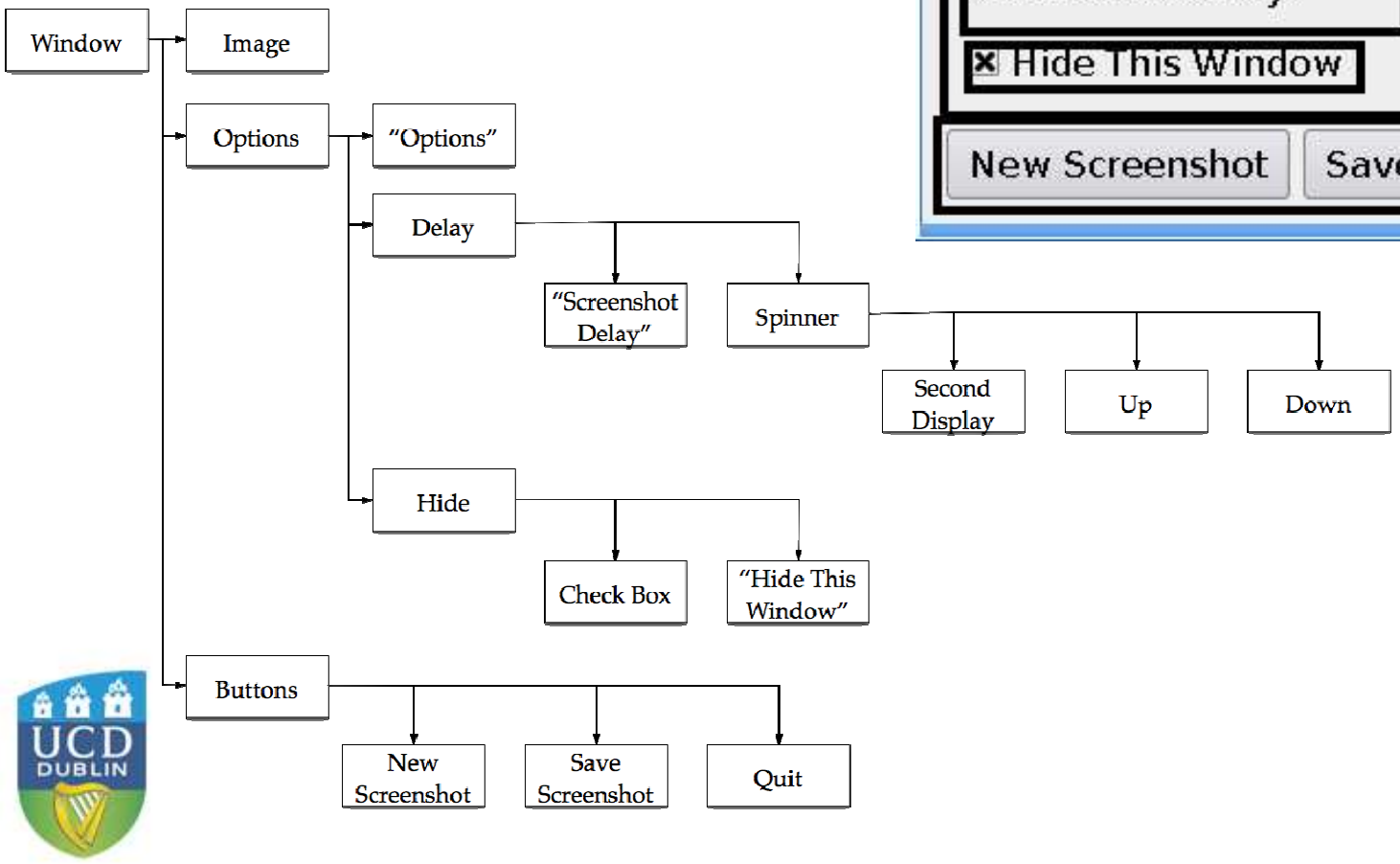
- Mouse events occur at mouse location
- Keyboard events occur in *active widget*
  - i.e. last widget *chosen* by user
  - arguably easier for user

Event Location

- In either case, events have a *location*
  - a cursor location, or
  - a specific window or widget
- How do we *find* the right widget?

Visual Hierarchy





Visual Layout

- Each group is *horizontal* or *vertical*
  - how children are laid out visually
- What is the layout of the window?
- Positions can be given directly
  - or left up to a *layout manager*

The Main Event Loop

```
int MouseX= Mouse.getX();
int MouseY= Mouse.getY();
int WheelPostion = Mouse.getDWheel();
    boolean  MouseButonPressed=
Mouse.isButtonDown(0);
```

```
if(MouseButonPressed && !MouseOnepressed )
 {
 MouseOnepressed =true;
  MyArcball.startBall( MouseX, MouseY, 1200, 800);
 dragMode=true;
 }else if( !MouseButonPressed)
 {
 MouseOnepressed =false;
 dragMode=false;
 }
 if(dragMode)
 {
 MyArcball.updateBall( MouseX  , MouseY  , 1200, 800);
 }
```

Command Hierarchy

- Commands are *delivered* to *target*
- If target doesn't process it, who does?
- Someone more important
  - i.e. recurse *up* the visual hierarchy
  - or follow some other rule
    - i.e. there is a *command hierarchy*

Hierarchy Problems

- Requires custom widgets all the time
  - less reuse of code
- Live variables were good, but they trigger full update every time
- Better to update only *relevant* views

Message Passing

- Each widget has a *message* it passes
  - to one or more *target* objects
- Which then process the event
- Implemented as messages, signals/slots, coded values, &c.
- MFC, Qt, BeOS for examples

Message Problems

- Message class is hard to define
- Each target type of data is custom
- And data doubles as controller
- Solution: separate events from data
  - *Model-View-Controller* paradigm

Model-View-Controller

- Current GUI paradigm
- Data is stored separately in a *model*
- Display is shown in a *view*
- Events are processed by a *controller*
- Separates visual & logical processing

Event Handlers

- Separate controller for each object
- Attached explicitly to data object
- Only process desired events
- Usually through virtual functions
- Example: Java Swing

GLUI

- 3D controls (e.g. arcball)
- Live variables
- Simple to add to GLUT
- Limited set of widgets
  - e.g. no file dialogs
- Single theme (old X-Windows)

Fltk

- Wide range of widgets
- Single theme (old X-Windows)
- No 3D-specific controls
- Event processing all in one function
- OpenGL view available

Qt

- Wider range of widgets
- Commercial, so well-supported
- Uses OS-native widgets
- Simpler event processing
- Somewhat trickier install / compile
- Limited 3D controls, but OpenGL

Java AWT

- Simple hierarchical processing
- Lots of widgets
- Platform-independent
- No 3D support worth mentioning
- No OpenGL, either

Java Swing

- Event Handlers, so good model
- Complex to program for simple tasks
- Java performance penalties
- Poor 3D support
- OpenGL just barely available

MFC (windows)

- Yes, it does OpenGL
- (updated for windows 10)
- Uses a variation of message passing
- Fairly messy code structure
- But lots of people use it

Interactive 3D scenes

- Implementing 3D scenes / games requires that the 3D element interact with each other in a logical manner
- Collision detection is the first step but you can develop a full physics engine for a 3D environment
- The most popular physics engine for 3D games and simulations is HAVOK
- Developed in Dublin

Collision Detection Basics

- Bounding Box or a Bounding Sphere
- Check if an object is within a range of another using a basic calculation using a box .
- This is fast normally only 6 sides to detect if it's a cube.
- Not the most accuracy, a better approach would be to use a sphere to check. Using the centre point of object and a radius.
- The most accurate approach would be to check vertex directly and see if they intersect ,