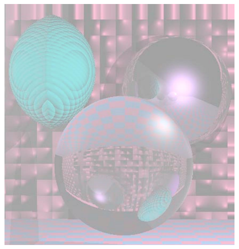
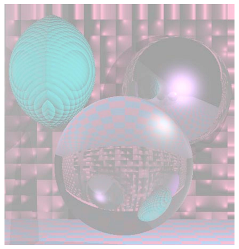


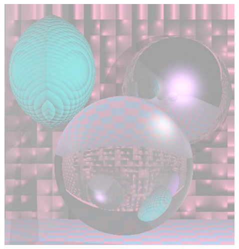
CS 4300 Computer Graphics

Prof. Harriet Fell
Fall 2011
Lecture 8 – September 22, 2011



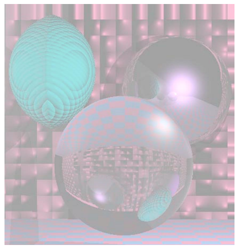
GUIs

- GUIs in modern operating systems
- cross-platform GUI frameworks
- common GUI widgets
- event-driven programming
- Model-View-Controller (MVC) architecture
- common user interaction techniques



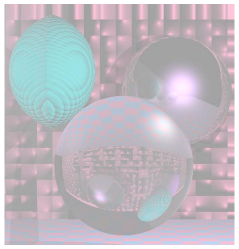
GUIs in Modern Operating Systems

- all modern desktop operating systems support a *graphical user interface (GUI)*
- *these are also called windowing environments because the most common paradigm, initiated in the early 80’s at Xerox’ Palo Alto Research Center (PARC), is to have a desktop where one or more overlapping windows may exist, each containing the GUI for a currently running application*



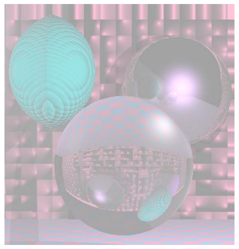
X Window

- the standard windowing environment for most modern variants of Unix (except OS X)
- has been around a long time but is continually updated
- variant used in most modern GNU/Linux distributions is currently managed by the x.org foundation
- X is a client-server architecture
- typically, a single instance of an X server runs on the machine, and has the responsibility for all direct interaction with output and input devices
- individual applications, such as Firefox, are X clients



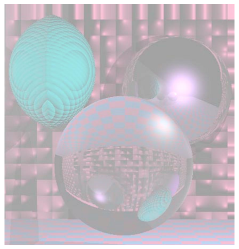
X Clients

- X clients can communicate with the X server over several different types of connections
  - standard TCP/IP sockets
    - this enables the X client and server to actually run on different machines on the network
    - note that the roles of “client” and “server” can be non-intuitive here
  - several other Inter-Process Communication (IPC) mechanisms
    - including “Unix domain sockets” and shared memory
    - these generally focus on improving performance in the case where both client and server are running on the same machine



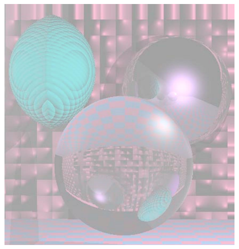
X Protocol

- the most important part of the X system is the *protocol that defines the communication between client and server*
- *the X protocol is an open standard*
- *different organizations can implement both clients and servers, and if they all stick to the defined protocol, the programs will inter-operate*



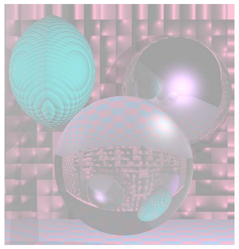
X Servers

- there exist X servers that run on both Macintosh OS X and on Microsoft Windows
  - this means that you can, in theory, run an X *client on a remote machine (e.g. a GNU/Linux machine to which you have established an SSH connection), and have that program display its interactive GUI on your local machine, which may be running Windows or OS X*
  - *also, this can ease porting of applications, since most of the GUI code can remain the same, assuming that an X server is available on the target platform*



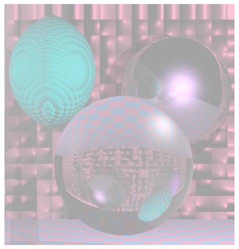
Macintosh OS X

- the original Macintosh OS was one of the first commercially successful GUI systems
  - copied many aspects of earlier prototypes from PARC (overlapping windows, mouse, etc)
- the modern version, OS X, is actually a Unix variant, with a GUI adapted from an earlier system called NextStep
  - the main GUI framework is called Cocoa, and is natively programmed in Objective-C
  - also comes with an X server, mainly used to ease porting of Unix applications



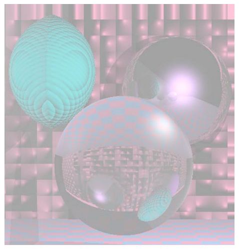
Microsoft Windows

- currently holds the largest market share
- several X servers are available as 3rd party software



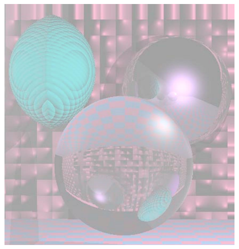
Cross-Platform GUI Frameworks

- because X Window, OS X, and MS Windows all require different application code, there now exist a number of libraries which ease the work of porting applications among the three major desktop OS
- these all provide a set of standard widgets—including windows, buttons, toolbars, etc. (more details later today)—which “look and feel” similar on different OS



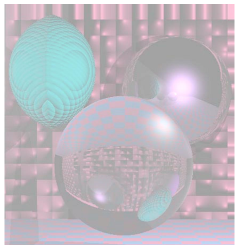
GTK+

- is the “Gimp ToolKit”, which evolved out of initial work on the GNU Image Manipulation Program (GIMP)
- written in C, but has bindings for many other languages
- main toolkit used in the GNOME desktop environment in GNU/Linux
- LGPL



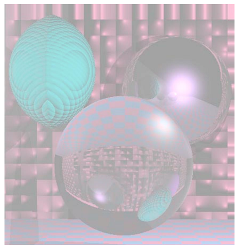
Qt

- was originally developed by the Norwegian company Trolltech, which was recently bought by Nokia
- written in C++, but has bindings for many other languages
- main toolkit used in the KDE desktop environment in GNU/Linux
- LGPL



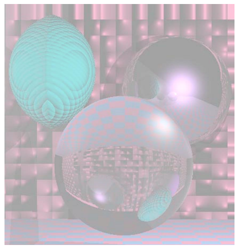
JFC or the Java Foundation Classes

- Abstract Window Toolkit (AWT)—the original Java GUI framework
  - largely supplanted by Swing, but still comes into play in many cases
  - Java is intended to be a cross-platform applications development environment
  - AWT attempts to map different platform-specific GUI frameworks into one least-common-denominator API
  - when you create widgets in AWT, you are directly creating widgets in the underlying OS-specific GUI framework



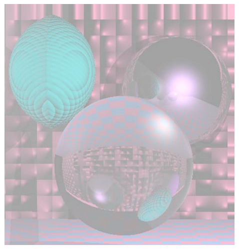
JFC - continued

- Swing—introduced to supersede AWT in Java 1.2
  - unlike AWT, the architecture of swing is to implement most widgets directly in Java
  - only the most basic windowing functions are used from the underlying OS-specific GUI framework (via AWT)
  - most widgets inside the window are entirely rendered in Java
  - this allows a consistent “Java look-and-feel” across all platforms
  - also allows support for more advanced features, such as high quality antialiased rendering, that are not in the least-common-denominator of the OS specific frameworks
  - can be slower than AWT, but modern implementations of Swing are highly optimized
- Java2D—the actual drawing APIs in JFC



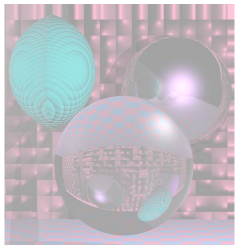
Common GUI Widgets

- most of these can be demonstrated with the SwingSet demo included with most Sun Java Development Kit downloads (search for a file named “SwingSet2.jar”)



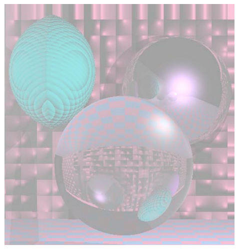
Windows

- some frameworks, in particular MS Windows, use the term “window” to apply to nearly any rectangular widget on screen
- the actual outer *container of an application is specifically called a “top-level” window or “frame”*



Windows

- *window (frame) decorations*
  - *title bar*
  - *border and resize handles*
  - *window buttons*
- *menu bar*
- *Toolbar*
  - *often just provides a convenient replication of the same actions available from the menu bar*
  - *this is a good thing: the menu bar is complete, but can be complex and inconvenient; the toolbar may not be complete, but it's simpler and more convenient*
- *status bar*
- *child windows*
  - *also called "internal frames" (Java) or "MDI" (Multiple Document Interface) (Windows)*

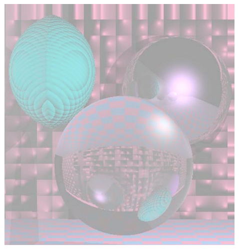


Buttons

- “regular” buttons
- Radio buttons
  - RadioButtonDemo.jnlp
- toggle buttons, aka checkboxes
  - CheckBoxDemo.jnlp

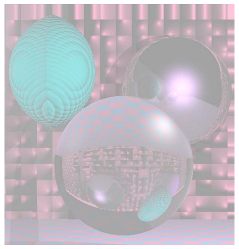
Sliders

esc, Select image, View, Formatting Pallet



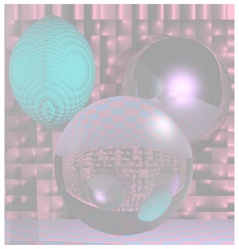
Widgets in Java

- http://download.oracle.com/javase/tutorial/uiswing/components/componentlist.html
- combo boxes
- dialog boxes
- file choosers
- standard “option panes” and message dialogs



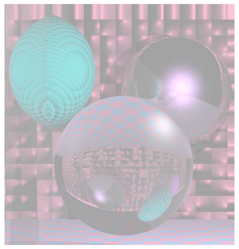
text layout

- may provide a variety of features including
  - font rendering
  - text justification
  - HTML or other “rich content” layout
  - text selection and editing
  - lists, tables, and trees
  - progress bars
  - scroll bars
  - split panes and tabs
  - tooltips



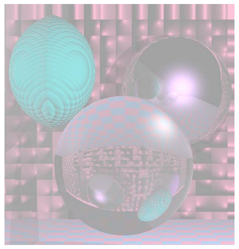
Event-Driven Programming

- the computation requirements of GUI programs differ from more “traditional” programs
- the program may have nothing to do for long periods as it “waits” for the user to do something
- multiple things can be going on at once in different parts of the GUI
- the actual tasks the program needs to perform may evolve at runtime as the user e.g. opens and closes documents



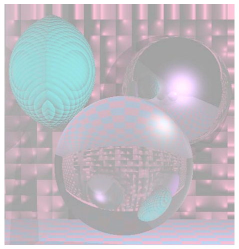
event-driven style

- main idea: a variety of *events may occur asynchronously*
  - *triggered either by the user (e.g. hitting a key or moving the mouse)*
  - *or by the system (e.g. a window from another application is moved on top of our window; an object is dragged from one application to another; the system is shutting down)*
- *application code specifies which events it is interested in handling*
  - *e.g. by registering event listener or callback function*



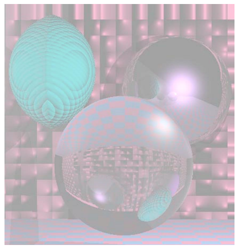
overall structure of an event-driven system

- *loop forever*
  - *wait for an event (without burning CPU)*
  - *dispatch: see if any handlers have been registered for the event, and if so, invoke them*
- *event handler code gets invoked as necessary*
- *unhandled events may be handled in a default way by the GUI framework or by the OS, or may simply be dropped*



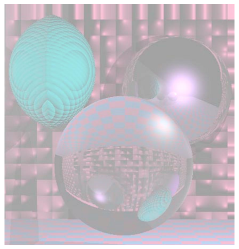
Event Handler Code

- **typically all runs from within a single thread**
  - events may come in various orders, but are typically at least processed one at a time
  - reasons for this are essentially about managing complexity and ensuring *thread safety of all the data structures that implement the GUI*
- *a good reference on concurrency as it relates to GUI programming is chapter 9 of Java Concurrency in Practice*



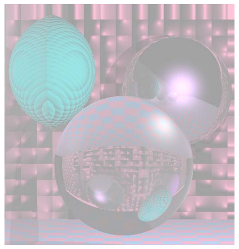
Implications of single-threaded design

- Keep your event handling code short and fast. If you spend a lot of time handling one event, you may be blocking the processing of later events (they will typically be queued).
- If you need to make modifications to any GUI data structures (e.g. opening a new window, or adding a widget to an existing window, or even changing the label of a button) outside of an event handler, you must take special care to ensure thread safety. In Java, one way to do this is to use `SwingUtilities.invokeLater()` or `invokeAndWait()`.



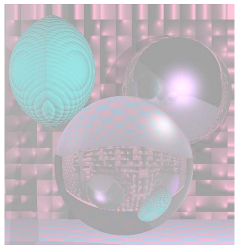
Model-View-Controller (MVC) Architecture

- the **model** is the set of core data structures defining the state of your application
  - e.g. in a drawing application, this could be a list of all objects (line segments, circles, curves, etc) currently in the drawing, along with all the current settings of their parameters
- one or more **views** of the model may be open; each shows a *depiction of the model; each view may have a particular viewpoint, e.g.*
  - *a multiplayer game could have different views showing the game world from the perspective of each player*
  - *a drawing program could have one view that actually shows the drawing, and another view that shows a textual list of all the objects in the drawing.*



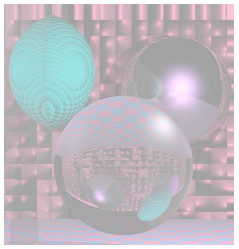
Model-View-Controller (MVC) Architecture

- the **controller** is all the event handling code processing events that may
  - alter the model itself, e.g. adding a circle in the drawing program
  - modify the state of views, e.g. the view from the perspective of a specific character must change viewpoint when the character moves
  - add or remove views
  - change the state of the program, such as minimizing or quitting



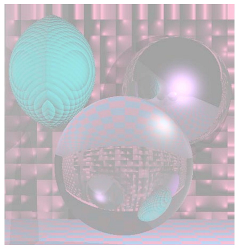
Common User Techniques

- picking and selecting
  - the user clicked the mouse. How does your application know *what was clicked?*
  - *what if multiple graphical objects are on top of each other?*
  - *the user may want to pick more than one thing at a time*



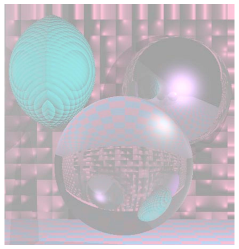
cut and paste

- basic idea is well known
- possible complexity: cut and paste is sometimes meant to work even between applications
- the OS (or at least the desktop environment or windowing system) must manage a shared resource called the clipboard
- what is the format of data in the clipboard?
- how does your application know that it is ok to change the contents of the clipboard?
- how does your application know what to do with any kind of data that the user may try to paste from the clipboard?



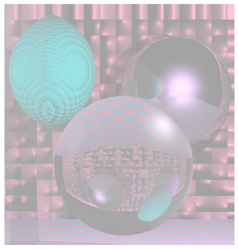
dragging

- user presses mouse button down over an object
- while continuing to hold the button, user moves mouse; object "follows along"
- user releases button; object "stays put"
- "object" can be either
  - a distinct graphical entity, e.g. an image in an image manipulation program, or
  - the viewpoint of the user itself: this is *navigation, which we will cover in more detail later in the course*
- *one complexity of implementing dragging is that separate events are typically delivered for the mouse press, each incremental motion of the mouse, and the mouse release*
  - *no guarantees that you will get these in any particular sequence!*



Drag and Drop

- a special case of dragging which is essentially a shortcut way to cut and paste
- sounds simple, but actual implementation can involve a lot of engineering and debugging
- again, the OS may become involved to support Drag and Drop across applications



Modes and dialog boxes

- a graphical application is “modal” if it can be put into a state where only some of its functionality is available, or if some special functionality is only available in that state
- common cases: modal dialog boxes, “wizards”
- can be a good thing, but also an argument for avoiding modality