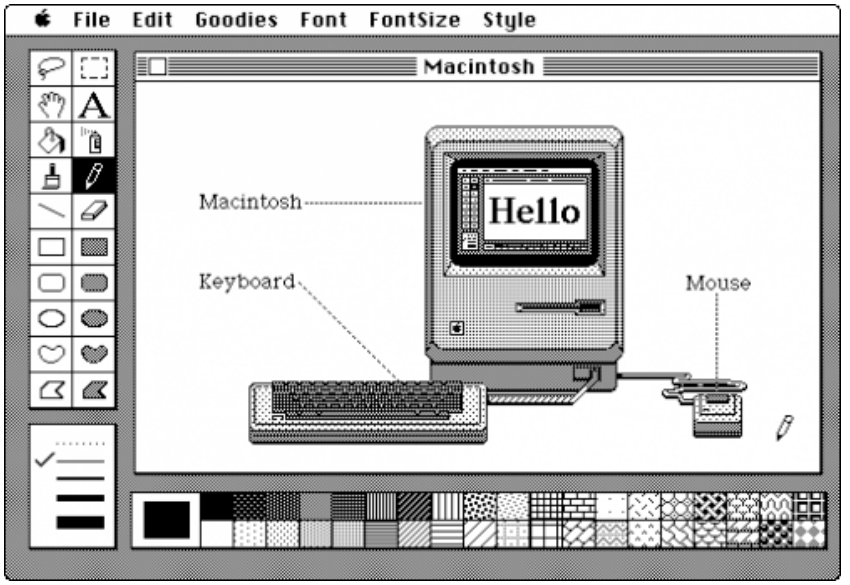
Modern GUI Systems

Life after X Windows

Apple Macintosh GUI (1984)

- Hardware interface
  - High resolution, high refresh graphics display
  - Keyboard
  - Pointing device (e.g., mouse)

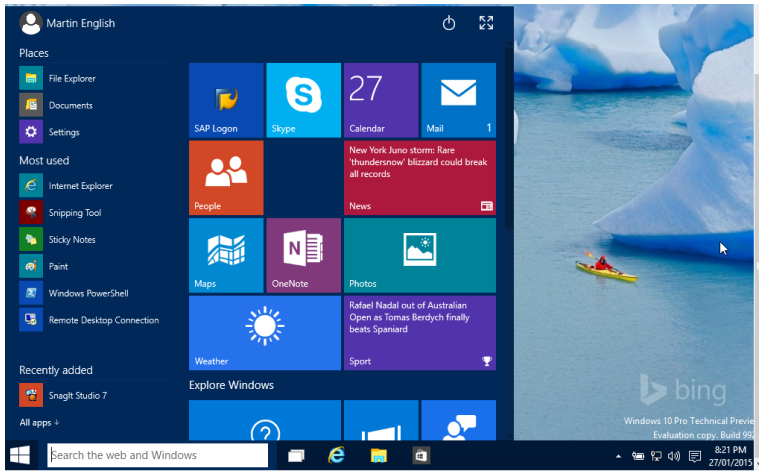
- WIMP interface
  - Windows, Icons, Menus, and Pointer



window icon menu pointer

Modern GUI (present)

Desktop (WIMP)



Tablet (Touch/Pen)

Smartphones (Touch)

Hybrids (WIMP+Touch +Pen)

window icon menu pointer

Is X Windows still applicable?

- X Windows hasn't aged very well
  - Based on standard C, not C++/OO
  - Built using conventions at that time (enums, int constants)
- Modern toolkits are object-oriented, and have better abstractions to handle events, drawing.
  - Required to make systems more extensible.
  - Also helpful to scale-out to more input and output devices.
- Event model assumes a single event loop
  - As we'll see, this scales poorly!
  - Would prefer this to be handled by the underlying system (i.e. not at the application level)

What stays the same?

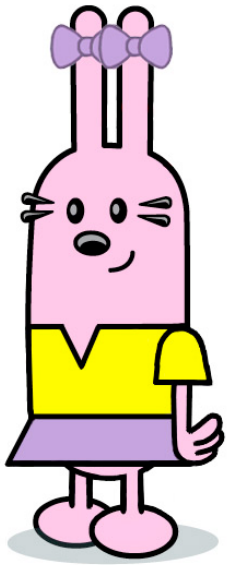
- Core requirements haven't changed
  - Need to manage events from multiple input devices
    - Events routed to appropriate application/windows
    - More devices supported but underlying mechanism for handling events is the same
  - Primarily 2D interfaces
    - Still need optimized 2D graphics
      - Clipping, double-buffering etc.
  - Input is still primarily text + positional
    - Variation in keyboards, but we still use standard layouts
    - Variation in pointing devices, but we still need a pointing device to return x,y coordinates (i.e. what to activate)

What's changed?

- GUI toolkits are more prevalent
  - Stock components (widgets) often provided by OS vendors.
  - Supports standard look-and-feel with minimal coding.
  - Have been expanded to support additional devices/platforms.
- New input modalities: speech, touch, pen
  - Non-mainstream but becoming more popular
  - Often complementary
- New output modalities: VR, AR
  - *Definitely* non-mainstream
  - Challenges to each

Widgets

- Widget (control, component) is a generic name for parts of an interface that have their own behavior: buttons, progress bars, sliders, drop-down menus, spinners, file dialog boxes etc.
  - Can have their own appearance
  - Receive and interpret their own events
  - Put into libraries (toolkits) for reuse
- Modern systems supply a toolkit of standard GUI components; provides applications with a standard look-and-feel, and common event model.



Widget from ***Wow Wow Wubbzy***

What has replaced X Windows?

- We have multiple operating systems/platforms/toolkits
  - Market share split: Microsoft (desktop), Google (mobile)
  - Apple (desktop+mobile)
- OS vendors typically provide toolkits and development tools/languages, often intended to lock-in to their platform.
  - Microsoft Windows: C#
  - Apple macOS: ObjC/Swift
  - Apple iOS: ObjC/Swift
  - Google Android: Java/Android
- Multi platform tools are rare or have limited capabilities
  - e.g. Java, Javascript/CSS/HTML (truly cross-platform)
  - e.g. Swift, .NET core (OSS but limited)

What's a programmer to do?

- Decision 1: Decide which platforms you want to support
- Decision 2: Choose appropriate languages/tools
  - Platform-specific vs. cross-platform languages and tools
    - e.g. Java for Android, Swift for iOS, C# for Windows, or build web interfaces using JS/CSS
    - e.g. JS/CSS using Electron, or Java everywhere
  - "Best" tools and support is often platform-specific.
- Decision 3: Stock widgets vs. custom
  - Widget toolkits and common look-and-feel were dominant for years (e.g. standard-looking Windows apps).
  - Shift towards heavy customization, and custom controls
    - Allows for more creative expression, removes toolkit limitations
    - e.g. a streaming music app without a title-bar