Worldwide Developers Conference

Imaging Under Rhapsody

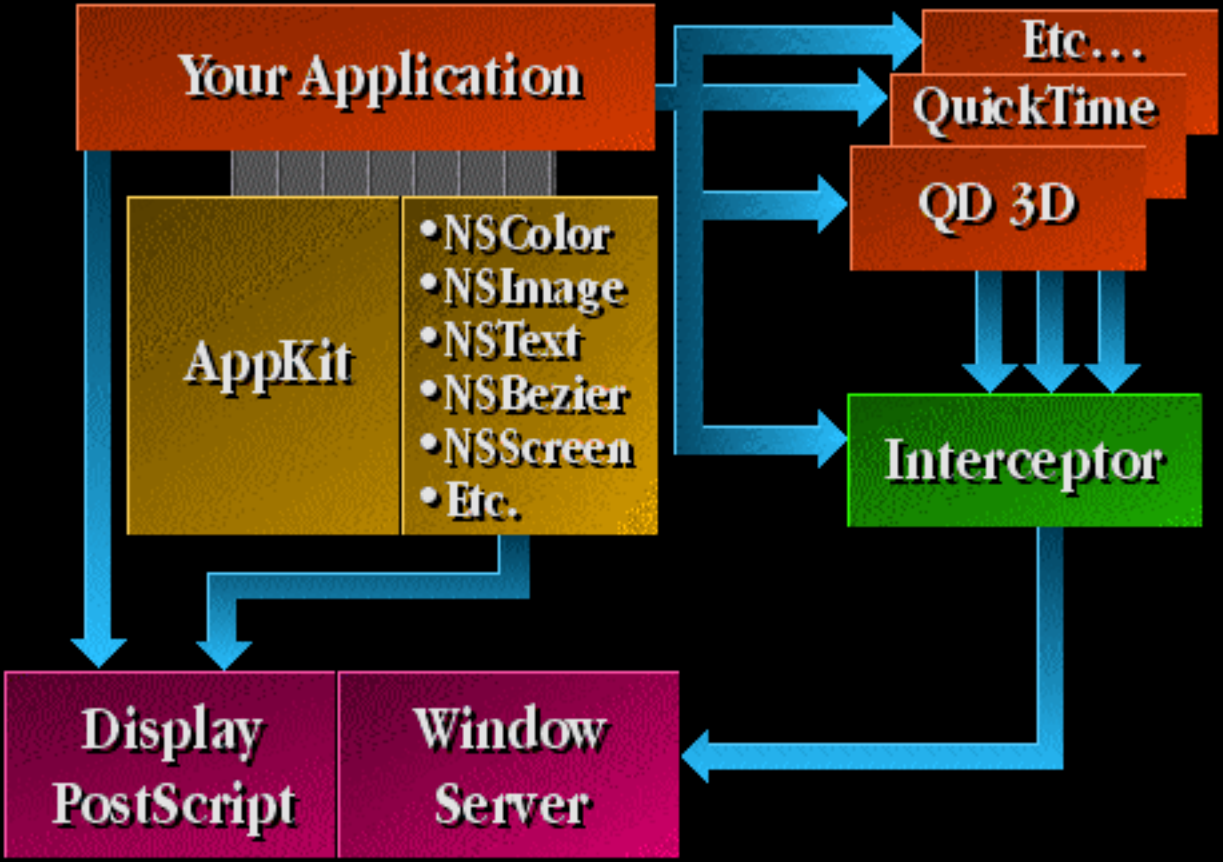
Michael Peirce

Rhapsody Graphics Group Manager

Graphics Session Roadmap

- Rhapsody Graphics Overview
  - Michael Peirce
- Using Graphics in the AppKit
  - Jeff Martin
- Display PostScript
  - Peter Graffagnino
- Interceptor
  - Mike Paquette
- Q & A

The Graphics Architecture



What to Use When

- AppKit Graphics Classes
  - Covers all the common graphics needs
  - 90%+ of applications should use AppKit graphics classes exclusively

What to Use When

- AppKit Graphics Classes
  - Covers all the common graphics needs
  - 90%+ of applications should use AppKit graphics classes exclusively
- Display PostScript
  - Used by smaller percentage of applications that need power of PostScript language

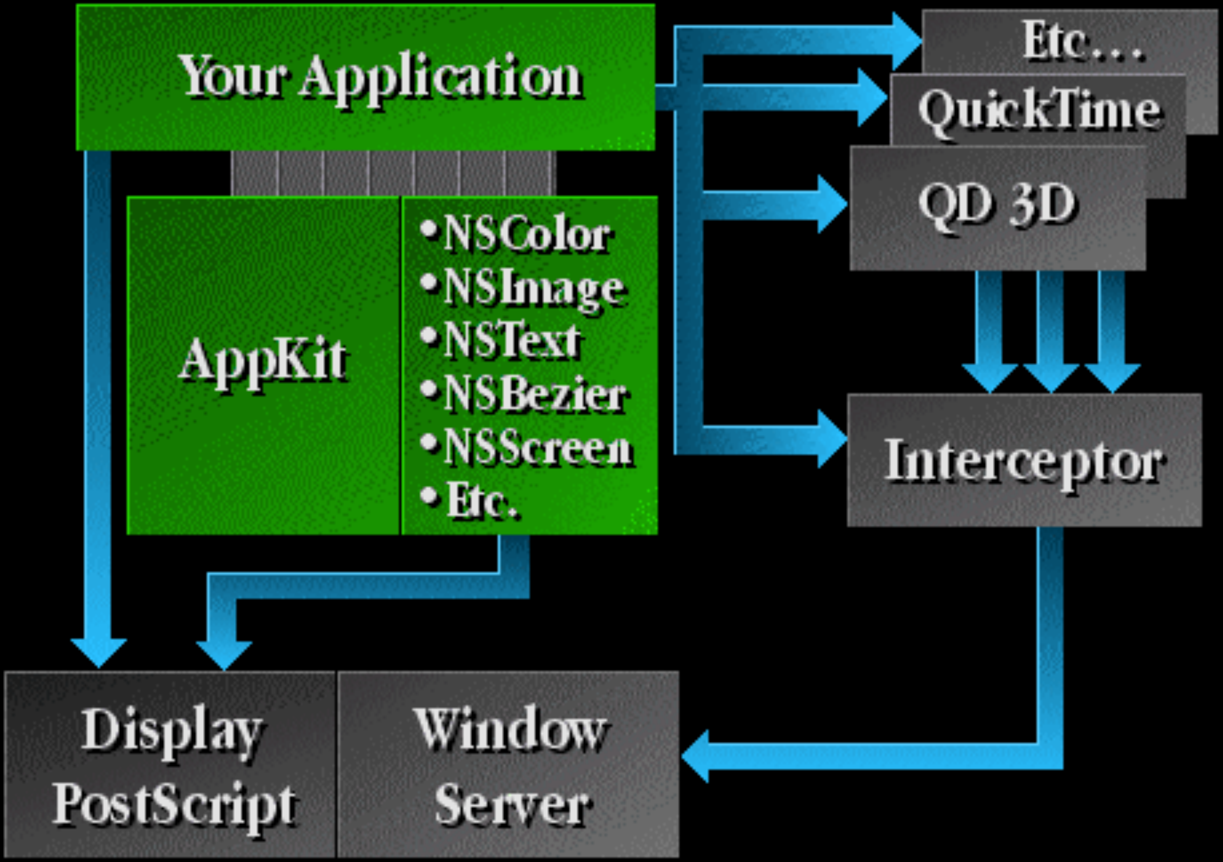
What to Use When

- AppKit Graphics Classes
  - Covers all the common graphics needs
  - 90%+ of applications should use AppKit graphics classes exclusively
- Display PostScript
  - Used by smaller percentage of applications that need power of PostScript language
- Interceptor
  - Very few applications need this
  - Available for access to display buffers and low level pixel access

Graphics Session Roadmap

- Rhapsody Graphics Overview
  - Michael Peirce
- Using Graphics in the AppKit
  - Jeff Martin
- Display PostScript
  - Peter Graffagnino
- Interceptor
  - Mike Paquette
- Q & A

AppKit Graphics Classes



AppKit Graphics Classes

- Object-oriented graphics model
- Fully functional
- Easily extensible through subclassing
- Portable
  - Currently hosted on DPS
  - Can be hosted on other graphics APIs

AppKit Core Graphic Classes

- Graphics state
  - NSColor, NSFont, NSTransform, NSDrawingStyle
- Vector Graphics
  - NSBezierCurve
- Image Graphics
  - NSImage
- Graphics Context Management
  - NSGraphicsContext

NSColor

- Represents a specific color—possibly with transparency information
- Is associated with a color space
  - NSDeviceCMYKColorSpace or NSDeviceRGBColorSpace for example
- Primary methods
  - colorWithCalibratedRed:green:blue:alpha:
  - set

NSFont

- Represents a font at a given point size
- Encapsulates font metric information
- Primary methods
  - fontWithName:size:
  - set

NSTransform

- Represents an affine transform
  - i.e., a 3x2 transform which preserves parallel lines
- Primary methods
  - translateXBy:andYBy:
  - rotateByDegrees:
  - concat
  - set

NSDrawingStyle

- Represents line attributes
  - lineWidth
  - lineCap
  - lineJoin

NSDrawingStyle

- Represents line attributes
  - lineWidth
  - lineCap
  - lineJoin
- Primary methods
  - setLineWidth:
  - setLineCap:
  - setLineJoin:
  - set

NSBezierPath

- Represents all vector graphics primitives
- Provides simple methods for drawing lines, rects, glyphs, etc.
- Primary methods
  - pathForRect:
  - moveToPoint:
  - lineToPoint:
  - stroke
  - fillRect
  - strokeLineFromPoint:toPoint:

NSImage

- Represents all operations on images
- Creates images from image data (PICT, TIFF, GIF, JPEG, etc.)
- Provides bit blitting (compositeToPoint:operation:)
- Provides imaging with arbitrary transform (drawAtPoint:,drawInRect:)

NSGraphicsContext

- Controls graphic state operations
- Flushing and synchronization
- Save and restore graphics state attributes
- Primary methods
  - flush
  - wait
  - saveGraphicsState
  - restoreGraphicsState

High Level Graphics Objects

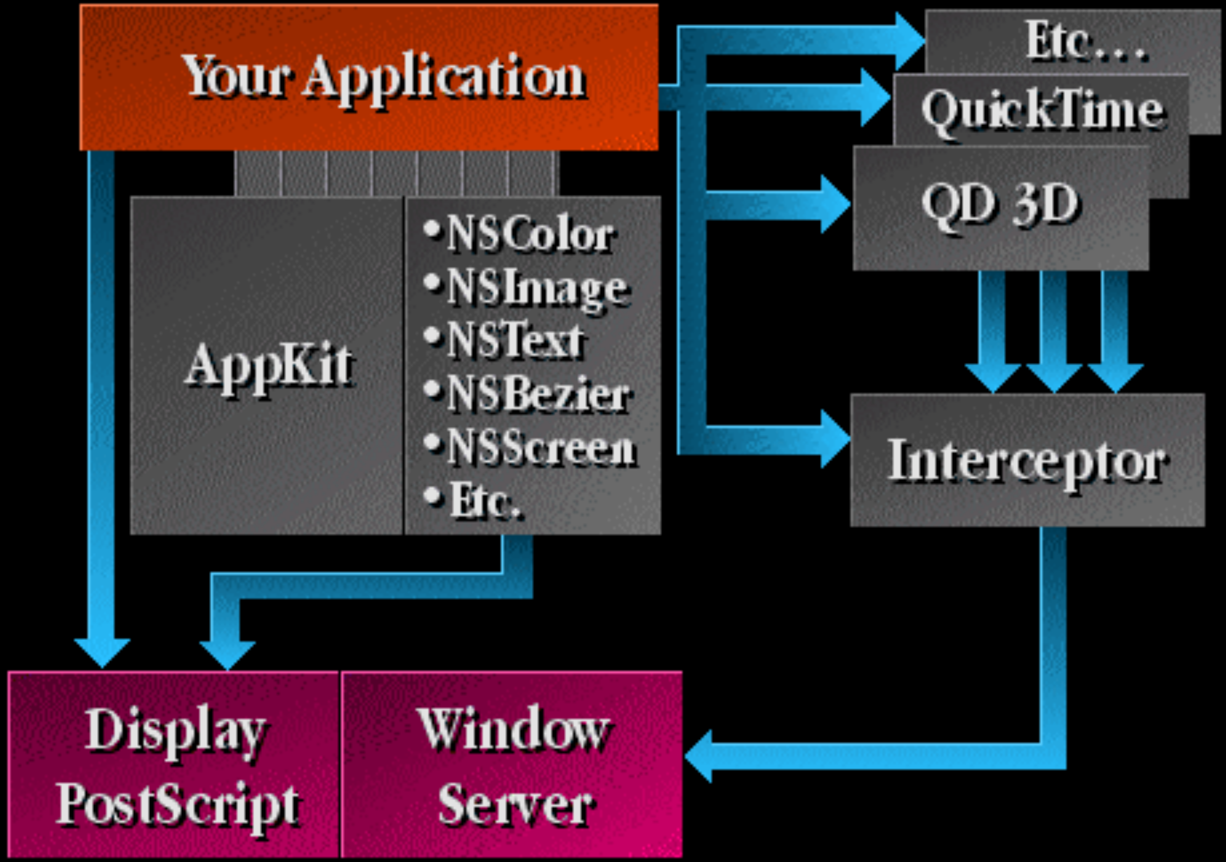
- NSView
  - Hierarchies of coordinate systems
- NSText
  - Used to draw and edit text
- NSWindow
  - Represents windows and graphics devices
- NSScreen
  - Represents physical video devices with information about their size, locations, depth, etc.

AppKit Graphics Demo

Graphics Session Roadmap

- Rhapsody Graphics Overview
  - Michael Peirce
- Using Graphics in the AppKit
  - Jeff Martin
- Display PostScript
  - Peter Graffagnino
- Interceptor
  - Mike Paquette
- Q & A

Display PostScript



Apple's Display PostScript Window Server

- PostScript Imaging Model
- Extensions for alpha and compositing
- Every color has alpha when drawn
- Porter-Duff compositing supported as generalized blit
  - Similar to PhotoShop's layers
- Three buffering modes for graphics
  - Non-retained (client repaints)
  - Retained (no client repaint, immediate)
  - Buffered (no client repaint, buffered)

WindowServer

- Based on core PostScript Level 2 code from Adobe

WindowServer

- Based on core PostScript Level 2 code from Adobe
- Client/Server (wire protocol is binary encoded PostScript language)

WindowServer

- Based on core PostScript Level 2 code from Adobe
- Client/Server (wire protocol is binary encoded PostScript language)
- Apple-specific DPS operators for

WindowServer

- Based on core PostScript Level 2 code from Adobe
- Client/Server (wire protocol is binary encoded PostScript language)
- Apple-specific DPS operators for
  - Compositing and Alpha

WindowServer

- Based on core PostScript Level 2 code from Adobe
- Client/Server (wire protocol is binary encoded PostScript language)
- Apple-specific DPS operators for
  - Compositing and Alpha
  - Window Management

WindowServer

- Based on core PostScript Level 2 code from Adobe
- Client/Server (wire protocol is binary encoded PostScript language)
- Apple-specific DPS operators for
  - Compositing and Alpha
  - Window Management
  - Input Management

WindowServer

- Based on core PostScript Level 2 code from Adobe
- Client/Server (wire protocol is binary encoded PostScript language)
- Apple-specific DPS operators for
  - Compositing and Alpha
  - Window Management
  - Input Management
- Window and Input Management operators not part of the “public” API, use AppKit objects

Apple Enhancements to DPS

- Consistent DeviceRGB interpretation across displays
- Gamma corrected and optimized dithering
- Common case imaging optimizations (identity and scale)
- Leverage Mach for efficient IPC
- Improved CMYK to RGB conversion

Apple Enhancements to DPS

- Improved color rendering performance
- Integer font metrics
- Lazy depth promotion
- Backing store compression

Future Enhancements to DPS

- Update to PostScript 3
- Update system font collection
- Integration with ColorSync
- Improved support for TrueType fonts
- Anti-aliasing of text and graphics

DPS Performance

- PPC version is more than 266 times faster than Laserwriter II/NT X PostScript
- Apple has heavily optimized “real world” usage of DPS
  - Special case, tuned code for common imaging and marking operations
  - Buffered windows minimize app redraws
  - Sophisticated backing store management minimizes memory cost

DPS Performance *(cont.)*

- Conventional acceleration available for window move and screen fills

DPS Performance *(cont.)*

- Conventional acceleration available for window move and screen fills
- Most drawing occurs to backing buffers, so conventional acceleration is trickier

DPS Performance *(cont.)*

- Conventional acceleration available for window move and screen fills
- Most drawing occurs to backing buffers, so conventional acceleration is trickier
  - Researching migrating active backing stores to off screen VRAM (if available)

DPS Performance *(cont.)*

- Conventional acceleration available for window move and screen fills
- Most drawing occurs to backing buffers, so conventional acceleration is trickier
  - Researching migrating active backing stores to off screen VRAM (if available)
- Can run "device layer" of DPS in a separate thread on SMP machines

DPS Performance *(cont.)*

- Conventional acceleration available for window move and screen fills
- Most drawing occurs to backing buffers, so conventional acceleration is trickier
  - Researching migrating active backing stores to off screen VRAM (if available)
- Can run “device layer” of DPS in a separate thread on SMP machines
- Device layer can take advantage of MMX style instructions and fast/wide memory

Advantages of Apple's DPS

- PostScript is great news for publishing applications!

Advantages of Apple's DPS

- PostScript is great news for publishing applications!
  - EPS is everywhere

Advantages of Apple's DPS

- PostScript is great news for publishing applications!
  - EPS is everywhere
- Basic architecture has been shipping for 8+ years (as NeXT)

Advantages of Apple's DPS

- PostScript is great news for publishing applications!
  - EPS is everywhere
- Basic architecture has been shipping for 8+ years (as NeXT)
- Improvements (Level 2, color, etc.) can be made without breaking applications

Advantages of Apple's DPS

- PostScript is great news for publishing applications!
  - EPS is everywhere
- Basic architecture has been shipping for 8+ years (as NeXT)
- Improvements (Level 2, color, etc.) can be made without breaking applications
- Ubiquitous buffering
  (clients not involved in damage repair)

Advantages of Apple's DPS

- PostScript is great news for publishing applications!
  - EPS is everywhere
- Basic architecture has been shipping for 8+ years (as NeXT)
- Improvements (Level 2, color, etc.) can be made without breaking applications
- Ubiquitous buffering (clients not involved in damage repair)
- Multi-depth backing store (color only paid for when used)

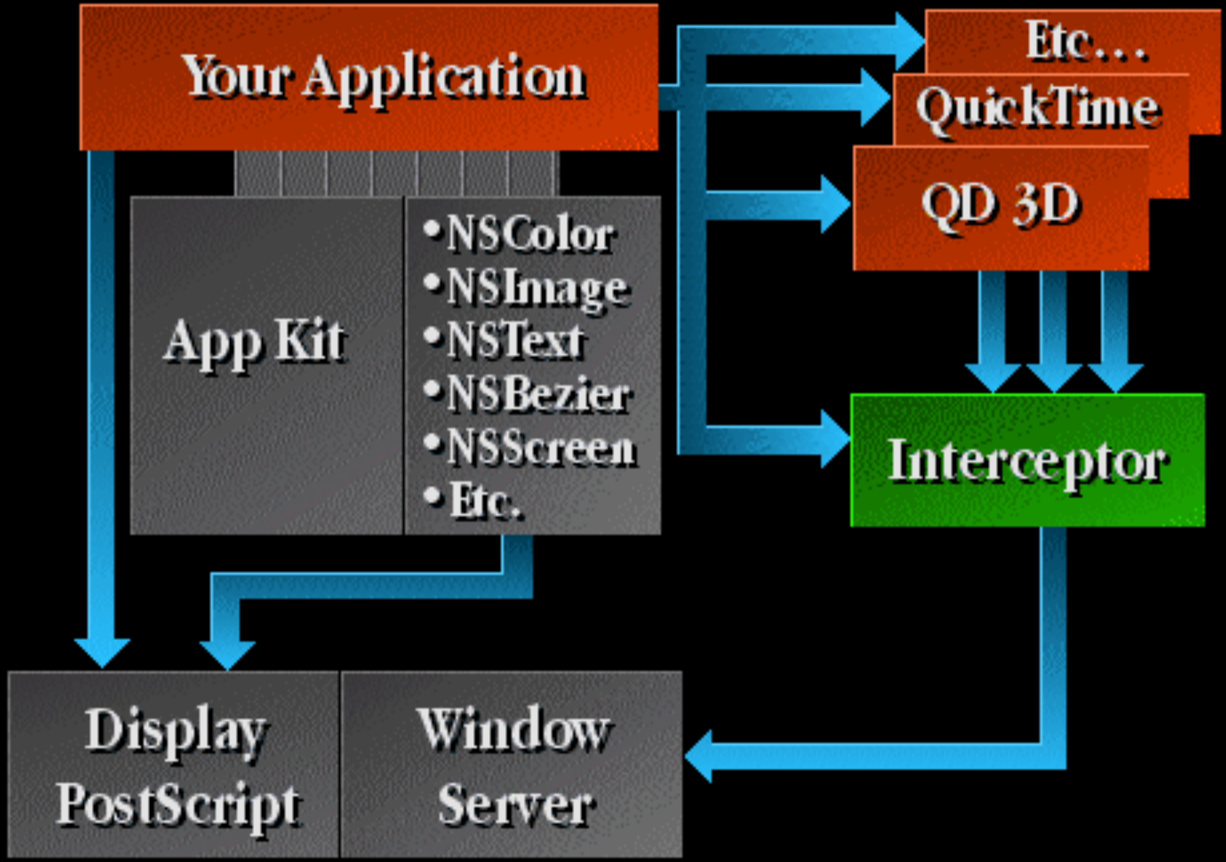
Advantages of Apple's DPS

- Compositing + Alpha =
- Depth independent blitting
- Client/Server model allows remote display
- Truly framebuffer independent
  - Very hard to write an application that doesn't run optimally on all displays

Graphics Session Roadmap

- Rhapsody Graphics Overview
  - Michael Peirce
- Using Graphics in the AppKit
  - Jeff Martin
- Display PostScript
  - Peter Graffagnino
- Interceptor
  - Mike Paquette
- Q & A

Interceptor



What is Interceptor?

- Mechanism to directly access display memory
- Works with Window Server to maintain geometry
- Speeds incremental display updating
- Supports specialized hardware (DMA, Acceleration)
- Fully supported, public API

Interceptor Features

- Memory mapping
  - For framebuffers which are linearly mappable, the interceptor package can be used to map the framebuffer memory directly into the client process
- Clipping notification
  - For a given area of interest, the interceptor package can synchronously notify the client of changes in window visibility

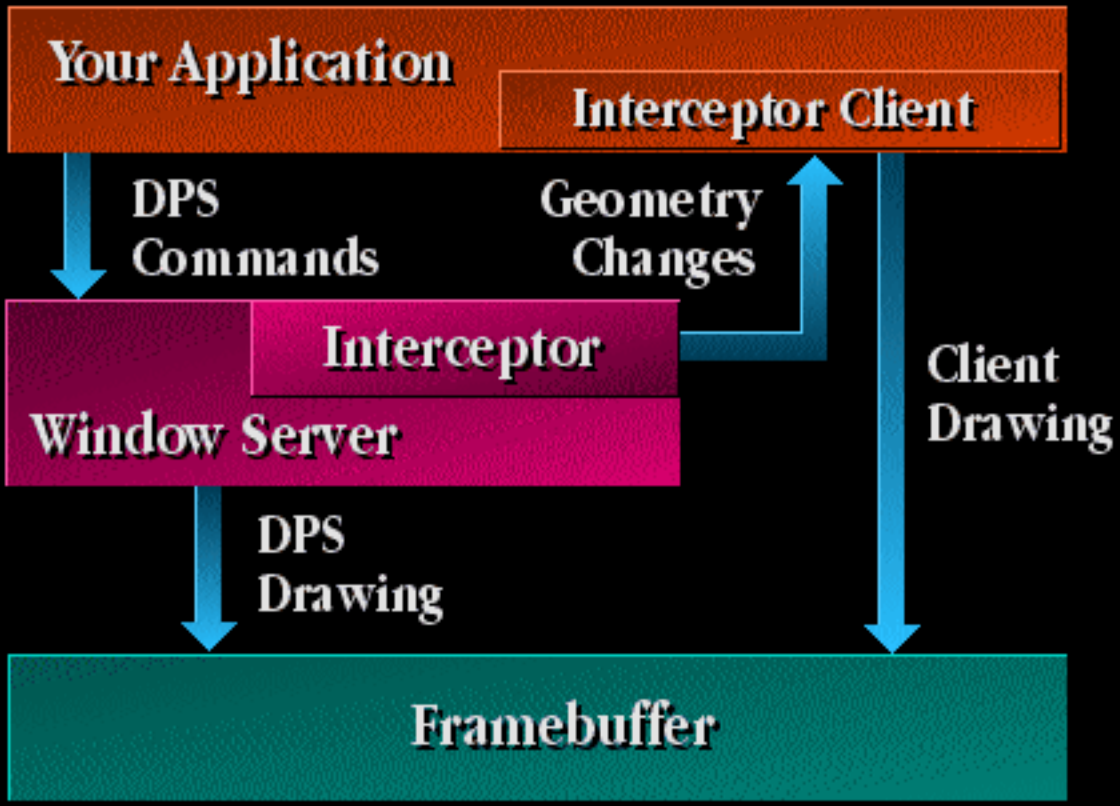
More Interceptor Features

- Movement notification
  - Window movement events can be synchronously handled by the client
- Direct compositing
  - Client can composite data directly into a window
  - Allows the client to exploit the Window Server's clipping logic to draw a bitmap directly into a clipped window

Interceptor Architecture

- Window Server manages window geometry and visibility
- Changes in visibility are synchronously sent to a proxy thread in the Interceptor client
- The proxy thread waits until the client is not drawing, then updates the geometry information
- Orchestrated to prevent client from "coloring outside the lines"

Interceptor Architecture



High Level Interceptor Classes

- NSSimpleBitmap
  - Abstract superclass for NSDirectBitmap and NSDirectScreen
- NSDirectBitmap
  - Drawing surface for a rectangle in a window
- NSDirectScreen
  - Drawing surface for a physical framebuffer

Low Level Interceptor Classes

- NSShape
  - Represents visible and obscured regions
- NSDirectPalette
  - Represents palettes for 8-bit indexed displays

Interceptor Do's

- Blue Box uses it
- Alternate drawing mechanisms
  - QuickTime Movies
  - QuickDraw 3D
- Full screen games
- Screen savers
- Live video display

Interceptor Don'ts

- Usually not the best way to blit to screen
  - Not hardware independent
  - Check out NSImage classes in AppKit

Interceptor Demos

- QuickDraw 3D and QuickTime
  - These use NSDirectBitmap to draw into a window
  - They are examples of giving an alternate drawing mechanism direct access to the display within a window

Interceptor

Demo

Q & A

- Engineering
  - Peter Graffagnino, Michael Peirce, Mike Paquette, Jeff Martin, Andrew Barnes, Eric Schlegel
- Marketing
  - Carla Ow-Chu
- Evangelism
  - Ken Bereskin

Worldwide Developers Conference