Porting Games to Mac OS X

Timothy J. Wood
The Omni Group

Subjects we'll be covering

- Why your game should be cross-platform
- General approaches to portability
- How to integrate with platform-specific services under Mac OS X

Subjects we won't be covering

- Carbon
- CodeWarrior (or other IDEs)
- Console development

Online resources

http://www.omnigroup.com/community/developer/gdc2001

- Presentation PDF file
- Example code
- Links to other online documentation

- Contacting Omni
  - **business@omnigroup.com**
  - **tjw@omnigroup.com**

About Omni

- Mac OS X only development house
- Programming NEXTSTEP/OpenStep/Mac OS X since the late ‘80s
- Ported a bunch of games to Mac OS X
- Very happy to port your game too!

Why design to be cross-platform?

- Economics. If your game is written portably, the additional effort to add another platform will yield more **$$$** than it will cost
- If you are going to license your engine, it will be more attractive to licensees
- Great way to find bugs in your code
- Effort carries over to your next title

Planning your game

- Decide on the supported platforms early
- Avoid APIs not supported on all targets
  - GUI libraries
  - High-level networking toolkits
  - Movie/audio playback

Implementing your game

- Add glue layer to insulate your core code
  - Including types
- Document this layer in its own header(s)
- Put all the platform-specific files in a directory for the relevant platform
- Compile early and often on each target
- Delete dead versions of files from your disk. This is what revision control systems are for!

Write C versions of all routines

- New targets can use these while porting
- Excellent debugging/documentation aid
- Define a unified macro for each CPU you'll be supporting

Write C versions of all routines…

```
#if !defined(C_ONLY) && (defined(ppc) || defined(__ppc__))
  #define use_ppc_optimizations
#endif

#ifdef use_ppc_optimizations
void myCoolFunction(É) {
  É something ppc specific É
}
#endif

#ifdef C_ONLY
void myCoolFunction(É) {
  É a generic C version É
}
#endif
```

- Note the C_ONLY macro which can be used to turn off CPU specific optimizations

Avoid compiler-specific behavior

- Don't depend upon struct elements or bitfields getting layed out a particular order or alignment
- Don't turn off compiler warnings -- try to fix them when at all possible
- Don't depend upon ordering of C++ static constructor calls
- Define macros for things like `__stdcall`, `__declspec`, and `inline`

Platform-specific functionality

- File I/O
- Network I/O
- Memory management
- Threading
- Code loading/unloading
- Controller input
- 2D video access (screen resolution, etc)
- 3D video access
- Audio/Music playback

BSD/POSIX APIs

- File I/O (fopen, fread, fclose, etc.)
- Network I/O (sendto, recvfrom, select, etc.)
- Memory management (malloc, free)
- Threading (pthreads)
- Lots of good documentation on the web

Code loading/unloading

- OS X uses ***dyld*** for loadable Mach-O code
- Simple ***dl*** API wrapper for ***dyld*** available in the Darwin Apache project
- Loads ***bundle*** Mach-O files which you can build directly from ProjectBuilder
- CodeFragment Manager works fine for PEF/CFM libraries (Bink, for example)

`http://www.opensource.apple.com/projects/darwin`

Input

- InputSprockets not fully working in Carbon/PEF
  - Doesn't work at all With Mach-O
- HIDManager not finished yet
- Mouse and keyboard input through normal Cocoa event mechanisms plus a few extras
- For portability try to avoid querying on the current state of a device button -- use events instead of polling
- Try to only use relative mouse events

Keyboard Input

- Keyboard up/down events handled via Cocoa
  - Unicode characters
  - Function keys, page up/down, etc. defined in the vendor specific Unicode range
- Modifier flags (shift, control, command, etc.) handled via ‘flags changed’ Cocoa event
- Keyboard repeat can be turned on and off

Mouse Input

- Ignore mouse button up/down events
  - Use system defined event w/subtype 7
  - Transmits button state for up to 32 mouse buttons
- Mouse move notifications via drag events
  - Event carries screen clipped coordinates
- Relative mouse movement can be obtained from CoreGraphics
- Mouse scaling can be disabled

Input Gotchas…

- Keyboard repeat and mouse scaling not automatically reset. Your app must do it.
- 'Other' mouse button down/up/dragged events added since Public Beta, but not yet documented.
- Currently no way to disable cmd-tab so you need to be able to deal with it.

Code: Mouse Scaling

```
NXEventHandle eventStatus;
NXMouseScaling originalMouseScaling,newScaling;

eventStatus = NXOpenEventStatus();
NXGetMouseScaling(eventStatus, &originalMouseScaling);

newScaling.numScaleLevels = 1;
newScaling.scaleThresholds[0] = 1;
newScaling.scaleFactors[0] = -1;
NXSetMouseScaling(eventStatus, &newScaling);

NXCloseEventStatus(eventStatus);
```

Code: Keyboard Repeat

```
NXEventHandle eventStatus;
double oldRepeatInterval, oldRepeatThreshold;

eventStatus = NXOpenEventStatus();
oldRepeatInterval = NXKeyRepeatInterval(eventStatus);
oldRepeatThreshold = NXKeyRepeatThreshold(eventStatus);

// No repeat events for 40 days and 40 nights
NXSetKeyRepeatInterval(eventStatus, 3456000.0f);
NXSetKeyRepeatThreshold(eventStatus, 3456000.0f);

NXCloseEventStatus(eventStatus);
```

Code: Window Setup

```
NSWindow *window;

[window setAcceptsMouseMovedEvents: YES];
```

Code: Getting Events

```
NSDate *distantPast = [[NSDate distantPast] retain];
NSEvent *event;
NSAutoreleasePool *pool;

pool = [[NSAutoreleasePool alloc] init];
event = [NSApp nextEventMatchingMask: NSAnyEventMask
         untilDate: distantPast
         inMode: NSDefaultRunLoopMode dequeue: YES];
if (event) {
   // Handle the event
}
[pool release];
```

Code: Examining an Event

```
NSEvent *event;
NSEventType eventType;

eventType = [event type];
switch (eventType) {
    É
}
```

Code: Simple Button Events

```
case NSLeftMouseDown:
case NSLeftMouseUp:
case NSRightMouseDown:
case NSRightMouseUp:
case 25: // New undocumented 'other' mouse down
case 26: // New undocumented 'other' mouse up
  // Ignore these events -- we'll find out about
  them via the super mouse button event
```

Code: Super Button Event

```
case NSSystemDefined:
    if ([event subtype] == 7) {
      unsigned int buttons;

      buttons = [event data2];
      // buttons is a bitfield of 32 mouse button states
    }
```

Code: Mouse Movement

```
CGMouseDelta deltaX, deltaY;

case NSMouseMoved:
case NSLeftMouseDragged:
case NSRightMouseDragged:
case 27: // New undocumented 'other' mouse dragged
  // Ignore the contents of these events -- just
  use them as a trigger to call CoreGraphics
  CGGetLastMouseDelta(&deltaX, &deltaY);
```

Code: Scrollwheel

```
float deltaX, deltaY, deltaZ;

case NSScrollWheel:
    deltaX = [event deltaX];
    deltaY = [event deltaY];
    deltaZ = [event deltaZ];
```

Code: Keyboard

```
unichar c;
NSString *characters;

case NSKeyUp:
case NSKeyDown:
 // Event can contain multiple characters.
 // If shift is down, theyÕll be upper case.
characters = [event charactersIgnoringModifiers];
charCount = [characters length];
for (charIndex = 0; charIndex < charCount; charIndex++) {
  c = [characters characterAtIndex: charIndex];
}
```

Code: Keyboard Flags

```
unsigned int newFlags, oldFlags, changed;
BOOL isDown;

case NSFlagsChanged:
  newFlags = [event modifierFlags];
  changed = newFlags ^ oldFlags;

  if (changed & NSShiftKeyMask) {
       isDown = newFlags & NSShiftKeyMask;
  }

  oldFlags = newFlags;
```

Code: Keyboard Flags

```
NSAlphaShiftKeyMask
NSShiftKeyMask
NSControlKeyMask
NSAlternateKeyMask
NSCommandKeyMask
NSNumericPadKeyMask
```

Demo

Display Mgmt - CoreGraphics

- Enumerate display devices
- Enumerate supported modes on each device
- Change current gamma curve
- Hide and show the cursor
- Disassociate the cursor and mouse
- Position the cursor

Code: Listing Display Devices

```
CGDirectDisplayID  displayList[MAX_DISPLAYS];
CGDisplayCount     displayCount;
CGDisplayErr err;

err = CGGetActiveDisplayList(MAX_DISPLAYS,
                             displayList,
                             &displayCount);
 if (err != CGDisplayNoErr) {
    ...
}
```

Code: Listing Graphics Modes

```
CGDisplayCount displayIndex;
CGDirectDisplayID selectedDisplay;
NSArray *displayModes;

selectedDisplay = displayList[displayIndex];
displayModes = CGDisplayAvailableModes(selectedDisplay);
[displayModes retain];
```

Code: Examining a Mode

```
NSDictionary *mode;
unsigned int value;

mode = [displayModes objectAtIndex: modeIndex];
value = [[mode objectForKey: kCGDisplayWidth] intValue];
```

Code: Display Mode Keys

```
kCGDisplayWidth
kCGDisplayHeight
kCGDisplayBitsPerPixel
kCGDisplayBitsPerSample
kCGDisplayRefreshRate
kCGDisplayIOFlags
   kDisplayModeStretchedFlag
```

Code: Cursor Control

```
CGEventErr err;
err = CGAssociateMouseAndMouseCursorPosition(boolValue);

CGDisplayErr err;
err = CGDisplayHideCursor(display);
err = CGDisplayShowCursor(display);

CGPoint point = (CGPoint){x, y};
err = CGDisplayMoveCursorToPoint(display, point);
```

Code: Gamma Control

```
CGGammaValue gamma;
CGDisplayErr err;

gamma = 1.0 / someValue;
err = CGSetDisplayTransferByFormula(
                display,
                min, max, gamma,  // red
                min, max, gamma,  // green
                min, max, gamma); // blue

// Restores ALL displays
CGDisplayRestoreColorSyncSettings();
```

Demo

3D - CoreGraphics/OpenGL

- Windowed mode
- Full screen mode
  - Lock a device (or all devices)
  - Change video resolutions
  - Make sure your game can recreate all textures
- Create a pixel format
- Create a GL context
- Set up the drawing area (window or full screen)
- Bind the GL context to the drawing area

3D - NSOpenGL, AGL, CGL

- NSOpenGL for Cocoa
- AGL for Carbon
- CGL underlies both NSOpenGL and AGL
- Querying CGL context for info
- OpenGLInfo application
- Link whichever framework you are using from /System/Library/Frameworks

Code: OpenGL Setup

```
// collect the needed information
newMode = [displayModes objectAtIndex: modeIndex];
fullscreen = some random computaion;
depthBits = another random computaion;
width = [[newMode objectForKey:kCGDisplayWidth] intValue];
height = [[newMode objectForKey: kCGDisplayHeight] intValue];

// If we are NOT fullscreen, then we MUST have the same color
   depth as the current display mode
colorMode = fullscreen ? newMode : oldMode;
colorBits = [[colorMode objectForKey: kCGDisplayBitsPerPixel]
             intValue];
```

Code: Configure the Display

```
// Figure out what display mode we want and switch to it.  Must
   do this before creating the GL context.  Otherwise, the GL
   context will say 'invalid drawable' if the old display mode
   and new display mode don't have the same bit depth.
if (fullscreen) {
    err = CGDisplayCapture(display);
    err = CGDisplayHideCursor(display);
    err = CGDisplaySwitchToMode(display, displayMode);
}
```

Code: Create a Pixel Format

```
#define ADD_ATTR(attr) \
do { \
    attrCount++; \
    attrs = realloc(attrs, sizeof(*attrs) * attributeCount); \
    attrs[attrCount - 1] = attr; \
} while (0)

NSOpenGLPixelFormat *pixelFormat;
NSOpenGLPixelFormatAttribute *attrs;
unsigned int attrCount = 0;

attrs = malloc(sizeof(*attrs));
```

Code: Create a Pixel Format

```
if (fullscreen)  ADD_ATTR(NSOpenGLPFAFullScreen);
ADD_ATTR(NSOpenGLPFAColorSize);
ADD_ATTR(colorBits);
ADD_ATTR(NSOpenGLPFADepthSize);
ADD_ATTR(depthBits);
ADD_ATTR(NSOpenGLPFADoubleBuffer);
ADD_ATTR(NSOpenGLPFAAccelerated);
ADD_ATTR(NSOpenGLPFAScreenMask);
ADD_ATTR(CGDisplayIDToOpenGLDisplayMask(display));
ADD_ATTR(0);

pixelFormat = [[NSOpenGLPixelFormat alloc] initWithAttributes:
                attrs];
free(attrs);
```

Code: Create a Context

```
context = [[NSOpenGLContext alloc]
             initWithFormat: pixelFormat
               shareContext: nil];
[pixelFormat release];
```

Code: Bind Context To Drawable

```
if (fullscreen) {
    CGLContextObj cglContext;
    cglContext = [context cglContext];
    err = CGLSetFullScreen(cglContext);
    if (err) // switch to old mode, release display, show cursor
} else {
    NSRect rect;
    rect = NSMakeRect(0, 0, width, height);
    window = [[NSWindow alloc] initWithContentRect:rect
   styleMask:NSTitledWindowMask backing:NSBackingStoreRetained
   defer:NO];
    [window orderFront: nil];
    [context setView: [window contentView]];
}
```

Code: Context Parameters

```
long zero = 0;

[context setValues: &zero forParameter:NSOpenGLCPSwapInterval];
```

Code: Current Context

```
[context makeCurrentContext];

[NSOpenGLContext clearCurrentContext];

[NSOpenGLContext currentContext];
```

Code: Swapping Buffers

```
[context flushBuffer];
```

Code: Tearing Down

```
[NSOpenGLContext clearCurrentContext];
[context clearDrawable];
[context release];

 if (fullscreen) {
     CGDisplaySwitchToMode(display, oldMode);
     CGDisplayRelease(display);
     CGDisplayShowCursor(display);
} else
    [window release];
```

Demo

Audio

- CoreAudio
- Carbon Sound Manager
- CD Streaming
- QuickTime

CoreAudio

- Lowest layer user-land sound API
- Low latency with synchronized output
- Uses a worker thread to feed the DMA engine
- Callback function must be thread-safe
- Callback function gets a pointer to the time that the returned samples will be played
- Currently only supports floating point samples
  - This is useful for mixing

Carbon Sound Manager

- Works great with Cocoa and Carbon apps
- Built on top of CoreAudio
  - Slightly higher overhead
  - Callback function must also be threadsafe

Code: Finding Mounted CDs

```
struct statfs *mounts;
unsigned int mntCount;
const char *mountName;

mntCount = getmntinfo(&mounts, MNT_NOWAIT);

mounts[mntIndex].f_flags & MNT_RDONLY == MNT_RDONLY;
mounts[mntIndex].f_flags & MNT_LOCAL) == MNT_LOCAL;
strcmp(mounts[mntIndex].f_fstypename, "cddafs")) == 0;
strcmp(mounts[mntIndex].f_mntonname, MY_CD_NAME)) == 0;

// Inside the CD mount directory will be a series of .aiff
// files, one for each audio track on the disk
// (in AIFF-C format)
```

Code: Setting up CoreAudio

```
propertySize = sizeof(outputDeviceID);
status = AudioHardwareGetProperty(
         kAudioHardwarePropertyDefaultOutputDevice,
         &propertySize, &outputDeviceID);

propertySize = sizeof(bufferByteCount);
bufferByteCount = SAMPLES_PER_BUFFER * sizeof(float);
status = AudioDeviceSetProperty(outputDeviceID,
         NULL, 0, NO,
         kAudioDevicePropertyBufferSize,
         propertySize, &bufferByteCount);
```

Code: Setting up CoreAudio...

```
status = AudioDeviceAddIOProc(outputDeviceID,
                              audioDeviceIOProc,
                              userInfoPointer);

status = AudioDeviceStart(outputDeviceID,
                          audioDeviceIOProc);
```

Code: CoreAudio I/O Proc

```
OSStatus audioDeviceIOProc(AudioDeviceID inDevice,
                           const AudioTimeStamp *inNow,
                           const AudioBufferList *inInputData,
                           const AudioTimeStamp *inInputTime,
                           AudioBufferList *outOutputData,
                           const AudioTimeStamp *inOutputTime,
                           void *inClientData);
```

Code: CoreAudio I/O Proc

```
float *outBuffer;
float scale = (1.0f / SHRT_MAX);

outBuffer = (float *)outOutputData->mBuffers[0].mData;

// Scale samples to floats in the range [-1..1].
for (sampleIndex = 0; sampleIndex < sampleCount; sampleIndex++)
    outBuffer[sampleIndex] = samples[sampleIndex] * scale;

// Fill in zeros in the rest of the buffer
for (; sampleIndex < SAMPLES_PER_BUFFER; sampleIndex++)
    outBuffer[sampleIndex] = 0.0;
```

Code: CoreAudio Shutdown

```
status = AudioDeviceStop(outputDeviceID,
                         audioDeviceIOProc);

status = AudioDeviceRemoveIOProc(outputDeviceID,
                                 audioDeviceIOProc);
```

Code: SoundManager Setup

```
SndChannel        *soundChannel
ExtSoundHeader     soundHeader

void _callback(SndChannel *sc, SndCommand *cmd);

err = SndNewChannel(&soundChannel, sampledSynth,
                    initStereo, _callback);

_callback(soundChannel, NULL);
```

Code: SoundManager Callback

```
short *buffer;

memset( &soundHeader, 0, sizeof(soundHeader));
soundHeader.samplePtr = buffer;
soundHeader.numChannels = 2;
soundHeader.sampleRate = rate44khz;
soundHeader.encode = extSH;
soundHeader.baseFrequency = 1;
soundHeader.numFrames = SAMPLES_PER_BUFFER / 2;
soundHeader.sampleSize = 16;
```

Code: SoundManager Callback

```
SndCommand     playBufferCommand, doCallbackCommand;

playBufferCommand.cmd = bufferCmd;
playBufferCommand.param1 = 0;
playBufferCommand.param2 = (int)&soundHeader;
SndDoCommand(channel, &playBufferCommand, true);

doCallbackCommand.cmd = callBackCmd;
doCallbackCommand.param1 = 0;
doCallbackCommand.param2 = 0;
SndDoCommand(channel, &doCallbackCommand, true);
```

Code: SoundManager Shutdown

```
SndDisposeChannel(channel, true /*quietNow*/);
```

Demo

QuickTime

- Can be used to decode MP3
- Can also be used to play movies (of course)
- See web site for example code

Demo

Common pitfalls in porting to PPC

- Memory bandwidth
  - Newer machines are much improved
  - Avoid back-to-back load/stores
  - Load several blobs of data, operate, store them all
  - PPC has a ton of registers unlike x86 -- use them!
  - Can use cache control instructions
- Byte swapping
  - PPC is big endian
- Float/int conversion expensive

More pitfalls

- ObjC and C++ can't live in the same module currently
  - Apple is devoting engineering resources to this
  - Come to WWDC to hear about it
- Use C interfaces to connect the two languages
- Possibly a good idea to make your platform API in pure C anyway

PPC specific optimizations

- frsqrte -- about 16x faster than 1.0/sqrt(x)
  - Is an estimate of 1/sqrt
  - Improve the precision with Newton-Rhapson refinement (see the PPC manual)
- fsel -- replace simple if/else assignments
- lwbrx and friends -- load/store and swap in one instruction

Performance monitoring tools

- Sampler application and sampler command line tool
- OmniTimer framework
  - Uses PPC TBR register to get high precision timings
  - Available with the rest of the examples on the web
- MallocDebug/ObjectAlloc/OmniObjectMeter
  - Leak fixing applications available for OS X

How does OS X stack up?

- Performance similar and often better than OS 9
- One crashing game doesn't take down the whole machine
  - Users are going to switch from OS 9 for this alone
  - Especially important in keeping developers efficient
- Quick time to deployment
  - Can typically bring up a well structured game in a couple days
  - Allows for extra time tweaking for performance

Advanced topics

- Optimizing for Velocity Engine
  - Can provide huge gains for the right kind of tasks
  - GCC and CodeWarrior provide C bindings
- Symmetric Multiprocessing
  - Minimize the number of synchronization points
  - Minimize the data flowing between processors

References, Q&A

http://www.omnigroup.com/community/developer/gdc2001

Additional references will be linked from this site

Contacting Omni

- **business@omnigroup.com**
- **tjw@omnigroup.com**