Technical Q&A QA1385

Driving OpenGL Rendering Loops

Q: How do I drive the drawing loop of my Cocoa OpenGL application?

A: On Mac OS X 10.4 and later, applications should use a Core Video display link (`CVDisplayLink`) to drive the drawing loop of a Cocoa OpenGL application. For a compatibility with Mac OS X 10.3 and earlier, they should use a Cocoa timer (`NSTimer`).

OpenGL applications should avoid frame tearing and wasting resources drawing pixels the user will never see. To do this, applications should not draw faster than the display can refresh and should swap or draw during the vertical blanking interval. `CVDisplayLink` provides a separate high-priority thread to notify the application when a given display will need each frame making sure the update won't overlap with a previously rendered frame. How often a frame is requested is based on the refresh rate of the display device currently associated with the display link. Listing 1 shows an example of how to use `CVDisplayLink` to drive your drawing loop. For more details about the `CVDisplayLink` functions, please see the CVDisplayLink Reference.

Listing 1: Sample Core Video display link

```
@interface MyView : NSOpenGLView
{
    CVDisplayLinkRef displayLink; //display link for managing rendering thread
}
@end

- (void)prepareOpenGL
{
    // Synchronize buffer swaps with vertical refresh rate
    GLint swapInt = 1;
    [[self openGLContext] setValues:&swapInt
forParameter:NSOpenGLCPSwapInterval];

    // Create a display link capable of being used with all active displays
    CVDisplayLinkCreateWithActiveCGDisplays(&displayLink);

    // Set the renderer output callback function
    CVDisplayLinkSetOutputCallback(displayLink, &MyDisplayLinkCallback, self);

    // Set the display link for the current renderer
    CGLContextObj cglContext = [[self openGLContext] CGLContextObj];
    CGLPixelFormatObj cglPixelFormat = [[self pixelFormat] CGLPixelFormatObj];
    CVDisplayLinkSetCurrentCGDisplayFromOpenGLContext(displayLink, cglContext,
cglPixelFormat);


    // Activate the display link

    CVDisplayLinkStart(displayLink);

}


// This is the renderer output callback function

static CVReturn MyDisplayLinkCallback(CVDisplayLinkRef displayLink, const
CVTimeStamp* now, const CVTimeStamp* outputTime, CVOptionFlags flagsIn,
CVOptionFlags* flagsOut, void* displayLinkContext)

{

    CVReturn result = [(MyView*)displayLinkContext getFrameForTime:outputTime];

    return result;

}


- (CVReturn)getFrameForTime:(const CVTimeStamp*)outputTime

{

    // Add your drawing codes here


    return kCVReturnSuccess;

}


- (void)dealloc

{

    // Release the display link

    CVDisplayLinkRelease(displayLink);


    [super dealloc];

}
```

When using a Cocoa timer (`NSTimer`) on Mac OS X 10.3 and earlier, there are a couple of things to keep in mind. It is imperative to turn on vertical synchronization to prevent frame tearing and to use an appropriate time interval. Listing 2 shows an example of how to use `NSTimer` to drive your drawing loop.

`NSTimer` is a general purpose timer. It is not a timer tied to the display device. The interval, and the instant the timer started firing, have no relation to when the vertical refresh happens. This means that the simple approach of creating a timer at "60.0" Hz is doomed to fail -- the timer will drift in relation to the actual refresh rate, and you will drop or double frames. It also means that the timer calls the application at an arbitrary point into the refresh, which reduces the amount of time available to prepare drawing, if the application is blocking on vertical synchronization.

When vertical synchronization is enabled in your OpenGL application, during each retrace period, when the timer fires, the application starts preparing data for the next frame; when the drawing is done, the application blocks waiting for the next vertical retrace. The timer can not fire until the swap completes. In order to let your application have enough time to submit the drawing commands, you will want to let the timer fire as soon as the application returns to the event loop. Recall that there is no correlation at all between when the timer fires and when the vertical refresh occurs. If you use a fairly big time interval (such as 0.0167 seconds to yield 60 frames per second), the timer may fire anywhere in the retrace period, giving your application the remainder of the frame to prepare the data for the next frame. You should set the timer to a very small interval such as 0.001 seconds or 1000 fps. This makes sure that the timer fires right after the swap completes, giving your application the entire retrace period to do the drawing.

Notice that if vertical synchronization is not enabled in your OpenGL application, by creating a timer with an exceptionally small interval (such as 0.001 seconds or 1000 fps), the application will burn a lot of CPU time just firing off the timer, even though the drawing loop hasn't even completed its last run. The net effect of this will be moderate to severe performance problems, depending on how busy the drawing loop is and how fast the machine can service the timer. However, if you synchronize buffer swaps to the vertical refresh rate, this small time interval will not overdrive the pipeline, because the timer does nothing when the application blocks during swap regardless of the time interval, thus taking no extra CPU time.

Listing 2: Sample drawing loop timer. (Make sure you enable vertical synchronization, otherwise, this timer with the small time interval of 0.001 seconds (1000 fps) will burn up CPU time.)

```
// Synchronize buffer swaps with vertical refresh rate
- (void)prepareOpenGL
{
    GLint swapInt = 1;
    [[self openGLContext] setValues:&swapInt
forParameter:NSOpenGLCPSwapInterval];
}


// Put our timer in -awakeFromNib, so it can start up right from the beginning
-(void)awakeFromNib
{
    renderTimer = [NSTimer timerWithTimeInterval:0.001   //a 1ms time interval
                                target:self
                                selector:@selector(timerFired:)
                                userInfo:nil
                                repeats:YES];


    [[NSRunLoop currentRunLoop] addTimer:renderTimer
                                forMode:NSDefaultRunLoopMode];
    [[NSRunLoop currentRunLoop] addTimer:renderTimer
                                forMode:NSEventTrackingRunLoopMode]; //Ensure
timer fires during resize
}


// Timer callback method
- (void)timerFired:(id)sender
{
    // It is good practice in a Cocoa application to allow the system to send
the -drawRect:
    // message when it needs to draw, and not to invoke it directly from the
timer.
    // All we do here is tell the display it needs a refresh
    [self setNeedsDisplay:YES];
}
```

Document Revision History

Date	Notes
2013-01-02	Removed an extra [in Listing 2. Fixed a link.
2009-05-01	Corrected a typo.
2008-12-23	Added using CVDisplayLink. Pointed out enabling VSYNC when using NSTimer.
2004-10-04	New document that using Core Video display links (CVDisplayLink) or Cocoa timers (NSTimer) to drive an OpenGL rendering loop