

Using Mac OS X for Real-Time Image Processing

Daniel Heckenberg

*Human Computer Interaction Laboratory
School of Computer Science and Engineering
The University of New South Wales
danielh@cse.unsw.edu.au*

Abstract With appropriate hardware, Mac OS X provides a capable platform for real-time image processing (RTIP). This paper provides an overview of available video capture hardware and presents development strategies to achieve high performance, low latency image processing. As the requirements of real-time image processing differ significantly to those for video playback or editing, different hardware and software techniques are appropriate. In particular, QuickTime and OpenGL may be configured for high performance RTIP applications using the methods described.

These techniques have been established in the process of developing video-based interfaces for Human-Computer Interaction at the University of New South Wales HCI Group. The results and approaches presented have been gathered from system documentation, the Apple development community and my own development and experimentation.

Introduction

Real-time image processing (RTIP) promises to be at the heart of many developments in computer technology: context aware computers, mobile robots, augmented reality and the subject of my research — video-based interfaces for human computer interaction. These applications have significant demands not only in terms of processing power: they must achieve real-time, low latency response to their visual input.

Whilst most modern operating systems provide a wealth of multimedia features, these features are usually oriented towards the playback or recording of media rather than processing in real time. Different media representations and handling mechanisms are often necessary for real-time processing. The operating system itself must also be capable of efficient, low-latency response and processing. Mac OS X provides a robust operating system with excellent latency performance and a rich multimedia framework that can be applied, with some care, to RTIP applications.

Suitable live image sources are also required for RTIP. Once again, general purpose or recording/playback oriented devices are not necessary suitable for this application domain. As a relatively young platform, Mac OS X does have limited driver support for video input hardware. Suitable hardware for which drivers are available will be compared and discussed.

Real-Time Image Processing

A platform for real-time image processing must provide the following

- high resolution, high frame rate video input
- low latency video input
- low latency operating system scheduling
- high processing performance

Sampling Resolution:

In the most general terms, image processing attempts to extract information from the outside world through its visual appearance. Therefore adequate information must be provided to the processing algorithm by the video input hardware. Precise requirements will, of course, depend on the algorithm and application but usually both spatial and temporal resolution are important. Broadcast video provides a practical reference point as most cameras provide images in formats derived from broadcast standards regardless of their computer interface (analog, USB etc).

Standard	Spatial Dimensions	Frame Rate
NTSC	720 x 480	30 fps
PAL	768 x 576	25 fps

Table 1: Broadcast video standards

We note that higher resolution in both spatial and temporal sampling is desirable for many applications.

Low latency video input:

All video input systems have intrinsic sources of latency in their hardware and transmission schemes. Indeed, the relatively sparse temporal sampling (frame rate) typical for video can itself be thought of as a source of latency equal to the frame duration. Higher frame rates therefore allow for lower latency and more responsive RTIP systems.

Additional latency occurs in the transmission of video from the camera to the computer interface. The sequential nature of almost all video frame transmission also imposes latency equal to the frame transmission time (which is usually close to the frame duration in order to minimise bandwidth requirements). This applies to digital transmission schemes over USB or Firewire just as it does to analogue transmission.

Applications which use video as part of a feedback loop (through a human user or electromechanical means) often have tight demands on the total latency in this feedback loop. For human interaction, common candidates for upper bounds on acceptable latency are:

- threshold of perceived causality (~50ms) [1]
- threshold of perceived synchronicity (e.g. music ~10ms) [2]

Given that the frame duration of a broadcast standard based video device will be at least 33ms (for NSTC 30fps) and we expect to have at least two frames of latency in the video input device (camera and transmission system) additional latency must be minimised if we are to stay close to these target figures.

Low latency operating system scheduling:

Once the video signal arrives at the computer it will be processed and passed between a number of software components. These components will depend on the type of video capture hardware in use, but generally and in the minimum case there will be a driver component and an application that performs the image processing. The driver is responsible for receiving the

transmission and presenting the video frame as a buffer of pixels and is of course provided by the operating system vendor or hardware vendor. This pixel buffer is then processed by the application which would then typically produce some output for the user or provide information to other application software running on the system.

The ability of the operating system to respond to incoming video data and to schedule each of these software components to run as soon as its data are available has a crucial impact on system latency. If no input data is to be lost, buffering (and hence additional latency) must be introduced to cover both lag and any variation in when data is available and when it is passed to the next component. This lag and variation is related to system interrupt latency and scheduling latency.

Fortunately Mac OS X has excellent low latency performance even under heavy system load as evidenced by its reliable behaviour with low latency audio software [3].

High Processing performance:

Image processing algorithms are very bandwidth and processor speed intensive. High bandwidth memory architecture, effective caching and high performance processors are necessary for an RTIP platform. AltiVec is an important factor in achieving good performance, as image processing algorithms are usually highly parallel and therefore well suited to SIMD optimisation. Recent changes in Macintosh hardware architecture are also very promising for RTIP, in particular the emphasis on memory bandwidth in the Power Mac G5 range.

Video Capture Hardware

Video capture hardware performs the vital role of handling the reception of the video signal into the computer and presenting to the processor in a suitable form. Some hardware integrates both camera and digitisation functions together, such as the DV video cameras, and USB Webcams. Other systems perform only digitisation of an analog video signal provided by an external camera. These devices are then connected to a suitable system bus (PCI, Firewire or sometimes USB).

Suitable devices for RTIP must provide high resolution, high frame rate video at low latency. Making the video signal available in an uncompressed format to the image processing software with low CPU overhead is also important. These requirements unfortunately exclude many common video input devices which provide only low quality input or introduce latency through their compression or transmission schemes. The common classes of devices with drivers available for Mac OS X can be considered for their suitability.

USB hardware

Both cameras and digitisers are available which use the common and convenient USB for communication to the host computer. Unfortunately the low bandwidth of USB 1.1 (11Mbps) is insufficient to convey high resolution video at high frame rate. Most devices are limited to 320x240 pixels at 30fps. Some devices provide higher resolution at lower frame rates. Other devices achieve acceptable frame rates and resolution but they must employ a compression scheme such as MPEG to limit their data rate for USB. The MPEG compression schemes not only degrade the visual quality of the incoming signal but usually add latency to the video

input stream. USB 2.0 offers sufficient bandwidth for high quality video but is not yet available on Mac hardware.

Firewire hardware

Like USB, Firewire offers the convenience of an external bus but crucially, Firewire does have sufficient bandwidth to convey high quality video.

DV hardware

The most common video devices used with Firewire, DV devices, are not usually suitable for RTIP as they use a compression scheme which involves significant CPU usage for decoding and adds latency to the video signal.

IIDC hardware

More recently, the Instrumentation and Industrial Digital Camera (IIDC) specification [4] has standardised a protocol for high performance imaging over the Firewire bus. The specification allows great control of frame rates and resolution although particular cameras usually only implement a small subset of this configuration range. A great variety of cameras is available from cheap webcam style devices to industrial grade cameras. Some of these devices are ideally suited for RTIP and Apple has provided a generic driver in OS X 10.2 which exposes many of their features. However, it is still complicated to extract good performance for these devices and this topic will be explored in the software section of this document.

DFG/1394-1

One particular Firewire device deserves special mention as it provides an excellent feature set for RTIP. The DFG/1394-1 digitises analogue video in NTSC or PAL format onto Firewire in a device specific (uncompressed) protocol [5]. Drivers are available for OS X which expose the device as a QuickTime video digitiser and offer many useful configuration options, such as field-rate digitisation [6].

PCI hardware

The most traditional hardware for RTIP is the combination of an analogue camera and a PCI-based video digitiser. This approach can offer excellent performance as the video digitiser can perform useful preprocessing and move the video frame buffers via DMA. This style of hardware is well supported by QuickTime. Unfortunately there are very few devices available with drivers for OS X.

Quicktime Video Capture Support

Apple's QuickTime architecture is the primary Operating System support for video-based applications on OS X. QuickTime (QT) offers a model of time-based media which facilitates the handling of video data, compression formats and various capture hardware. Unfortunately QuickTime's model of real-time video input is based around the recording of the input stream to disk, and providing screen based video previews to assist this. Such video capture has significant differences from RTIP requirements in the following ways:

Recording	RTIP
Throughput more important than latency	Latency can not be traded off for throughput
Compressed formats are ideal for recording	Uncompressed formats are required for image processing
Demands high priority control of system	Must coexist with other software

Table 2: Differences between recording and low latency capture schemes.

These factors have important consequences for the way that video capture devices are treated and how critical situations are handled. The recording model tries to avoid dropping frames at all costs by adding buffers to the video stream and demanding priority scheduling. An RTIP system would usually prioritise latency over the dropping of frames and therefore introduce as few buffers to the video stream as possible. Furthermore, if critical time deadlines are not being met (such as processing time for other parts of the RTIP system or frame drops due to frame handling taking too long) the behaviour of an RTIP scheme will be different to that of a recording scheme.

Sequence Grabber

High level support for the capture of sequences of Audio and Video in QuickTime is provided by the Sequence Grabber component. Apple has started to provide support for low latency capture through the introduction of new configuration options in QuickTime 6 (e.g. the seqGrabLowLatencyCapture flag for the SGSetChannelUsage function [7]). However, to achieve satisfactory results it is necessary to bypass the SeqGrabber component and access the video digitisation hardware directly.

Video Digitisers (vdig)

Video digitiser hardware is presented in QuickTime as a vdig component, providing a standardised interface to control each hardware device. Direct, programmatic control may be obtained over low level configuration of capture hardware such as pixel formats used for digitiser output or transmission and buffering schemes used. This control is necessary for high performance RTIP. Despite some loss of convenience through avoiding the Sequence Grabber, programming a vdig directly is reasonably straightforward.

To illustrate the advantages of performing direct vdig capture, the following section presents the stages in development of a RTIP capturing system for an IIDC compliant camera system.

High Performance IIDC Capture

Firewire based IIDC cameras offer high quality image capture with the convenience of an external data and power bus. A single Firewire input provide the power and data interface for multiple cameras. These characteristics make the development of a high performance capture scheme for IIDC worthwhile.

A number of obstacles exist to high performance capture from IIDC devices using QuickTime on Mac OS X. These obstacles are explained in the following sections, along with methods to overcome them.

IIDC Pixel Formats

The IIDC specification presents a variety of standard operating modes which are combinations of image dimensions, frame rate and pixel format. QuickTime does not expose direct control of the pixel format used by the camera on the Firewire bus. Furthermore QuickTime does not provide support for all of the IIDC modes even in the earliest released version of the IIDC specification (v1.04). This has the consequence of excluding the use of some combinations of resolution and frame rate that a given camera may support.

Mode	Image Dimensions	Pixel Format	Bits/pixel
0	160 X 120	YUV(4:4:4)	24bit/pixel
1	320 X 240	YUV(4:2:2)	16bit/pixel
2	640 X 480	YUV(4:1:1)	12bit/pixel
3	640 X 480	YUV(4:2:2)	16bit/pixel
4	640 X 480	RGB	24bit/pixel
5	640 X 480	Y	8bit/pixel

Table3: IIDC v1.04 Format Modes [4]

The combination of mode 2 and 30 fps is the only configuration that allows 640 x 480 pixel colour capture at 30fps using many cheaper IIDC devices. This mode, using the YUV 4:1:1 pixel format, is unsupported by Apple's vdig. Only 640 x 480 pixel colour capture at 15fps or 320 x 240 pixel colour capture at 30fps are achievable using such cameras and Apple's driver.

Fortunately a third-party vdig provides additional support for these modes and cameras. The IIDC vdig from IOXPerts [8] may be configured to use the correct mode by requesting 640 x 480 pixel colour capture at 30fps. This driver will then use YUV 4:1:1 pixels for communication between the camera and the driver. A pixel format conversion to a YUV 4:2:2 format that is supported by QuickTime is performed internally in the driver before passing the buffer as output. A thorough discussion of YUV pixel formats and their treatment in QuickTime may be found in IceFloe 19 [9].

Low Latency Capture

Once the vdig has been configured appropriately the capture cycle may be initiated. The cycle consists of three steps:

1. ask the vdig to capture a frame into a buffer
2. perform image processing on the buffer
3. return buffer to vdig

The pixel formats used in capture and image processing are often YUV based and in the case of IIDC cameras we have seen that this is necessary to achieve the frame resolution and rates that we require. Such frames are captured in QuickTime using a set of calls prefixed with VDCompress. [10]

VDCompress capture calls are asynchronous which allows the capture process to be started and then periodically checked for completion without stalling the CPU for the duration of

frame capture. This is crucial for high performance as it allows the CPU to perform other processing.

Many vdig drivers can support outstanding capture buffers simultaneously which allows the capture of the next frame to be overlapped with the processing of the current frame without adding any buffering latency to the system.

Code fragment 1 presents an outline of the entire capture process[11]. The vdig is configured to the appropriate capture dimensions, frame rate and pixel format. It is then queried for the image description of the frames it will return which should be the driver's best attempt to match the configuration requests. The vdig is then told to start capturing the first frame. Finally a timer-based polling function is set up which should be run at a frequency greater than the desired frame rate.

```
SetupVDig()
{
    VDSetDigitizerRect() // determines cropping
    VDSetCompressionOnOff(vdComp, 1)
    VDSetFrameRate() // set to 0 for default
    VDSetCompression() // compressype=0 means default
    VDGetImageDescription() // find out image format
    VDGetDigitizerRect() // get vdig cropping
    VDRestCompressSequence()

    VDGetImageDescription(&imageDesc);

    VDCompressOneFrameAsync();

    SetupVDigPolling(myVDigPollFunc(), pollPeriod);
}
```

Code Fragment 1: vdig Setup (based on [11])

The polling function, outlined in code fragment 2, checks the status of the vdig and performs processing on completed frames. Overlap is achieved by commencing the new frame capture before the current frame processing begins. When the processing is completed, the frame buffer is returned to the vdig. Some hardware supports multiple outstanding capture requests which allows for further overlapping to be performed.

```
myVDigPollFunc()
{
    if (!VDCompressDone(&queuedFrames) && queuedFrames)
    {
        VDCompressOneFrameAysnc();
        myProcessFrame();
        VDReleaseCompressBuffer();
    }
}
```

Code Fragment 2: Overlapped Asynchronous Capture (based on [11])

Efficient Display of Video Sequences

Even if the RTIP system does not require the display of video as part of its output, it is always important to be able to monitor and preview the video stream at various stages of processing. QuickTime includes functions which perform hardware accelerated display of buffer with some pixel formats and appropriate conversions for buffers of many other pixel formats. As the accelerated formats have changed in OS X from previous versions of Mac OS this topic deserves some treatment.

Accelerated Pixel Formats

The formats that receive hardware accelerated display under OS X are those which can be treated directly as textures in the underlying OpenGL graphics system. Presently, these formats are:

Name	FourCC	OpenGL format	Bits per pixel
Monochrome	'raw '	GL_LUMINANCE	8 bits per pixel
RGB	'raw '	GL_RGB	24 bits per pixel
RBGA	'raw '	GL_RGBA	32 bits per pixel
YcbCr (YUV) 4:2:2	'2yuv'	GL_APPLE_ycbcr_422	16 bits per pixel

Table 4: Hardware Accelerated Pixel Formats

Unfortunately many common YUV style video formats are not part of this list and therefore must be converted in the process of image display. In particular the component video pixel type 'kComponentVideoPixelFormat' ('yuvu' or 'yuv2') which is the common interchange format for many of the codecs in QuickTime [9] is not able to be directly displayed without an implicit, but CPU expensive, format conversion. Many vdig components produce 'yuvu' data, including the IOxperts IIDC driver, resulting in relatively poor performance if the frames are displayed.

Fast display of Image Sequences

The Image Compression manager in QuickTime allows for efficient conversion and display of a series of images with identical format. By only configuring the conversion at the initialisation of the sequence rather than upon each image transfer, some efficiency is gained. These QuickTime calls are prefixed with DecompressSequence and are documented in Inside Macintosh — QuickTime API [12].

Avoiding extraneous frame copying

In the relentless pursuit of performance it is important to reduce any unnecessary data copying, in particular copying of video frames which may be many megabytes in size. QuickTime image operations generally operate on Gworlds which may be created to refer to a particular image buffer (using GworldFromPtr). In the case that we want to display a series of buffers originating from a vdig component we are faced with a choice: to create a new Gworld for each buffer that we receive, or to copy that buffer into an image buffer for which we have created a Gworld previously.

In practice it is possible to avoid either overhead by simply creating a `GworldFromPtr` and changing the image pointer in the corresponding `Gworld` structure that is passed to the QuickTime functions [13].

OpenGL image display

By using OpenGL for image display rather than QuickTime, very high performance may be achieved. Apple's OpenGL extensions [14] allow fine control over the details of texture uploading; the process of moving images from main memory onto the display adapter. YUV images can be asynchronously transferred across the high performance AGP bus without CPU intervention.

Three Mac OS X specific OpenGL extensions together provide a highly optimised pixel transfer scheme. The `GL_APPLE_client_storage` extension forces the application's image buffer to be used directly for texturing, rather than making and then using a copy on the display adapter. Control over texture caching and memory mapping is achieved through the `GL_APPLE_texture_range` extension. Finally, support of YUV 4:2:2 pixel format (`GL_APPLE_ycbcr_422`) for OpenGL textures means that in some cases, video may be obtained from the driver, processed and displayed without any format conversion or buffer copying. Apple's "Texture Range" sample provides an example of this complete process for RGB images [15].

System Profiling

Specialised development tools are required to understand and analyse the time-performance of software. Apple provides one such application, Shikari, in its suite of Computer Hardware Understanding Development (CHUD) tools. Shikari can perform detailed sampling of the run-time behaviour of software allowing for thorough analysis of the time taken by every part of the code comprising an RTIP application. Furthermore, Shikari is able to use the symbols present in any framework to provide very useful information about the time spent and functions performed in other code upon which the RTIP application depends. It is therefore possible to scrutinise QuickTime and system calls to observe how the API functions are being implemented and to compare their performance after configuration changes.

Conclusions

Mac OS X offers all of the necessary features for the development of high performance RTIP applications, although careful choice of peripherals and software techniques are required. Using the recommended hardware and techniques outlined in this paper, low latency high performance video capture and display is possible using the QuickTime architecture and OpenGL. A general framework has been presented for the development of RTIP applications on Mac OS X.

Acknowledgements

The techniques outlined in this paper have been developed with the use of Apple's sample code library and assistance from members of the QuickTime API mailing list. Thanks in particular to Milton Aupperle, Ben Bird, Chris Clepper and Steve Sisak for their invaluable contributions.

References

- [1] VON HARDENBERG C. ET AL (2001) *Bare-hand Human-Computer Interaction* Proceedings of the ACM Workshop on Perceptive User Interfaces.
- [2] WESSEL D. AND WRIGHT M. (2001) *Problems and Prospects for Intimate Music Control of Computers* Proceedings of New Interfaces for Musical Expression.
- [3] MACMILLAN K, DROETTBOOM M. AND FUJINAGA I. (2001) *Audio Latency Measurements of Desktop Operating Systems* Proceedings of the International Computer Music Conference.
- [4] 1394 TRADE ASSOCIATION (1996) *1394-based Digital Camera Specification* Version 1.04, August 9.
- [5] — *Product Information for DFG/1394-1*
<http://www.theimagingsource.com/prod/grab/dfg13941/dfg13941.htm> accessed 4/5/2003
- [6] — *Product Information for Mac OS X drivers for DFG/1394-1*
<http://dfg1394.outcastsoft.com/> accessed 4/5/2003
- [7] APPLE COMPUTER INC *Documentation for SgSetChannelUsage* Inside QuickTime: API Reference
<http://developer.apple.com/techpubs/quicktime/qtdevdocs/APIREF/SOURCESIII/sgsetchannelusage.htm> accessed 4/5/2003
- [8] — *Product Information for Universal Firewire Webcam driver for OS X*
<http://www.ioxperts.com/dcam.html> accessed 4/5/2003
- [9] APPLE COMPUTER INC *QuickTime Ice Floe Notes — Ice Floe Dispatch 19 — Uncompressed Y'CbCr Video in QuickTime Files*
<http://developer.apple.com/quicktime/icefloe/dispatch019.html> accessed 4/5/2003
- [10] APPLE COMPUTER INC *Controlling Compressed Source Devices* from *Inside Macintosh: QuickTime Components*
<http://developer.apple.com/techpubs/quicktime/qtdevdocs/INMAC/QTC/imVideoDigComp.1b.htm> accessed 4/5/2003
- [11] SISAK, STEVE *Correspondence on QuickTime-API* mailing list 9/4/2003 and 12/5/2003
<http://lists.apple.com/mailman/listinfo/quicktime-api>
- [12] APPLE COMPUTER INC *Working With Sequences* from *Inside QuickTime: API Reference*
<http://developer.apple.com/techpubs/quicktime/qtdevdocs/APIREF/SOURCESV/workwithsequences.htm> accessed 4/5/2003
- [13] BIRD, BEN *Correspondence on QuickTime-API* mailing list 10/4/2003
- [14] APPLE COMPUTER INC *OpenGL Extensions Guide*
<http://developer.apple.com/opengl/extensions.html> accessed 4/5/2003
- [15] APPLE COMPUTER INC *TextureRange Sample Code*
http://developer.apple.com/samplecode/Sample_Code/Graphics_3D/TextureRange.htm