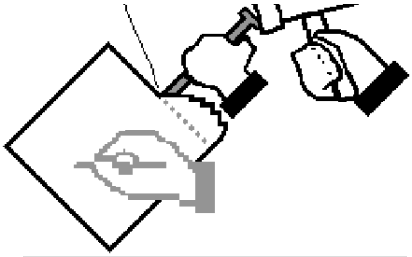


MacHack '95

The Tenth Annual Technical Conference for Leading Edge Developers

Table to Contents

Welcome

Welcome to the 10th MacHack:

I have a confession to make. I wasn't actually at the first MacHack. Aimée Moran was the Conference Manager then, and I'd just had a baby, but I can look in the files and see how much has changed.

In 1986, the first MacHack had 73 people attending to hear 18 sessions over 2½ days. Sessions ran from 9 AM to 5 PM. The hot issue was the hierarchical file system. Opinion was divided as to whether or not it was "a good thing". The Plus was new, (wow! a whole 1 megabyte of RAM!, this speeds right along at almost 8 MHz), floppy drives now were for 800K, SCSI ports were the forefront of hot hardware, and hackers stayed up all night to hack in a fully equipped machine room with four 512e single floppy Macintoshes. We were even allowed an account on ARPA-Net, a government run electronic data exchange system to get messages between the brand new Expotech office and the conference sponsors at the University of Michigan.

In 1995, we expect 300 attendees to attend almost 70 sessions over 3, 24 hour long, days. ARPA has become the Internet and we have a domain registered to MacHack. Cyberdog, OLE, and OpenDoc are hot issues. And hackers will stay up all night to hack in a fully equipped Machine Room with almost 100 computers, most running at speeds well over 60 MHz, it's own ISDN connection, and enough hardware and software goodies to impress even hardened hackers.

We have tried to put together the best MacHack possible. MacHack would not be possible without the dedicated efforts of many, many volunteers, and the generous contributions of many, many companies. If you'd like to become one of them for next year's conference, please let us know.

Enjoy!

PostScript Picture
Carol/epsO

Carol Lynn
Conference Manager

Conference Structure

There are many different things going on simultaneously at MacHack '95:

Presentations

Papers given by individuals, case studies, etc., are being presented. Please see the complete schedule for times and dates.

Sessions

Sessions take place each day of the conference roughly from 10am to Midnight. Panels take place in the various Ballrooms and Session Rooms. These are the main programming events. Sessions consist of one or more speakers on a specific topic. For more information, please see the complete schedule.

Business Sessions

Sessions on topics of business interest are scheduled for 9am through Noon on Thursday, Friday and Saturday in room 10. Business Sessions must be registered separately, see registration for details.

Roundtables

In addition to the panels and presentations, there will be "Roundtable Sessions" Scheduled all throughout the conference. Roundtables will consist of a volunteer moderator, attendees, a Macintosh, add a dedicated area. Topics will be scheduled as and when the moderator desires. Session topics can be as specific or as general as desired. Anyone may add a roundtable to the schedule at any time. Check with Operations in Room C to add a Roundtable.

Code Clinics

Code Clinics are the place where some of the most talented code-hackers attending actually dissect, debug, and comment on code. Real, hands-on session, up close and personal with that code. Code Clinics are scheduled every late afternoon in Room 5.

Machine Room

The Ballrooms A&B are this year's "Machine Room." Here you will find Macintoshes and other "toys" to play with to your heart's content. The Machine Room is open 24 hours from 6pm on Wednesday to 9am on Sunday. Please cooperate with the security stationed at the door; all personal equipment being brought into the room must be registered with the guards. No briefcases or bookbags will be allowed in the Machine Room. There has been some missing equipment in past years that make this necessary.

There will also be PowerBook hookups in the Lobby with access to the Machine Room.

Hack Contest

Sponsored by the MacHax™ Group. If you have questions on the rules or how to enter, how to get your hack on the CD or others, go to the opening session scheduled for Wednesday in the wee hours of the morning. At other times grab Scott Boyd and ask him. Hacks will be shown at midnight Friday in the combined Ballroom (C&D) area. Winners will be announced at the Hack Awards Banquet on Saturday evening.

Thanks to:

The Management and Staff of MacHack '95 would like to extend our special thanks to the following companies and individuals whose generous contributions make MacHack a more fun and educational experience.

Ameritech
ISDN line setup

Apple Computer, Inc.
Communications and Collaboration – InterNet Router & IP Gateway software
ETO & Mac OS SDK Product Marketing – ETO & Mac OS Software & Right to Copy
Developer Press – Macintosh Programmer's Toolbox Assistant CD-ROM & Inside Macintosh CD-ROM, Complete Inside Macintosh Hardcopy
Developer Support – Software and machines, Friday Night Hack Show Pizza, ISDN line
Developer Tools Product Marketing– Apple Media Tool, HyperCard 2.3
Internet Server Marketing – The Internet Server Software CD
Open Doc Evangelism – 30 PowerMacs, "Batman Forever" movie tickets, carrying bags
System 7.5 & 7.5.1 Product Line – System 7.5 and 7.5.1 Update Software & Right to Copy
Mike Zivkovic & Pat Harding – ETO Hardcopy Documentation

Black Box
network equipment

Chrysler Motor Co.
machine room setup

Digital Ocean
wireless network equipment

Excel Software
technical brochures and a disk

Farallon Computing, Inc.
network machines

Ford Motor Co.
Projection equipment

MacTech Magazine
Hack Show beverages

Metrowerks
software, T-shirts, CD's, additional publicity

Microsoft Corp.
machines and network hardware, Friday Night Hack Show Pizza, ISDN line

MGM Studios
"Hackers" movie promotional items

Mitech
CDR, monitors, networking bridges, and other hardware

MSEN
internet provider

Newer Technologies
memory and speedup chips

Anna O'Connell
Babysitting

Pictorius, Inc.
Prograph CPX

QC
debugging software

Quasar Knowledge Systems, Inc. (QKS)
QKS Host CD, SmalltalkAgents

Rustnet
Internet provider

Shiva Corp.
connection devices

Symantec Corp.
software, keynote pizza

Wayne State University
machine room staff

Wedimeyers
wire and connector, etc

and more - see daily updates for the latest information

Committee:

Chair:
Sheila Wallace, Fluent Software

Sessions:
Leonard Rosenthol, Aladdin Systems

CD-ROM:
Brian Bechtel, CD Character, DTS, Apple Computer Inc
John Wallace, Fluent Software

Equipment:
Doug Houseman

Papers:
Richard Clark, General Magic
Shane Looker, Software Engineer, GlamerWare Software

Internal Networking:
Jay Weiss, Keane, Inc.

External Networking:
Bob Desoff, (Inter Net) Ameritech

Code Clinics:
Marshall Clow, Aladdin Systems, Inc.

Pre Conference Training:
Gary Kacmarcik
James Plamondon, Microsoft

Round Tables:
Timothy Knox, Ameritech

P.R. Advertisment:
Neil Ticktin, Editor-in-Chief/Publisher, XPLAIN CORP-MACTECH MAG.

Hack Contest:
Scott Boyd, MacHax™ Group
Greg Marriott, Just Some Guy, General Magic, Inc.

Volunteer Coordinators:
Gerry Felipe (Pre-conference)
Paula Smith (On-Site)

Apple Liaison:
Rick Fleischman, Product Mgr. - Dylan, Apple Computer Inc.
Jordan Mattson, Apple Computer,Inc.

Key Note:
Chris Allen, Consensus Developmen
Brad Kollmeyer, Software Design Engineer, Microsoft
John Kalb, Liberty Software

T-shirts:
Brad Serbus, Sofrware Design Engineer, Microsoft

Program Book:
Darryl Wattenberg, D&W Media Productions

Management:

For Expotech:

Conference Managers
Carol Lynn
Bobbie Meganck
Aimée Moran

Administrative Assistants
Katie Scallen
Heather Hogan

Volunteers:

Lou Abundis
Mark Birac
Michael,Birac
Joseph Cotter
Josh Dady
Mike Davis
Bob Desoff
Brian Duck
Brad Grupczynski
Brian Duck
Yuri Komarov
Thomas Knox
Bob Meyers
Lorrn Olson
Ronald G. Robinson
Dave Shuman
Doug Stoyer
Darryl Wattenberg
Mike Young
Steve Yuhasz
Joseph A Zayac

Keynote: Chris Crawford

Chris Crawford entered the personal computer revolution in 1977. After teaching himself enough hardware to design and build his own computer, he taught himself enough software to program it. He started at Atari in 1979 as a game designer, and was quickly promoted, first to supervise a training group for programmers, then to lead a games research team at Atari Corporate Research, reporting to Dr. Alan Kay. Following the collapse of Atari, Mr. Crawford began working as a freelance computer game designer. Mastering the Apple Macintosh in six months, he wrote the best-selling game, “Balance of Power” and was profiled in national media, including The New York Times Sunday Magazine and Newsweek. Since then, Mr. Crawford's prolific output includes “Balance of the Planet,” “Patton Strikes Back,” and “The Global Dilemma.” Crawford founded and served as Chairman of the Board of the Computer Game Developers’ Conference; he edits and publishes “Interactive Entertainment Design.“

Client/Server

If you thought that Client/Server architecture was on the outs, think again! Hear about real life experiences developing these system and why they are still better for many situations. Copland changes the Toolbox Many managers are changing under Copland, and we don't mean a reorg! See what's happening with some Macintosh staples as the Window Manager, Menu Manager and more!

Driving with Copland

Writing drivers will never be the same! Listen to those who've been there tell all about how much more interesting driver writing can be. Developing for PCI Ready to replace your NuBus cards with the new kid of the block? Hear how!

Cross-Platform Development with the Win32 API

This two-hour tutorial session will explain how an application writtenonce, to the Win32 API can be delivered on Windows 3.1, Windows NT (including both the x86 and such RISC chips as Alpha, MIPS, and PowerPC), Windows 95, the 68K Mac, PowerMac, and a number of flavors of Unix -- all with a single source tree, minimal effort, little retraining, and low cost. If you want to write your code once, and have it run everywhere, attend this session.

CyberDog

If you thought Apple was ignoring the Internet - you were wrong! In a compelling use of OpenDoc, CyberDog brings the net right into your documents. The authors will show you how it works and how you can extend it or bend it to your will.

Developing for PCI - Allan Foster

Ready to replace your NuBus cards with the new kid of the block? Hear how!

General Magic

It's finally out and the folks from GM can't wait to evangelize you to come play with their new toys. Magic Cap and Telescript will both be covered.

MacOS Futures

Just because Copland is coming doesn't mean that we have to wait for new improvements to the current MacOS. Apple Engineers will tell you about upcoming changes to the System 7.5 family and what it means to you.

All about the OpenDoc Framework (ODF)

If you thought Apple didn't need another class framework - think again! The OPF makes writing parts a snap and it's authors will show you how.

Build an OpenDoc Part

Applications of the future will exist as OpenDoc parts. Learn all you need to know to start building your new and exciting parts today! All about the OpenDoc Parts Framework (OPF). If you thought Apple didn't need another class framework - think again! The OPF makes writing parts a snap and it's authors will show you how.

OLE

Come to this session to learn how you can implement, in your application, support for OLE — the technology that is already being used to integrate shipping Macintosh applications from Microsoft, Adobe, Caere, and others. Learn about the features and technologies underlying the next version of Mac OLE, which will make OLE even more compelling.

OpenDoc Overview

Find out about how compound documents should work! Apple, Novell and other industry leaders have formed an alliance to deliver the best tools for the job and our experts will tell you why!

SOM

Learn all about the IBM System Object Model that forms that basis for OpenDoc and many of the new toolbox goodies in Copland.

Business Track

Sessions on tech support - how to do it right, opportunities and changes in the Macintosh market.

Copland Overview
An overview of what Copland (System 8) is all about. What areas of the Macintosh OS will be affected? What does it mean to users? To developers? To hackers?

Copland Kernal Architecture
Details of the new microkernal that is the guts of Copland. The new I/O subsystems, memory architecture, multitasking & threading and more!

Copland File System
Look at what they've done to our file system!! See the changes in both architecture and API that will make writing file related code and patching it a lot more fun.

Debugging
Learn about the available tools for finding those elusive bugs from the folks who wrote them! Get some hot tips!

Fanfare & the Appearance Manager
What did they do to my Macintosh??? Apple brings colors, patterns and the personal touch to the Macintosh in 1995 and your applications had better look good.

Frameworks
Looking for tools to improve your development - class libraries and frameworks are a good way to go. Here about the ones in use today from PowerPlant and the Think Class Library to Sprocket, the new kid of the block.

Getting the most from Universal Headers
Apple's new headers are both pain and pleasure for most developers. Hear about why Apple chose this new direction and how you can make your own headers conform.

Everything you wanted to know about the Internet
Ready to hop on that Information Highway either as a user or developer? Then you should show up here! We'll will talk about everything from setting yourself up as a network node to what state of mind you need to think Unix is actually fun!

Writing an OpenDoc Container
If your application wants to allow things to be contained (embedded) within it, then this session is for you!

Hacking OpenDoc
This one is for all the real hackers out there! If you thought that Apple was going to deliver something that couldn't be hacked up - you're wrong! The authors tell just how open OpenDoc is!

OpenTransport 1
Take the first step towards the Macintosh networking world of tomorrow. Learn what's happening with AppleTalk, TCP/IP and more!

OpenTransport 2
Looking to do neat and interesting things at a lower level on your network? Maybe writing your own network protocols or improving Apple's? Then this session is for you!

Introduction to PowerPC Assembly

Even though Apple says not to write assembly code for the PowerPC, that's never stopped us! Listen to a long time assembly weanie teach you about the PowerPC and moving your 68K skills into the future.

Optimizing for PowerPC

For those that want to squeeze that extra cycles out of their software, come hear from folks who've been in the trenches doing it!

Programmer Burnout

QuickDrawGX

Drawing pretty pictures has never been this much fun! Apple's new imaging system gives you control that even the folks at Adobe can drool over. Get your feet wet on GX here!

Business Track

Contract programming - finding and keeping contracts, setting rates, case studies (or don't do what I did), project management - theory and reality.

Development Environments

Looking for those better, faster tools to get your code out to market quicker and with less bugs. Check out the latest from the folks at Symantec & Metrowerks.

DLLs for the Mac

If you think that Windows folks have us beat with DLLs - your missing a lot. There are a number of options for dynamically linked libraries on the Mac and some of the folks who helped write them will tell you more.

Dylan

Apple thinks that the future of development is in dynamic languages and they've even weathered a lawsuit to bring you their vision. The product manager for Apple's Dylan technology will give you the show and tell you've been waiting for!

Interactive TV

Introduction to PSP (Personal Software Process)

The PSP (Personal Software Process) is a set of 'professional' practices for software development that provides a framework for measuring your process. Documented in the propular book, A Discipline for Software Engineering, by Watts Humphery. PSP provides a framework for 'life-long' process improvement at thelevel of the individual engineer. As a result professionals plan,design, and schedule better. Better bids, cleaner code, and fewer defects (read bugs) result.

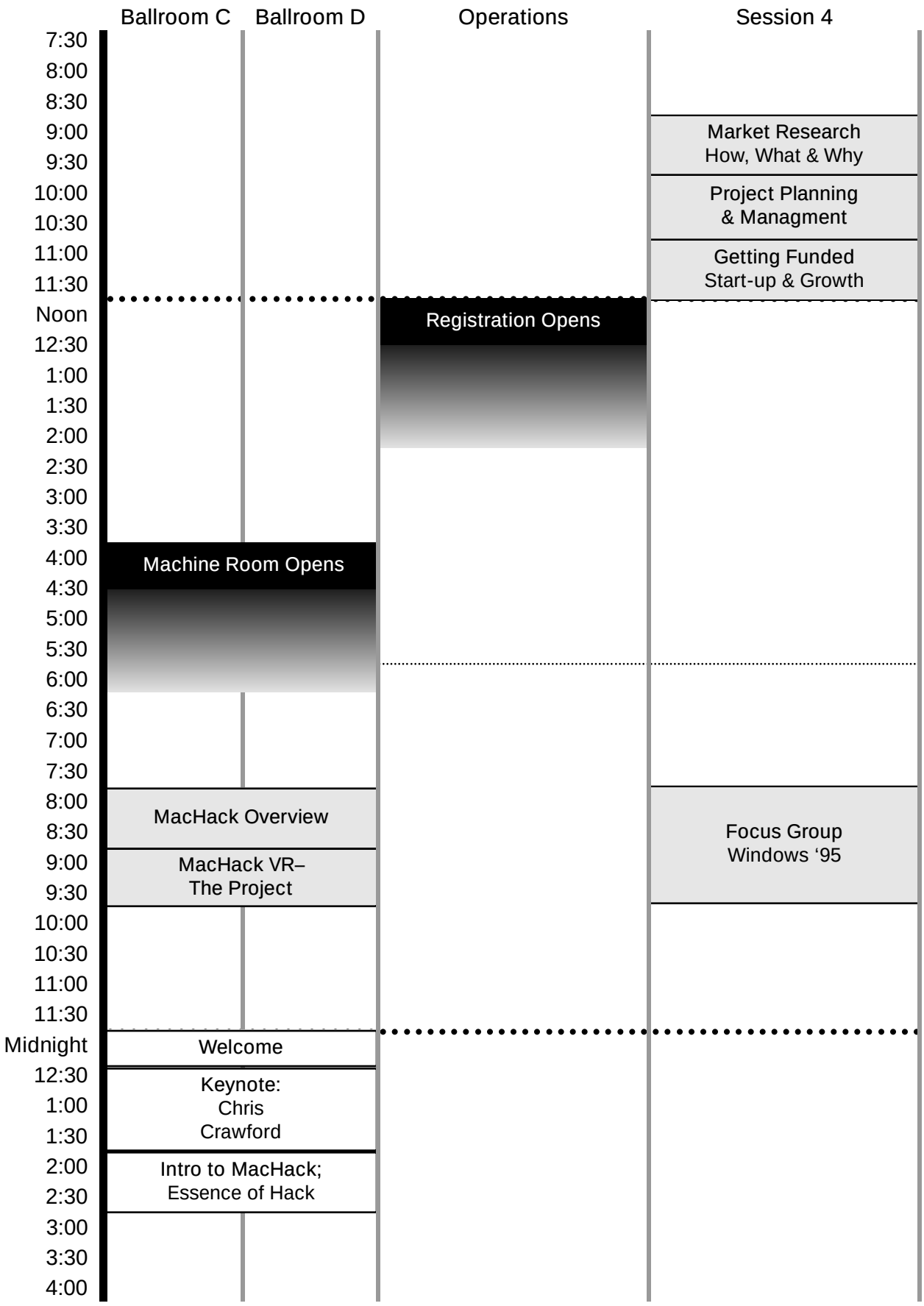
Women & Hacking

Why aren't there more women in software development?

Business Track

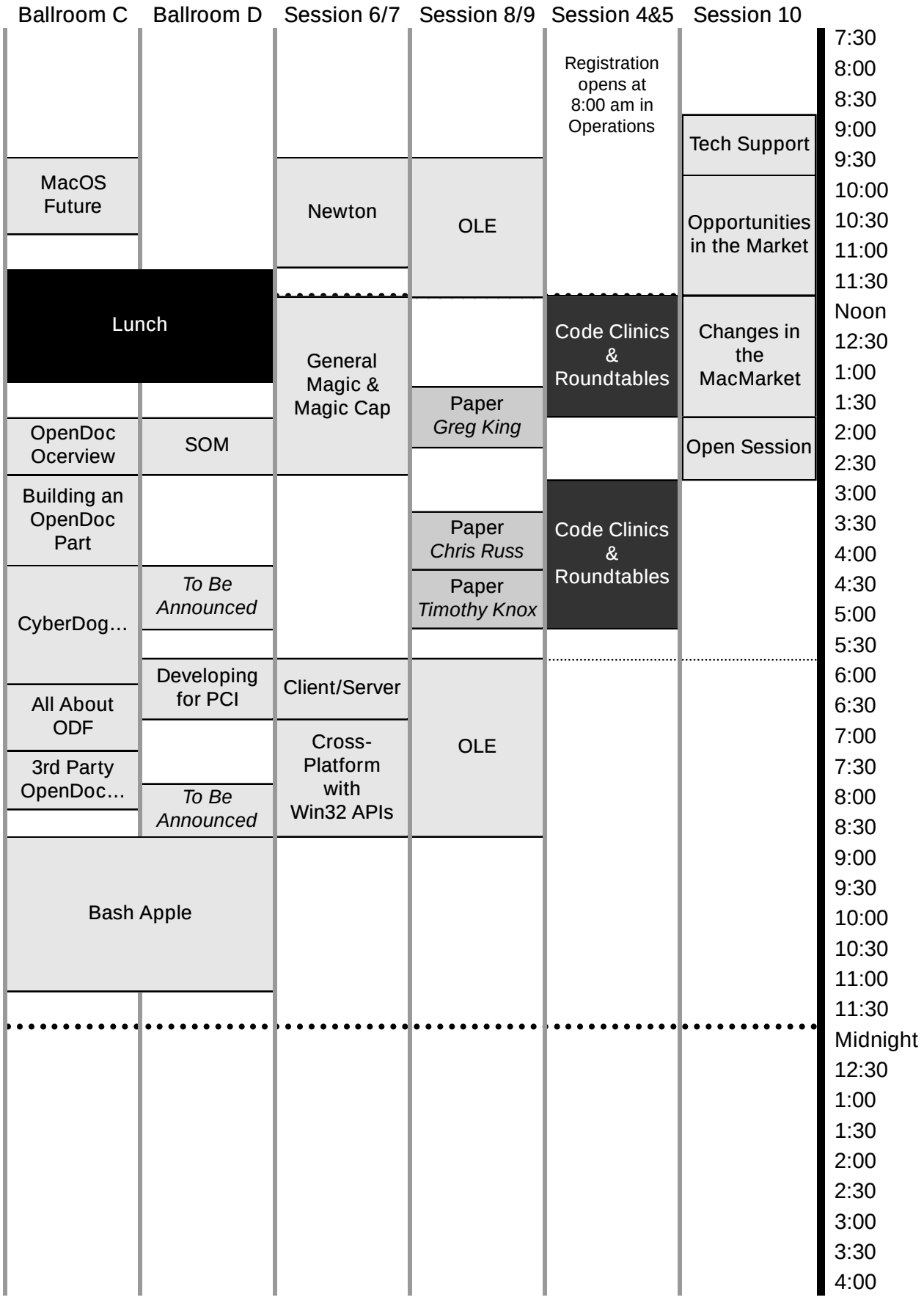
Copyrights, trademarks, intellectual property rights. Internet and clipper updates.

Wednesday June 21, 1995



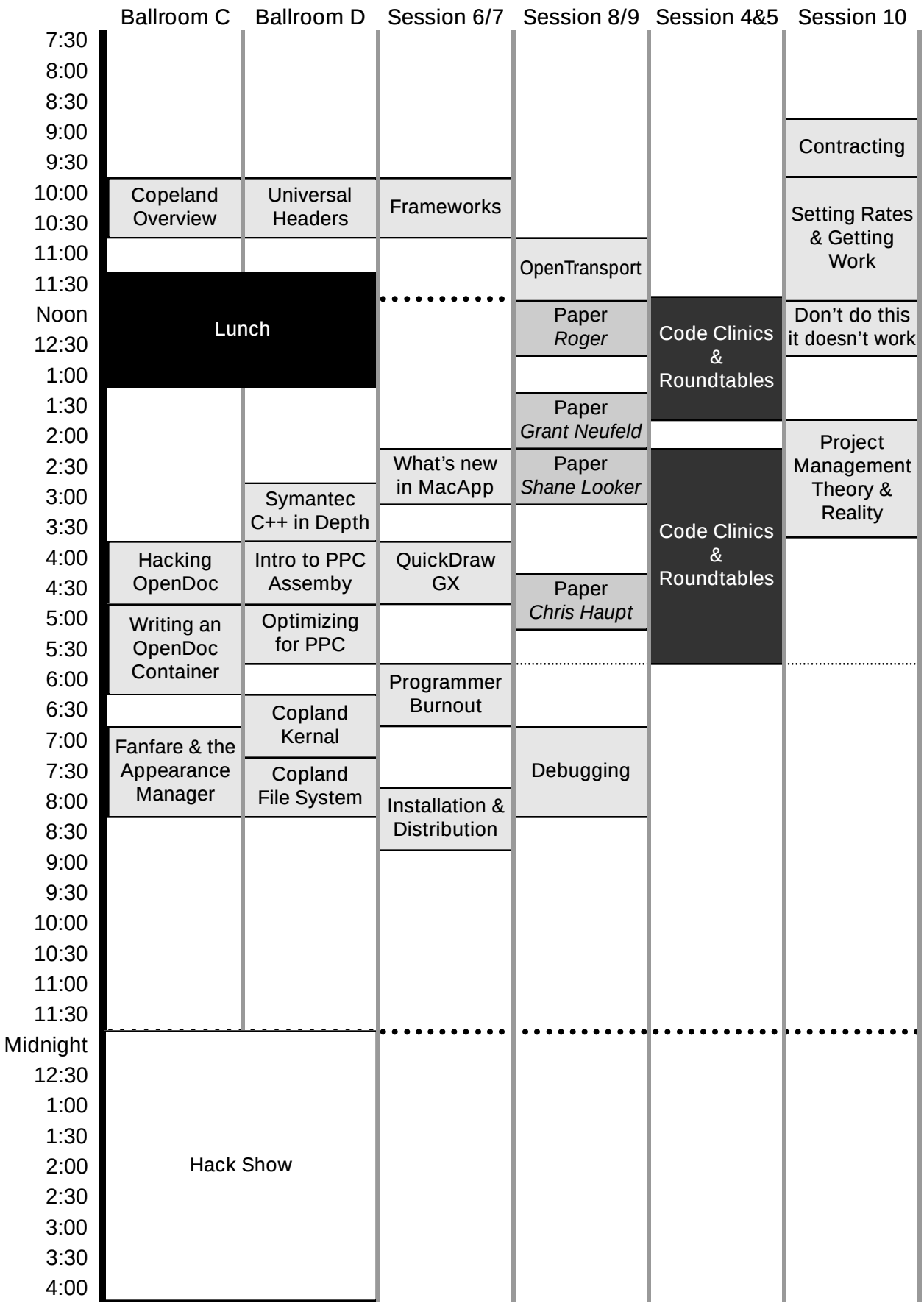
Please check the daily updates for sessions and events which may be moved, cancelled or added.

Thursday June 22, 1995



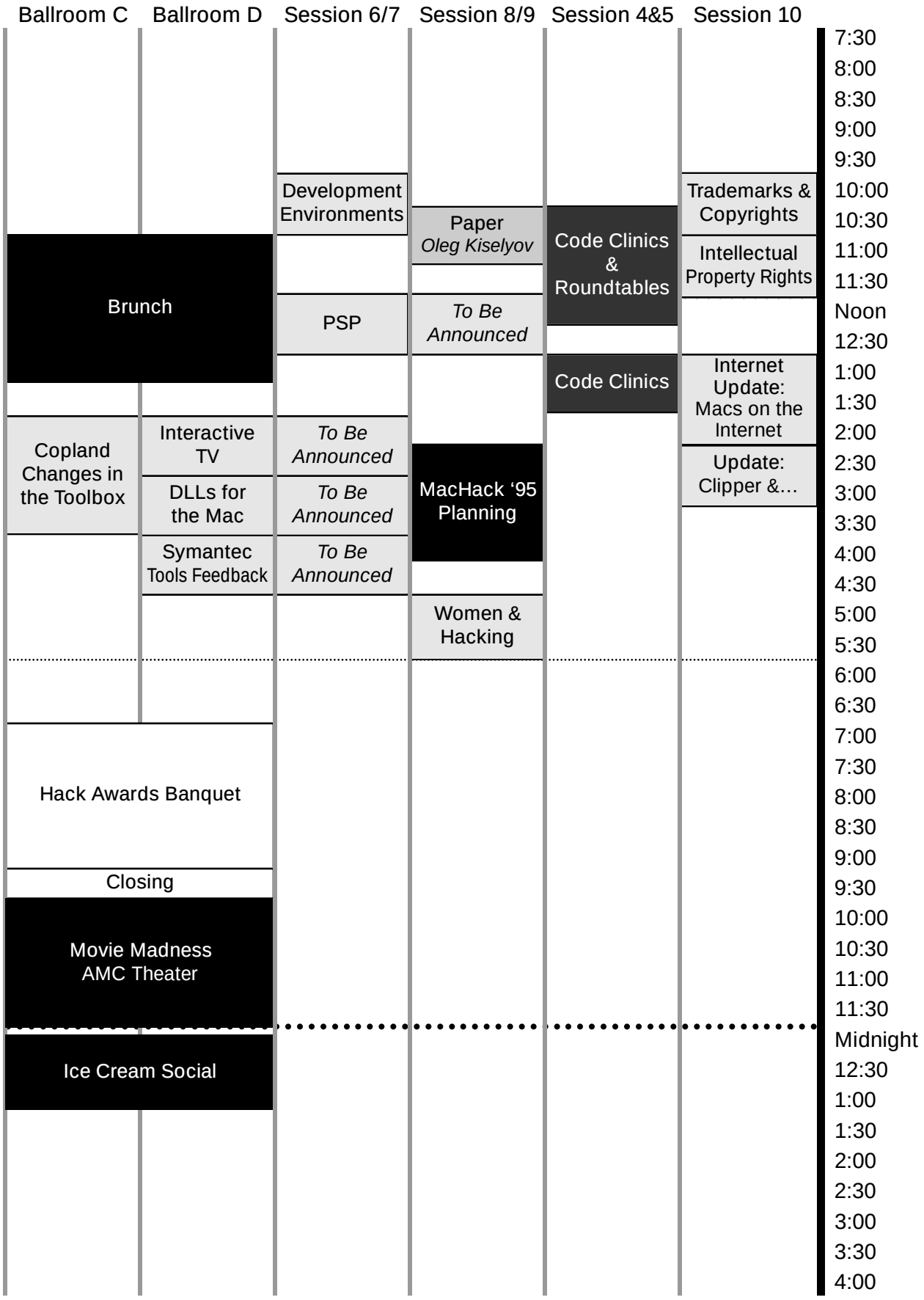
Please check the daily updates for sessions and events which may be moved, cancelled or added.

Friday June 23, 1995



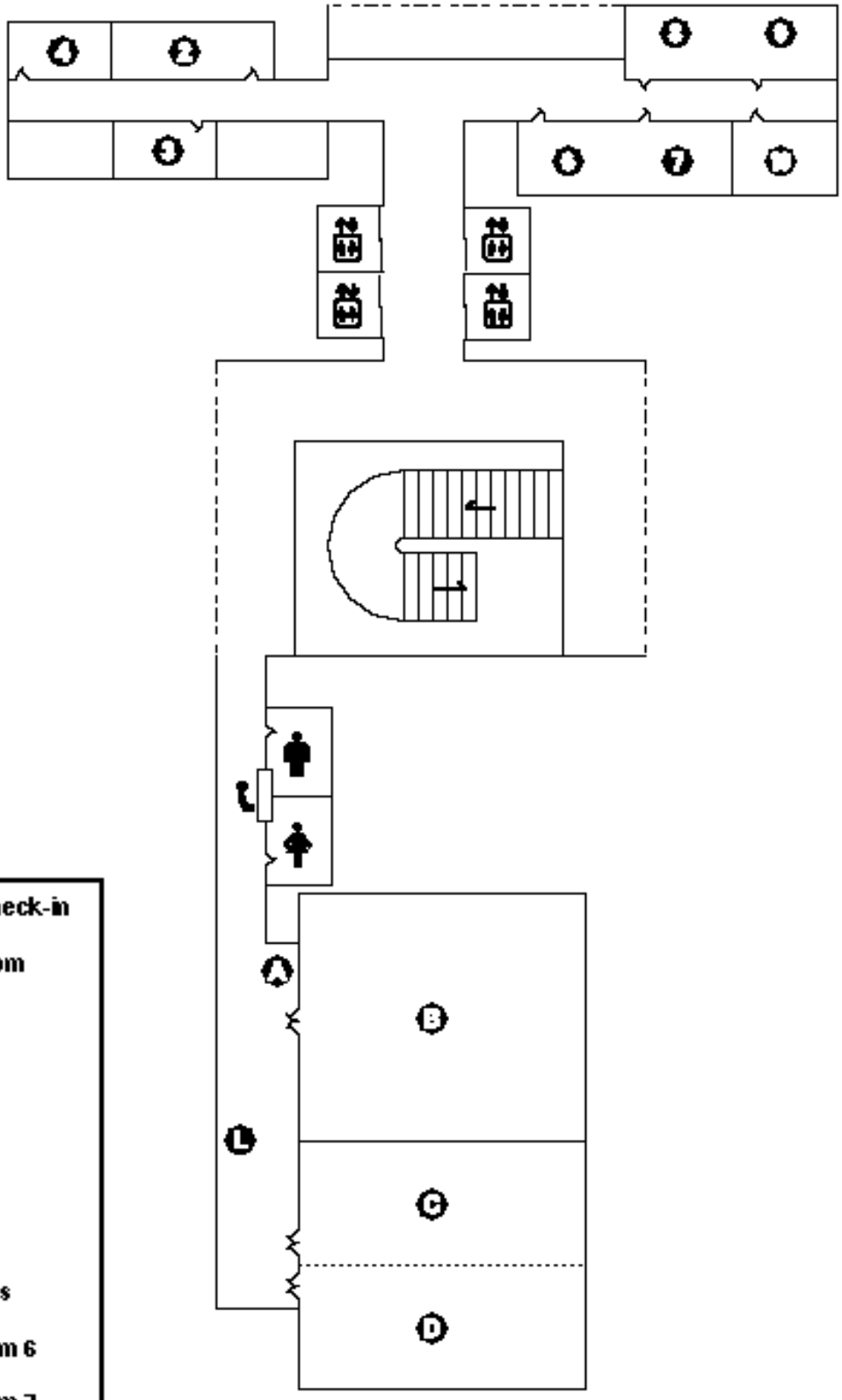
Please check the daily updates for sessions and events which may be moved, cancelled or added.

Saturday June 24, 1995



Please check the daily updates for sessions and events which may be moved, cancelled or added.

Conference Map



- A Hardware Check-in
- B Machine Room
- C Ballroom C
- D Ballroom D
- L Lobby
- 3 Operations
- 5 Code Clinics
- 4 Round Tables
- 6 Session Room 6
- 7 Session Room 7
- 8 Session Room 8
- 9 Session Room 9
- 10 Session Room 10

MacHack is a non-smoking conference

The Care and Feeding of a Small Developer: Or How I Stopped Worrying and Learned to Love the BOM.

Christopher Haupt, CyberPuppy Software, Inc.
cfh@netcom.com

This paper examines some of the issues that must be considered in the process of becoming and succeeding as an independent commercial software developer. Whether the reader is currently a hobbyist, student, or professional programmer who is ready to strike out on his or her own, this paper provides some guiding principles culled from the experiences of the founders of the author's company and anecdotes from others that can provide some food-for-thought to the new entrepreneur. Areas touched on in this survey include: the philosophical aspects of starting your own company, legal and financial matters, product definition, ways of getting your product onto the market, and developer/publisher arrangements. The paper then presents the concept of a small developer cooperative— an entity specifically aimed at addressing some of the more onerous issues discussed above.

Introduction

This paper provides an overview of a number of important areas for thought when contemplating the leap into independence. It is something of a combined essay and a pointer to more reading. In being somewhat general, I hope to provide a catalyst for your own thinking and research on starting a development business. The ideas here are culled from the personal experience of starting CyberPuppy Software three years ago. When we started, none of the principals had any formal business experience. So in some sense, we've had the best education possible, the real world. My intention with this document is to impart some of that experience, but to also form an introduction to deeper discussion, both in the form of talks and other writings. I look forward to hearing from others with ideas to share and areas to explore.

Going Independent: A Philosophical Sojourn

Have you found yourself approaching the end of your college program not knowing what you wanted to do? Have you written a cool game, placed it on the nets as shareware, gotten that first check and thought to yourself, "gee, could I make a living doing this?" Perhaps you have woke up one morning wondering why you are dragging yourself up at such an early hour to make the commute to work. Maybe you've just gotten through the last week of a seven day-a-week, fourteen hour-a-day beta rush and are perplexed why you did it for someone who may or may not thank you and is seemingly taking all of the credit.

These and other reasons are often the incentive for folks to contemplate the entrepreneurial life-style. Why are you thinking about it? Is it curiosity? A really novel idea that just *has* to get out to others? Fame? Fortune?

One thing you must really enjoy is hard-work and pain. According to recent industry reports, during 1994, only 4% of software firms were profitable [Gloster95a]. In general, of the roughly one million businesses started in the US each year, 40% will fail during their first year. Within five years, more than 80% will have failed. Of those surviving, roughly 80% of those will fail after five years [Gerber95]. In our industry, the generally accepted odds are worse. Approximately 1 in 10 companies survive their first year. The emotional and financial strain these odds place on the new entrepreneur should be considered when evaluating your career path.

The most important first step to becoming an independent developer, regardless of the eventual way you go into business, is to understand your own motivations. What are your goals? Make a list of the primary things that motivate you. Is it money? Working on your own schedule? Cool new technology? All are valid reasons, but without knowing what you want, it is nearly impossible to plan a business that will make you happy.

After identifying your motivations and desires, you must next take stock of your resources. What are your strengths? Michael Gerber [Gerber95], describes three base skill classifications: The Entrepreneur, the Manager, and the Technician. The Entrepreneur is the dreamer, the part of your mind that creates new ideas and thought processes. The Entrepreneur is always wondering, and often interrupts the functions of the other two traits. The Manager is the organizer trait. It concerns itself with the details of running the day-to-day operations of a business. The Manager makes sure all of the t's are crossed and the i's dotted. The Technician is the doer trait. It gets the actual work done. It is the portion that will bring the dream of the Entrepreneur into reality under the schedule of the Manager. Most technical people will favor the Technician trait, and they will start their business in this mode. The important thing here is that you must recognize each trait as important, and balance their contributions to make

the business successful. Gerber proposes a "Business Development Process" that assists this balancing act.

Once you understand yourself, you can get on with the task of organizing your business. There are a number of excellent resources for understanding the issues of entrepreneurialism and small business. If you are totally naive to the ways of business, you may want to investigate some of the many general books available in any good library or book store. [Brandt82] provides a list of "ten commandments" to keep in mind during the formative and operational phase of a new business. This source is a good place for getting started. It doesn't go into great detail on any one subject, but rather points the way on such details as business plans, distribution of controlling interests in partnerships, planning for growth, etc..

Legal Bungee Jumping for Fun and Profit

After exploring your goals and motivations and getting the first frightening glimpse at what the basics behind running a business really are, you are poised to make the leap. It was at this point in my experience that I began to appreciate what I was missing from formal business training. Perhaps there was some value in those business classes others were taking while in school. All I remember are long summer nights in the graphics lab while my business student "colleagues" were kicking back there second keg. I feel better now. Having spoken with many a business owner, it appears to be true that the School of Real Life provides much more useful training than that coursework could have. Still, I guess it would have been useful to learn their jargon and formal theory.

But I digress. You are going to start your own business, fine, but what form should it take? At this point in our journey, I become somewhat US-centric. Fill in some of the following paragraphs with the appropriate information from your home country.

First, what are going to name yourself. While not mandatory for some of the various business entities, creating an identifiable mark for yourself is very useful. Additionally, your company name is legal property. As such, it adds value to your company, especially over time as it becomes well known among your customers. Choosing a name other than your own really requires a search of current trade names to avoid conflict with other companies. You don't want to start your business with a copyright or trademark infringement suit! Assuming you can find a unique name you like, you will need to register it. Depending on your state, you register with your city, county, or the state itself. This is generally an inexpensive process, although the search process can have some cost behind it if you have your lawyer do it.

Once you get your name, you can now get things like bank accounts using that name. You will be using your "DBA" (Doing Business As) certificate for this purpose.

OK, you have a name, but what kind of entity will your company be?

Each choice has its pros and cons, and the best way to analyze them is to have a specific idea of what you want to accomplish, who you want to work with, and what your relationships are going to be like with other companies and customers.

A sole proprietorship is what you are if you are starting your company by yourself. You do not need to fill out any formal paperwork. The proprietorship requires some minimal extra tax preparation—you need to fill out an extra form along with your personal tax forms. One negative aspect of a proprietorship is that it does not protect you from legal attacks. You are liable for all suits, and your personal assets can be attached to in the case of a judgment against your company.

If you are forming a company with one or more other people, then you can create a partnership. Like a proprietorship, you need to do very little legally to form the partnership. The partnership will provide some tax benefits. Like a proprietorship, partners are still individually liable in the case of lawsuits or debts. To make things more scary, if any one partner should be the target of a particular problem, all partners are liable. It is very important to have a formal agreement between you and your partners which delineates your stake in the entity.

Finally, there are corporations. Here your get both tax implications and significant personal liability protection. Additionally, you can have multiple shareholders, providing different ways for allowing investment in your company. Corporations are much more formal, legally and from an accounting point of view. As such they are the most expensive entity to create.

Which one do you choose? Well, it depends. How comfortable are you with risk? Are you alone or do you have partners? How much starting capital do you have? The most important thing to do at this point is to find an accountant and lawyer you are happy with. You must decide what your tax situation is—or how you would like it to be. You must also figure out how much liability you are willing to expose yourself to. Your lawyer and accountant can help you with these questions. Bob Schenot [Schenot94] provides a good description of some of these points from the viewpoint of a shareware author.

Case Study Part One: A History Lesson

My company, CyberPuppy Software Inc., was formed in 1992 by myself and two partners. We were college faculty who literally met and came up with a product idea over a pizza lunch. We were talking about the state of children's software, and just decided "Hey!

We can do a better job at kid's creativity software than most of that junk..." (Yeah, we were a tad naive, but we did know a thing or two about our subject matter.) At the time, the most popular, and some might say only real successful children's creativity product was Kid Pix. We decided to do a technological leap frog, so Kid's Studio was born.

We decided to incorporate for a couple of reasons. We wanted to minimize our personal exposure to lawsuits, and we wanted to be able to have multiple shareholders for investment purposes.

We wanted to both develop and publish our concept. We didn't have a lot of money, and didn't really know many people in the industry we were leaping into. We figured that at worst, we could develop the product in a few months, and through a grassroots effort, get boat-loads of copies into the hands of excited children whose parents had wads of spendable cash. Then we woke up.

You Got A Company, Now What?

Let's assume that you've gotten this far. Now you have a company, and you have some great product ideas. You've decided you want to do entertainment software. You have some money in the bank, enough to live off of for a while, but not enough to publish your own product. What next?

Now you have to decide the initial direction of your company. Do you go ahead with your product idea, simultaneously looking for a publisher? Do you decide to go grassroots, shareware, or just give it away? Do you shelve your idea and look for some contract work to build up a bigger war chest?

All are valid approaches, and there are probably many others. Let's assume that you and a friend decide to go ahead with building your product. Because of your funding situation—in this case, let's assume you are self funded, [Gloster95b] is a good discussion about other bootstrapping and financing options—you choose to do a strategy/puzzle game in the same vein as Oxyd, Tetris, or some such. You won't need a huge staff, you can coerce some local college art talent to provide you with some nicely rendered custom graphics. A friend who is a MIDI wiz can provide some snappy music and sound effects on the cheap. You avoid high costs and legal problems by not using copyrighted content.

The technical aspects of such a project are left to the reader as an exercise. Of course, a word to the wise is to make sure you are well organized, [McConnell93] and [Maguire94] should definitely be required reading for the new team.

Let's address two more business related problems. First, what should your game or program do, and second, how do you make money?

Market Research is More Than Comparing Safeway to Acme

What should your product do? How can you make a lot of money with it? These are the kinds of questions you will have to answer through market research.

Usually, the first product ideas for a new company come from the founders' sense of what interests them. You should classify these projects by what I call the "esoteric-filter". Is the program a real niche type with a potentially limited audience, or would it be of wider interest. This is the difference between a beer-bottle labeler program and DOOM. Usually, this judgment call can be safely made via gut instinct. If you are having trouble, however, it is important to get some potential customer feedback.

The first step is to understand where the portion of the industry you want to compete in is going. Frequent analysis appears in industry journals, often sponsored by the Software Publisher's Association (SPA) or independent research firms. [Wiegner95] provides one such general outlook, showing the growth patterns for the personal computer software industry as a whole.

An easy next step for your research is to know the competition. Are there other products out on the market, either retail or shareware/freeware that are similar? If so, is your product different enough to pique the interest of owners of competing products as well as new customers? If not, go back to square one. In either case, get your hands on as many different products as you can. Make a list of features that work for you, and those that don't. Ask users on the net what they think about the different products. Target users who represent the customers you hope to capture. If you are developing educational software, for instance, ask individuals and school representatives.

Competitive research helps you refine your feature list. It also shows you what the currently hot products are. For instance, as of this writing, firstperson maze games are very popular. After DOOM took off, a number of popular follow-on products came out for most platforms. You can use this kind of information to determine if an area is getting too saturated. If there are more than a dozen of this kind of program, is the market space getting too crowded? Also, does any one vendor have a majority of the market share for that kind of product? You can get this information from industry analysis. Sources such as the SPA publish breakdowns of key market groups. Specific subsections of the industry are often served by special trade organizations. Entertainment (and education to some degree) is served by the Computer Game Developers' Association (CGDA), which publishes a regular newsletter with market analysis. Check out the Internet news groups for a sense of the pulse of a particular area. What products are being mentioned

most often? Which ones are getting slammed? Why? [Schenot94] and [Moore92] both provide additional hints on refining your product idea. You should try to develop a product that fits within a hot category (or an anticipated hot category if development is lengthy). Your product should in some sense "leapfrog" over the competition. What is the WOW factor? This may be better technology—like a real color, real-time rendering engine in your game—or better functionality—as in easier to use features, more options, etc.. [Radin94] and others agree, if your project is just another me-too type of product, you may as well throw your startup money away. Plan to blow the competition away. Test your ideas against the common mistakes described in [Marriner92]. Marriner enumerates the most common mistakes made when nearing launch time of your product. Many of these can be caught and fixed at the product definition phase.

Unless you are starting out with a lot of cash and a full team of programmers, designers, artists, etc., you should concentrate on what a small, focused team does best. Go for the niches in which the larger players just can't seem to compete. As a startup, you have the flexibility to change instantly as the market moves. Considering the production values of today's multimedia games and products often call for US$500K plus budgets, look for those cracks in the market where you can work on establishing a beachhead. [Estvanik95] discusses such strategies. If you are doing entertainment, look for "genres that can produce crisp, tight little games: intricate puzzles, strategy games, even wargames." Utility programmers might shoot for projects that can add to the latest and greatest OS features. Or, be retro, support all of those users out there with older hardware and older software. Frequently the larger companies must migrate their packages on to follow the customers on the middle bulge of Figure 1. Take advantage of the folks sitting on either side. Both could be lucrative.

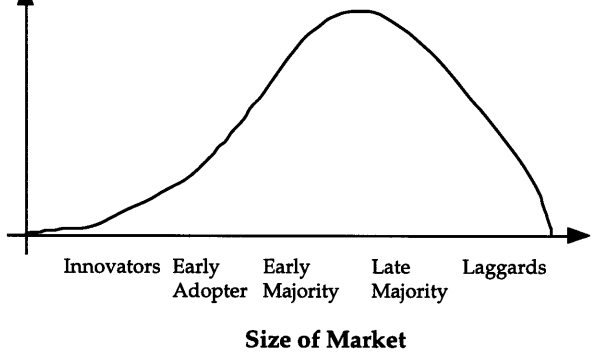


Figure 1. Technology Adoption Curve

Case Study Part Two: The Wrath of Pup

In 1992, when we formed CyberPuppy, we had a mission in mind. We had been looking at the children's creativity market and were not impressed with what we saw. The majority of schools and homes that we interviewed were using KidPix, a cool, but dated painting package. We wanted to give kid's something more. Being that one of our personal interests was story telling, we decided that we wanted to give kids all of the potential tools that they would need to put together a full presentation. Our philosophy has always been that too many adults don't give children enough credit for innate intelligence and creativity. Given very powerful tools, kids will do amazing things that many adults would be hard put to accomplish. We modeled are plans for products around this philosophy and an observation I developed that computer users in general can be classified as in Figure 2.

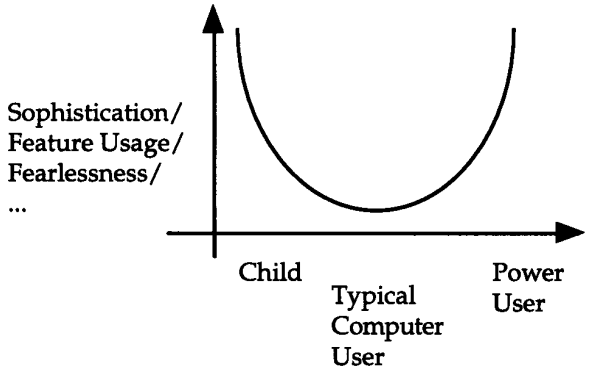


Figure 2. User Sophistication Characteristics

Kids and power users tend to sit on either high point of the curve, while the majority of users fall in the trough. It is interesting to note that as users get older, they tend to sink in their usage of power features on a day to day basis. It is also interesting to note that as the computer user population ages, this curve will change, flattening out somewhat. There will always be a majority in the dip, though, as the mentality changes to a "gotta get work done" mode.

But I digress. Following the above, we came up with Kid's Studio, a presentation and movie making package. It was designed to completely leap over the competition of the time, both in overall functionality and performance. A year after it was on the market, it was recognized by the SPA as the best creativity/productivity tool for educational use. We had finally broken through!

Turning SuperWizMagic Into Big Bucks

The name of the game is customers, customers, and more customers. From the customers, all else flows. Hopefully that includes a healthy dose of cash and credit card numbers.

It can seem counter-intuitive at times, but for your first break into the market, giving away as much product for little or no cost is often one of the best strategies. Keep in mind that if your plans are to be more than just a contract developer, your most important asset over time will be your customer list.

If your customers are happy with your products, they will tend to continue to buy both new products and upgrades to current programs. For many products the cost-of-goods (COG) is less than US$2, but the potential upside is five or ten times that, at least.

Continuously search for new and creative ways to gather mind share and build that list. [Kawasaki90] is one of the ultimate sources for cultivating an evangelical outlook. You will make customers happy, and possibly garner the attention of the press as well.

Okay, you can give stuff away, but how do you make any money? Here I briefly touch upon four paths. The first three, shareware, direct marketing, and low cost retail, make the assumption that you are your own publisher, or that you have a significantly flexible enough arrangement with a publisher that you can target some channels on your own. The fourth approach is to become a "pure" developer, linking up with a publisher to handle marketing, sales, distribution, etc..

[Schenot94], [Radin94], and others agree that a good marketing and sales plan will involve a mixture of channels approached together. While it is possible to scratch out a living by pursuing just one channel, a concerted effort can have significant advantages. Here, I will briefly discuss a few channels, leaving the process of building a plan of attack for selling your software as an exercise.

Shareware

Shareware is perhaps the channel with the greatest confusion associated with it. This can probably be traced to its evolution over time. During the earliest days of microcomputers, shareware was the source for many new and exciting products. Often these programs would be free. Shareware would often be written by individuals who were primarily interested in solving an interesting problem, or creating a neat piece of software that they would use. Today, much of that motivation is there, but profits seem to play a bigger role. The shareware channel is being used more often by companies of various sizes to promote their product and build a customer base [Gerrold94]. From that base, they hope to lure users into buyingthe commercial counterparts of the shareware product. The try-before-you-buy for little or no cost model of shareware provides a powerful sales tool.

According to surveys conducted of shareware authors, shareware is not the road to quick fortunes [Garrett90], [Corbier94a], [Schenot94]. Of the surveys conducted in 1990 and subsequently in 1994, the average selling price has stayed about the same (approximately US$15 for Mac programs, US$30 for DOS and Windows) [Corbier94b]. According to the surveys, an estimated 1% to 10% of the users of shareware programs actually register. Most authors concur that the actual numbers are probably closer to the low end of that range. The average income over a product lifetime period is $3220 according to [Garrett90].

There are certainly some standout exceptions to these numbers, but for each of those highly visible money makers, there are hundreds of zeros. The key to success is to build a high quality product that will be useful to the largest possible segment. Your attention to technical detail and your marketing research both play large roles. If your goal is to grow your development company, then you should take advantage of the shareware channel to attract a customer base you can later upsell into. The income you derive should be considered a nice secondary benefit.

To maximize your number of registrations, you must make the process as simple as possible. Because your software will likely travel around the world, you need to consider the problem of support and payment from different locales. While this can be done by an individual, there exist a growing number of registration services that you can outsource this work to. Appendix 1 lists several such services. The benefit of using a service is that they can take a variety of credit cards, in addition to other forms of payment. Surveys and individual interviews have shown that the greatest number of payments comes through the credit card avenue. (Especially since many of the international issues of exchange are hidden from you.)

Direct Marketing

Soliciting business through the mail, telephone, and increasingly, email or newsgroups is the venue of direct marketing. There are really two important aspects to direct marketing. First, it is your best avenue for keeping in touch with your current customers. The people who have bought your earlier products, and even better, have registered, are the most likely to spend more money on followon products or new programs from your company. The second aspect to the direct approach is that it can garner new customers if implemented well.

If you plan your product such that it can grow with new features, new levels, new tools, etc., you can ascertain a constant income stream, assuming, of course, that it achieves some degree of popularity. Episodic games, utilities, and content-consuming programs are good examples of these kinds of products. [Schenot94] reports approximately 33% of active users of a product will upgrade to a new major version. Our experience at CyberPuppy corroborates this; a major upgrade offered through direct mail to our customers last year saw a 36% response rate.

Communicating with current customers is relatively easy. You already have the names of people who are supposedly interested in your product—after all they

bought it and bothered to register. But what about the second aspect of direct marketing, gaining new customers?

This is the domain of the rented list. Mailing lists are available from a number of sources. Often, lists can be rented from major computer (and other) magazines.

The trick to direct marketing when trying to generate new customers is to find and rent the most focused name list possible. [Schenot94] provides a good starter table of list brokers on pages 146-147. Magazines that cater to your target audience can be contacted directly.

Once you have a list, the rules of thumb are to test, test, and test some more. Be sure that the contract you sign when renting a list will allow you to do some test mailings. Different brokers have different policies. What you want to be able to do is send out several different small mailings, say of about 200 pieces, and discover what combinations of mailings give the best results. After you are satisfied with the response rate, then you can go ahead with a larger drop. Direct mail that is focused can be very effective, however, a decent campaign is not cheap. You can expect to spend a good US$5000 on a 10,000 piece mailing. A good response rate will be between 3% and 8% Johnson92].

Because you generally can use a rented list once per payment, you want to try to as quickly and efficiently as possible capture some feedback from customers. (Using a list more than once without renting it is a Very Bad Thing. Most lists have some number of "guard names" used by the owner to determine if someone is stealing list usage. If you are caught, you may be black listed from using that, and perhaps other, lists ever again.) Every name that you get from a reply card is a name you can permanently keep in your database. Sometimes it pays to get the user to contact you with a reply card or call even if they are not going to buy immediately. Such names are excellent candidates for repeat mailings. Often it takes several pieces to build up some customers' comfort level in a company and product. This approach certainly makes more sense when your product is more expensive and you can better afford this war of attrition.

There is a whole art form to designing direct mail that targets new customers. [Schenot94] [Radin94] [Stone] are all excellent sources for guidance on a direct mail campaign. One of the easiest things to do is to start saving your junk mail. You will notice that many of the mailing you collect are similarly designed, even though one may be for a computer product and the next for a music club. Some key elements include:

- The product and its benefits should be simple to understand
- The purchase of the product should entail low risk to the buyer
- Response should be very simple: include a postage-paid card, 800 number, etc.
- The offer should have an expiration date clearly shown on the mailing
- The mailing should be as tightly focused as possible;
- Code your mailings so you can determine which approach is getting the best response
- Involve the customer with the response: use stickers, check boxes, puzzles, etc.
- Use repeat mailings (of your legal names!)

Low Cost Retail (LCR)

Low Cost Retail is a growing avenue for a number of small developers. You may have seen some of the racks in non-computer stores that indicate a LCR publisher. The software products in these racks usually form some kind of "meta-line" of the LCR publisher. Products are low cost, and appeal to impulse buyers and computer neophytes. Often, the products are shareware items that have been dressed up with a new, "exclusive" interface or set of features.

The product mix in this channel appears to be mostly games, educational products, and some home productivity items. [Estvanik95] discusses LCR from a games point of view while [Schenot94] provides a more in-depth treatment, and includes a list of LCR publishers looking for products.

Others

Many other channels exist. Bundling, Internet and On-line commerce sites, CD-ROM collections, and self publishing are just a few more areas to explore. The sources quoted in the above sections are good places to start learning about these opportunities.

Publishers

You may decide that the problems of getting the product out on the market, getting it built, handling the sales calls, etc. are just not what you wanted to do. If developing cool products of either your own design, or excepting work-for-hire from an interesting company, are more your speed, then you should consider finding and linking up with a publisher.

The advantage of this avenue is that you can generally concentrate on the technical aspects of your business. You still have work to do in running your company, but the marketing and sales, distribution, and support portions can be mostly forgotten.

There are two general routes to working with a publisher. The first is more of a contract programming deal. In it, you work with the publisher to implement a project that some other entity has designed. In

essence, you are strictly a programming shop—the creative aspects of coming up with a new product are left to others. I won't go into this option.

The second route is to design, build, and test a product of your own. You only need the publisher to get the product out onto the market and to supply some needed income. There are a number of variations on this, ranging from the programmer shop described above to a fully affiliated label. I will concentrate on the middle ground.

Developer/publisher arrangements work much the way an author of a novel works with a book publisher. The developer comes up with an idea, researches it, implements it, then tries to find a publisher who thinks that it will appeal to a wide audience. The relationship with a publisher often starts at the conceptual phase, especially once you are an established developer with a successful track record. As a new developer, unless you have created a significant work at another company, a publisher will want to see your "portfolio". In essence you bring a full featured prototype, or possibly a finished app, in for review. Different publishers will specialize in different software, obviously. If you have a new game, find all of the companies that you can that currently have games of the same genre. At this discovery stage, networking is the most important thing you can do. Go to industry shows, developer conferences, user groups where companies will be visiting, and so forth and bring your best work. The process is like interviewing for a job, just a bit more intense.

With some good fortune, you will be able to match up with a publisher that believes in your product and your development team. At this point, the fun begins to create a contract. Most developer/publisher arrangements work on a royalty basis with some sort of cash advance. Typical royalty amounts fall within the 8% to 15% range of net receipts. This amount depends greatly on your past experience, product type, and negotiating skills. It is also dependent on how much work has yet to be completed on the project. Often when projects are in earlier stages of development, a cash advance will be granted to assist in the completion of the program.

Royalty percentages seem really low, but you must bare in mind that the publisher is taking the greatest portion of risk. Your product may never get completed, needs to be built, marketed, distributed, advertised, and may still bomb in the marketplace. In contrast, percentages for affiliates or pure distribution agreements can often have 75% going to the developer and 25% to the other party.

Case Study Three: Luck Helps In Finding a Partner

When CyberPuppy was searching for someone to help us publish Kid's Studio, we didn't really know what kind of arrangement we were looking for. We had assumed that we would actually publish the product, and just look for distribution assistance. Being relatively new at the game at that point, we didn't really realize that the publishing aspects alone were so expensive. A proper product launch can easily run US$400K; marketing, sales, and manufacturing costs will usually be twice or more what development cost.

Our lucky break came while visiting as many potential companies as possible. We were going around showing our early versions of the product, until a fortuitous meeting with Maxis. Maxis was in the process of setting up an affiliate label program and wanted to expand its product line with other creativity and children's products. Our product fit the bill, and we became affiliate number two. It was that experience that gave us the best education on what getting a product sold through to customers really takes.

Later in Kid's Studio's life, the product was sold to Storm Software. Storm wanted to enter the consumer space, including children's software. Storm had evaluated a number of products, but its staff most enjoyed Kid's Studio. CyberPuppy actually received the initial call from Storm asking about the product's availability from out of the blue. Sometimes your product will be your best calling card, especially if you are getting it into customer's hands in some fashion. Keep in mind that LCR and some of the CD-ROM distribution channels often find their products by scanning the nets for excellent shareware.

Software Development Contracts: Roadmaps to Fortune or Ruin?

Upon finding the publisher of your dreams, you must hammer out a contract which will explicitly detail such things as deliverable schedules, royalties, payments, and other legal aspects of the deal. The two most important tools needed at this stage in your business are your ability to negotiate and a lawyer who deals with software contracts on a frequent basis.

Negotiating a developer/publisher deal is largely effected by your experience in the genre and your credibility. To earn these attributes, it takes time and hard, high quality work.

Regardless of whether you are a neophyte or an experienced developer, many of the issues are the same. [Gloster95a] points out a number of key points that you should carefully consider and negotiate to the best of your ability.

The first point covers royalty rates. As described earlier, in this area you may have the least room for negotiating. Larger publishers have the clout to demand smaller rates, and the reverse is usually true of smaller publishers. Why go with a larger firm

then? Generally, you have to assume that they have more resources at hand to sell units. More units, even at a lower royalty, can mean more money for you. Even more importantly, if you can negotiate access to customer lists, you can build up your database much more quickly. Often publishers will accept so-called "step-up" royalties. Step royalties are those calculated over time based on the number of units sold. While you may start at a lower percentage X, after n units are sold, you will start getting X + Y percent. Often you can negotiate multiple steps.

Keep in mind, though, the reality of selling many units. A trap some developers fall in to is to accept a really low initial royalty, with significant incremental steps for large unit sales. In today's blood-bath market, selling 50,000 or more units is significant, and selling over 100,000 qualifies as a best seller.

Regardless of royalty rate, you will want to work out a royalty advance schedule that meets your anticipated needs. Payments are generally made on completion of milestones agreed upon in the contract. Be certain that the advance you receive covers all of your calculated costs. Include all salaries, development costs, and a buffer to cover unforeseen expenses, delays, problems, and something you can put away for times between projects.

The second important issue is royalty calculations. The dollar amount used to calculate the developer's payment can wildly effect the amount received.

Usually a net revenue base is used. This is the amount left after the publisher adjusts his gross by deducting expenses and return reserves. Keep in mind that the publisher's gross rarely starts at the street price. Publishers will usually sell product to distributors and stores for 50% to 60% of the standard list price. The publisher will try to deduct as much as possible from this gross before calculating the developer's cut. It is in your best interest to define the deductions as narrowly as possible. [Fishman94] and [Gloster95a] provide good general descriptions of these calculations.

The third point includes royalty recoupments. The compensations you receive during milestone payments are normally considered advances against future royalties. The publisher will specify how those advances are to be repaid in your contract. Depending on how this is structured, it will greatly effect when you will see any future income from your product.

The standard method is to not pay the developer until his royalties earned at the current percentage of net revenue exceed the advance. This procedure can take a significant time, especially if the product sells slowly or is a low cost product. Even worse for the developer, given today's cost of goods, it is reasonable to assume that the publisher will have made a decent profit long before he had to start paying the developer again.

There are a number of ways to work around the standard method. [Gloster95a] suggests several. The developer could negotiate a "shadow" royalty rate on the first Z units sold. This rate would be at higher than his normal rate, thereby paying back the advance at a quicker pace. The developer and publisher could also agree that when a specified number of unit sales are achieved, the advance is considered paid off. Perhaps an even better technique is to negotiate an alternate unit of calculation. In this method, some portion of each unit sold will go towards the repayment, while the remainder will be paid out to the developer. In this way, you will see some cash flow immediately.

A fourth major point is that you need to consider what is covered by the contract, and what you will earn a royalty on. Publishers will want all rights to future related works derived from the product. A publisher wants to be sure that if the program is a hit, he can cash in on sequels, new uses, ports to other platforms, etc.. You should be sure that your royalties will be extended to cover these new items. It is usually possible to keep royalties coming even if the new works are implemented by other developers, albeit at a lower amount. Keep in mind that a publisher who paid for the development of the product will usually be considered the owner of the code and related materials. If you have developed technology which is of general use to your work, and have included this in the current product, you will need to either exempt it from publisher ownership, or license it back to yourself if you are selling it.

Gloster's last major point for consideration is coverage of termination issues in the contract. Standard publisher agreements will allow the publisher to reject milestone components and subsequently terminate the contract if the problem is not resolved. Boiler-plate agreements will often state that such cancellation is "at publisher's discretion, for any or no reason." It is extremely important to nail down several issues. You must know who the publisher's representative is to be that will be authorized to accept or reject milestone deliverables. That person should have a fixed amount of time in which to respond to each delivery. If after the expiration of that time no word is received, it can be assumed that the milestone was acceptable to the publisher. If the deliverable is rejected, the reason for the rejection should be clearly stated and there should be a reasonable period of time given to the developer in which the problem can be corrected.

These five major areas are key points in the developer/publisher contract, and are usually the focus of much negotiation. There are, of course,

many other standard contractual components.

The general structure of a developer/publisher agreement is shown in the prototypes provided in [Fishman94]. [Brown95] describes the key structural points of such agreements and explains the importance of each section. Most of the time, the initial contract will come from the publisher, and will be a boiler-plate document with minor modifications. Usually this means that it is heavily slanted in favor of the publisher. You will need to carefully go over each provision to ascertain it meets your needs and the agreed upon stipulations of your negotiation. You will also want to be sure that special items you have agreed upon are included. Occasionally, the publisher's lawyer will just send along the generic boiler-plated contract, and whole issues may be left out.

For those who want to get a head start on some of the legal aspects revolving around the development contract, several decent resources exist. [Fishman94], [Landy93], and [Smedinghoff94] are books that describe key agreements and provide the forms necessary for doing it your self. They are all written by lawyers, and for simple matters, they are quite satisfactory. One of the most important issues you will want to research is that of owning what it is you are selling. This will be demanded of you by your publisher, and you publisher contract will require certain warranties and representations to that effect. This is especially true of copyrights and licensed materials. The above references provide some good tools for collecting required assurances.

Dealing With People: A Necessary Evil

One of the most difficult areas for Technologists is in negotiation and management of people. To create a successful business, you will need to have some extroverted blood within yourself or on your team. While this topic is certainly broad enough for whole books, I wanted to explicitly mention it as something to think about, especially given the deep and often stressful issues of contract negotiation.

If you feel uncomfortable talking to large numbers of people, negotiating a deal, or managing a team, you should either work towards breaking down that barrier or find someone you trust that will join your company. A couple of useful resources we keep around include: [Freund92], which is a great general treatment of the negotiation mind-set; and [Whitaker94] and [Maguire94], both of which examine the often fun yet tricky road of making a technical team work.

And Now for Something Completely Different

Okay, I have now gone on at some length regarding some of the issues you will need to consider in starting your business endeavor. But what about A big issue that seems to occur with coop-like structures is political infighting. As the coop grows, it often fractures along different lines. Some forethought would be required to keep the process fair. Perhaps the size of a coop would need to be managed as well.

A slight variation on the coop idea is to have what I call a publisher pipeline. Especially in the case where one development company has a good relationship with a publisher, it may be possible to have that developer act as a conduit between the publisher and new developers. This could work in a number of ways. In one, the developer almost has a pseudo-affiliate type relationship with the other developers. In this case, the main developer acts as the communication pipe to the publisher. The arrangements between the pipelining developer and the new small developer could possibly be in the form of a sub-contracting arrangement.

Another pipeline approach would have the primary developer act more as an agent for the new small developer. In this case, the pipelining developer puts the publisher and the new developer in-touch, and if the deal works out, gets a finder/agent's fee for the transaction. This is probably the simplest of the above arrangements.

As you can see, there are many ways that developers could work together in a mutually beneficial way. It is our hope that some form of these (or other) relationships can be built. We are hoping to start the dialog process now, and see if there is enough interest to follow through with more formal activities.

Fin

The purpose of this mini-treatise has been to provide enough coverage of a broad area to give you important issues to think about on the path to forming your own development company. If it is helpful enough to point you in the direction of further research, then I feel it has been successful in its mission. The document as a whole is an ongoing project with CyberPuppy Software. We are trying to pull together such resources for our own ongoing use, but also for sharing with the larger community as a whole. If you have information you feel would be useful to new developer-entrepreneurs, please feel free to send it along and I will include it in our archive. We hope to continue this dialog dynamically on the nets and in person; please do contact us if you wish to participate.

Bibliography

Brandt82] Brandt, Steven C., *Entrepreneuring: Ten Commandments for Building a Growth Company.* New York: Penguin Books USA Inc., 1982.

[Brown95] Brown, Marc E., "Software Development Contracts." *Dr. Dobb's Sourcebook*, No. 239 (1995), pp. 61-64.

[Corbier94a] Corbier, Daniel, *Shareware Author &User Case Study.* 1994. (This shareware document is available on the FTP site listed in the Resources section of this document. email: CORBIER@delphi.com)

[Corbier94b] Corbier, Daniel, *Shareware Author & User Case Study Summary.* 1994. (Available as in [Corbier94a].)

[Estvanik95] Estvanik, Steve, "Guerrillas in the Mix: The Lone Wolf Transmogrified." *The CGDA Report,* Vol. 1 No. 4 (1995), pp. 10-12.

[Fishman94] Fishman, Stephen, *Software Development: A Legal Guide.* Berkeley: Nolo Press, 1994.

[Freund92] Freund, James C., *Smart Negotiating.* New York: Simon and Schuster, 1992.

[Garrett90] Garrett, Kevin, Shareware Author Survey. Available on Apple Developer Mailing CD-ROM.

[Gerber95] Gerber, Michael E., *The E Myth Revisited: Why Most Small Businesses Don't Work and What to Do About It.* New York: HarperCollins Publishers, Inc., 1995.

[Gerrold94] Gerrold, David, "Why Do You Think They Call It a Medium?" *PC Techniques,* Vol. 5 No. 4 (1994), pp.106-109.

[Gloster95a] Gloster, Dean M., "The Art of the (Multimedia) Deal: An Overview of Key Points in Negotiating CD-ROM Development Agreements." To be published in an upcoming journal. Contact author at glosterd@alhooked.net

[Gloster95b] Gloster, Dean M., "Beyond Bootstrapping: Financing Alternatives for Game Developers." *Computer Game Developers' Conference Proceedings,* No. 9 (1995), pp.30-35.

[Johnson92] Johnson, Dave, "How Direct Mail Saved Our Company: Overcoming Fear of Channel Conflict." *Apple Direct,* May 1992.

[Kawasaki90] Kawasaki, Guy, *The Macintosh Way.* Glenview, Illinois: Scott, Foresman and Company, 1990.

[Landy93] Landy, Gene K., *The Software Developer's and Marketer's Legal Companion.* Reading, Massachusetts: Addison-Wesley, 1993.

[Marriner92] Marriner, Leigh, "Ten Common Product Launch Mistakes." *Apple Direct,* January 1992.

[Maguire94] Maguire, Stephen A., Debugging *The Development Process.* Redmond, Washington: Microsoft Press, 1994

[McConnell93] McConnell, Steve, *Code Complete.* Redmond, Washington: Microsoft Press, 1993.

[Moore92] Moore, Geoffrey, "So What's Your New Program All About?" *Apple Direct,* February 1992.

[Radin94] Radin, Dave, *Building a Successful Software Business.* Sebastopol, California: O'Reilly & Associates, Inc., 1994.

[Schenot94] Schenot, Bob, *How To Sell Your Software.* New York: John Wiley & Sons, Inc., 1994.

[Smedinghoff94] Smedinghoff, Thomas J., *The Software Publishers Association Legal Guide to Multimedia.* Reading, Massachusetts: AddisonWesley, 1994.

[Sternberg95] Sternberg, Sam, "The Business Guide." An electronic compendium of notes on integrating your business with Internet resources. Contact author at samsam@vm1.yorku.ca.

[Stone] Stone, Bob, *Successful Direct Marketing Methods.* Chicago: Crain Books.

[Whitaker94] Whitaker, Ken, Managing Software Maniacs. New York: John Wiley & Sons, Inc., 1994.

[Wiegner95] Wiegner, Kathleen, "Outlook 95." *Software CEO Magazine,* Vol. 2 No. 2 (1995), pp. 36-37, 40.

Appendix 1: Resources

Associations:

Association of Shareware Professionals (ASP)
545 Grover Road
Muskegon, MI 49442

Computer Game Developer's Association (CGDA)
555 Bryant St. Suite 330 Palo Alto, CA 94301 415-856-4263 report@cgdc.com

Internet Mailing Lists:

Internet Marketing List (Discusses marketing efforts on the nets) send email to: LISTPROC@POPCO.COM with the body of the message containing: SUBSCRIBE INET-MARKETING

Marketing List (Discusses general marketing resources and issues) send email to: majordomo@mailer.fsu.edu with the body containing: SUBSCRIBE Market-L

Net Archives:

Personal Computer Developer Business Archive
URL: FTP://x2ftp.oulu.fi/pub/msdos/programming/biz

Shareware Registration Services:

Kagi Shareware, 1442-A Walnut Street #392, Berkeley, CA 94709-1405 USA shareware@kagi.com fax: 510-652-6589

MicroGenesis, ATTN: Brad McQuaid or Steve Clover 4055 Park Drive Carlsbad, CA 92008 USA, voice: 619-729-2898, Toll-free Orders only line: 800-294-1302, Internet: bmcquaid@crash.cts.com, America Online: Bmcquaid, Compuserve: 75231,2277

Public (software) Library (PsL), P.O. Box 35705, Houston, TX 77235 USA, voice: 713-524-6394, voice: 800-242-4775, fax: 713-524-6398

Creating a Consistent 3D Interface

by Shane D. Looker

The Macintosh is a popular computer in part because of its consistent graphical user interface. The interface metaphor has been changed in a number of applications from a flat 2D interface to a gray-scale 3D interface. While this is not necessarily bad, the mixing of 2D and 3D interface elements can look rather poor. This paper covers the major elements of a 3D interface and gives guidelines for implementing the interface in a consistent user friendly way.

The Macintosh user interface has been in use for eleven years now. It has been updated by Apple's Human Interface researchers in response to new requirements of the Macintosh as well as in response to developer conventions and experiments.

The most recent major change has been the 3Dish look of dialogs and scroll bars introduced in System 7. While these changes are small deviations from the older black & white elements they signaled the coming of pseudo-3D interface looks from the subtle to the hideous.

Many developers, unfortunately implement a pseudo-3D look in their primary windows, but neglect to carry the look through other windows and dialogs in their application.

But First a Word From Webster's

Before this paper continues, I need to define what a pseudo-3D interface is. I use the term pseudo-3D to distinguish the interface drawn on the screen from the direct manipulation interface for 3D objects, such as defined for QuickDraw 3D. While this may seem bulky and unnecessary, as immersion interface systems become practical, this distinction will become more important.

No matter what the appearance on the screen, all the graphics used in the Macintosh user interfaced are rendered in 2D, trying to give a 3D illusion.

Ruminations on User Interface

The core of good user interface is consistency. The Macintosh interface is consistent throughout applications (excepting Microsoft applications, of course), and with elements such as standard windows and controls, the same look and feel is available across all applications. There are also a standard set of menus in all correctly written applications.

If consistency is so important for good interface design (which it is) why are there so many variations on a theme for pseudo-3D user interface design? The answer of course is that nobody has set down (and published) a consistent set of rules for how a pseudo-3D interface should look. A number of issues have been addressed (such as general look using MacApp in "Working In The Third Dimension" by Jamie Osborne and Deanna Thomas, in Develop 15, November, 1993), but the guidelines have never been complete. Also new interface elements crop up at the rate of one or two every year. This paper is an attempt to gather and codify a consistent interface style. As such, this tends to be a living document that will change slightly as time goes on.

Elements for a Pseudo-3D Interface

There are a number of elements to a pseudo-3D interface which need to be addressed during the visual design phase of a project. Trying to take a nearly completed application and throwing some gray into windows does not cut it.

Primary user interface elements are: windows; dialogs; controls; menus; popup menus; and window/dialog content display. Each of these areas needs to be examined in light of the new interface appearance.

The primary way of implementing pseudo-3D is with gray coloring schemes. Unfortunately, not everybody agrees on the gray shades that should be used for the interface. It is important to remember the purpose of the interface: to assist users in accomplishing tasks, not to hit them over the head with how clever you are. Subtle colorings work best. Using the lightest shade of gray (kRGB8BitGray1 in Table 1) for the general backgrounds works best for several reasons. First the light background is close to white, leaving the most contrast between dark text and the background. Second, if you pick a darker gray, any additional coloring needed in the interface (such as shadowed areas) must become darker still to provide contrast with the background.

Table 1 shows the numeric values for eleven different gray shades evenly spaced along the brightness spectrum. Most of the colors are provided in the standard 256 color palette on the Macintosh, but using that are not in the palette will map to the closest gray value available. This is the palette that should be considered standard when designing a pseudo-3D interface. Note that gray colors are always found along the axis of a color wheel, where each color component, red, green, and blue, are equal in value.

Constant Name	Decimal value	Comment
kRGB8BitGray1	61166	Lightest gray
kRGB8BitGray2	56797	
kRGB8BitGray3	52428	
kRGB8BitGray4	48059	
kRGB8BitGray5	43690	
kRGB8BitGray6	34952	
kRGB8BitGray7	30583	
kRGB8BitGray8	21845	
kRGB8BitGray9	17476	
kRGB8BitGray10	8738	
kRGB8BitGray11	4369	Darkest Gray

Table 1
Grayscale Values

For the best consistent results, use kRGB8BitGray1 for the background colors, kRGB8BitGray4, for the shading areas of the interface, and standard white for contrast areas against the Gray4 areas. Also remember to following standard Macintosh lighting conventions, light comes from the top left corner of the monitor.

Some psychological element of this interface style should also be considered during layout of the interface. Elements that appear closer to a user seem more accessible, and appear more inviting. Certain elements such as push buttons should be flush with the primary surface in general. Putting buttons in a recessed area makes them feel restricted, and less likely for the user to push.

Windows and Dialogs

The standard window on the Macintosh is a flat object. It appears to float over the desktop for a small pseudo-3D illusion, but many people want to enhance that effect. One common (and cheesy) way to add "pseudo-3D" on a window is to just make the window gray. This doesn't look 3D, this just looks gray. A much better approach is to add a small bevel to the edge of the window content area, so the content area appears to become a raised work surface, instead of a flat sheet of paper. In many cases, however, it is best to leave the window flat, especially if you have a document-centric application such as a graphics program or word processor. Instead, any special areas of the window can be raised, such as a built in tool palette.

The second most common place for 3D windows is in dialogs. Using the standard modal and movable modal dialogs give a window that has a 3D frame. Adding a gray background to this dialog will usually give you a good starting base for the dialog. The best way to do this is to use a dialog template which has a background color of your standard gray. The reason this is important is because the standard frame has a small border around the inside that is drawn in the window background color. Without a custom dialog color table ('dctb') this is drawn in white, while you are trying to draw the background in gray, giving an annoying border to the dialog.

Pseudo-3D Controls

Controls in a 3D interface are more problematic. In the Develop article, a set of drawing adorners were supplied to fake the 3D look, but they have several drawbacks. First of course, is that you need to be developing in MacApp, using only standard views to make use of them. Second, the classes need to be reworked to fit into MacApp 3.1.

A better solution in general is to use the public domain 3D Buttons CDEF by Zig Zichterman. This CDEF creates pseudo-3D buttons, checkboxes, radio buttons, and icon buttons which follow the suggestions from Develop, with the addition of button text and icons shifting when they are pushed, but are nicely self isolated in the standard Macintosh way. Source code is also provided. (If you do use the CDEF, contact Zig and send him a copy of your software, he deserves it.) The only slight drawback to using this CDEF is that you will need to create CNTL resources for each of the 3D controls you use in a dialog box.

Figure 1
Samples of 3D Controls

Menus and the Menu Bar

One of the most important aspects of the Macintosh interface is the use of menus in the menu bar. There seem to be three possibilities when dealing with pull down menus and the pseudo-3D interface: Give them a gray background, use custom MDEFs to make them appear as if they have a frame around the edge, or leave them alone.

I've seen interfaces where menus were colored with a background gray only. In this case the menu doesn't look 3D, it looks gray. Don't even bother.

The second possibility is to actually write a custom MDEF that gives a raised border around all the edges of the menu when it is popped up. This can look nice (especially when combined with cut-lines as described later), but clashes slightly with the standard flat menu bar appearance. Also, since the content is raised into a work surface, the psychological effect is that it should be able to be positioned like a window (see above). This works well for tear-off

menus, but the vast majority of menus are not tear off.

I am also leery about combining raised pane menus for tear-offs with flat menus for not tear-offs, although the different format could give useful information to a user about the ability to tear-of a paned menu.

As mentioned above the menu bar is also normally flat. It can be colorized easily to a gray color, but it is non-trivial to make a pseudo-3D menu bar. (It also creates some interesting philosophical problems with the position of the light source. Is it in the top-left corner of the main monitor, or does the light actually shine from the apple as has been rumored?) Even if the color is changed in the menu bar, when the user switches applications, the menu bar color would then change, which removes the perceived "solidity" of the menu bar. Also, there are a number of utilities that add icons to the menu bar which include gray in their icons. Making a gray menu bar merely makes seeing them more difficult.

The third possibility for menus is to leave them alone. Keep them as they have been for years. I tend to think this is the best solution, since adding pseudo-3D to the menus doesn't seem to enhance their usefulness, but can cause some visual distraction (interface noise) to the user.

Popup Menus

Popup menus pose a slightly different problem, however. A popup menu appears in the content area of a window or dialog. Often that area will have a gray background, which can cause problems with popup behavior.

There are two ways to implement a standard popup menu. Draw the popup title and non-popped state yourself, or use the standard popup CDEF supplied in System 7. The easiest way to do popup menus now is with the System 7 popup CDEF, but it has a major problem: it doesn't honor the control color table associated with the control (as far as I can tell at least.) When it draws the popup menu title, it either performs an EraseRect to white under the title or when popping up the menu it does an InvertRect which breaks with a color background.

The best solution to this is to either draw the popup the old way, using your own code to draw the title and non-popped state of the menu , or to use the System 7 CDEF to draw the actual popup area, and handle the title drawing (and hilighting) on your own. The latter method requires additional effort and doesn't work well in dialog boxes, because your code is not called when the popup CNTL resource is hit by the user. This requires additional filtering in the dialog's ModalFilter.

No matter which way you choose to handle the popup problem, the popup title must be drawn on the standard gray background when not in use, and drawn in white when the popup menu is hilighted. (Remember that the menu should pop-up when the title is selected, also.) You can not simply call InvertRect on the title rectangle, as is normally done on white backgrounds. Inverting on color is not well defined. You will normally end up with a black rectangle with the title drawn in an unreadable yellow color.

To fix this problem, you need to actually fill the rectangle with black, then set the TextMode to srcBic to draw the title correctly on top of the black rectangle. The following code snippet shows an easy way to draw the title of a popup, in either normal or hilighted mode.

```
// Assuming that the graphics state is
// saved and restored around this code
oldTextMode = qd.thePort->txMode;
if (hilighted)
{
    FillRect(&titleRect, &qd.black);
   TextMode(srcBic);  // Draw white on
black
}
else
{
    RGBForeColor(&stdGrayBackColor);
    PaintRect(&titleRect);
    RGBForeColor(&blackColor);
    TextMode(srcOr);  // Draw black on gray
}
GetFontInfo(&curFontInfo);  // Could be
moved out
MoveTo(titleRect.left + 1,
    titleRect.top + curFontInfo.ascent);
DrawString(titleString);
TextMode(oldMode);
```

Window Content

The content of a window or dialog is the most important part of a user interface. It is where primary information is displayed and/or it is where the user works. By following standard rules, you can assist the user in working efficiently and in getting the most our of your pseudo-3D interface.

There are a number of pseudo-3D interface elements which can be added to a content area to aid the user. But when you add elements, make sure the help the user and are not being thrown in just to "spice up" a window that you might think is dull. Elements that can be very effective are: raised and lowered panes, iconic buttons, cut lines, and tabbed panels.

Panes and Frames

Raised or lowered panes are nice convenient ways to group items together for presentation. The pane is drawn as a chiseled border around an area with a two pixels border. The coloring of the border provides the

illusion of height or depth. In my experience, two pixels of frame width are much stronger than a single pixel, especially with a light gray background color. Since the light chiseled edge of the frame is white, more area is needed for the eye to pick out the contrast between white and light gray.

Figure 2
Lowered and Raised panes

In the 2D interface world, we have used gray or black lines to outline a group of items for a number of years. These items are usually checkboxes or radio buttons, and often have a label associated with them. There are two different ways to group these items together in the pseudo-3D interface; by placing them on a raised pane, or by using a box composed of cut line (described in detail, below). Figure 3 illustrates the different techniques. The grouped items, which often are relatively flat, may be put on a raised pane, moving them closer to the user, seeming to make them more accessible. Alternatively, cut lines can be used to frame the grouping, leaving the objects on an "island" in the window.

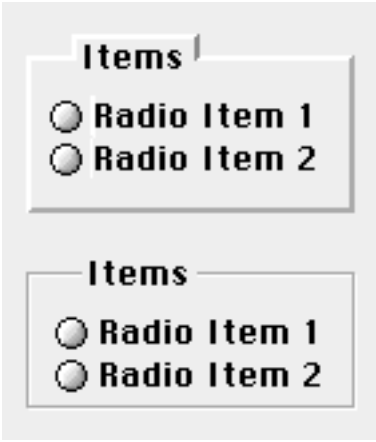


Figure 3
Two Techniques for Grouping

A lowered pane should normally be used for static or "containered" items, such as static text, editable text, or scrolling lists. The interface needs to provide visual cues to the user about which items can be directly manipulated, such as editable text, or a scrolling list, as opposed to items such as explanatory text. There are two things that can help make the difference apparent. First, when a user can edit text or scroll a list, put it on a white background. Static text should be on the same color background as the window content. Second, the inside edge of the pane should have a one pixel black border. This gives distinct edges to the work area. Static text in a frame should not be bordered except by the chiseled frame.

Figure 4.
Static vs. Edit Text Panes

There are several appearances which should be avoided if possible with using raised and lowered panes. Nesting panes within each other leads to distracting height illusions, with items poking out of the screen at the user or appearing to recede into the distance. These should almost always be avoided.

Placing raised panes next to recessed panes can also sometimes create strange effects, making an interface hill effect. Panes above one another with different levels don't seem to be as distracting as side by side pane differentials.

Iconic Buttons

Iconic buttons are a variation of standard push buttons which have a picture representing a function that will be performed when they are pressed. In general, they should only perform the same functions as menu items (i.e. verbs in grammar). The icon on a button should be changed only if the action of the button changes.

Iconic buttons should not be used to show state data by changing their in/out pushed state. To show status information, use icons which are not raised from the surface of the window. This signals to the user that

they can not be directly manipulated and are for informational purposes only.

Cut Lines

A cut line is used to separate areas of a window. It consists of one dark gray line above and one white line below the dark line, making a two pixel cut across the screen horizontally. If the cut runs vertically, the dark line is on the left side of the cut. They dark gray needs to be at least as dark as the gray color used for the darkened sides of a pane, although a gray one or two levels darker can also be used to enhance the effect of they cut.

Multiple cut lines can be used to form a box for framing, although the drawing becomes a little trickier. With a cut line box, the white edge of the cuts are offset slightly to the right and down for the bottom and right edges respectively. Figure 5 shows a blow up of a cut line box, placed on a black background to enhance the visibility of the individual parts. Figure 3 shows a normally drawn cut line box. The cut line box can actually be drawn quickly as two simple rectangles, as long as the white line rectangle is drawn first.

Figure 5
Blow Up of Cut Line Box

Tabbed Panels

Tabbed Panels are the newest up and coming interface element. They can be difficult to draw correctly, and the interface to multiple panes can be tough to do correctly, as demonstrated by Microsoft Word 6.0.

There should never be more than a single row of tabs in a window. Multiple rows cause ambiguity to the user. When a tab is clicked and it isn't in the first row, each row of tabs must shift so the currently active tab is positioned along the top edge of the current panel. This is bad because interface elements should not be moved around on a user, and each tab is a distinct interface element.

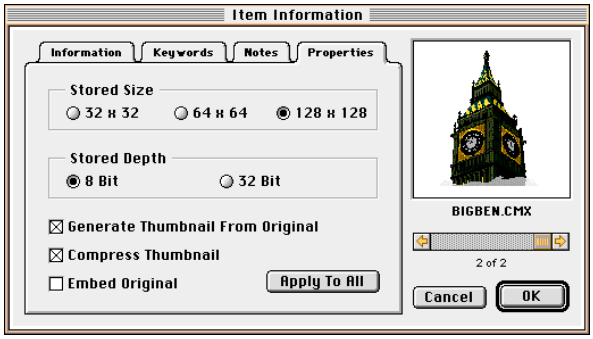


Figure 6
A Tabbed Pane Dialog

When tabs are used in a dialog, the OK and Cancel buttons should be on a level *outsize* the tab panel area, as those buttons affect the *entire* dialog, not just the current panel. Any buttons that appear on a panel must apply only to that panel. Failing to follow this rule causes the user to be uncertain about the results of pushing a button.

Tabs should be drawn carefully to maintain the illusion of a rounded edge. Figure 8 is a blow-up of two tabs showing the left and right edges of a panel.

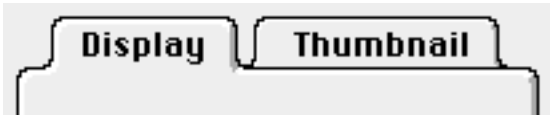


Figure 8
Tabbed Panel Edges

Note the wider border on the right edge of the panel. Also the tab which is not active, does not have a heavy raised surface, as does the tab marked “Display”. This helps the user realize that it is in the background and not the immediate tab. Note also the black edge line which is required to firm up the existence of the raised surface in this interface element. Without a distinct edge, the cards fade into the surface and lose all 3D appearance.

The largest drawback to implementing tabbed panes is the difficulty in drawing the tabs programmaticly. One potential solution is to create the tab edges in a paint program, then load the images and draw them to the screen instead of drawing them with code each time they are needed.

Surprises and More

While creating a pseudo-3D interface may not seem difficult, there are a number of small problems that will crop up as you write and test code. One surprise I ran into was that hilighting broke in some places. If items were draw on the gray window surface itself, they could usually be hilighted, but if the item were something like edit text, or a scrolling list, on a white background, the hilighting would just stop working. After reading through the appropriate areas of

QuickDraw, I finally found the problem. Hilighting only works on pixels that are the same color as the background color of the window. With a gray window background, when drawing a white surface over it, hilighting would always break.

There are two possible solutions to this problem. Depending on your case either one might be appropriate. The first solution is to set the background color of the window to white (or whatever color surface you are working on) before trying to hilight something. The second possibility is to actually use the system hilight color and programmaticly draw your own hilighting.

You might also discover that if you raise the contents of a window, the standard Macintosh grow box no longer looks correct in the corner of your window. To fix that problem you will need to write your own grow box drawing code, and implement a custom version of GrowWindow. This is a non-trivial task, but really needs to be done to make the pseudo-3D interface look natural in your application. And on top of that, your grow box growing code will need to determine the color tingeing currently used by the Macintosh, so your grow box matches the colors found in the rest of the window frame. (This is documented in Tech Note TB 33 – Color, Windows & 7.0.)

One other note: These guidelines will almost certainly change in the future (under the next major release of the MacOS, most likely) and will have to be updated when that release becomes a practically reality in the programmer's life.

Special Thanks

The author would like to give special thanks to George Storm of MacXperts, Inc. for helping iron out some of the issues surround the pseudo-3D interfaced elements outlined here. In particular, his work on tabbed panes clarified their appearance and functionality .

Reaching The Author

If you are interested in arguing points presented here, or wish to make comments or contributions to the guidelines I've set forth in this paper, you can reach me electronically at:

`Looker1@aol.com`

I would like to maintain this paper in a revised version that can be used as a reference to people who are interested in implementing a pseudo-3D interface in their application. This document will be available online via the World-Wide Web and/or ftp service in the future.

The Design of Interactive Television Applications

Timothy D. Knox

The next generation of applications will be multimedia, and interactive television applications. In this paper, I will provide some information about creating interactive television (ITV) applications, along with some specifics about the approach taken by Ameritech.
First, I will detail the Hybrid Fiber-Coax (HFC) network. I will begin with the Video Operations Center (VOC), out to the Video Serving Office (VSO), to the Video End Office (VEO), and finally to the Customer Premise Equipment (CPE).
Next, I will detail the client/server nature of the application, and discuss the server complex. I will look at the various elements of the server complex and the services they provide. I will also briefly touch on the interfaces to the outside world. (I.e. How Video Information Providers (VIPs) get their data onto the platform.)
Subsequently, I will look at the Set Top Terminal (STT), and describe the hardware and operating system support provided. I will also look at how applications are partitioned between the server and the STT.
Next, I will briefly look at the development environments available, such as MacroMedia Director (tm), Kaleida Lab's ScriptX (tm), and others.
Finally, I will detail some of the human factors issues involved with designing ITV applications. For example, there is no keyboard, or mouse. Also, the interface must look good on a low-res television, rather than a hi-res monitor. Counterbalancing these and other factors, there is the very high bandwidth available from the server to the STT. This allows full motion video to be an integral part of the interface, something still difficult on desktop systems.
I will conclude with a brief look at where the technology is going, and where I believe things will be in five to ten years.

Programming, telephony, and cable television are converging. Instead of three unrelated fields, we are getting to a point where there is some overlap. This will only increase with time. I am working for Ameritech on a next generation interactive television environment, which will bring all three areas together.

Ameritech is developing an advanced, high bandwidth cable network, along with new set top and server technology that will enable us to deliver fully interactive television, advanced computer connections, and who knows what in the future.

What kind of network will be put in? We looked at a pure fiber optic network. This has a tremendous advantage in bandwidth (in excess of 1GHz). However, the cost of FTTC (Fiber To The Curb) would be prohibitive at this time.

Consideration was also given to using existing copper in an ADSL (Asymmetric Digital Subscriber Line) configuration. While this would be far less expensive than FTTC, it has several problems as far as bandwidth goes. First, it only offers downstream (to the house) bandwidth of around 2Mbs, barely adequate for a single MPEG-1 video stream. Also, the upstream signalling capacity is only 9600bps (hence the name Asymmetric), which, while adequate for today s needs would likely fall short in the future.

We instead chose to build a Hybrid Fiber-Coax (HFC) network. In this layout, the backbone of the network is fiber optic, and then at various neighborhood distribution points (known as Video End Offices (VEOs)), it is converted to coaxial cable. Each VEO serves 500 homes, with four coaxial cables. This offers the advantage of a high bandwidth backbone, and a relatively low cost local loop. Even so, the coax is specified with 750MHz bandwidth, a significant increase over current cable TV, which is specified at 450MHz.

What exactly do all these bandwidth numbers mean? Well, a single analog television channel requires 6MHz, which means that traditional cable TV operators can offer about 70 analog channels (4506 = 75, however, the remaining 30MHz is used to provide separation).

However, that same 6MHz can be encoded to carry digital information. One encoding is 64 QAM (Quadrature Amplitude Modulation), which provides 27Mbps in a 6MHz channel. A newer system, which we will be using, is 256 QAM, which provides a usable bit rate of about 40Mbps.

How does this relate back to digital video? Well, MPEG-1 requires 1.5Mbps (not coincidentally the bit rate deliverable by a CD-ROM). MPEG-1 will run at higher bit rates with a concomitant increase in quality (in fact, it is often used at 3Mbps, which provides roughly VHS quality), but in essence MPEG-1 remains a consumer standard.

On the other hand, MPEG-2 was expressly designed to provide broadcast quality compression, and starts at 6Mbps, going as high as 20 to 25Mbps. The difference is not merely higher bit rates, but also some changes and improvements to the compression algorithms. Therefore, one 6MHz analog channel can carry approximately 10 3Mbps MPEG-1 streams, or about 6 6Mbps MPEG-2. The current plan for bandwidth allocation is 450MHz for 70 analog multicast

channels, and the remainder divided up as 30 channels for digital multicast, and 10 channels for switched interactive services.

There will be three kinds of programming sent to the viewer. The first is Analog MultiCast (AMC), which is based on the traditional cable model, though there will be a few simple enhancements, that make it closer to the interactive models. The second kind is Digital MultiCast (DMC), which will allow greater interaction. For example, DMC would enable NVOD (Near Video On Demand). A number of other applications would also be possible, though DMC would not permit fully interactive applications. For that, one needs Switched Interactive (SI). This is the most interesting model for programmers, and the one that we shall spend most of the rest of our time considering. However, be aware throughout the rest of the paper that this is only one model, and that two others exist, and will be used.

The network is based on the telephony model, to some extent. It starts with a VOC (Video Operations Center). There will be one VOC in the network (or at most, a small number). Most programming will originate at the VOC. The antenna farm will be located there. There will also be many racks of computers and disk drives for the interactive portions of programming, which I will detail later.

Programs will flow out of the VOC to a VSO (Video Serving Office). There will be one VSO per major market area (e.g. Chicago, Detroit, Milwaukee, &c). The VSOs allow for adding locally originated programming. It is expected that the Ameritech region will have six VSOs, one for each major city.

Each VSO will feed a number of VEOs, which will act somewhat like central offices from the telephony model. A VEO will be an unmanned mechanical site that will serve as a bidirectional interface between the fiber network and the coaxial network.

Finally, the signal will flow down the coax to an individual subscribers home. In the home, there will be one of two Set-Top Terminals (STTs). One will be the analog STT, which will fairly similar to current cable TV STTs. It will have a CPU, and be able to run some pretty primitive programs, but it will not represent much of an advance over the state of the art.

The other STT is, not surprisingly, the digital STT. This will incorporate a PowerPC CPU, several MBs of RAM, several MBs of ROM, an MPEG decoder, and graphics hardware. They are expected to run the PowerTV operating system (created by PowerTV). I will discuss this in more detail later.

A typical Service Application (SA) consists of two parts: the Client Part of the Application (CPA), and the Server Part of the Application (SPA). The chief application running on the STT is the application navigator. The navigator allows the user to select a service application to run.

When the user selects an SA, the CPA is downloaded to the STT by the navigator. The CPA calls the Session Manager (SM) to request a variable bandwidth connection to the server complex. The SM connects the STT to an Interactive Application Server (IAS), which actually runs the SPA. The IAS may also talk to an Interactive Information Server (IIS). The IIS is essentially a video pump; in other words, its primary purpose is to stream video out to the different SAs. There are a number of other computers in the server complex that provide various and sundry functionality, such as event data collection, credit card authorization, and media asset loading and cataloging.

Because the SM can establish connections of varying bandwidth, the CPA requests only as much as it needs. For example, in a movies on demand application, the CPA might initially request only, for example, 20Kbps, which is enough to download the titles of movies available. However, once the viewer has selected a movie, the CPA would request a 3Mbps MPEG-2 connection over which the movie could be sent.

The SM allows the IISs and the IASs to be used to optimum advantage, especially as compared to systems that require each SA to know exactly what servers they are talking to. Also, since the IAS runs the application, while the IIS is used only for streaming video, it allows for better load balancing.

The STT OS will provide a set of APIs that will seem very familiar to Macintosh programmers. The OS will be event driven and multi-threaded. The STT imposes some interesting restrictions on programming. The chief one is the small memory footprint (no more than 2MBs, approximately). However, the CPA only needs to provide the code to control the display, and to get viewer choices. The bulk of the application resides on the server. For example, any relational or object database systems will live on the server.

Take a home shopping application as an example. The viewer will select, for instance, the Sears Home Shopping Service in the application navigator. (See Disclaimer.) The CPA will be downloaded. It will display the opening menu. The user selects the home electronics section. The CPA sends a request up to the SPA to access the RDBMS, and send down the Home Electronics section of the catalog.

After receiving the home electronics index, the viewer might select televisions. This results in another request to the SPA, this time for the section regarding TVs. The viewer will browse the different TV offerings, and finally select the one to buy. At this point the CPA will call on standard services that will display a credit card authorization box. The information will be gathered and sent up to the server, which will

then call out to get approval. Having received approval, the viewers shipping address will be verified (presumably the same as the viewing location). Then a TV will be sent right out.

Now, we come to the question that all real programmers are asking right about now, which is: What is the programming environment? What tools can we use? The answer is, the environment is many and varied. Oracle MediaObjects, Scala InfoChannel, and Sybase Gain Momentum are all possible candidates. It is also possible that Kaleida Labs ScriptX or MacroMedia Director could be used. The development environment must support both STT and server programming. Generally, there is a scripting language available for the STT, often with the ability to support external code resources written in C or other high level languages.

It is quite probable that the CPA will never need to explicitly call the SPA. The STT OS will provide APIs to connect to the SPA by way of RPCs or CORBA calls. In this way, the CPA can be insulated from the details of the network, and can live as though it were a full and complete application.

Obviously, the SPA must understand that it is running on a network. It is still being decided what remote mechanism will be used. It could be RPC, it could be CORBA, or perhaps even some combination thereof.

It should be noted that interactive television application development involves several paradigm shifts, as follows. One, programming for television versus programming for a computer monitor. Two, programming in a client/server environment, rather than all in one. Three, programming for a small footprint, high bandwidth environment, rather than large footprint, low bandwidth. Four, programming for a remote control, rather than for a mouse and keyboard.

First, programming for television. This presents several unique features. One, television is inherently a low resolution device. While even a small monitor can easily display 24 or more lines of text, consisting of 80 or more characters per line, a television can t generally manage much above twelve lines of twenty characters, and still retain legibility. After all, the NTSC standard was designed for optimum display of continuously moving images, not large sharp-edged stationary objects like text. Also, things like one pixel lines, while quite nifty on a monitor, just don t work on a TV. Thirdly, television uses NTSC color space, which is NOT the same as the RGB color space used by most computers and monitors. It takes great skill to choose colors that will look good on a television.

The second paradigm shift is from programming for a mouse and keyboard to programming for a remote control. Because there is no general pointing device, a Macintosh-like interface with windows, menus, icons, and such, would be extremely difficult to implement. Instead, an interface with perhaps six or a dozen hot spots would be used. Then, instead of a pointer, there would be a moving focus. The focus would be shifted from hot spot to *hot spot* by means of arrow keys on the remote. The focus would likely be indicated by lots of motion (perhaps a video clip or color-cycle animation). There would also be a select button on the remote, acting much like the button on a traditional mouse; to wit, it would select whatever hot spot had the current focus. A consequence of this is to force users back into a very modal, almost menu-driven, interface. While such an interface is considered less helpful for general purpose computer applications, for ITV applications, that is exactly what the user wants. Just as no one complains of the modal nature of bank s ATM, because it is designed to accomplish only a very few extremely simple tasks, ITV applications would be similar.

The third paradigm shift is to the client/server model. Of them all, I believe this will be the least traumatic. This particular change has been coming for some time now, and we have developed tools to simplify this. It is likely that the programming model will incorporate either OSF DCE RPC s; or even a CORBA implementation. Most CORBA implementations either have, or will soon have, C++ bindings for their object request brokers. This allows the systems programmers to hide the network stuff inside some C++ classes, and lets the set-top application think it is simply talking to a large class library. This is especially useful considering the rather limited work area that the set-top provides to applications. We can put all the difficult and large stuff on the server, which will have considerably more horsepower to run the server parts of the applications.

The fourth paradigm shift is programming for a small footprint, high-bandwidth environment (i.e. the set-top), rather than large footprint, low bandwidth environment typical of most microcomputers today. This affects such things as general programming practices, as well as the design of the user interface elements. It is likely, for instance, that code will be highly segmented (as in the early days of Macintosh), with each segment loading, running, setting up the load of the next segment, and exiting. With the bandwidth available for downloads (several megabits per second), this will be far less inconvenient than it was in the old days. There will also likely be a greater push to use high-level scripting languages, as they can accomplish more per byte than all but the most tightly written assembler. In the matter of user interface, the easy availability of almost unlimited bandwidth on demand means that certain things that would be difficult or impossible on a microcomputer become not just easy but necessary on the set-top.

For instance, consider a movies on demand application. This lets the viewer select a movie from a list of dozens or hundreds of movies, which is then beamed to his (and only his) TV. In a PC based implementation, a likely interface would be a list of movie titles, perhaps with a graphic still from each one to help the viewer identify it. However, on a set-top, the screen could be (for example) quartered, with each quarter showing a video clip or commercial for a different movie. The sound and motion would indicate which clip currently had the focus, and then the viewer could try before you buy.

Actually, there is another important difference between a microcomputer and a set-top. A micro is a general purpose computing device whose primary purpose is to run user applications, and enable them to accomplish their work. On the other hand, the purpose of the set-top is primarily to prevent theft of services, and only secondarily to serve the viewer. Thus the set-top needs elaborate decryption hardware or software, as well as an elaborate protocol set to convince the Head End Controller (HEC) that it is talking to a legitimate set-top, and that it has the right to access the services that the set-top is requesting (in other words, did the viewer pay for the movie in question, for instance).

What, then, does the future hold? Well, in the short term, I think the advantage accrues to the newcomers to the cable TV market, such as the telcos, because they do not have the large investment in obsolete technology that traditional cable providers are stuck with. However, movies on demand, and even home shopping with automatic delivery of products is not going to make this the hottest market under the sun. In the long run, it will be services that make or break the marketplace. Offering the service applications the consumers want is what will sell the service. The industry is looking for the next Pong, or the next VisiCalc; some application which would be reason enough for Ma & Pa Average to buy the system. (After all, techno-geeks like us will have already bought in, but as history shows, we are too small a market segment to keep all but the most narrow markets afloat.) Could it be interactive games? Maybe. Or perhaps it is CD-ROMs online. But probably, the next killer application is something all of the pundits have completely overlooked. If you know what it is, please, talk to me and we can discuss getting you on our platform.

Acronomicon

ADSL	Asymmetric Digital Subscriber Line
AMC	Analog MultiCast
CORBA	Common Object Request Broker Architecture
CPA	Client Part of the Application
DCE	Distributed Computing Environment
DMC	Digital MultiCast
FTTC	Fiber To The Curb
HEC	Head End Controller
HFC	Hybrid Fiber Coax
IAS	Interactive Application Server
IIS	Interactive Information Server
ITV	Interactive TeleVision
MPEG	Moving Picture Expert Group
NVOD	Near Video On Demand
OSF	Open Systems Foundation
RDBMS	Relational DataBase Management System
RPC	Remote Procedure Call
SA	Service Application
SM	Session Manager
SPA	Server Part of the Application
STT	Set Top Terminal
VEO	Video End Office
VOC	Video Operations Center
VOD	Video On Demand
VSO	Video Serving Office

Disclaimer

A dream of an ultimate OS

Oleg Kiselyov
CIS, Inc & University of North Texas
303, N.Carroll, Suite 108 Denton TX 76201
oleg@ponder.csci.unt.edu, oleg@unt.edu, http://replicant.csci.unt.edu/~oleg/ftp/

This is a dream, and as such, it is made of shreds of reality shuffled and rearranged in sometimes bizarre combinations. It has been brewing on for over 10 years out of dissatisfaction with many of the major modern Operating Systems. Indeed, it is glaringly obvious that the only thing a user does at the terminal is requesting/reading/modifying textual information, arranged mainly into tables/scrollable lists. Nevertheless, the user often has to use different and completely disparate commands to achieve exactly the same modification, for example, to delete an item (line) from a list as in: removing a line in a text document, removing a file, killing a process (that is, removing it from the list of active processes) and canceling a print job. Furthermore, despite the fact that the OS is swarmed with tables – from a hierarchical database of files to yellow pages maps, to a hash dictionary of an object file archive (library), to relatively flat databases of IP routes, current processes, users, and code revisions – common database functions like inserting a record into a "table" with hashing a key, retrieving a record/field using a simple/concatenated key and linking tables, are conspicuously missing among the core kernel services. This paper is an attempt to imagine what an OS would look like and how it would work if looking for a word 'foo' in something and deleting/closing/stopping this something, be it a paragraph of text, a network connection, a subscribed newsgroup, a process – would all require roughly the same sequence of mouse clicks/keystrokes, and would be understood and interpreted in the same spirit by the operating system.

Introduction

Operating Systems are commonly viewed from two major standpoints: managing computing resources, and hiding hardware idiosyncrasies while putting a *friendly* face for a user. IMHO, providing a user-friendly interface is the primary OS responsibility: after all, a CPU doesn't need any OS to run, nor does it care much which code, system or user, it currently grapples with. Nevertheless, with a notable exception of MacOS, MagicCap and Newton's OS, operating systems have been thrusting upon a user a flood of disparate interfaces, commands, and actions – all are for doing essentially the same thing: filling in and editing some list/table. Moreover, even internally every major OS component – a file system, network services, user management, terminal management, to name a few – each implements and manages its own private incarnation of a simple database.

It literally springs to mind that database services and text/list editing are a core activity ought to be supported on the very fundamental level of OS. The paper flashes a few images reflecting some particular examples of how that unification could be done and how one can work with it. Here is the preview:

MacOS is one of the very few systems around close to the ideal: TextEdit has been elevated to the level of a standard system (toolbox) service. It was the strong statement that the OS is not only about managing files and processes. Furthermore, deleting a piece of text, a file, a directory, a file server connection – all can be accomplished by the same action: highlighting and dragging into trash. Still, one can go a little bit further: for example, a list of processes conceptually isn't much different from the list of files. One can imagine the Finder manage (arrange, get info, duplicate, trash) files and folders that are not necessarily ordinary files and folders, like processes, open TCP connections, newsgroups, print jobs active and pending, to-do- tasks, etc.

Deep down, an operating system is nothing but a manager of many databases. Indeed, a file system, the process table, routing tables, list of known AppleShare servers, revision control system (projector) data, Think C projects - they're all databases. Unfortunately, despite a sizable share of common functionality and interface, every one of them is implemented and managed separately. For example, hashing, searching, cashing are used in many of those databases (if not all of them), but each application implements and re-implements them in its own way. One would think that if a database manager were a universal core program, one could take trouble really polish hashing and caching, once and for all. It really looks like a very good deal to trade a multitude of "custom" database managers for one well-designed distributed database manager.

Conventional databases are usually implemented on the top of a file system. But the file system itself is a database. Mac's HFS and Novell's file system even use btrees and other "advanced" indexing schemes typically found in full-blown "industrial strength" databases. For another thing, a hierarchical model is not the only db model out there. So, why not to have an Adabas-like or DB4-like or Oracle-like database instead of the file system? After all, querying the database for a Jan 1994 sales, and clicking on folders

"Sales", "1994", "January" are very related activities. Only, a full-blown database provides a more flexible interface for linking and querying records. Modern databases have all facilities for storing images, sound files, movies, etc. big binary objects. I wonder, what else do we need files for?

Unification of user interface and the underlying database functionality has one more important advantage: ease of linking or mixing together, objects. I imagine documents made of not only chunks of text, but folders and applications themselves (or links to them). Indeed, a document with a picture usually has the text and the image in separate places, with additional information (link) as to how and where the image should be displayed. Why not to store likewise links to menus, applications, or remote servers, or precompiled headers, or *mailto:* or other *form* or *anchor* as well? The desktop is probably going to look just like a ***homepage.***

The rest of the paper shows how one can come to these points and where he can go from there.

Everything is just editing

It is indeed. A document I'm editing is a list of lines; if one edits a file in emacs, he presses \C-K (I press PF4) to delete a line. When we list a contents of a directory, we also get some sort of table or list (especially using ls -l or view-by-name). To remove a line from this table, we use a different command, rm. When we want to see which processes are currently running, we use ps, or top, or ProcessWatcher. Again, we end up with a table, but now we need to use kill process_id (not even a process name) to "delete a line" from the table. To delete a line from the table of queued print jobs, we use yet another command, lprm print_job_id.

Thus deleting a line, removing a file, or killing a process or a route or an ARP entry all boil down to essentially the same thing: removing a row from some table. This is no coincidence. The uniformity runs very deep, both on the level of implementation, and on the level of presentation. Indeed, there is not so many ways how one can manage a collection of objects: it is usually some kind of list or tree. Also, there is not so many different tricks how this collection can be presented to the user and be manipulated by him. The only way people work at a terminal is browsing and editing, that is, moving the mouse, typing and pressing the PgDn key, if you think of it. Disparate interfaces are not a consequence then of some fundamental differences in user or system activities, they are simply result of evolution: different subsystem/services were written by different people and modified by even bigger crowd.

Macintosh definitely stands out of the crowd. Many functions within MacOS are accomplished by exactly the same action (say, removing is by dragging into trash, opening is by double-clicking). This is especially true with a Drag&Drop Manger installed. Even UNIX is moving towards some unification, take for example CDE, or a proc filesystem. The latter is really neat: indeed, with UNIX approach "everything is a file" one could only wonder why a process should be any different (and why it took so long to implement and popularize this idea). However, the unification isn't complete. While it is possible to open /proc/1024 to get hold of a process with id 1024 (just to find out who owns this process and when it was created, if for nothing else), one cannot rm /proc/1024 to kill the process, and one cannot do ls /proc/1024/open_files to see the list of all open files for this process. Though why not?

Since the list of processes conceptually is not much different from the list of files, why don't we have a folder "processes" on a Mac, which has "files" standing for processes. So I could use the standard Finder operations, View-by, GetInfo, Trash, Duplicate to manipulate the processes. Usenet News hierarchy is very similar to that of a filesystem (as a matter of fact, this is how it is stored and managed on an NNTP server). Nuntius newsreader presents the news hierarchy as some directory tree, as folder's and files in a view-by-name mode. Alas, Nuntius had to emulate much of the Finder's functionality to manage these newsgroups-folders. It would be much easier to develop and use, say, printer and network managers, an FTP utility or a newsreader, if one can tell the Finder: here, this is a list of files, manage it as you usually do with a list of *files*, just tell me when you're about to trash something.

Rearranging file icons within a folder view and rearranging paragraphs within the document are essentially the same activity. If it is unified, the overhead and code duplication can be significantly cut down. There is also a good side effect from that: a document can contain folders and/or applications (icons of other applications, etc.). So, we kind of get the hypertext for free.

The luster and dull of plain text

Configuration of a UNIX system is specified and controlled by a huge tangle of *plain text files*, /etc/hosts, sendmail.cf, syslog.conf, inetd.conf, /etc/uucp/Systems to name just very few. .INI files on some other systems are also plain ASCII. Even MacOS caved in a little bit with System Folder:Hosts, although it is a very isolated example on a Mac. Of course, just because symbols displayed on screen must be ASCII, it doesn't mean that the information on disk should be the same. Still, ASCII configuration files abound, for a very simple reason: they can be modified with any text editor from ex and edlin upwards, and can be viewed and created even with-

out an editor, with a cat command. Note, accessing a block of /etc/hosts file is not much easier than reading a /machines tree of Next's netinfo database. ASCII wins simply because tools for handling text files are present even on a naked system. As long as a file system is treated differently than a database, the makers of netinfo, NIS or any other database have to support tools for converting a database table or set of records into a plain text file, and back.

But it does not have to be this way. If a database manager is implemented as a core system service, and the system always comes with a very primitive database access tool, the gordian knot of system configuration files disappears. The situation is almost ideal on a Mac, where ResEdit is this universal database editor. Much (if not all) system configuration can be set up by opening a resource and toggling a few buttons or retyping strings or adjusting colors, without any need to learn syntax of a specific configuration file (and without any need for a system to waste time trying to parse that text file and report an error if any). Unfortunately, ResEdit and a set of templates for system resources do not come bundled with MacOS. But SimpleText always does. That's why System Folder:Hosts is a plain text, though it would have been much better as a resource.

The very idea of an application as a merely collection of code and configuration resources with a common name is beautiful. It is even possible in some applications (LaserWriter Utility, for example) to add/delete menu items and corresponding functionality just by adding/removing appropriate resources, without any need to recompile/relink the application. It was with pain that I read a recommendation to refrain from creating code resources with PowerPC native code (which should be moved to the data fork instead). Now an application has a database managed by the resource manager, and a database managed by a fragment manager.

Everything is Database

And this is not exaggeration: an operating system is permeated with sets, lists, tables, and other collections of something. This something starts with fundamental data structures like a process control block or a page table and continues to include i/o request queues, list of open windows, resources. Also, SCCS, RCS, termcap/printcap, networking databases (name domain service, name resolver, /etc/hosts, /etc/networks, /etc/services, routing tables), whatis database for man pages: one can hardly spend a millisecond working in a computing system without hitting some kind of database query. Note, every of these databases supports basic select/insert/delete functions; advanced options like concatenated keys and linking among several tables are often required, too. Still, almost each database has its own implementation of these common functions. One can argue that managing process control blocks and virtual pages does require tailored and very tuned in implementations. However, even a couple of seconds delay in removing a SCCS revision or a host entry is not a big deal at all. Furthermore, many of system databases are implemented as "flat file" with linear search, oftentimes just because it was simpler. Using a well-designed universal database instead of a multitude of quick fixes actually promises a boost in performance (besides, it is just cool).

Having a universal database frees the system or an application from many chores, like time stamping or permission checking, etc. But universality offers another clear advantage: an ability to link all kinds of records/tables, which is so much pain to do now. For example, a link between two records representing files is no different for a universal database manager than a link between a record in the "Users" table, a record in the "Processes" table, a couple of records in the "Files" table and the "Print jobs" table. There is no longer need for multiple IDs and messing with them. Many-to-many links are also possible. Moreover, this improves performance: list of all processes belonging to user joe can be found faster by a database query rather than with ps aux | grep joe. Surely any database can do better than a dumb search, like in netstat -a | grep finger (which is used to finger fingerer). Many similar scripts are just database queries, and not very efficient ones. Makefile would be also easier to generate and maintain. It would also be possible to link #include directly to the includee (which would obsolete the intricate art of specifying compiler's -I and -L flags and trying to predict which of several possible time.h files the compiler would actually pick up).

File system is a database, and can be a better database

Hierarchical organization of a file system is not the only possible organization. Besides, the widespread use of symbolic links and aliases makes a file system look more like a network database, though kludged (everyone who spent some time reanchoring aliases can tell that). So why not to start with a networking database with good indexing capabilities, something of, say, Adabas? It was very efficient, implementing its own storage management and transparent multi-level indexing; its journaling capabilities are much better than those of UFS (or HFS for that matter). Or take a FileMaker: there are several applications on info-mac for cataloging the contents of CDs and floppies in a FileMaker database. So why a file system could not be like this database from the outset?

Nesting folders is what gives the file system its hierarchical flavor. Directories are just view of a certain

subset of files, selected according to some criteria (and named). Thus, a directory can be thought of as a view of database records, a saved database query. It follows immediately that a file can appear in as many "directories" (views) as one wishes to. For example, one can be a view of all files tagged as "sales reports", another may be a view of files modified within five days. In a sense, "searching" a file system and creating and populating a folder becomes the same activity. Moreover, any file modification would be visible in all views. Since saved views are database objects, one can reference views within views if one so wishes. There is no required hierarchy: you can have two views be referenced from within each other, or create any other network of views that best suits the problem.

Every database record (item) should have some mandatory attributes like time stamps, owner, permissions, kind (document). Beyond that, the user (or application/creator) can add anything he wants. For some records, the body is just a binary object. Records of kind "image" (or in table "images") can have additional attributes like the image width, height and depth, a key of a compressor method, etc. Thus, listing all images 512-pixel wide and with depth at least 8 with a private colormap should be as easy and elementary as viewing files in a folder by date.

Viewing and editing *files* should not be a problem as long as (even a naked) OS has some rudimentary database browser. It may look like a basic no-frills record browser in Paradox (displaying all record fields in columns or as *name=value* pairs). Of course, an OS should have an ability to generate better looking views/reports (as Paradox does). Still, the basic browser is necessary and useful (like cat or SimpleText) in a desperate situation.

Having a general purpose database instead of the file system does not mean that users should forget everything they knew about how to grapple with a computer. Almost all old acquired skills can be used without any change. Indeed, a path to a file in a hierarchical file system is just a list of keys how to locate a file. This becomes especially obvious if one looks at URL, say, http:/somehost/foo/bar.html. This URL does not actually imply that we want to access a file bar.html in the directory foo under the DocumentRoot on the host somehost. If foo is a script, then bar.html is just a parameter passed to that script, and can be interpreted as a file name, or anything else the script wishes. In short, not everything between slashes in URL is a name of a directory. It is just a key specifying an object we want to take a look at. The same argument holds for the new file system: one still can locate a file by listing some of its attributes separated with slashes. The advantage of the database approach is that one could put a wildcard in *any* of the "directory" names, or one can consider such file attributes as modification date, size etc. as some "directory names" and use them in the "path". In a sense, running find and listing a "directory" would be exactly the same activity.

Many industrial strength database managers (Oracle for one) support transparent access to remote databases, with possible local caching to improve performance. It looks then that one doesn't need NFS at all: distributed database manager takes care of remote mounting, authorization, etc.

Mock-up 'hello world' session

It is probably not going to look much different from how we work on a Mac now. Once Mac is powered up, we see a Desktop: a some particular view of the database. It contains references to other database objects like the table of processes, the table of system data (which have attributes System and UserConfigurable: Control Panels in short), the table of recently used objects, etc. Suppose I located a C compiler (well, I wish there were a better language by then, say, Dylan), either on the Desktop, or in one of the subviews, or just doing a plain search for anything with attributes application and C. Once the "C Project Manager" is launched, it prompts as usual to open or create a project. I create a new project and a new file, which is automatically tagged with the project name, C, text, etc. attributes. In a sense, a ThinkC project plays the same role as a directory does now. Of course I can define and set other attributes I wish (some of them could be inferred from the context or from the some other db object). This is like entering data in a database table: one can say, make a new record like a previous one, and then change just a few values. I can also attach a comment to a new C program, and make it as big as I wish. Words from comments can be used in a file search, too.

When writing a "hello world" code itself, I can say "enter a db object", get a query window and find an object with attributes "data, C declaration, owned by system, having something like 'standard io' in comments". Or I can say "C declaration for printf()", and hopefully the editor can fill in some default attributes for the query. Of course I can go ahead writing a C code and then click on printf() and tell the editor to find and include some C .h file that declares this function. If the editor inserts something like #include <standard io> I can click on it and see what this package has.

It looks like one can have the functionality like that even tomorrow: doing away with files as we know them does not mean breaking everything, every habit, every skill and every application. On the contrary, most of the old functionality would apply, and some

applications even become simpler (the C preprocessor does not need to go into trouble trying to figure out where that .h file is).

Legal Update: Selected Legal Developments Of Interest to MacHack 95 Participants

Roger H. Leemis, Attorney[1]
Southfield, Michigan

Over the last decade computers, and the software to run them, have become increasingly important to society. The computer industry as a whole, both software and hardware, continues to grow in size and sophistication as an industrial sector. Microprocessors and personal computers are found in ever more products, in most businesses and in many homes. This relatively new technology now pervasively influences our economy and society in general. It is hardly surprising that the computer industry is involved in legal matters before Congress or regulatory agencies, in private litigation, and even in criminal prosecutions. Software developers and MacHack conference participants may find it helpful to be aware of legal developments concerning computers or software. This paper discusses selected developments over the past year illustrating how the legal process is responding to this new technology.

Introduction: Computers and the Law.

The legal system pervades modern American society. One might almost say that it is the virtual infrastructure of the economy. Notwithstanding current calls for litigation reform, the legal system itself will continue to be important because it provides the basic assumptions for commercial transactions. As with any industry, beyond general rules governing us all, there are statutory enactments pertaining specifically to that industry. Because software and computers are inherently concerned with patents and copyright, that statutory regime is uniquely federal[2]. Other important areas of regulation arise as computers become commonplace throughout society; regulation by way of telecommunications policy is an example of that. This paper discusses a wide range of issues which have in common that computer software or hardware is somehow involved.

The majority of the cases and statutes discussed are federal because most of the legislating is taking place on the national scene. In part, this results from the predominance of issues involving intellectual property and interstate commerce. It is probably also true that the broad regulation of such technology is more efficiently done on a national level. All three branches of the federal government are involved in this process. Congress of course legislates and the judiciary decides disputes under existing law. The executive branch has a more complex role because it both enforces existing laws and enacts regulations to carry out policy; regulatory agencies often can wear more than one hat.

It may be observed that developments in the law always lag behind the most recent technologies, forever playing catchup. This is inherent in the legal process because busy engineers and software developers have a head start: usually, a need for regulation is neither apparent nor discussed until after the new technology comes on the scene. The legal developments discussed herein illustrate how, over the past year or so, the legal community has been grappling with issues raised by the computer industry. The paper covers selected areas in which there has been significant news since January, 1994. Some issues, including current patent law developments and the general law of privacy, are beyond the scope of this paper. The cases were selected to help the reader understand how the law may be evolving as the legal process accommodates and adjusts to a major new technology.

Recent Legal Developments.

Antitrust. Though vigorous competition is the norm in the computer industry, not everything is permitted; throughout our economy, monopolies are frowned on and certain forms of unfair competition are prohibited. Obvious examples are the outlawing of collusive price fixing and bid rigging. More generally, marketing practices may be prohibited if they unduly restrain competition. Competitors or customers may bring civil suits for damages for unfair competition; the Justice Department's Antitrust Division enforces federal trade regulation; and states may enforce their own monopoly statutes.

Pride of place must go this year to Microsoft Corporation, which has been involved in two major antitrust cases. The first is the well-known action by the Antitrust Division alleging a variety of unfair trade practices[3]. While the original investigation had been broad, the Justice Department ultimately reached a settlement, widely seen as quite favorable to Microsoft, which would not have required major modifications to Microsoft's marketing practices. The proposed settlement had to be submitted for court approval, the point when things got very interesting. A group of competitors hired, anonymously, an attorney to represent their interests; his brief opposing the settlement detailed many complaints about Microsoft practices. Judge Stanley Sporkin, a former head of

enforcement at the S.E.C., reacted strongly and refused to approve the proposed settlement. (Among other things, he wanted Microsoft to agree not to make vaporware announcements to preempt the competition.) This places the Justice Department and Microsoft in the unusual position of jointly appealing Judge Sporkin's decision, both arguing that he abused his discretion. Most commentators expect that the settlement will stand[4].

The other big antitrust concern for Microsoft was its proposed merger with Intuit, which had been under review by the Justice Department for months. When the government announced it would not approve the merger, Microsoft sued. One immediately notable aspect was that Microsoft and the Justice Department were allies in appealing Judge Sporkin's decision while simultaneously fighting over the Intuit acquisition. It is not only politics that makes for strange bedfellows. In mid-May, Microsoft suddenly announced that it was dropping both the litigation and the proposed merger because it would have taken too long to resolve. It remains to be seen whether Microsoft will face future challenges; heightened scrutiny seems a given. As Anne Bingaman, head of the Antitrust Division, recently said: "We have become a kind of Microsoft complaints center. And we take them very seriously[5]." Hardly comforting words for Microsoft.

Intellectual property. Lawyers speak of "intellectual property" law as encompassing patents, copyright, trademarks and trade secrets. It is readily apparent that the computer industry is deeply involved with them all. The law has wrestled with some esoteric distinctions, such as copyright for the mask used to make an integrated circuit or potential patent protection for software or an algorithm, subjects beyond the scope of this paper. Two cases in the copyright area illustrate the continuing importance of intellectual property and its impact on software development.

Perhaps the most-followed copyright infringement clash has been the "GUI Wars," Apple Computer, Inc v Microsoft Corp, wherein Apple complained that Microsoft and Hewlett-Packard had infringed on its graphical user interface. This may be the classic "look and feel" case and dates back to the Lisa Desktop and Windows 1.0. After a series of preliminary decisions over four years (familiarly referred to as Apple I - Apple V), the trial court ultimately concluded that any copying was covered by an old license from Apple to Microsoft going back to Windows 1.0. Last fall, the U.S. Court of Appeals affirmed[6]. Especially given the license, the Court of Appeals found that only virtually identical copying could have been an infringement; because the graphical interface was "partly artistic and partly functional[7]," similar features or standard treatments (like a trash can icon instead of a delete key) were to be treated as ideas and were not protected by copyright[8]. Press accounts focused on the other issue in the case, that Microsoft and Hewlett-Packard were entitled to recover their attorney's fees from Apple. This case is significant because the expression of similar ideas in relatively standard ways from program to program may not create copyright problems; however, software developers still need to tread carefully in this evolving area.

Another well-known, closely followed, and even more technical copyright infringement action has been the suit brought in Massachusetts wherein Lotus Development Corp. had charged that competing spreadsheet programs Quattro and Quattro Pro infringed on its copyrighted program, Lotus 1-2-3[9]. The critical issues were not those of wholesale copying or "piracy," but instead involved particular keystrokes which related to executable operations. In 1993, after a nonjury trial, the federal district court held that copyright protection could extend to a program's menu structure, its organization, and even the first letters of command names[10]. The judge had previously reviewed the Quattro programs and concluded that the "look and feel" metaphor used in discussing similar graphical interfaces was not dispositive of copyright protection[11]. The district court had rejected arguments that the menu command hierarchy had been dictated by the functions involved, particularly the user macros[12]. Though offering insight into how courts weigh such claims, the district court determination of infringement is now of historical interest only. The U.S. Court of Appeals recently reversed, holding that the menu was not copyrightable, because the copyright laws provide that copyright protection does not extend to, inter alia, a "process, system, [or] method of operation . . . regardless of the form in which it is described, explained, illustrated, or embodied in such work[13]." The First Circuit reasoned:

We think that "method of operation" . . . refers to the means by which a person operates something . . . . Thus a text describing how to operate something would not extend copyright protection to the method of oper- ation itself; other people would be free to employ that method and to describe it in their own words.[14]

Both the First Circuit and the Ninth Circuit have recognized that there are a few common ways in which users interact with software and have limited copyright law accordingly.

Commercial Law: The Computer Business. We begin with an interesting, if not earthshaking, case. The famous astronomer Carl Sagan did not have a good year. His problems began when Apple Computer assigned the code name of "Sagan" to one

of its products in development; though for internal use, these names are of course widely known. Apple's practice has been to honor those named. Sagan apparently did not feel especially honored and had his lawyers write a suitably lawyerly "cease and desist" letter. Apple did so, redesignating the product "BHA." While that sounds like a food additive, Dr. Sagan heard through the grapevine that this stood for "butt-head astronomer" and that they didn't mean Beavis' friend[15]. Sagan sued; it's the American way. His claim was that he had been defamed by the name (even though not literally true). Last December, his suit was dismissed. To make matters worse, he recently learned that he was suffering from a potentially fatal anemia; he is now recovering from a bone marrow transplant. The anemia is believed to be unrelated to the lawsuit. Apple's latest new product designation, Copland, for its pending System 8 operating system, honors the American composer Aaron Copland, who died in 1990.

One important obstacle to greater use of electronic data interchange is the lack of adequate "assurances," in the broadest sense, to support regular commercial transactions. Downloading information from an online database is one thing; entering into a binding agreement or engaging in financial transactions is something else again. Privacy issues and the development of cryptographic standards are but part of the puzzle; standards for verification, confirmation, subsequent documentation and auditing are other needed elements. A United Nations working group has completed a draft model law on electronic data interchange in international trade[16]. The proposed model law will address admissibility of EDI messages and computer "signatures," among other things. The working group is reportedly also considering financial questions, including electronic promissory notes or bills of lading. This could bring greater legal certainty to home shopping via cable.

Bankruptcy Law. Given the continued growth in the industry, it may seem premature to discuss any aspect of bankruptcy law. Nonetheless, success is not guaranteed to all competitors in any industry - - just ask Wang Laboratories. Whether from an eventual general economic downturn, increased competition or a specific misjudgment, there will continue to be bankruptcies within the industry. One of the more complex aspects of bankruptcy practice involves a debtor's statutory right to terminate (called "rejection") ongoing contracts after a bankruptcy filing[17]. However, Congress has limited the debtor's right to reject in certain classes of cases, such as real estate leases, because rejection would be too disruptive. Imagine a software company which goes into bankruptcy and then suddenly terminates its long-term licensing agreements. Congress has provided special protections for licensees of "a right to intellectual property[18]." While no reported recent cases involved software, cases involving a french fry vending machine and a secret formula for rum[19] suggest the statutory protections will be given a broad and flexible interpretation. Licensees of intellectual property will, however, have to meet the requirements of the Bankruptcy Code, which could be very significant in the case of a pending dispute with the developer.

Employment and Bankruptcy. Many people work as independent contractors in computer-related fields, including as programmers, software developers or sales representatives. The IRS position on independent contractors generally is beyond the scope of this paper. However, consider this question: What happens if an independent contractor is unpaid? As a general rule, all creditors have to wait (sometimes forever) to get paid. There is a longstanding exception for salaries, so that employees don't lose their last couple of paychecks (limited to $4,000 within the preceding 90 days). But technically, an independent contractor is not an employee. In the Wang Laboratories bankruptcy, the court treated independent sales representatives as the equivalent of employees[20]. The Bankruptcy Reform Act of 1994 amended the Bankruptcy Code to confirm that treatment[21]. However, this only applies to individuals or to corporations with a single employee and only if at least 75% of earnings in the preceding year came from the debtor. Most importantly, this only applies to independent sales representatives; other independent contractors are on their own in the event of bankruptcy. Persons doing significant work on that basis would be advised to review their agreements.

Employment Law. The implications of computer-based communications in the workplace continue to unfold, especially in the area of privacy on the job. Most employees are not aware that they probably do not have any privacy rights for e-mail or voicemail at work. One recently filed suit illustrates the issue: a McDonald's franchise manager was terminated after his supervisor listened to romantic voicemail messages from a fellow employee with whom he was having an affair. The terminated manager and his wife have sued, alleging privacy law violations[22]. From time to time, similar incidents make the news. Never forget that computer-based communications are accessible to the system operator; at work, the system operator is accessible to the boss.

Criminal Law. Criminalization of conduct is perhaps society's ultimate expression of disapproval (surpassed only by becoming the butt of the opening monologue on a late night talk show). Several criminal statutes address computers or software, sometimes treating them as unique subject matter. Other statutes merely assume computers are but one of sev-

eral options open to the criminally inclined. For instance, many federal statutes require that the crime take place across state lines; legally, it matters little how an object is physically transported, or whether a communication occurs by mail, telegraph, telephone or computer. Often, a computer is not even mentioned.

The most notorious case of the year, and the one most fitting the public conception of a "computer crime," started with the break-in and stealing of files belonging to Tsutomu Shimomura, a computer security expert with the San Diego Supercomputer Center. This was widely seen as an affront and a personal challenge[23]. Rising to that challenge, a number of computer security experts succeeded in tracking down the alleged perpetrator, Kevin Mitnick, who was arrested and charged in Raleigh, N.C.[24] Perhaps surprisingly[25], Mitnick was charged only in regard to fraudulent cell phone codes and programming devices, not under the specific federal statute for fraudulently or otherwise improperly accessing a computer or, through interstate use of a computer, damaging a computer system[26].

One of the year's criminal cases of some national notoriety, the "Jake Baker" prosecution[27], arose in nearby Ann Arbor. Jake Baker is the college student who had posted a violent sexual fantasy, using the name of a fellow student. An alumnus contacted the University of Michigan after reading the posting. The original criminal complaint charged the interstate or foreign transmission of a "threat" to injure another[28]. Public attention focused on the fact that, as graphic as the story was, it did not express an intention to act in the future; it clearly was a fictitious account of a nonexistent crime. That, and the University's suspension of Baker, caused most press reports to focus on the First Amendment issues. Since then, the U.S. Attorney has obtained a superseding grand jury indictment, which drops the story and substitutes instead a series of e-mail messages with someone using the name "Arthur Gonda," believed to be located in Ontario, but about whom nothing else is known. These conversations concern mutual desires or fantasies suggesting a serial killer wannabe. Baker is charged with five counts of transmitting threatening communications regarding a threat to injure another person[29], though there is no clear identification of a victim. As is often the case, the computer is treated as incidental to the offense even though news accounts prominently feature this as a case about computers and the Internet. According to both prosecution and defense, this is the first time the federal threat statute has been applied to the Internet[30]. The prosecution has quite candidly stated that Baker is being prosecuted to deter others. At the end of May, U.S. District Judge Avern Cohn was considering a defense motion to dismiss.

Yet another computer case, this time out of nearby Farmington Hills, involves "stalking" by computer. As have many states, Michigan recently enacted criminal laws proscribing stalking, meaning "willful . . . conduct involving repeated or continuing harassment of another . . . [31]." Both stalking and aggravated stalking require "unconsented contact," which in Michigan specifically includes: "[s]ending mail or electronic communications to that individual[32]." The specific inclusion of "electronic communications" has attracted public notice because most stalking statutes do not cover computers. In the Farmington Hills case, a disappointed suitor was charged when he kept sending e-mail after a brief relationship ended[33]. State District Judge Frederick L. Harris denied the defense request to dismiss on grounds of vagueness and because the statute was overbroad; however, he did limit the prosecution's proofs. Both sides are seeking leave to appeal[34].

A matter of much public debate is the issue of children having access to pornography or potentially being contacted by adults with bad intentions. This is the subject of Sen. Exon's proposed bill, also discussed herein. The particular subject of "child pornography" (generally, explicit depictions of those under 18) was dealt with in the Protection of Children Against Sexual Exploitation Act[35]. The law is worth mentioning for two reasons: First, it was recently amended to specifically cover distribution of visual depictions of minors by computer. Second, the statute was recently upheld against a constitutional challenge (in a case not involving a computer). In United States v X-Citement Video, Inc, the Supreme Court concluded that the requirement for "knowingly" transporting such material covered both the sexually explicit nature of the depiction and the age of those depicted[36]. However, this is still a very broad law, especially since the depiction of the minor need only be sexually explicit; it need not be obscene.

As this paper was going to press, news broke of a prosecution in Los Angeles for possession of child pornography obtained over the Internet. The City Attorney expressed his concern that computer distribution of child pornography "may become the preferred method of distribution[37]." The man came to the attention of the city's "sexually exploited child unit," which corresponded with him and obtained incriminating pictures. At that, the defendant may be fortunate because he is only charged on a city misdemeanor complaint, with a maximum of only 1 year and $1,000. The corresponding federal charge, discussed above, is a serious felony. City Attorney James K. Hahn seemed aware of at least some of the issues raised by on-line communications when he was quoted making the following distinction: "Certainly if you

see something flicker across your computer monitor, then you are not in possession. But I think if you go to the difficulty of downloading then it's yours[38]."

Telecommunications and the "information superhighway". Last year's hot debate in the telecommunications area involved privacy, especially enhanced wiretap capabilities and the Administration's notorious "clipper chip" encryption proposal. It appears that the clipper chip is dead, at least for now. The proposal to require a hardware standard for encryption was widely attacked on a number of fronts, including impracticality and lack of flexibility as well as privacy. (The government would have retained a key, enabling it to read private communications.) Federal agencies are still working on the clipper chip for their in-house use, however.

On the other hand, the Communications Assistance for Law Enforcement Act[39] did pass in the closing days of the last Congress. This law will require "telecommunications carriers" to assure that their systems are wire-tap friendly, that is, "capable of — (1) expeditiously isolating and enabling the government, pursuant to a court order or other lawful authorization, to intercept, to the exclusion of any other communications . . . ." all signals of the carrier[40]. While there are also supposed to be privacy protections, the plain intent of the act is to require the phone company to guarantee that the calls it carries can be tapped, at least by the government. As FBI Director Louis Freeh has explained, "We just want to make sure we have a trap door and key under some judge's authority . . . .[41]"

Congress is considering a sweeping telecommunications bill this year which is expected to be the subject of considerable debate this summer. Senator Jim Exon (D. Neb.) has proposed the "Communications Decency Act," which is intended to protect children from pornography or pedophiles on the Internet. These proposed amendments are intended to criminalize use of any "telecommunications device" to make any "obscene, lewd, lascivious, filthy, or indecent" communication; make harassing calls; or to make "any indecent communication for commercial purposes" to a minor. This is a sweeping proposal indeed; perhaps its best feature is a provision that a BBS operator who is not responsible for content has some protection from liability.

The potential liability of bulletin board operators and similar service providers is a significant problem that is only beginning to be addressed. Some on-line services do monitor content; does doing so make them more or less liable? Under the Exon bill, that might constitute "editorial control over the communication" and may increase liability. Do they warrant to subscribers that children using their service will be protected from obscenity or harassment? And what of their liability if either infringing (that's how lawyers say "pirated") software or defamatory comments are disseminated using the bulletin board? These questions are just beginning to be raised; more detailed treatment is beyond the scope of this paper. In one very recent case, Prodigy faces potential liability in a multi-million dollar libel suit because its content guidelines and "board leaders" gave it a sufficient degree of editorial control[42]. The plaintiff corporation contends it was defamed by a posting under a false name.

Cross a photocopier and a telephone and you have the fax machine; add a computer and you have the ability to send multiple facsimile copies at the press of a button. It did not take long for some unknown pioneer to invent the so-called "junk fax," unsolicited facsimile advertisements, broadcast to lists of fax machines. Congress reacted to junk fax complaints by passing the Telephone Consumer Protection Act of 1991[43]. The statute prohibited the unsolicited distribution by facsimile of commercial advertisements. Advertisers who were shut off from this means of advertising promptly sued to enjoin the FCC from enforcing the act. This past February, the Court of Appeals held that the statute was constitutional, even though it did not prohibit all unsolicited faxes[44].

Concluding thoughts: the year ahead.

It seems incumbent on any author of a "year in review" article to venture an opinion as to the areas in which there are likely to be significant developments in the year ahead. Predicting what others may do is fraught with peril, especially when speaking of institutions such as the federal government. Still, two areas stand out as worthy of special attention in coming months; each is likely to be a significant influence on the shape of the industry in years to come.

The first area to watch is the Microsoft antitrust situation generally. Even if the settlement proposal rejected by Judge Sporkin holds up on appeal, Microsoft may well face further challenges to its marketing practices. The now-abandoned acquisition of Intuit clearly indicated a corporate intention to become a significant player in financial software, to the point that bankers were made quite anxious about the proposed merger[45]. The increasing extent to which Microsoft combines various software programs may also become an issue. This is an area of some considerable uncertainty. As one insightful commentator explained, "I thought it was pretty amazing that Microsoft announced plans to bundle into the next release of Windows (Windows 95) the software for Microsoft Network . . . .[46]" The legal concern is that if Microsoft sells different software programs in one big "bundle," businesses that compete with part of Microsoft's product line will be shut out; some critics have long claimed that Microsoft, as a operating sys-

tem developer, has had an unfair advantage in applications software. Microsoft is more than just a dominate player in the industry; any antitrust developments affecting Microsoft will have a substantial impact beyond the MS/DOS world. The marketing practices of all software companies, and even the shape of the computer software industry, could well be affected because antitrust policy can fundamentally shape the way an entire industry does business[47]. Restrictions on software development practices could become an important part of doing business. More generally, litigation will likely play an increasing role in the growth and rationalization of the computer industries. Enough to borrow a phrase from Clausewitz[48] and say that in today's economy: Litigation is the continuation of marketing by other means.

The second area of significant future development is that of telecommunications regulation. Presently pending legislation to substantially revise federal telecommunications policy virtually assures significant change. More generally, public fascination with, and concern about, the Internet will likely translate into continued attention from policymakers. One issue likely to receive continued attention is that of privacy, ranging from children's access over parental objections to sexually-oriented material, to public cryptography and wiretapping, to workplace privacy.

The Internet is already becoming a metaphor for the continued extension of computers into the lives of more and more people. Telecommunications policy is a logical, and likely, place to regulate computer-based activities. Indeed, this may become the principal basis for future regulation of computer usage. That would be ironic, considering that the computer industry itself has grown despite, so far, the rejection of various proposals for direct government control. This is not entirely without precedent: the first substantial regulations of automobiles were motor vehicle codes and parking meters.

Endnotes

1. The author wishes to acknowledge Bernard L. Meyer and G. Burgess Allison, who have contributed to the author's understanding of computer technology.
2. U.S. CONST. art I, sec 8, cl 8, authorizes Congress to secure "to Authors and Inventors the exclusive Right to their respective Writings and Discoveries." Similarly, the "commerce clause," art I, sec 8, cl 3, authorizes the regulation of interstate commerce. Note to readers: Citation form generally follows the Michigan Uniform System of Citation rather than the format customary with law reviews.
3. United States v Microsoft Corp, Civ No 94-1564 (DDC 1994).
4. MacLachlan, "Microsoft Stalled by Tunney Act," National Law Journal, February 6, 1995, p A6 col 2. (The Tunney Act requires public comment and judicial review of the proposed settlement, but limits judicial modifications of those settlements.)
5. "Micromanaging Microsoft," Detroit News, May 24, 1995, p 10A, col 1.
6. 35 F3d 1435 (9th Cir 1994)
7. Id, 35 F3d at 1444.
8. In general, an idea alone cannot be copyrighted or patented because only the expression of the idea is protected.
9. Lotus Development Corp v Borland Int'l, Inc. Civ No 90-11662-K (D Mass).
10. Lotus Development Corp v Borland Int'l, Inc. 831 F Supp 223 (Keeton, J, D Mass 1993).
11. Id, 799 F Supp 203, 220 (D Mass 1992).
12. Id, 799 F Supp at 212-214. The opinion compares the programs even at the level of individual keystrokes.
13. 17 USC 102(b).
14. Lotus Development Corp v Borland Int'l, Inc. Dkt 93-2214 (1st Cir March 9, 1995)(emphasis added).
15. Wall Street Journal, April 11, 1994, p A1 col 4.
16. "International commerce law draft complete," Automatic I.D. News, Vol 11, no 4 at 54 (April, 1995).
17. Generally, 11 USC 365.
18. 11 USC 365(n).
19. In re Prize Frize, Inc, 150 BR 456 (9th Cir BAP 1993) and In re Ron Matusalem & Matusalem of Florida, Inc, 158 BR 514 (Bkcy SD Fla 1993), respectively.
20. In re Wang Laboratories, Inc, 164 BR 404 (Bkcy Mass 1994).
21. The new language is codified at 11 USC 507(a)(3)(B).
22. Huffcut v McDonald's Corp, reported in National Law Journal, February 13, 1995, B1, col 2.
23. "Taking a Computer Crime to Heart," New York Times, January 28, 1995, p 17 col 3.
24. United States v Mitnick, Crim No 5:95-Cr-37-01-BO (ED NC March 9, 1995).
25. Then again, perhaps not. In the hours before his arrest, Mitnick apparently made a number of cellular calls, which were monitored. The charges actually brought are certainly easier to explain, and prove, than the original Shimomura break-in. Mitnick was charged under three different subsections of 18 USC 1029, including one count of possessing "device-making equipment," a com-

puter capable of reprogramming a cellular phone [section 1029 (a)(4)]; one count of possessing fifteen or more "unauthorized access devices," e.g. cellular access codes [section 1029(a)(3)]; and twenty-one counts of placing unauthorized, cellular calls [section 1029(a)(1)].

26. 18 USCA 1030 (West Supp 1995).
27. United States v Baker, CR 95-80106 (ED Mich); filed sub nom, United States v Alkhabaz. Jake Baker is the name he was known by at the University of Michigan; it was not just a nom de byte. The unknown "Arthur Gonda" was also indicted.
28. 18 USC 875(c).
29. 18 USC 875 (c).
30. "Attorney: Student's fantasies harmless," Oakland Press, April 28, 1995, p A8.
31. 1992 PA 260, eff January 1, 1993, MCLA 750.411h. Stalking is a one year misdemeanor. On the other hand, "aggravated stalking," is a five year felony; it has the additional element of: either violation of probation or a restaining order (as, during a divorce); or making threats; or prior stalking conviction. 1992 PA 261, eff January 1, 1993, MCLA 750.411i.
32. MCLA 750.411h(1)(e)(vi); MCLA 750.411i(f)(vi) (emphasis added.)
33. People v Archambeau, 47th District Case No 94-4-4039 (Farmington Hills, Michigan).
34. People v Archambeau, Oakland Circuit Case No 95-DA-6342 AR (Pontiac, Michigan); personal communication with Thomas W. Cranmer, counsel for defendant.
35. 18 USC 2252 et seq.
36. __ US __, 115 S. Ct 464, 130 L Ed2d 372, 63 USLW 4019, 4023 (November 29, 1994).
37. "Man Held in Child Pornography by Computer," New York Times, Friday, May 19, 1995, p A8, col 1.
38. Id (emphasis added). But consider the "knowing" state of mind required under the federal statute, recently upheld by the U.S. Supreme Court, which may not require downloading. That flicker could turn out to be the defendant's future flashing before his eyes.
39. Pub L 103-414, 108 Stat 4279, October 25, 1994.
40. Id, section 103(a).
41. Testimony, Senate Appropriations Committee, May 11, 1995.
42. Stratton Oakmont Inc v Prodigy Services Co, Case No 31063/94 (Supreme Court, Mineola New York), reported in Wall Street Journal, May 26, 1995, p B2 col 4. (In New York, the Supreme Court is the general trial court; the ruling was on a preliminary motion.)
43. Codified in part at 47 USC 227(b).
44. Destination Ventures, Ltd v Federal Communications Comm'n, 46 F3d 54 (9th Cir 1995). The court accepted evidence of the burdens of such faxes when the statute was enacted and refused to speculate on future technological developments.
45. Paré, "Why the Banks Lined Up Against Gates," Fortune, May 29, 1995, at 18.
46. Allison, "Technology Update," Law Practice Management, May/June 1995, pp 12, 16.
47. The breakup of the Bell System is proof of that. For another example, consider the great divide between moviemakers and theaters; a long-ago antitrust decree prevented the major studios from owning theater chains.
48. The aphorism is commonly referred to as, war being the continuation of politics "by other means."

A Macintosh based real-time, multi-tasking data acquisition system.

Susan E. Grocott and Gregory B. King***

*Prior Data Sciences, Suite 120, 1550 Enterprise Rd., Mississauga, Ont., Canada L4W 4P4.
**SCIEX, a division of MDS Health Group Ltd., 71 Four Valley Dr., Concord, Ont., Canada L4K 4V8.

This paper describes the design of the software system used by the PE-SCIEX API-300 mass spectrometer for acquiring data in a real-time, multi-tasking fashion using Macintosh computers. The system services the instrument in real-time and saves the data to disk without relying on cooperation from other executing applications. The system consists of separate, independent components for instrument control, instrument communications, and data processing running concurrently and in communication with each other. This paper concentrates on the design of the instrument communications component where the real-time, multi-tasking functionality is implemented.

Introduction

Modern scientific instrumentation can rapidly produce large amounts of data that must be saved and processed. This is not only true for imaging techniques like magnetic resonance imaging (MRI) but also for two dimensional methods such as liquid chromatography/mass spectroscopy (LC-MS) where repeated spectral scans are acquired over time.

It is a relatively easy task to write a program to service an instrument if it only runs in the foreground and allows only minimal user interaction during data acquisition. It would be better to write a program that is capable of running in the background without any degradation of performance. This would allow the user to perform other tasks during data acquisition. This means that the program must be able to service the instrument in a real-time fashion despite what other running programs or the user may be doing.

The Macintosh environment offers major challenges to writing data acquisition software. It is user interface oriented, not multi-tasking in the true sense, and not real-time. This means that foreground tasks and user interaction receive priority over background tasks. However, it includes support for multi-media applications which can be used to implement a real-time, multi-tasking application. In this paper we will show how various features of the Macintosh ToolBox can be used to create a real-time, multi-tasking data acquisition application. Such a system is used to control the API-300 mass spectrometer produced by PE-SCIEX.

It is important for future compatibility and support that only stable features of the operating system be used and that patches be avoided. This way the source code should be compatible with future versions of hardware and the operating system.

Functional requirements

At PE-SCIEX we have specified a multiple module data acquisition system using a client-server model. The basic system (see Figure 1) consists of a background task that communicates with the instrument, a foreground task for controlling the instrument, and another foreground task for data display and processing.

The background instrument communications task acts as an instrument server while the instrument control task is the client. This shields the control task from hardware dependencies. Other tasks, including third party software, can also be running. This allows for maximum flexibility since we can tailor modules for specific functionality and substitute modules to meet changing requirements.

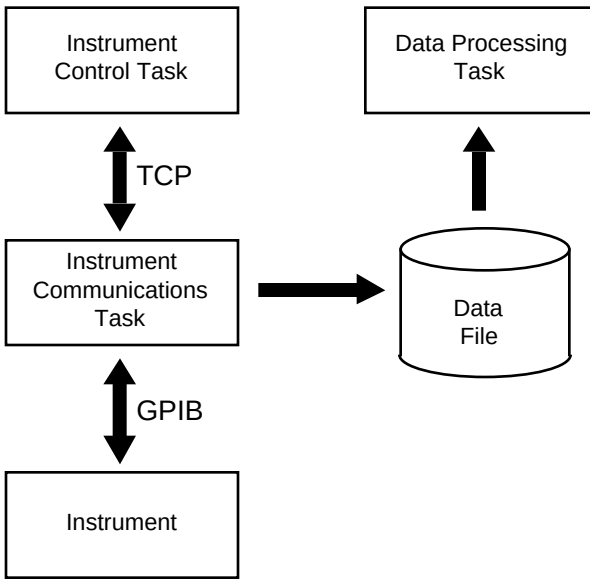


Figure 1. API-300 Software Architecture.

The background task handles all instrument communications and writing of data to disk. It must be able to perform these functions regardless of what other tasks may be doing and do so in real-time. It must also communicate with the instrument control task in a manner that doesn't interfere with its prima-

ry function, but still be responsive enough to handle user requests from the control task in real-time.

The communications task must write data to disk as it is being received from the instrument instead of keeping it in RAM until the end of the run. This is necessary because the total amount of data may easily exceed the amount of available RAM. This also makes the data immediately available to the processing task and minimizes the possibility of data loss due to catastrophic failure.

The instrument control task supplies the user interface for the instru-ment. It takes user input and builds commands for the instrument which are then sent via the instrument communications task. It communicates with the instrument communications task via TCP/IP. Several different control tasks have been written, each with a UI tailored to a specific application.

The data display and processing task does not have to communicate directly with the other tasks. It features the capability of reading data files that are still in the process of being acquired. This gives the user the ability to process the data as it is being acquired. It must conform with good Macintosh programming practices and periodically yield execution to other tasks during lengthy tasks. Since there are no special requirements for this task to support data acquisition it will not be discussed in this paper.

System configuration

The hardware consists of an Apple Power Macintosh 8100/80 computer with 24MB of RAM and a hard drive. The actual amount of RAM required is not fixed, but must sufficient to hold all executing tasks. Virtual memory can severely impact system performance and is not used. The actual interface to the PE-SCIEX API-300 mass spectrometer is either a National Instruments SCSI-GPIB-A SCSI to GPIB interface or an NB-GPIB/TNT NuBus GPIB interface card.

The operating system is Apple's System 7.5 with Apple's MacTCP 2.06 (TCP/IP driver) and National Instruments' NI-488 INIT 5.0 (GPIB driver) installed. The TCP and GPIB drivers must be correctly configured for the installed hardware. Other system extensions are optional.

The API-300 is a high performance triple quadrupole based tandem mass spectrometer designed to be interfaced to LC systems. It can scan the mass range from 10 amu to 3000 amu with a minimum step size of 0.05 amu and can be operated in both MS and MS/MS modes. Data can be generated at a rate of several megabytes/min on a continuously sustained basis at the maximum scanning speed of the API-300.

The PE-SCIEX API-300 itself contains two processors, a Motorola 68332 and a Motorola 68340, running a real-time operating system (Byte-BOS 683XX). The instrument is controlled by a custom post-fix instrument control language interpreter running on the 68332. The 68332 controls the instrument while the 68340 handles communications through a GPIB (IEEE-488) interface.

Design

The instrument control task is a standard Macintosh application that gathers user input and formats command sequences for the instrument and the communications task. Figure 2 shows the experiment editor window from the Tune application. This application is used for real-time adjustment and optimization of instrument parameters. Tune creates a scanning method from information that the user enters into this window and sends it to the instrument when a run is started.

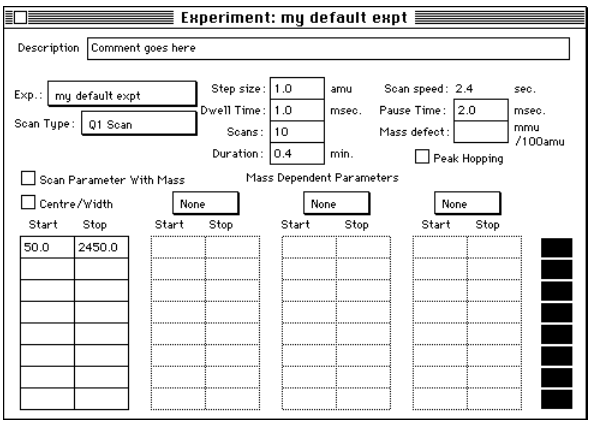


Figure 2. Experiment editor window from the Tune application.

The control task first attempts to open a TCP connection to the communications task and then requests a connection to the instrument before attempting to send any commands to the actual instrument. The communications task only allows one active connection to the instrument at any given time and will deny subsequent attempts to connect. When the control task is finished it must relinquish the connection so that other control tasks can connect.

The instrument communications task is implemented as a normal Macintosh application called MacDAD (Macintosh Data Aquisition Daemon.) It has to manage bi-directional communications with both the instrument control task and the instrument itself as well as writing data to disk and handling Macintosh events. This is accomplished through a multi-threaded design (see Figure 3), the use of circular buffers, and a custom disk caching scheme.

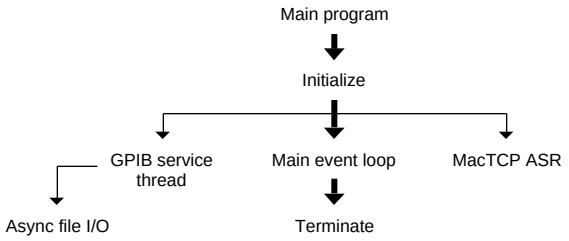


Figure 3. MacDAD execution threads.

All communications to and from the instrument are through message packets. Each message packet includes a header describing the type and amount of information contained in the payload, routing and reassembly information, a variable sized payload, and a checksum. The payload itself can be either commands or data. Routing information is necessary because commands can originate from either the instrument control task or directly from MacDAD itself. The instrument uses the source code of a command packet as the destination code of the reply that is generated.

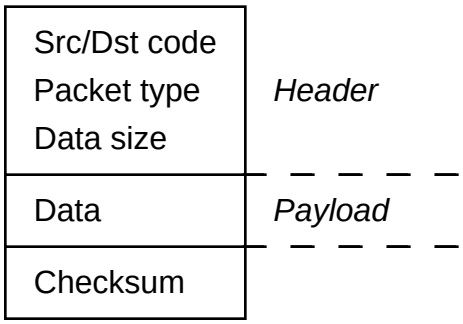


Figure 4. Instrument message packet structure.

Incoming packets are enqueued in a circular buffer before being dequeued and sent to their destination. Separate buffers are used for each direction (see Figure 5) and separate threads handle enqueuing and dequeuing. This allows the reading and writing processes to be decoupled. Packet processing can be done either before enqueuing or after dequeuing. Semaphores are used to control access to shared resources such as the circular buffers or communications interfaces by separate threads.

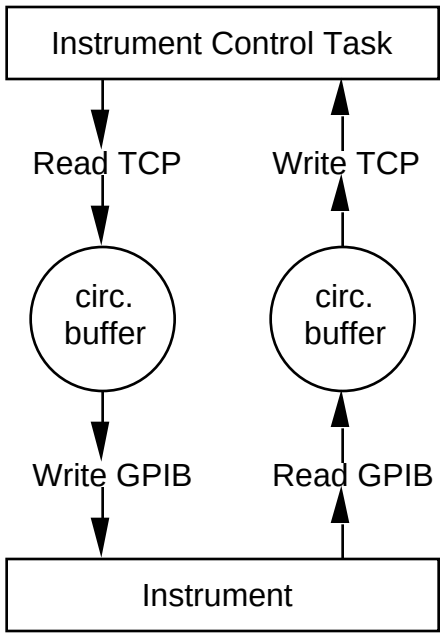


Figure 5. MacDAD data flow.

Multi-threaded design is the key to implementing a multi-tasking, real-time system. Macintosh execution threads are either cooperative or pre-emptive. Cooperative threads are only executed when other threads yield execution to them, they cannot pre-empt other threads. Pre-emptive threads will interrupt other threads when they need to execute and do not rely on cooperation from other threads. Pre-emptive threads can be implemented using interrupt service routines, vertical blanking tasks, the Time Manager, or the Thread Manager (except on PowerPC's). However, pre-emptive threads are restricted as to what they can do and have limited access to the ToolBox [Reference 1].

Time critical tasks are performed using pre-emptive threads which do not depend on other threads yielding time. Tasks which are not time critical are handled from the main event loop which is executed cooperatively. Other tasks of intermediate priority are handled using asynchronous service requests.

The main execution thread spawns a GPIB service thread and a MacTCP ASR task before it enters the main event loop. The GPIB service thread will also spawn asynchronous file write requests when data has to be written to disk.

The GPIB service thread is pre-emptive because incoming packets from the API-300 are time critical. The instrument signals that it needs servicing by asserting a GPIB control line. The current Macintosh implementations of the GPIB interface (NuBus and SCSI) are implemented via polling. Because polling degrades the real-time performance of the system it is done using the Time Manager so that polling intervals can be tuned for best performance.

The MacTCP ASR is also pre-emptive, but because

TCP events are not as critical, the event is stored and processed in the main event loop. When reading from a TCP stream, all available data is read and enqueued for further processing.

The writing of packets to either the GPIB or TCP is of low priority and is always done from the main event loop using synchronous device calls. The main event loop is responsible for handling Macintosh events, handling TCP events, writing instrument packets to TCP, writing TCP packets to the instrument, handling requests from other (pre-emptive) threads, and handling internal errors (see Figure 6).

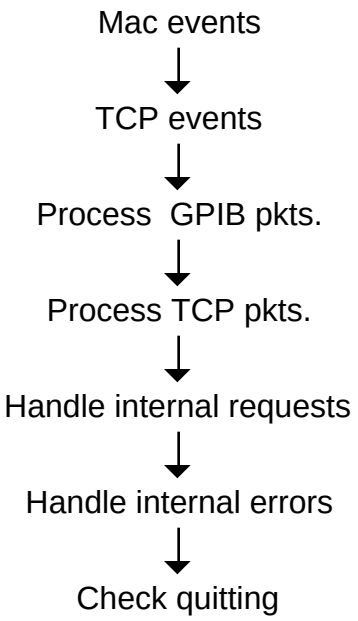


Figure 6. MacDAD main event loop tasks.

The writing of data to files is done both synchronously and asynchronous-ly. Operations done from the main event loop are synchronous while those performed from a pre-emptive thread are asynchronous. When a pre-emptive thread has to perform a synchronous file operation it posts a request to the main event loop to perform it.

The GPIB Poll Task

The Time Manager poll task is responsible for both servicing the API-300 and processing the received data. Before the task actually runs it checks the environment to ensure that it is safe for it to execute. The task first checks global semaphores to see if shared resources used by the task are available. Then it checks to make sure that it isn't interrupting an I/O operation and then if there is sufficient stack space available. If any of these conditions are not met the execution of the task will be postponed.

The presence of I/O operations is checked by walking through the driver unit table and checking each driver's drvrActive flag and queue header. Stack space is checked by calling the ToolBox StackSpace trap [Reference 2]. The StackSpace call works despite published warnings [Reference 3], an alternative would be to compare the contents of the stack pointer register with the current stack base pointer.

The poll task checks the state of the GPIB SRQ line. When SRQ is asserted, indicating that there is data to be read, the task will serial poll all GPIB devices in turn to determine which device has to be serviced. Multiple devices are supported. The data read in from the API-300 is processed immediately instead of being queued for later processing. This violates the guidelines for interrupt time tasks but is necessary in order to satisfy the requirement that incoming data be processed without cooperation from other tasks. Because this processing can be lengthy, the polling and processing is actually done from a deferred task installed by the Time Manager task. This sacrifices a small amount of real-time performance for greater compatibility with the Macintosh OS.

When processing instrument data, the task determines which need to be written to disk and which need to be sent back to the instrument control task. Data is written to disk directly from the task using asynchronous file I/O while data destined for the instrument control task is enqueued for later transmission, via TCP, from the main event loop.

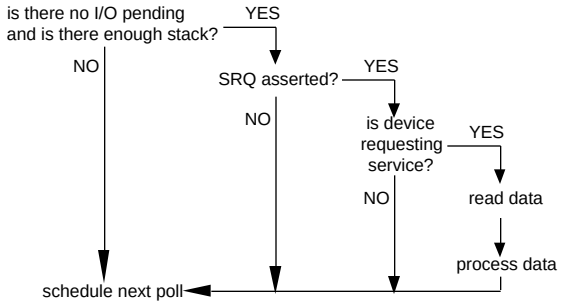


Figure 7. GPIB polling logic.

The poll task is executed at one of two polling intervals, a normal long interval for when the system is idle and a short interval for when the system is busy. The short interval is used when postponing a poll to reduce latency and when data is being read from the instrument in order to quickly flush the instrument buffers. When all data has been read the system will revert back to the normal polling interval. This scheme is a compromise between real-time performance and system loading.

Accessing the GPIB Interface

The driver supplied by National Instruments is only used to initialize the GPIB interface and to write to it. Polling and reading use custom code that directly accesses the GPIB hardware. Directly accessing the

hardware is done for performance reasons, using chained asynchronous driver calls in the poll task degrades real-time performance.

In order to poll and service the API-300 the poll task must first determine the state of the SRQ line. If it is asserted, the task must then serial poll the instrument to see if it is requesting service, and if it is the task must read the data from the instrument. These actions are normally done using the ibline, ibrsp, and idrd driver calls [Reference 4].

MacDAD replaces these driver calls with a set of functionally equivalent custom routines that directly access the GPIB hardware. These routines use the same parameters as the driver calls they replace.

Disk Caching

The use of a custom disk caching mechanism is important in tuning the performance of the system. The size of data packets returned by the instrument varies widely, as does the time interval between packets. The two limiting cases are small packets returned very frequently and large packets at a low frequency. The conditions will vary depending on the type of analysis chosen by the user and the type of sample being tested.

The normal disk caching mechanism where flushing is triggered by the amount of cached data only works well for the large packet case and even then the size of the cache affects performance. In the case of small packets being returned at a fairly high frequency, cache flushing would occur infrequently and can cause the Macintosh to fall behind in servicing the instrument whenever it occurs. This is because the time to flush the cache is greater than the time between packets. The larger the size of the disk cache, the worse this problem becomes because it takes longer to flush the cache.

All file writing operations bypass the regular Macintosh disk cache. Synchronous writes are immediate while asynchronous writes use a custom disk caching mechanism where the trip point for flushing the cache is determined by both the amount of data cached and the time since the last flush (see Figure 8). We also use a double buffering scheme where new data is cached in a second buffer while the first is being flushed to disk. This also eliminates the need for multiple parameter blocks since there is never more than one asynchronous operation pending and data can be accumulated in the second buffer while it is pending.

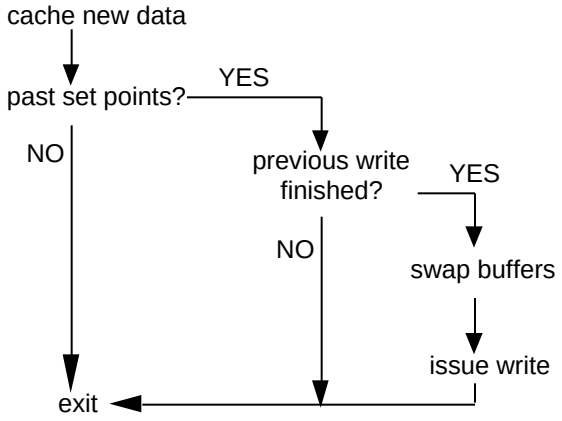


Figure 8. Disk write caching.

By flushing the cache based on elapsed time a smaller amount of data is written when packets are small, reducing the amount of time required to flush the cache. Combining the two mechanisms allows both large and small packet cases to be handled. In addition, control of the cache parameters is taken away from the user.

Summary

We have presented an overview of the software system used to control a PE-SCIEX API-300 mass spectrometer. It is both multi-tasking and real-time and the system meets its data acquisition performance goals. But because data acquisition has priority over all other tasks, acquiring data at the maximum rate can compromise the performance of other tasks. Performance at lower data rates is adequate to simultaneously analyze data and print results while acquiring.

The robustness of the system has been proven by daily usage and overnight and weekend batch runs in our test labs under a wide variety of conditions. In addition, special stress testing using limiting cases are also performed. Robustness is important because the system is intended for continuous operation.

Separate instrument control and communications tasks allows for the remote control of instruments. Only the instrument communications task needs to run on a computer that is directly connected to an instrument, the control task can run on any other networked computer.

Communications between the instrument control and communications tasks could also be performed using PPC instead of TCP. Using TCP has the advantage of allowing the instrument control task to be a non-Macintosh application. Because of the large amounts of data that can be transferred, the use of AppleEvents is not recommended.

In order to make the system compatible with virtual memory all buffers used by the instrument communications task must permanently reside in physical memory. This guarantees that they will always be

accessible to interrupt time tasks. However, virtual memory imposes a performance penalty on the overall system because other tasks will be swapping their storage in and out of physical memory. Therefore the use of virtual memory is not recommended.

Native PowerPC versions of some of the component applications have been built. Since the drivers and most of the ToolBox are still not native, only those applications which do a lot of data processing would benefit from converting to native code. In the case of MacDAD, which does little processing, the emulated version actually runs ~10% faster than the native. This is most likely due to the overhead in switching from native to emulation.

Acknowledgments

The authors wish to thank Ronnie Bishop and Dan Repich at National Instruments for their help in guiding us through the internal workings of GPIB interface boards. Thanks also to the other members of the API-300 team with whom we (and MacDAD) had to interface with: Peter Buckley (firmware), Peter C. Gzowski (Tune), and Enzo Iantosca (firmware).

References

[1] "Inside Macintosh: Processes", Apple Computer, Addison-Wesley, Aug. 1992.

[2] "Asynchronous Routines on the Macintosh", Jim Luther, Develop Mar. 1993 (13), pp. 5-30.

[3] "More on Asynchronous Routines in Issue 13", Develop Sep. 1993 (14), pg. 5.

[4] "NI-488.2 Software Reference Manual for Macintosh", National Instruments, Oct. 1993.

Macintosh Task And Process Priority Management

Grant Neufeld

The modes of organizing computing tasks are expanding and evolving due to the introduction of new technologies, techniques, and paradigms such as parallel processing, independent/intelligent applications, and document-centered processes. It is no longer acceptable to force the user to wait to regain control of the computer; the era of the watch cursor is over. The user must have the perception of instant access to commands and the ability to interrupt processes. Hence, to meet user expectations, and to accommodate multi-processing requirements, the design of task execution must be restructured.

Instead of a simple model of defining task execution — as either currently in progress or scheduled for later execution — applications will need to be, among other things, responsive to interruption at almost any point in their execution. The increasing number of processes operating simultaneously on modern computer systems has also created a need for prioritization — including the designation of tasks that might never execute — and fragmentation of tasks — division into sub-processes or independent processes.

In this paper, I will explore design methods available for task prioritization, fragmentation and inter-process cooperation. This exploration will emphasize the division of tasks into three major classes: 'Immediate' user interaction oriented, 'background' the actual work of the process, and 'optional' tasks that can be skipped if there is not sufficient processor time available. I will also examine these methodologies in the context of emerging technologies such as OpenDoc and preemptive threading. Regarding implementation issues on the Macintosh, I will focus on high-level events, threads, and the process related managers. I will also examine the use of cooperative and helper applications for task distribution, including those working across networks of computers. Through this discussion, I will provide a basis for application task design in the context of current and future Macintosh development.

Introduction

The goal I have in mind in setting out to write this paper, is to provide a direction for making use of unused processing facilities—maximizing system productivity in order to get more done quicker. I was lead to the ideas in this paper by consideration of the emerging field of agent software, and by awareness of the potential processing power being wasted by idle systems. As we give computers more independence, this wasted processing can be put to use for the performance of tasks that will not require human intervention. Some of this is being done today with background virus and disk maintenance applications, but I will point out in this paper that many more—if not the majority—of the tasks performed on computers can be designed to utilize available processing more effectively.

Most users imagine that computers should help them get as much done as quickly as possible. Unfortunately, that isn't always the case. A consequence of this is a reduced willingness to perform some tasks on computers. A complicated graphic filter which 'ties up' the computer for what are perceived to be—and perception is a key issue—extended periods of time is reduced in value to the user because of those constraints. By reducing the time it takes for tasks to complete (at least the perceived time) users will be more inclined to actually utilize those tasks.

The vast majority of user's don't care at all about code optimization and how fast a programs routines can run. They care only for the performance of the applications they use [5, p.676]. Performance is a measure of how quickly a user can accomplish their work, not how quickly a program can perform its internal functions. A program can have completely un-optimized code and still out-perform its competitors providing it makes it easy for users to get where they want to be.

With this distinction in mind, I'm going to ask for the world. I want computer systems to give me the perception of instantaneous response. I want most of my work to be done for me—before I ask for it. I want computer systems to utilize 100% of their processing capabilities all of the time.

These are not unreasonable requests. In this paper, I'll explain why.

Definitions

I will begin with some definitions to clarify key terms used throughout this paper.

Task: A set of work to be completed. May be divided into stages and/or sub-tasks.

Stage: A single, non-subdividable, portion of a task.

Process: A sequence of work.

Priority: How much preference is to be given to a task/stage/process over others for use of available processing—and potentially other—resources.

Weight: The relative priority given to a task or process. Measured in an arbitrary scale dependent on the system of priority management in use.

Agent: An application or task that independently performs tasks for users.

Quantum: time interval in which a process is allowed to run. [8, p.64]

Task Modeling In The Software Design Process

The software development process, as discussed here, is divided into six primary stages as presented by McConnell [5; Chapters 1 & 3]:

- Problem Definition
- Requirements Analysis
- Architecture
- Construction
- System Testing
- Future Improvements

The consideration of task priority and status should begin immediately following the definition of the problem and should continue through all the remaining stages. Every stage of design except the problem definition is affected by, and affects, process and task design.

As an illustration for this discussion, I will define an example problem as being "Users do not always spell words correctly."

At the requirements stage of application design, the results needed to solve the defined problem should be determined. In the example, the results needed are correctly spelled words. All tasks performed by the application should work toward achieving these results.

At the architecture stage the actual tasks and subtasks are defined and, depending on the scale of the application, the stages of the tasks are determined. In the example, I will define three major tasks (other than initialization and cleanup) which make up the components of the application loop:

check for misspelled word
confirm spelling with user
correct spelling

Sub-tasking the first of these, the following tasks are determined:

retrieve word
compare with dictionary
identify result of comparison

At the construction stage, the actual code for the functions which will perform the tasks must be written. A pseudo-code function might look like:

```
put the next word into variable theWord
search dictionary for theWord
if theWord was found return correct
else
    load suggestions as thread process
    ask user to confirm or correct theWord
    if user confirmed return correct
    else
        change theWord to user selection
        return incorrect
    end if
end if
```

Note that the load suggestions function is called as a thread process. What is meant by this is that the suggestions are loaded in while the user is being interacted with. Some existing spell checkers use this method to prompt the user with a misspelled word as soon as it is found and to start loading in suggestions while the user is looking at the word. This is a more effective method than making the user wait until all the possible suggestions have been loaded or waiting for the user to select a 'suggest' button. This type of implementation is discussed further in the section on anticipatory tasks.

As well as the normal testing of individual functions to see that results match intent and no bugs are present, it is crucial to test whether the defined tasks, of which the functions form the stages, actually provide the results they were designed to. This may seem obvious, but in actual practice is not always followed and is included here as a firm reminder.

Finally, the vast majority of applications are subject to upgrades and improvements (unfortunately, in practice, not necessarily at the same time). When modifying an existing application, all the stages described above should be followed. The new tasks, or changes to existing tasks, should be identified and the procedures applied at each stage should be adjusted accordingly.

Considerations in Designing Tasks

A number of factors and options need to be dealt with in the design and implementation of software tasks.

Primary among these is which resources will be needed and/or useful to the performance of the task. This includes memory, storage, data files, other applications/processes and results from other tasks, among other things.

Developers should be considering which tasks can be initiated without all of their pre-requisites first completing, as in UNIX pipe operations. Tasks should be designed to start as soon as possible, even if some of the pre-requisites needed to complete the task are not yet available or done. An example of this is found in UNIX pipe operations which allow the dependent process (the receiver of the pipe output) to process the piped information as it receives it, not requiring the dependent to wait for the depended upon process to complete. Under most circumstances, this will allow the task to complete more quickly.

Processes need to be able to deal with required facilities being down. If the process has other work it can perform in the mean time, it should work on that until the missing facility is available or it finishes all it can do without the facility. The process can also switch to an alternate (perhaps less useful or desir-

able) facility to accommodate its needs if one is available.

If a process does have to block because of a missing facility, it may be appropriate to notify the user or an owner/scheduler process so that the decision of whether to wait for the facility or cancel the process may be made. Not all processes will require this interaction; optional processes may be summarily canceled. [1, p.72]

"Transparency of Processing" [8, p.385-387] is important while at the same time, it is essential that users maintain control and the perception of control over the tasks on their systems. The user must at any moment be able to change task priorities, add tasks and cancel tasks. They must also be able to 'see' the processes if they want to—but not have to if they don't.

Good choices for feedback are progress bars and showing the results of a task up to its current stage (as in Netscape's display of html documents up to the point it has downloaded). Simply identifying the task's state as busy (as in a watch or beach ball cursor) does not tell the user anything of what work is being performed.

The splitting of processing for portions of tasks—whether to multi-processors, distributed processing, or other task handling applications—should in most cases be performed transparently to the user, requiring no intervention.

Another consideration in task design is which tasks can be run continuously, and also whether there may be tasks that could be added which would engage in continual processing. As an example, OpenDoc users will want dynamic documents that update themselves continuously (perhaps stock & news reports) with certain portions having periodic updates (news headlines) and others requiring continual updates (stock charts), possibly constrained to certain larger periods of time.

What is important to note is that the document components handling the news headlines and stock charts should continue to process their tasks regardless of whether the user is currently viewing them. They may be allocated fewer resources to do so, but the continual updating will mean that the user will be able to access the information with minimal delay.

It is also useful to consider who or what is initiating a task. Some possible task initiators are:

User
System
Software
Agent
Distribution (as in distributed processing networks)

Task Types

I will identify and attempt to define a number of distinct categories/classes of tasks. Individual tasks may fit in to multiple categories. I expect that there remain other valid categories which I have not considered here.

Anticipatory Tasks

One way to increase an application's performance is to perform part or all of a task's work before the user requests the task to be done. Amazingly enough, this advance processing does not require processors with built-in time-travel units. What is required is a model of task design which determines the earliest point at which a task can begin processing, regardless of the state of the user's input. I call this type of task definition 'anticipatory' tasks.

As an example, consider the task of spell checking text. The earliest point at which a spelling check task may begin is the point at which some text (any text) may be found. Some applications apply this by checking text as it is typed. I would suggest an even earlier point at which text can be found—when there are text documents available on local (or even networked) storage devices. Utilizing idle processing time, a spell checking task could scan drives for documents with misspelled words—compiling a list which a user could then view when they choose to. Various configuration options could be added to such a task, such as particular types of documents to skip.

An important aspect of anticipatory tasks is their level of feedback and intrusiveness to the user. Unless the user explicitly requests feedback (such as beeping when a word is misspelled) none should be provided except under special error conditions. As more anticipatory tasks are added to a system, the growth in unrestrained feedback could aggravate users, disinclining them to use the tasks. By the same token, insufficient feedback can also be seen as a problem area by users who want to know what tasks are taking place on their systems.

Almost any application that performs tasks that the user has to wait for will benefit from anticipating those tasks. In some cases this may require the application to have some level of intelligence and learning capability, and perhaps most applications would benefit from these. However, there remain numerous tasks which require no intelligence at all to anticipate.

For example, in the case of development environments, a number of tasks can be performed on a continuous basis: syntax checking, compiling, linking and maintaining the errors/warnings list. All of these can take place, while the programmer is entering code, as optional tasks using spare processing time (which is usually a lot when the only foreground task is the entry of code by the programmer). By perform-

ing these tasks continuously rather than waiting until the last minute (when the user actually requests the tasks to be done) there will be a significant increase in effective speed. Compiling will take place over a much longer period of time than with the 'last minute' compiling, but the time it takes to finish compiling once the programmer selects the compile command should seem almost instantaneous.

With this I come to the notion that software should operate in an evolutionary fashion; the user should be able to say at any instant "I want the results of the work you've done so far." In fact, the results should already be available for the user to work with—even though the work hasn't been completed the results itself should always be maintained in a 'completed up to this point' form while new changes and additions take place. Users should not be made to wait until every detail is complete before they can work with the few or many details already completed.

Optional Tasks

There are tasks which, while they will improve the results of a set of work, are not necessary for the user to get what they want.

An example 'optional' task is an anticipatory task to syntax check code while it is being edited. If there is sufficient processing time available the task should be performed because it will likely benefit the user. However, if the user has a 3D rendering program producing a graphic in the background while the user is editing code, there will not likely be sufficient processing time for the optional syntax check to take place. The user can still request a syntax check at any time, which will make the task a required one.

Any time-critical requirements for a process must also be specified. If an optional task has time critical requirements, and there isn't processing available for optional tasks, the task may be canceled because it isn't able to execute properly.

Timing Sensitive Tasks

Periodic tasks, such as mail downloading, may have a 'desired' time frequency specified, as well as a leniency factor to allow for—and restrict—delays during 'busy' periods. A more difficult scheduling trick would be to anticipate a busy period and schedule time sensitive processes early (within their leniency limits) to avoid it.

Dependent/Depended-upon Tasks

Some tasks will be dependent on the completion of other tasks before they can complete. There are two definable states for dependent processes regarding their dependence: having other work that can be done, and being unable to proceed (blocked) until the depended upon task(s) completes. An example is a task to spell-check a file which depends on a task to read the file.

In priority management, depended-upon task should not be aware of the processes that depend upon it for completion. The priority of the depended-upon process may have its priority affected by the dependent processes, but the system in use for prioritization can determine this without any input from the depended-upon task. It is, however, important for the prioritization system to track all dependencies.

The level of dependency should weigh (in proportion) upon the priority given the depended upon process, with each additional dependent processing adding weight as well. A scheduling system might also factor the number of dependents a process has in determining its priority.

Distributed Tasks

Individual users will not likely provide enough tasks for their systems to utilize all available processing. I would conservatively estimate that on the majority personal computers such as the Macintosh, at least 90% of the processing power is left idle. Consider the lack of productivity of systems during word processing when the only activity is user keyboard entry and the occasional formatting or save command. This 'spare' processing time can be utilized effectively for distributed tasks.

An example of a distributed task which can be spread out among many systems is the cracking of RSA encrypted keys. One such project involved about 100 systems across the internet and was done over the period of about 1 year.

Not all tasks benefit from distribution. Currently, the majority of distributed tasks are large, calculation intensive tasks such as graphics rendering which can be easily broken into small distributable chunks. A form of task that could benefit from distribution is the emerging field of agents, especially net agents. [1]

Distributed tasks will normally be treated as optional unless specifically designated as required. If a user's tasks begin to require more processing than will allow for processing of other systems' distributed tasks, the processing that has been done should be returned to the other systems and they should be notified that no further processing is available for the moment.

With multi-user applications it may be necessary to calculate priority determination based on priorities set by a number of users on different machines.

Eventually there will be a need for a Distributed Process Manager or a distributed process component within the Process Manager.

User Tasks

User tasks must be given the highest priority next to essential system tasks, except within extraordinary circumstances, or when the user has explicitly

allowed otherwise.

The two primary aspects of user tasks are interaction and feedback. Tasks should respond with feedback immediately to user input such as keyboard and mouse events. If the feedback can not be the completed results for a task, it should at least be an acknowledgment of the user's request and indication of its status.

Other Types of Tasks

System Tasks

Tasks performed to maintain and update the system. Examples include updating the clock and refreshing the video display.

Required Tasks

Quite simply, required tasks are those which must be completed, whether initiated by the user, the system or some other initiator.

Priorities

In order to properly schedule processing for the various types of tasks, a system of task priorities must be established.

Highest	High	Normal	Low	Lowest
user, system, timing sensitive	depended-upon	required	dependent	optional, anticipatory, distributed

Figure 1, Relative Priority of Tasks

User, system and timing sensitive tasks will take priority over all others.

Optional (including anticipatory and distributed) tasks will be given the lowest priority.

Dependent tasks will have their priority lowered in favor of depended-upon tasks which will have their priority raised.

The actual priority weight of a task should be calculated based on it's type, the weight of any dependent tasks and the weight of any tasks that have been indicated as being able to benefit from (but not depending on) the completion of the task. The latter determinant is more difficult to establish, and will probably not be used in many systems.

In addition to these basic factors, there may also be direct input from regarding the priority to be given a task, as well as individual weightings attributed to tasks by themselves and their 'parent' tasks or application. The priority for an application should also be a factor in the calculation of priority for its subprocesses. Decisions about which processes to distribute in distributed systems will be affected by priorities. It can get quite complicated without a clear delineation of priority control.

It is with this point in mind that I advocate later in this paper for the introduction of a comprehensive priority manager for the Macintosh operating system.

I suggest that individual processes should only be allowed limited say in determining their own priority. A system of priorities can only be effective if it is applied consistently throughout the whole system.

The priorities of tasks will also change based on changing circumstances. The user can change the priority, processes may indicate dependence or cancel their dependence, among other changing factors.

Other more complicated factors may be included in task prioritization. The time a task will take to complete a task can be considered; the shortest tasks should generally be done first. [8, p.67] The 'feasibility' of a stage of a task being completed within some time frame can also be used to affect its priority.

Prioritization must not prevent low-priority tasks from executing at all (unless they are optional). It should reduce proportionately the amount of processing resources available, but not to the point of preventing the task from reaching completion.

In another vein, faster, but lower-quality, methods for tasks should in some cases be performed first, with subsequent use of the higher quality method if time is available. An example of this is Netscape's "lowsrc" attribute for the image tag (IMG) in html. This allows a low-quality (and presumably significantly smaller) version of a graphic to be supplied for faster downloading; the user can then determine from the low-quality version if they want to take the time to get the high-quality image.

In more complicated systems, there may be negative effects from the use of low-quality processes. Marvin Minsky has given some arguments against this type of 'instant gratification' in (intelligent) learning systems [1, p.26].

All of the prioritization factors which affect application tasks apply equally to document tasks. OpenDoc documents will have many components which require processing at the same time, and some components may require other components to complete processing before they can do their work. (IE. spread sheet equation dependent on value set in another component.)

Some resources other than just processing may be useful to allocate based on priorities. For example, memory and storage limitations may mean that there isn't enough space for all processes to execute, so some processes may be completely prevented from being able to complete and others may have to wait for sufficient resources to be freed. For example, a process may take up a large amount of storage for scratch files which will be freed when it completes. Other processes may have to wait until it finishes to have enough space to work in. It may also be useful to consider doing processes requiring scratch space before those requiring permanent space, because if the latter is done first, the former may not have

enough room to work in.

Always keep in mind that the first priority is to get done what the user wants—that is the ultimate measure of a system's performance.

A Priority Manager

Throughout my work on this paper, I have continually returned to the need for a system based priority manager. Many of the prioritization controls I have discussed should be introduced at the system level to be effectively implemented.

The 'mechanism' for process/task scheduling should be in the system. The 'policy' should be available to the system and individual processes. [8, p.68-69]

The impending version of the Macintosh OS codenamed Copland will apparently support 32 levels of prioritization for processes created in the protected, pre-emptive area of memory.

> "New applications written specifically for Copland will be able to create protected processes that are pre-emptively multitasked at any of 32 levels of priority. These independent tasks will access the I/O subsystems directly but will not be able to use any Toolbox or QuickDraw routines to create user interfaces." [11]

A Macintosh Priority Manager should be responsible for, or provide a basis for, scheduling of processes and threads with the Process and Thread Managers.

When a required process identifies an optional process as being necessary for the completion of the required process, the optional process should be treated by the Priority Manager as required, and be given a priority level based on the priority of the required process. Because the effective priority calculated by the Priority Manager can differ from the specific priority given to the process, the optional process need not be aware that it is being treated as a required process.

A number of questions remain to be answered regarding the implementation of a complete priority manager.

How should permissions for processes to assign priority to other processed be determined? Protection may be needed against malicious or 'confused' processes which may attempt to negatively affect the execution of other processes by changing their priorities, marking them as optional, or whatever. Some sort of permissions scheme is needed to handle this. [1, p.53]

How should new or custom priority classes be identified? The system Priority Manager could use a method like AppleEvent descriptors to allow the addition of non-standard priority properties. Handlers for non-standard properties would have to be supplied either by a system extension or by an application using the property type. If no handler exists, the property should be ignored. There may be room for applications and extensions to expand or modify existing property handlers. The precedence or weight of individual properties in relation to each other must also be evaluated.

In this way, it could also be possible to add fields to priority records to identify total space required to work, total space wanted to work, total space needed to permanently allocate.

Some Necessary Priority Manager Functions

- flag a process's priority
- internal system calls from Process & Thread Managers to determine process to process.
- allocate/deallocate priority record
- set process (PriorityRec, processIdentifier); if processIdentifier is nil, calling process is used.
- set class; a priority record can have multiple classes
- set user defined class

Conclusion

One distinguishing thing about Macintosh programmers has usually been a willingness (whether out of genuine concern or necessity) to work harder so the user doesn't have to. What I am proposing will require more work on the part of developers in the design and construction of software.

When you start making it easier for the user to have their computer performing additional and/or anticipatory tasks without noticeably interfering with the user's active tasks, they will start using those facilities. It's like putting more ram or storage into a system — it will be used. This is about increasing the amount of work that can be got out of a system; even 'slow' processors have the potential to perform much more work. If a user doesn't have more work for their system to perform, the spare processing can be allocated for distributed processing by other systems.

I see two major additions/changes needed in the Macintosh OS. The first is the complete multi-threading of OS-components, as well as prioritization of system tasks. The second the introduction of a system wide manager for process priority control.

This paper has also covered many areas of task design and creation, as well as prioritization, which should be considered and hopefully implemented by software designers.

The less the user has to interact with the computer, the less user input required for work to be done, the better. This will allow greater productivity with less effort by end users.

Bibliography and Related Readings

I refer you to the following readings for more specif-

ic discussion of process design and applications, especially agent technologies. More comprehensive listings of additional readings may be found within these readings.

[1] *Communications of the ACM*. Volume 37, Number 7. July 1994.

[2] Haupt, Christopher. "The Construction of a TCP/IP to Apple Event Gateway for use in Distributed Computing Experimentation", MacHack Proceedings 1992.

[3] Horwat, Waldemar. "Communication Abstractions in Concurrent Processing" MacHack Proceedings 1993.

[4] Looker, Shane D. "Parallel Processing on a Macintosh Network", MacHack Proceedings 1993.

[5] McConnell, Steve. *Code Complete: A Practical Handbook of Software Construction.* Microsoft Press. Washington, U.S. 1993.

[6] Robinson, Patrick G. and James D. Arthur. "Distributed Process Creation Within a Shared Data Space Framework" in *Software—Practice And Experience.* Pp. 175-191. Volume 25, Number 2. February 1995.

[7] Sisak, Steve. "Adding Threads to Sprocket" in *MacTech Magazine.* Pp. 41-52. December 1994.

[8] Tannenbaum, Andrew S. *Modern Operating Systems.* Prentice-Hall, Inc. Englewood Cliffs, New Jersy. 1992.

[9] Tannenbaum, Andrew S. *Structured Computer Organization, Third Edition.* Prentice-Hall, Inc. Englewood Cliffs, New Jersy. 1990.

[10] Thelen, Randy. "Threading Your Apps" in *MacTech Magazine.* Pp. 48-55. November 1994.

[11] Howard, Stephen. "Copland revealed at WWDC" in *MacWeek*. Volume 9, Number 20. May 15, 1995. <URL:http://www.ziff.com:8006/~macweek/mw_051595/news1.html>

[12] Dreyfus, Paul. "Copland: Technology for Customers' Sakes" in *Apple Directions.* June 1995. <URL:http://www.info.apple.com/dev/appledirections/jun95/newsstratmos.html>

Contacting the Author

gneufeld@ccs.carleton.ca
aa917@freenet.carleton.ca
grant@idc.com
http://arpp1.carleton.ca/grant/
http://www.carleton.ca/~gneufeld/

Video: Implications of a 60 year old technology; How to deal with its idiosyncrasies

J. Christian Russ, Reindeer Games, Inc.

NTSC video is the mainstay of American entertainment, Television, Videodisk, Videotape, and now Quicktime. But when performing video effects in Quicktime, there are some things to be aware of. A lot of compromises were made in video to make it look as good as it does. It depends upon many properties of the human visual system. This paper discusses some ways of improving video, helping WYSIWYG, and making compression better than thought possible.

History of TV

The origins of television come from a paper entitled "The Iconoscope–A Modern Version of the Electric Eye" 1934, V. K. Zworkin (RCA Victor). This was a description of the first practical TV image pickup tube. In conjunction with an earlier paper discussing raster scanning on cathode ray tubes, television became practical. Then came World War II.

Television took off after the war. NTSC video (and RS-170) started out in the 1940's as a standard for transmitting and drawing black and white images on a cathode ray tube (CRT). This was accomplished by scanning an electron beam that would strike a phosphor coating on the inside of a vacuum tube both horizontally and vertically at pre-set rates and modulating the intensity of the beam, in addition to synchronization pulses.

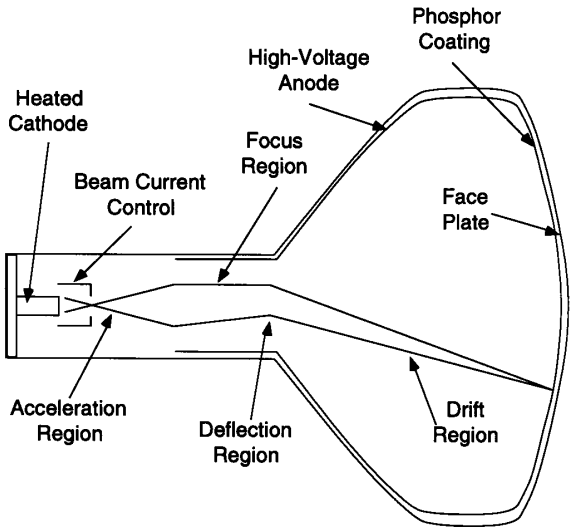


Figure 1 - **Cathode Ray Tube (CRT)**

There were three major problems to be solved:

1) There was a limited broadcast bandwidth.
2) A phosphor was needed that would stay on from one scan of the beam to the next.
3) The scan rate had to be high enough to redraw the screen in a reasonable period of time.

Add to that the fact the humans start to notice flicker when the frame rate gets below 24 frames per second (and some have seizures at 16 per second). The standard later evolved to support color, requiring even more data in the signal.

The current standard draws 525 lines on a CRT every 1/30 of a second. Each one of these lines is horizontal and goes left to right on the tube. This leaves 52µs (of the 64µs per line) to draw a horizontal line. Of the 525 lines, only 480 are defined within the viewing area, and most television sets are lucky to get 350 lines. With 480 lines visible and a standard 4:3 aspect ratio, there are 680 points horizontally. This requires at least a 6.1 MHz signal for black-and-white, and that simply isn't available for broadcast. In reality, the horizontal resolution is much less than that (3.3 MHz) which gives 340 real points on a line. The resolution isn't the same horizontally as it is vertically.

Interlace

There is a problem with drawing 400+ lines from the top of the screen to the bottom of the screen, especially with 1940's technology. The top lines have faded by the time the bottom ones are drawn. The solution was interlace.

Imagine that the scan rate is not one frame per 1/30 of a second, but rather two frames per 1/30 of a second (or one frame per 1/60 of a second). The first of these frames draws the odd numbered lines (1, 3, 5, …) and the second draws the even numbered lines (2, 4, 6, …). The lines from the even frame are drawn between the ones on the odd frame. This is called 2:1 interlace.

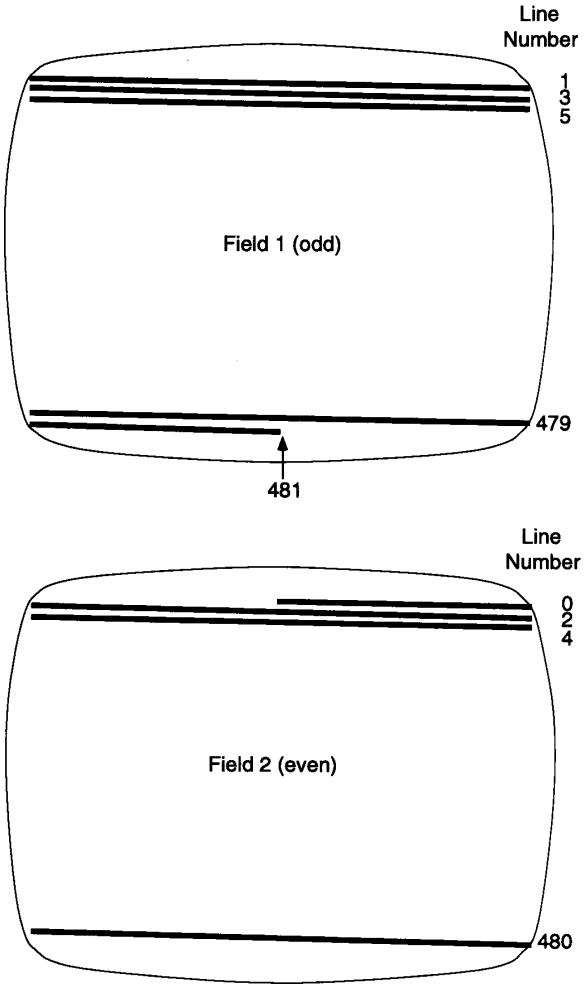


Figure 2 - **Interlace Fields: Odd & Even**

In this way, the amount of time from the top to the bottom of the screen is only 1/60 of a second for 262.5 lines, and cheaper phosphors can be used. [This can best be described as a video hack.] At 50Hz in Europe, the frames are 625 lines and thus 30% larger. Typically PAL is digitized at 768x512 with the same 4:3 standard aspect ratio as NTSC. PAL and SECAM use different color encoding schemes.

This interlace has some rather disturbing implications on image digitization if anything is moving in the field of view. More on this later.

Noise & Electrical Appliances

There are two kinds of noise: random noise and pattern noise. Random noise causes speckle throughout the picture, but pattern noise causes repeating patterns, and can often be removed with better cables, shielding, and grounding. Certain kinds of image processing can remove pattern noise, too.

Appliances with fans, compressors, microwave emitters, electron guns (TV's), RF crystals (computers, radios) can cause pattern noise.

Everything in a house (that has electricity) that can conduct a current is oscillating at 60Hz. including skin, pets, metal, containers of soda, etc. The wire that connects the camera to the computer acts as an antenna and also gets a 60Hz signal imposed on it that may or may not be in phase with the video signal. This can show up as interference patterns in the picture. This wire also picks up the pattern noise from household appliances.

Color

By the 1960's color television was on the consumer horizon, and there was a big problem; how to still allow the existing black-and-white television sets to work, and yet provide color for the color sets? The solution is another video hack, and explains why there is a strange transform to convert a color picture to gray scale. The NTSC solution was to separate RGB color into YIQ space.

YIQ Equations

$$\begin{vmatrix} I \\ Q \\ Y \end{vmatrix} = \begin{vmatrix} 0.60 & -0.28 & -0.32 \\ 0.21 & -0.52 & 0.31 \\ 0.30 & 0.59 & 0.11 \end{vmatrix} \bullet \begin{vmatrix} R \\ G \\ B \end{vmatrix}$$

Example: $Y = 0.30R + 0.59G + 0.11B$

Basically the idea was to compress the signal especially where people can't see the differences as easily. Its okay if the color is slightly wrong as long as the edge is in the right place. YIQ breaks the RGB image into three parts: Y (also called luminance) which is still treated as the B/W intensity image, I (standing for in-phase) is an orange-cyan axis, and Q (standing for quadrature) is a magenta-green axis. I and Q (combined signal called chrominance) are modulated at 3.58 MHz (interesting how this is the video clock-rate of an Apple II) on top of the Y signal. On a black-and-white set, all you see is the Y signal, because the higher frequencies (containing the color information) aren't shown.

Because the low-frequencies contains the intensity of the image, if there is noise, or some of the signal is lost, it is more likely that the color should be messed up, not the position of the edges. (NTSC stands for the National Television System Committee, but often referred by engineers and technicians as "Never The Same Color".) For compression purposes this noise is a big deal, especially since most of QuickTime compression techniques use RGB color.

Film is completely different. With motion pictures the entire frame is projected 48 times per second (each frame twice). No portion was drawn first or last, and everything fades away at the same rate while the next frame gets ready to be projected.

Digitizing Images from NTSC

A computer uses a frame grabber board to capture a video signal synchronized to the top of the frame and will digitize the signal into memory, either on the card itself, or over the bus. A lot depends upon how good the A/D (analog to digital converter) is because a poor one will produce a lousy image.

There is a bigger problem than the quality of the A/D, however. It is interlace. While it is true that the image is 640x480, there is the problem that the even lines (2, 4, 6, …) were digitized 1/60 of a second later than the odd lines (1, 3, 5, …). If there is a moving object this causes some nasty effects.

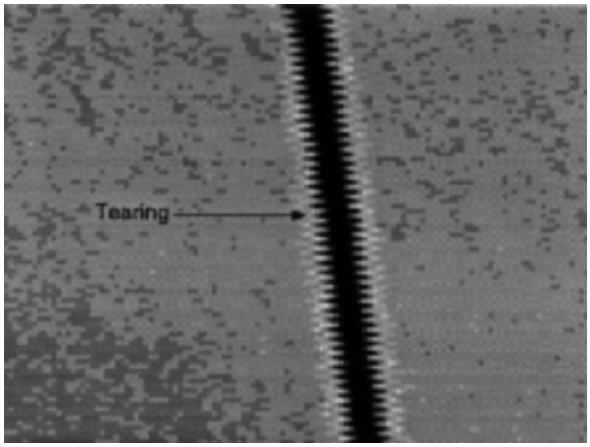


Figure 3 - **Edge Tearing**

The most common solution is to ignore the even lines and only keep the odd ones through a technique called "line-doubling". Unfortunately when people use this technique, they still assume that the frame rate is only 30 frames per second, where there are now really 60 frames. [I have never seen a VCR that allows still frames and uses line doubling that will show the even lines as well as the odd ones.]

The Video-Spigot™ and Apple's AV board (including the 840AV and 660AV) are in the same category. These boards perform line-doubling to solve the interlace problem, and go on to provide several sample down sizes for the video frame: 1/2, 3/8, 1/4.

> Note: the 3/8 size doesn't look as good when the video hardware samples down. It is generally better to grab the 1/2 size and sample down in ConvertToMovie™.

There is another problem caused by interlace. Aliasing is caused when a line is near horizontal and we use line doubling, or the method of just keeping one of the two frames.

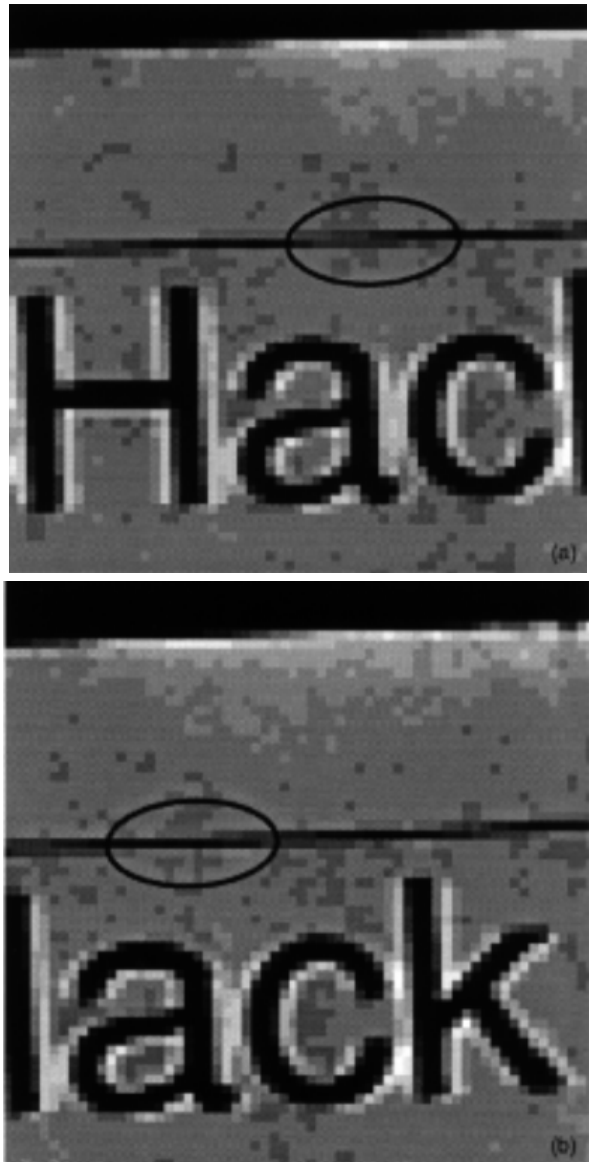


Figure 4(a)(b) - **Horizontal Aliasing**

If the object that has these aliasing effects is moving across the screen, sometimes falling across a odd line and being visible and sometimes falling across an even line and being dropped, moving Møire patterns result.

Digitizing BOTH the odd and even lines and averaging down to the desired size is vastly preferable to the line-doubling process for still scenes. For moving scenes, where there are no narrow near-horizontal lines, line-doubling is acceptable.

Dealing with Noise

Noise can come from a number of sources, but the primary source is the A/D. Here is where the quality of the A/D and the conversion circuitry that converts YIQ back into RGB comes into play. In order to simplify this problem let's assume that we have just a black-and-white signal. If the A/D has 6 bits then the video signal can be turned into 64 discrete values,

giving a signal-to-noise ratio of 64:1 or 36dB (decibels). Furthermore, the IEEE has only defined 140 levels in the video intensity, so there are slightly more then 7 real bits in the intensity contained within the video signal.

Signal-to-Noise (S/N) Ratio:

(dB) = 20 * LOG10(biggest - smallest)

# Bits	Dynamic Range	S/N ratio (dB)	
4	15	24dB	
5	31	30dB	
6	63	36dB	
7	127	42dB	
8	255	48dB	
12	4,095	72dB	
16	65,535	96dB	*CD-Audio*
18	262,143	108dB	

Now consider 15-bit color. Treat this as three 5-bit A/Ds, one each for Red, Green, and Blue. The bottom bit is correct somewhat more than half of the time, plus the original source was NTSC, so there is a significant impact on compression, especially when nothing is changing in certain areas of the image. The obvious solutions are to:

1) Adopt a digital video standard so that the digitization is done when the picture is captured, and everything stays digital after that,
2) Try to take the picture directly from a camera or video disk instead of video tape,
3) Get a better A/D chip, although NTSC video will still be a problem, or
4) Use frame averaging.

Frame Averaging

Frame averaging is the digital equivalent of a longer exposure time with slower film. The grain size is smaller and the amount of visible noise is less. The actual exposure time can't be lengthened with a video camera easily, but digital frame averaging does the same thing.

This method adds together a series of frames and keeps only the average of the pixels at each x, y point. If the scene is not changing, an A/D that was 5 bits (30dB) can be improved to 8 bits (48dB) by simply adding eight frames together. By adding 128 frames the quality can be improved to 12 bits (72dB), but since TIFF and PICT only store 8 bits per color channel, the extra averaging is usually wasted. (It also takes a lot longer.)

Simple averaging isn't terribly useful when something is moving in the scene. Obviously any areas within the video clip that are static should be replaced with their averages using statistics to get a better signal-to-noise ratio. But, what is a static area and what isn't?

Just as it is possible to compute a pixel's average over a series of frames, it is possible to compute its variance.

Statistics Equations:

n = number of pixels

x_i = pixel from image #i

$$\text{Average} = \text{SUM}(x_i) / n$$

$$\text{Variance} = 1/(n-1) * \text{SUM}((x_i - \text{Average})^2)$$

simplified...

$$\text{Variance} = 1/(n-1)*(\text{SUM}(x_i^2)-\text{SUM}(x_i)^2/n)$$

$$\text{Standard Deviation} = \text{SQRT}(\text{Variance})$$

Wherever the variance is low, the image is not changing much, so if we provide a threshold, or a maximum limit that the variance can be, the pixels can be selectively replaced for every frame in a video clip. This generally only works if the camera stays in a fixed position, but this is already a major improvement in image quality.

If the compressor uses frame differencing, then these static areas within the image require no storage beyond the key frame. The means that compressed file size goes down, playback speed goes up, and image quality improves.

It should also be possible to make an picture from the variances. It could be black-and-white or color, depending upon how the separate color channels are treated, but it could be used to segment the original for smart sweeps, dissolves, blue-screening, or other digital effects.

Conclusion

Video is not simple to digitize. The same trick that makes possible broadcasting a passable picture that degrades well, makes getting a really good picture difficult. Methods that reduce noise and improve color quality are well worth the trouble since they also improve compression and playback speed.

References

Huntsberger, David V. and Billingsley, Patrick (1981) *Elements of Statistical Inference, Fifth Edition* Allyn and Bacon, Boston, Mass.

Inoué, Shinya (1986) *Video Microscopy,* Plenum Press, New York, New York.

Russ, J. C. (1990) *Computer Assisted Microscopy,* Plenum Press, New York, New York.

Appendix: Gamma - Linear vs. Log

When watching television it is pretty easy to tell what was filmed with a studio camera from what was filmed with a hand-held video tape recorder. There are several indicators of this:

1. There is quick or shaky movement (home video).
2. The image is grainy.
3. The lighting looks "different."

In a lot of the action TV series the directors try to get a lot of interesting shots during the stunts. Unfortunately the cameras can be expensive, so they place the studio cameras in safe places and occasionally put disposable High-8 Camcorders on tripods in creative locations to see what they might get. These little clips get spliced in and are only on the TV screen for a few seconds at a time, and yet the viewer can tell right away what happened.

There are several reasons for this, as mentioned above, but the best reason is the third: the lighting looks different. This is because of a side effect of the types of cameras and how they respond to light. In the desktop publishing community this is called gamma.

In order to understand the gamma curves and when they are appropriate, it is necessary to look at the human visual system and film.

The human eye has about 5 bits of resolution (humans can see about 30 grey levels at a time), and that is distributed over the the intensities that it receives logarithmically. This provides excellent response in high and low light conditions even before the pupils react and the eyes adjust.

Photographic materials, film emulsions, paper, etc, all respond logarithmically as well, and typically film is vastly better than the human eye in what it can see all at once.

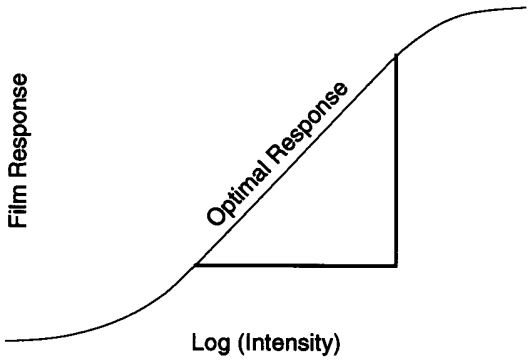


Figure A. Film has a logarithmic response to light for over a large portion of its response curve. Most photographers use this Optimal Response range.

A vidicon camera (or a tube camera) outputs voltage that is proportional to the log of the brightness that it sees. This voltage is transmitted to a television which has the same properties – the display shows the exponential of the voltage in the video signal. Therefore no correction was needed.

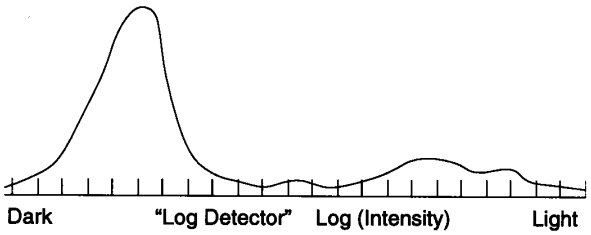


Figure B. The intensities are broken up into logarithmic steps. Since the histogram is also logarithmic, the intensities seem to have equal steps.

CCD cameras are a result of the VLSI revolution. The elements in the CCD camera build up voltage directly from the light that strikes each element. Internally, the voltage is linearly related to the amount of light that struck it. (This is excellent for low-light conditions.)

Cheap video cameras then turn this accumulated voltage into a video signal, but the lighting in the resulting image looks different on any display device that worked well with old video.

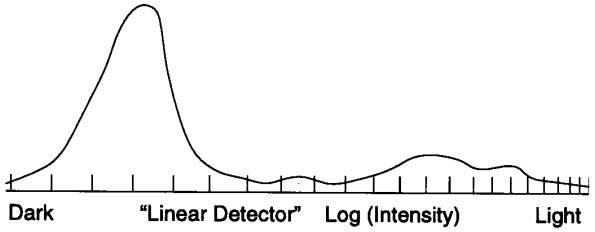


Figure C. The intensities are broken up into linear steps. Since the histogram is also logarithmic, the intensities seem to have un-equal steps.

The gamma curve is a mapping function that computes how a linear voltage should appear on a log device like a TV or Film Recorder. This gamma correction takes place inside of studio cameras so they behave in the same manner as film or old-style vidicons.

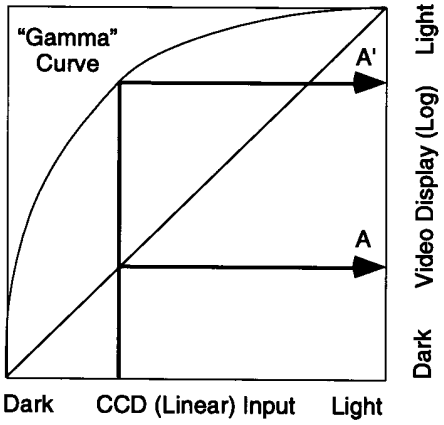


Figure D A gamma mapping function that takes a linear intensity and changes it so it appears correct on a logarithmic display device.

Unfortunately if there are only 8 bits in the signal, the darker portions of the images seem very choppy - the differences between low light levels seem very exaggerated and the image does not look good.

This problem is abated with linear output devices. If an image from a CCD camera or scanner is printed on a Laser Printer or with offset printing then everything is fine and no gamma correction is necessary. The gamma is needed to look at the image on the screen, but NOT on the printed page.

Conversely, if the image came from a log device and is displayed on a linear device (such as a printer), the curve needs to go the other way.

Device	Linear/Log	approx # bits
Input Devices:		
Human Eye	Log	5
Film	Log	18
Slides	Log	10
Scanners	Linear	8†
Vidicon	Log	7
CCD Camera	Linear	7
Studio Camera	Linear -> Log	12*, 7
Output Devices:		
TV Set	Log	7
LCD Panel	Linear	4–5
Prints	Log	15+
Slide Projector	Log	10
Laser Printer	Linear	6
Offset Printer	Linear	8

† Some scanners digitize 10 or even 12 bits, but generally are all linear response.

** Studio cameras use 3 chips, one for each Red, Green, and Blue light. The chips map their voltages into a logarithmic response within the camera and then it is send out as an RGB video signal. Once it becomes a video signal there are only ~7 bits left.*

Why C++ isn't very fit for GUI programming

Oleg Kiselyov
CIS, Inc & University of North Texas
303, N.Carroll, Suite 108 Denton TX 76201
oleg@ponder.csci.unt.edu, oleg@unt.edu, http://replicant.csci.unt.edu/~oleg/ftp/

With no intent of starting a holy war, this paper simply lists a few annoying C++ birthmarks that the author has come across developing GUI class libraries. The main snag appears to be that C++'s idea of objects, classes and the hierarchy of classes looks tantalizingly close to GUI's concepts of gadgets, widgets, window classes and subwindows. However, they are not quite similar: C++ was designed to be a "static" language with a lexical name scoping, static type checking and compile-time type construction/hierarchy. GUI objects, on the other hand, are inherently dynamic; they usually live well beyond the procedure/block which has created them; their hierarchy is defined to a large extent by event flow/capture and geometry/layout. Many GUI fundamentals such as parent-window-child subwindow hierarchy and event relaying (let alone memory allocation issues) are not supported in the C++ "core" (or supported as "exceptions" - pun intended). All in all, this leads to unnecessary bloating of the code, duplication of the window manager functionality, engaging in unsafe practices and foregoing of many strong C++ features (like scoping rules and compile-time type/method verifications). The paper lists a few major C++/GUI sores and illustrates them on simple examples.

Listed below is a commented list of common snags one runs into when implementing a GUI system in C++, especially a general purpose one. I hit mine while wrapping XVT's platform-independent GUI engine (still smacking of Windows) in a C++ class library. It appears that the pitfalls and workarounds (kludges) are fairly common.

1. Don't CREATE constructing (or at least, don't show it)

That is, constructors of Window, Dialog, etc. classes (and their descendants) are better not to open a window on the screen: they should leave the creation of a GUI object for a later time. Otherwise they deny any derived class an opportunity to modify appearance of the window or to do some initialization at the CREATE event. For example, suppose Basic_Window handles a plain regular black-on-white window:

```
class Basic_Window {
public:
    Basic_Window(Rect rect) { GUI_create_window(rect,visible,this); }
    virtual void handle_create_event() { set_background(BLACK); }
};
```

If I want merely to change the background, I'll derive

```
class MyWindow : public Basic_Window {
     virtual void handle_create_event() {
set_background(RED); }
public:
     MyWindow(Rect rect) :
Basic_Window(Rect rect)         {}
     ...
};
MyWindow my_window(default_rect);
```

Unfortunately, `my_window` would show up black, not red! When the `Basic_Window` "portion" of `MyWindow` is being constructed, the `Basic_Window()` constructor runs within the environment of the `Basic_Window` class. Therefore, until `Basic_Window()` finishes, it is `Basic_Window::handle_create_event()` that will be handling the CREATE event generated by `GUI_create_window()`. Virtual tables of `MyWindow` class kick in only when the base class is completely constructed, that is, when the window is already on the screen and it is too late to change its background color without flashes.

The easy work-around is to require every constructor either not to create a Window Manager object (and thus hold off the event processing until derived classes finish constructing), or, at the very least, keep the window hidden/disabled until further notice. It works, but it isn't very cool: for one thing, one always needs to remember to call a special function to show a window to the user. For another (and more important) thing, if somebody gave you some Window class and this class' constructor does go ahead and present a window, this Window class is "underivable": you can't make a derivative class that slightly and nicely modifies the appearance of a window the original class creates.

This example alone breaks a nice metaphor "window on the screen <-> window class object in memory". They are different beasts indeed: and it's possible to have a window class object without the corresponding screen window (and sometimes, the vice versa, see below).

2. Don't rely on virtual functions for relaying events

Suppose we have a `Basic_Window` class that shows some fancy picture in a window:

```
class Basic_Window {
```

```
public:
     virtual void repaint() {
draw_fancy_picture();  }
     ...
};
```

We derive `MyWindow` class to modify the basic functionality a little: draw some rubber-box-type rectangle over the picture (say, to emphasize some area of it):

```
class MyWindow : public Basic_Window {
     Rect rect;
     virtual void repaint() { draw(rect);
}
};
```

Unfortunately, the window associated with `MyWindow` object would show no fancy picture, only a rectangle in an empty window. Indeed, since `MyWindow::repaint()` overrides `Basic_Window::repaint()`, the latter function would never be called should the window need repainting. Note the fancy picture itself is a private property of the `Basic_Window`, something that `MyWindow::repaint()` can't get hold of. So, one always has to keep in mind to write `repaint()` like that:

```
class MyWindow : public Basic_Window {
     MyWindow(void) : Basic_Window() {}
     virtual void repaint() {
Basic_Window::repaint(); draw(rect); }
};
```

The constructor of `MyWindow` is spelled out here to emphasize that `repaint()` in a sense should work like a constructor: the first thing, it has to call its uncle, and then go on tinkering with its inheritance. This trick, *call the uncle first*, has to be done in every virtual function that has something to do with events, like `handle_resize()`, `handle_color_event()` etc. Doesn't it look like the standard "virtual" functionality being slightly inadequate?

Note, that during the construction of a derived object, the compiler implicitly and tacitly creates the underlying base object first (and saves us the hassle of doing it explicitly). In a sense, the compiler "relays" the object construction "event" down the hierarchy. Most of other events (expose, resize, etc.) have to be relayed in the similar way, too: alas, the compiler is of no help here, and we have to do relay all by ourselves.

3. Clashes of hierarchies

Besides a hierarchy "generic tool -> specialized instance" which C++ is based upon, GUI environments have another hierarchy of windows/subwindows (or dialogs/controls). The latter has little to do with specializing, and a lot to do with layout. Sometimes the hierarchies clash. Let's consider again the example of a picture shown in a window, and a small rectangle dragged around over the picture. One can implement this using two separate windows: picture and a rectangle. One of them has to be a parent of the other, that is, when the picture window gets hidden/closed, the rectangle has to drop out of sight too. This solution taps the smarts of the GUI window manager to figure out which window should get a repaint event. For example, if some part of the rectangle area becomes obscured and later exposed, only the rectangle window receives the repaint event. If the entire picture emerges from obscurity, both the picture window and the rectangle have to be redrawn (but the picture first).

On the other hand, one can think of a picture window with a drawn rectangle as a particular instance of just a picture window. So, we come the `MyWindow : Basic_Window` hierarchy of the previous section. Now, both the picture and the rectangle in it are represented by a single object of class `MyWindow`. `MyWindow::repaint()` receives all repaint events (method overriding works like the "exclusive or"). If `MyWindow` wants to be smart and avoid redrawing the picture when it's not necessary, it has to look into which area needs repainting, and call `Basic_Window::repaint()` only when it is absolutely needed. In short, `MyWindow::repaint()` should (re)implement what is already built into the Window Manager.

4. Destruction is tough, too

This is the famous problem of what should be destroyed first, a chicken or an egg: should a CLOSE event handler do `delete *this;`? Should the destructor call `close_window()`? Probably one needs both:

```
class SimpleWindow {
public:
     void handle_close_event(void)      {
delete *this; }
     ~SimpleWindow(void)         {
GUI_close_window(); }
};
SimpleWindow * a_window_ptr = new
SimpleWindow(...);
```

Suppose the user has clicked "OK" or "Dismiss" button, and the window is to be closed. The GUI server during the clean-up notifies the `*a_window_ptr` object that it's going to be destroyed: the server sends the object a CLOSE or TERMINATE event. This is going to be the last event sent to and handled by the object. Since the screen window is being wiped out, so should be the object: it has to delete itself.

Now let's suppose that the program decided to get rid of the object before that by doing

```
delete a_window_ptr;
```

somewhere in the code. The corresponding window on the screen should be closed, too (otherwise there would be something on the screen with all its data

and the event handler already disposed of). So one has to call the window manager and tell it about this. But this would result in a CLOSE event being sent to the object, which would delete itself one more time. Does it smell of danger here? Thus, one has to be very careful when disposing of window objects, and keep a special flag to tell if a screen window has been already closed. Again, the metaphor “a window object in memory <-> a window on the screen” is slain once again.

5. Out of scope

Static name scoping of C++ isn't very compatible with the dynamic nature of GUI objects. Creation and destruction of names (and on-stack objects) follow *static* lexical rules in C++. However, when the name of a window object goes out of scope, it more often than not means that the window itself should remain on the screen. Indeed, in the context

```
void on_show_button_hit(void)
{
     Picture picture(“file.pic”);
     PictureWindow window(picture);
}
```

one usually wants for picture to remain on the screen after the function finishes. So, one has to write

```
void on_show_button_hit(void)
{
     Picture * picture =
Picture(“file.pic”);
     PictureWindow * window =
PictureWindow(picture);
}
```

After the function finishes, the objects, `Picture` and `PictureWindow`, would be still alive; however, the pointers to them (their *names*) would be already disposed of. Once the object reference disappears, it's rather difficult to dispose of the object when it's no longer needed. Thus heap objects without names (and/or references) proliferate. The only clean way of handling this situation is making basic window classes self-threading etc., which in a sense, is a duplication of the functionality already built into the window manager (GUI server).

Of course all these snags aren't fatal. C++ is a universal and powerful language, capable of expressing every possible computational algorithm (because one can “build” a Turing machine in C++). Therefore, if an application demands dynamic features, like the ones built into Tcl/Tk, Scheme/Tk, PostScript, etc., one can always emulate/implement them in C++. But why not to use a well-designed “dynamic” language in the first place?