The Journal of the Association for Computing Machinery at UIUC: The

Banks of the Boneyard

Volume 17, Issue 2

From the Chair

by Mark Ashton

The computer industry has seen incredible growth over the last ten years or so. Computing power is applied in all branches of our economy, and the supply of qualified personnel has not kept up with the incredible demand. What does this mean for you and me? It means we get the royal treatment.

I thought that looking for a job would be an extremely high-stress situation. I knew that it wouldn't be hard to find a good job, but I was sure that recruiters would be playing hard ball. I fully expected batteries of tests, hard questions about my resume, and frank evaluations of my abilities. What I got was very different.

Companies are so desperate for qualified computer people, that they snap up our resumes at job fairs like hungry piranhna. It really was nice to be able to walk around Expo in shorts and t-shirt and still be taken seriously. Companies are trying all sorts of things to draw in geeks. I have quite a collection of free pens, t-shirts, and various other knick-knacks, thanks to job fairs. Lots of companies give out free food at info nights, and some of them even have social nights at bars, complete with free drinks.

My experience with interviews has been similarly gentle. Aside from one simple test, all my interviews have basically involved talking about whatever I want to with a nice person from the company. In fact, the majority of the time has been spent with me asking them questions about why I should work for them. I'm pretty sure in some industries, the job seekers try to convince the interviewer that they are qualified, but in computing it's the other way around; recruiters are always telling us why we should work for them.

We can reasonably expect the hot job market to continue for a few more years. There are currently around 300,000 unfilled computer jobs in the US, depending on which estimate you use. Sometime soon people will step up to fill in the void, and the market will cool down. Until then, we can sit back and enjoy the ride, and be glad that we are in so much in demand to do the things we'd be doing anyway.

Inside This Issue

Inside this special Conferemce issue of the Banks of the Bonyard, we give you SIG updates, Conference highlights, and the inside scoop on:

Java
Java 3D
3D APIs
Perl
CGI
Linux
Windows Programming
Free Software
History of Video Games
Digital Signal Processing
Rhapsody/OS X
ACM UIUC Workshops
And Oh So Much More!

So dig in and read all about it while you're waiting for the next Conference presenter. Enjoy!

3D APIs:
A Gentle Introduction

By Brian Klamik

First of all, since I'm probably addressing newcomers, an API is an Application Programming Interface. An API consists of a bunch of functions, data structures, and constants which a programmer can use to achieve something. Primarily, APIs provide a unified and abstracted way to interact with a piece of software or hardware. The sockets API simplifies network programming, so the developer doesn't have to know how to deal with TCP/IP protocols directly. Also, the sockets API eliminates the need for programmers to deal with the myriad of network cards directly, by forcing hardware vendors to provide a driver.

In the domain of 3D, there are two main types of APIs. At the highest level, there are scene graphs. Scene graphs are object-oriented APIs, meaning they are usually easy to visualize and deal with. Each object represents something many people can identify with: point light, shape, and texture. Even someone unfamiliar with computers would intuitively know that to dim the light would require changing some property on the light object in the scene. Here is an example of what typical scene-graph code looks like:

```
CSquare Square; CNode RootNode; CLight Light;
 RootNode.AddChild( Light);
 RootNode.AddChild( Square);
 Square.SetSize( 2.0);
 Light.SetColor( 1.0, 0.0, 0.0);
```

continued on page 12

The Banks of the Boneyard is a student publication of the UIUC Chapter of the ACM, the First Society in Computing, and does not necessarily represent, in whole or in part, the views of the University of Illinois, the Department of Computer Science, or the International ACM. Partially paid for by SORF and the Department of Computer Science.

Editing, Layout, and Design: Joe Doyle

The Association for Computing Machinery at UIUC
1225 Digital Computer Lab, MC-258
1304 W. Springfield Avenue
Urbana, IL 61801
(217) 333-5828

For more information, email acm@uiuc.edu
or visit our homepage at www.acm.uiuc.edu

A 5-Minute Introduction to DSP

By Dan Sachs

One of the buzzwords of modern society is "DSP" - you can hardly buy an amplifier or CD player without it having those three letters somewhere on the case. Most people know that a "DSP" is a "digital signal processor." And most think that the DSP is a mysterious black box that somehow makes sound different, using techniques that can't possibly be understood. In fact, a DSP is just a microprocessor, with certain characteristics that make it well suited for signal processing operations.

And the most important thing that differentiates a DSP chip from an ordinary microprocessor isn't esoteric. Signal processing operations don't actually require any specialized hardware to run. In fact, an ordinary PC can run perform almost any signal processing task. This is because the basic operation in signal processing is actually simple, everyday operation called a multiply-accumulate.

A multiply-accumulate operation takes two numbers, multiplies them, and adds the product to a third number. The result of the addition is then returned to the register from which the third number was taken. A string of multiply-accumulate operations, therefore, performs the sum of several product terms. In other words, a DSP is simply a chip designed to take the basic functions of a cash register (find the total price of two items at $2.99, three at $1.48, and one more at $15) and do it as quickly and efficiently as possible. This is the single most important signal processing operation because the basic filtering algorithm - the linear FIR (finite impulse response) filter - is simply repeated multiply-accumulates.

All digital signal processing algorithms act on a series of samples, separated in time. For instance, a CD contains samples taken at a rate of 44.1KHz, which means that the signal from the microphone is examined and recorded 44,100 times a second. Under certain conditions, the portion of the original sound from 0Hz to 22KHz (the range that can be heard by the human ear) can be reconstructed exactly from this sampled signal - that is, that the samples contain all of the information present in the original signal. (1)

At one point or another, you've probably played with a gadget called a "graphic equalizer" which can increase or decrease the strength of certain parts of an audio signal. And if you have a modern amplifier, it probably has a DSP which adds reverb to simulate a large concert hall. Both of these are examples of filters, and can be easily implemented using the linear FIR filter and the multiply-accumulate operation.

continued on page 13

Reflections Projections Sponsors

The 1998 *Reflections | Projections Conference* Committee would like to acknowledge the following companies for their generous donations:

Assocation for Computing Machinery (http://www.acm.org), for continuing patronage

Computer Discount Warehouse (http://www.cdw.com), for our label printer

Champaign Computer Company (http://www.c-computer.com), for discounted computer prices

Microsoft Corporation (http://www.microsoft.com), for software and operating systems

O'Reilly and Associates (http://www.ora.com), for technical reference books

Red Hat Software, Inc (http://www.redhat.com), for operating systems and t-shirts

University of Illinois Department of Computer Science (http://www.cs.uiuc.edu), for ongoing support

Official sponsors include Abbott Laboratories, Commonwealth Edison, Crowe Chizek, CSG Systems, CSI Inc, ITDS, Lante Corporation, Lucent Technologies, Motorola, PriceWaterhouseCoopers, STR, and Wolfram Research. Additional sponsors include AIS, Allstate Insurance, BDS, Clarity Consulting, Dazel, Geneer, Green Hills Software, GTE, Hewlett Packard, IntegrationWare, Lockheed Martin, Microsoft, Trilogy, and TRW.

This year's conference would not be where it is today without the help and support of these companies and organizations.

CGI, Easy As Pie: A Quick Intro to CGI.pm

By Jason R. Govig

For anyone who has created CGI applications using cgi-lib.pl or their own CGI processing routines will love the simple interface and advanced features of CGI.pm. People who have not written CGI applications may find it easy to learn. All one needs is a little knowledge of Perl, a handy reference, and a simple Perl module called CGI.pm.

CGI.pm is an object oriented approach to CGI, and the list of feature goes on and on, ranging from simple form generation and processing, to easily handling JavaScript and frames. There are two ways of creating a CGI object based on the form information sent from the browser, by object reference or by importing it to the standard namespace. For example, to create a CGI object and print a test page, one might use:

```
  use CGI;
  $query = new CGI;
  print $query->header;
  print $query->start_html('Test Page');
  print "Hi mom!\n";
  print $query->end_html;
```

or import it into the standard namespace like this:

```
  use CGI qw(:standard);
  print header;
  print start_html('Test Page');
  print "Hi mom!\n";
  print end_html;
```

Forms that only use one CGI object will benefit from the import to the standard namespace.

Form information sent to the script from the browser can be easily accessed and manipulated. Name/value pairs can be accessed and changed easily via the method param, deleted using the method delete, and appended to using append. State can then be saved by calling `$query->save(OUT);` here OUT is a filehandle, and retrieved by creating a new CGI object with a filehandle like: `$query = new CGI(IN);` This allows one to create and maintain state for users overtime without excessively using self-referencing URLs or cookies. Multiple sets of data can be written to a single file and read in again, allowing easy creation of simple guest books or other related net activities.

When creating new forms in the output html, CGI.pm handles everything. For example, when creating a text field with the following code:

```
  print $query->startform;
  print $query-textfield(name=>'Username',-
\size=>30);
  print $query->endform;
```

CGI.pm will print out the necessary form code using the name/value pairs stored in $query for the defaults. Therefore, returning to the script will present a text field with the name Username, and the value that was last entered by the user.

Those are the main highlights of CGI.pm, but there are many more features available. Most html tags are useable by method calls, adding JavaScript tags and custom html tag parameters to elements is as simple as adding an additional parameter to such methods, and environment variables (for the elite CGI application developers) are also accessible by method calls. Users of cgi-lib.pl will be relieved because of the backward compatibility and easy conversion to CGI.pm.

So grab yourself and install a copy of CGI.pm, read the relevant documentation, and see how it can work for you. Be creative, try things out, the are virtually no limits.

Additional Information:

CGI.pm documentation:
http://www-genome.wi.mit.edu/ftp/pub/software/WWW/cgi_docs.html
CGI documentation: http://hoohoo.ncsa.uiuc.edu/cgi/
Perl Institute: http://www.perl.org/

SIGWeb

By Valerie Franek

What's the deal with this "SIGWeb?" And what happened to WebMonkeys? Well, starting this semester, there's a new name for an old SIG. What was once WebMonkeys, the ACM@UIUC special interest group dedicated to all things web, is now SIGWeb. The new name more closely identifies with the international ACM's SIGLink and extends the topics to go beyond just hypertext, to include web design and graphics, as well as some web-related programming.

Plans for SIGWeb include workshops, presentations, activities on HTML, CGI programming, DHTML, JavaScript, web-safe graphics, and lots of other web technologies and topics. Of course, the group isn't limited to these, and ideas and suggestions from members are always encouraged.

If you're interested in learning more about web design, graphics, HTML, and more, come check out SIGWeb, meeting Wednesdays at 7:30pm beginning this fall. (If you'd like to share what you already know about these topics or have any questions about SIGWeb, feel free to contact me at sigweb@acm.uiuc.edu)

SigArt: The Artificial Intelligence Frontier

By Borislav Dzodzo

As soon as a few unsuspecting freshman were assimilated, our merry little clan was ready to have its most impressive year to date. The beginners and enthusiasts are being taught the basics of adaptive systems that lie at the core of artificial intelligence. Meanwhile the old timers are returning in anticipation of future adventures and spreading propaganda on campus. The previous year has proven that challenges of robotics can be overcome and mastered. With this in mind we plan for future robotics projects. So far the newcomers have been enlightened on the topics of neural networks, and genetic algorithms. But alas, all this would be worthless without a plan for a bright and brave future.

First, you can come to our meetings at 7:00 p.m., or you can contact sigart@uiuc.edu for details. Secondly, those rumors of our own, of an improved and evil robotic arm, will slowly become a reality. With the knowledge and faith in AI, and basic intuition of mechanics we will search for the best configuration of the arm and software. As our intense last year research pointed out, however, reality sucks. This is why it is imperative that we start our EOH project early this semester and design algorithms and machine robust enough to withstand the hardships of existence within this cruel unforgiving world.

SIG-BIO

By Mary Lee

The Special Interest Group for Biocomputing focuses primarily on the applications of computers to the biological sciences. Its name is supposed to be Biomedical Computing, which would emphasize medical informatics. In order to fulfill this, Andrew Dalke visited to give an informal talk on bioinformatics. There are also plans to visit biophysics or bioengineering labs around campus.

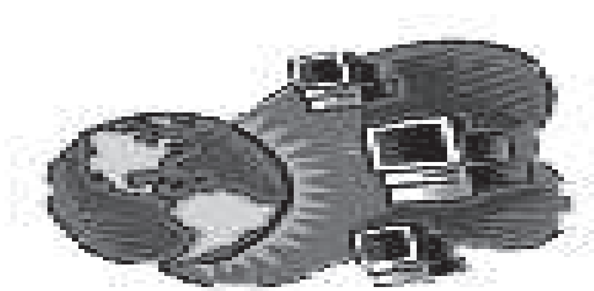
Our past projects reflect the varied interests of our members. The Virtual Anatomy Textbook expands every year and requests are made to continually update it from all parts of the world. It presently covers the lymphatic, nervous and digestive systems. The circulatory system most likely will be next.

Cyber Cell is going to be revitalized from its admirable first implementation in cgi and html into Java or C++. It will be a *Euglena* by itself with emphasis on molecular mechanisms or a world of Euglenas that interact to fight for food at a macromolecular model. Work on this project has been delegated into the user interface in which various foods will be induced into the environment.

The most exciting project this year possibly for EOH (Engineering Open House has been work in facial recognition and various methods that can be used to implement it. Frame captures from a video camera 1600x1200 should be available to take pictures at a resolution high enough to do proper calculations and analysis. For the overly ambitious, it might be developed on both the windows and linux platforms. This program will be compliant with the Human Authentification API written by the Department of Defense.

If you'd like to help on any of these projects or you'd like to find out more information, come to our informal meetings at 6:30pm on Tuesdays.

`http://www.acm.uiuc.edu/sigbio`
`mailto: sigbio@uiuc.edu`



SigNet

by Jason govig

SigNet is the Special Interest Group for Networking and Security. We do anything network related, from writing network code to networking personal computers. Currently, we are having several tutorials on networking programming and related topics, including a CVS tutorial for those interested in developing this year.

Our main project, Voodoo City, will now have a separate meeting time, immediately following the usual SigNet meetings. We will be able to discuss more details about the site and write more code easily with a separate meeting time. If you want to write networked games in Java, create dynamic pages in CGI, or design web pages and graphics, then this is the place for you.

So come along and join the fun! No experience necessary! SigNet meets every Monday in 1102 DCL at 7:30pm and Voodoo City will meet in the same location at 8:30pm.

www: http://www.acm.uiuc.edu/signet
email: signet@uiuc.edu

SIG UNIX

By Tony Sintes

I'm pleased to say that SigUNIX is off to a strong start this semester.

Currently, we're working on a generic Xload like application for Unix. Like Xload, our application will graph a resource over time. However, unlike Xload, this application will allow plugins. These plugins will allow the application to track any kind of resource. This resource can be anything from a file to an entire network of computers. Since we want to keep this application as general as possible, we're even considering allowing plugins for the communication between the resource plugin and the GUI front end.

This academic year, SigUNIX plans to host a number of workshops. Currently, we plan to host a UNIX security workshop, a GUI programming workshop, and a ZSH workshop. Dates and times are forthcoming.

Finally, SigUNIX will present a project during EOH. We're still working out the details but it looks like we'll be adding a security layer to the CORBA implementation that is currently used by the GNOME project. Other possibilities include a port of SSH, a secure TCP/IP stack, or the ever popular game.

As always, SigUNIX can only be as strong as its membership. So, whether you're a seasoned UNIX GURU or someone who's just trying to figure out what UNIX is all about, stop on by. We're always looking for new members.

SigUNIX meets on Thursdays at 8pm outside the ACM office.

`http:/www.acm.uiuc.edusigunix`

SIGArch

By Jason Gallichio

At the first "real" meeting of the Special Interest Group for Computer Architecture, members built what lies at the heart of nearly every computational giant -- an adder. For the new members, we breezed through the design of a full adder, and each person built their own with discrete gates. At the end, all the adders were hooked up together to reveal a massive four-bit adder rivaling even those of our illustrious forefathers.

Considering a four-bit adder was an Engineering Open House project of years past (albiet a more complicated and graphical version), hopes for the future of this unusual "hardware based" SIG look bright. Not only does there seem to be great interest from inexperienced members, but there were several members with more digital design experience expressing interest in reviving projects such as the scrolling sign, the automatic pop machine, and MPEG-Man, along with proposing new ideas of their own. This interest should propel our SIG back into having some of the "coolest" projects around. Great job guys. We're all looking forward to a great year.

`http://www.acm.uiuc.edu/sigarch`

email: sigarch@uiuc.edu

Workshops@ACM

By Steve Mycynek

Starting soon, ACM will be hosting several promotional workshops aimed at the general student population.

The opportunity to use computing knowledge to reach out the UIUC community is one motivational factor for these workshops. The other is to increase student awareness of ACM and its resources.

These workshops will be highly informative, fun, interactive, and will relate to the needs of everyone--not just engineers. Put another way, our goal is to give attendees a healthy does of information that is neither too technical nor too watered down and (gasp!) maybe even have a good time doing it.

This coming October, (exact dates to be announced), look for ACM's famous Linux Install workshop--organized by the Linux User's Group. Students tired of dealing with Microsoft can get a change to learn about alternatives to Windows from the ground up.

Soon after that, come and see our "Buying a PC" workshop, sponsored by our very own SigDAVE. Don't walk into your local electronics store blindly assuming bigger and more expensive is always better. Learn what you need, do not need, and why, and end up with a PC that's right for you.

Also, ACM's SigWeb will be hosting a "Building Better Web Pages" workshop near the end of the semester. If your home page is difficult to read, navigate through, doesn't live up to what you had envisioned, or just doesn't work properly, come see SigWeb for helpful pointers some time in early December.

MacWarriors

by Rick Roe

Greetings! MacWarriors is a special interest group within ACM dedicated to doing cool stuff with all things Macintosh--everything from tips and tricks for everyday MacOS users to programming with the latest Mac-related technologies to playing with PowerPC hardware. We've got the year off to a good start, and some cool projects and events coming up.

ACM's new Power Macintosh G3 just arrived last week, and will soon become the primary workstation for our upcoming development and publishing efforts. It's currently down in L510, but we'll be moving it up to the office as soon as we find a hub for it... and maybe clear off some desk space. :)

Well, the bad news is that our Apple reps had to cancel their appearance at the last general meeting, as they were busy with the iMac rollout and didn't have enough exciting things to talk about nor the time to comfortably get to another university afterwards. But the good new is that we'll instead be hosting the first of many Apple Tech Updates following this month's ACM general meeting on October 1st. Come to 1320 DCL at 8pm and check out the cool new iMac, the blazing-fast PowerBook G3 Series, and presentations on Apple's latest software offerings--the award-winning AppleShare IP 6.0 and WebObjects 4.0 server software, Mac OS 8.5 (the must-have update to Mac OS 8), and the next-generation Mac OS X operating system. Word has it there'll be a few giveaways as well.

The next item on our agenda is a series of workshops on how to make the most of the Mac OS's powerful AppleScript technology. It's been part of the OS since system 7.1, but most users still don't recognize its full abilities to automate and speed up routine tasks, easily configure systems and applications, and create custom solutions not otherwise possible using multiple programs. Come join us and see everything from the basics of the language to writing sophisticated programs like IRC bots, WWW CGI scripts, and lab administration tools. We'll also show off the useful new features and performance improvements of AppleScript 1.3 (part of the MacOS 8.5 upgrade due this month).

We're still considering ideas for a year-long project to show for the spring 1999 Engineering Open House; possible projects thus far suggested include a more full-featured replacement BroadCast (an AppleTalk communication tool), extending last year's QuickTime VR project (http://www.acm.uiuc.edu/macwarriors/eoh98/), working with the open-source Mozilla project, and various network-security implementations (ssh, scp, kerberos, etc.). We're also considering a gaming tournament or two and more programming workshops.

Feel free to come express your views, learn cool new stuff, and have fun in the campus Macintosh community with us! MacWarriors meets every Saturday at 3 pm in the ACM office, 1225 DCL.

http://www.acm.uiuc.edu/macwarriors/

Rhapsody and OSX Unveiled

By Steve Bytnar

Apple is heading toward the future with their latest development efforts.MacOS X (said MacOS 10) is Apple's response to demands that they produce a modern OS. Basically, Apple removed a large part of the MacOS APIs that were not viable for a modern operating system, and have defined the "Carbon" APIs which take advantage of preemptive multi-tasking to increase performance, protected memory for more reliability, and faster virtual memory through the use of the Mach microkernel.

Rhapsody v1.0, which has been renamed to MacOS X Server, repositions Apple in the OS market with a direction that will allow them to sell differentiated "workstation" and "server" operating systems. The name reflects that Apple's future OS strategy is centered around MacOS X.Rhapsody v1.0 is due to be released later this year, possibly late September or October. Rhapsody developer releases (DR) 1 and DR2 have both been Intel hardware compatible. However, Apple has declared that future versions of MacOS X will only run on Apple hardware, so do not expect an Intel version to be released, or even supported.

What makes MacOS X different from Rhapsody, or for that matter, MacOS 8.x of today? Apple is creating both new operating systems so that they run under the Mach microkernel. Mach is an industry known kernel which is very stable. Apple aquired NeXT Corporation in December of 1996, and the microkernel approach has stemmed from this. NeXT's product was NeXTStep and OpenStep which are commercial quality UNIX operating systems based the Berkeley Standard Distribution (BSD) v4.4. The most interesting aspect is that Apple's original plans were to have developers completely rewrite their software in order to make

continued on page

Windevils

By Ibrahim Merchant

Well, WinDevils is off to a good start. A lot of people are going to the Windows Programming Workshops and learning how to code in Windows 95. The workshops are at 11:00 AM on Saturday or Sunday. You can attend any one of these days every week -- they cover the same material (hope this is enough of a motivation to keep you sober the night before). If it isn't, you might be interested in something related to sobriety -- our EOH project. We are working on a 3-D drunk driving simulator. For those of you who want to do more than "research" for the game, you are welcome to join the project. In fact, we are working on an algorithm that compresses the feeling of being drunk right into the game... it is called M-Keg (just kidding).

Well, for those of you who do not want to do the simulator because it brings back painful memories of last Saturday night, WinDevils is also entering the National ACM Windows CE contest. We are going to create an application for the Windows CE platform. If you have any ideas or are interested, just send me an e-mail: imerchan@uiuc.edu. (Maybe we will create an application that goes with the 3-D drunk driving simulator -- an application and special device that can allow Windows CE to administer a breathalyzer test!) Windows CE would be able to allow you to play solitaire and perform sobriety tests -- a cops dream.

Well, thanks for all your time and if you have any comments or are interested in some of our projects -- e-mail to imerchan@uiuc.edu.

Oh yeah, I almost forgot. I want to send out a word to the Thrustmaster company for donating the "Steering Wheel" joystick for the 3-D Drunk Driving Simulator. Thanks again!

`http://www.acm.uiuc.edu/windevils`

SIGOPS

By Dave Wentzlaff

SigOps is the Operating Systems research and development branch of ACM. This year, we endevour once again into the roll your own OS workshop. In this workshop, you get a chance to develop your own 32 bit protected mode operating system from scratch. This year we are branching out our development into the Sparc architecture while in the past we focused on the Intel architecture, and this continues to be our main target. After the workshop ends we will start working on a killer EOH project that will include clustering and a common object system across operating systems designed by members. Also, we will be rewriting some of the tutorial that goes along with the workshop for a more organized reference.

So if you are interested in playing with operating systems, operating systems research, or other systems level programing, come to a meeting. Meetings are at 7PM on Tuesdays, so come check us out. You can also look at us on the web at www.acm.uiuc.edu/sigops. Hope to see you there.

`http://www.acm.uiuc.edu/sigops`

BUG@UIUC

By Vikram Kulkarni

The BeOS is an unique operating system. Unlike Windows9x/NT, MacOS, or even any flavor of UNIX (or Linux), the BeOS is not bogged down by legacy requirements. It is truly a fresh start. Be programmers didn't start with an old operating system and try and build upon it. They started with the idea of a MediaOS, an operating system that was designed from the ground up to cater to the needs of the power user. That is, anyone who finds something lacking the current crop of operating systems.

With this idea in mind, the Be programmers created the BeOS. The BeOS is a fully 'buzzword compliant' operating system. (skip the rest of the paragraph if you aren't interested in buzzwords) The BeOS is a microkernel based operating system that supports symmetric multiprocessing, preemptive multitasking, virtual memory, and protected memory. It has a modular, dynamically loaded, multithreaded I/O system and is pervasively multithreaded. The Be file system, BeFS, is a 64-bit, multithreaded, journaling file system with integrated attributes, indexing, and MIME-type based identification. The BeOS also has external file system support. The BeOS sports antialiased fonts, OpenGL, messaging, scripting, replicator services, and integrated Unicode font support. It is POSIX compliant and has a native bash shell. It runs on PowerPC and Pentium processors.

The Be Users Group (BUG) is dedicated to the support of BeOS users and developers. We meet every Wednesday evening at 7pm in front of 1225 DCL (the ACM office). This year we will be writing an application that uses the GPS system. As always, we are open to other ideas. We will also be giving development workshops. These workshops are a quick introduction to the BeOS API. If you've never programmed for the BeOS, these workshops will get you up a coding quickly.

`http://www.acm.uiuc.edu/bug`

SigGraph

By Brian Klamik

SigGraph is currently running our beginner tutorials and workshops to bring our members up to speed on graphics programming. We've also started our EOH project early this year. The project is a networked 3D game, of course. The details, so far, are that the players will pilot hovercrafts of some kind. The physics engine should be completed by the time this article is printed. It will be able to calculate thruster-surface interactions, collision detection, and collision forces. The physics engine and rendering engine will both use Oriented Bounding Box Trees to calculate thruster forces and efficiently render the map/ ground. Further plans probably will include usage of force-feedback input devices, 3D sound, and team player aspects. This project is meant to be an educational experience for all those involved. We will meet and discuss architecture and issues, so that everyone comes away with a good idea of how to build their own game.

We also are coordinating with SigMusic to start Sounds & Visions early. Sounds & Visions is a concert where SigMusic members creates the music and SigGraph creates graphics. The usual method of creation is that SigMusic hands off a tune to a SigGraph member who then creates a program, which might react to the music, or an animation. However, it is possible for a SigGraph member to give an animation to a SigMusic member to score. There really are no rules, except that it has to be in a showable form on the night of the concert.

SigSoft

By Erik Gilling

Sigsoft is getting off to a good start this year. We have several new members who are eager to learn and make great things. We have decided to spend the first several weeks on workshops to get all of our members up to speed. After that we plan on an EOH project.

Our workshops will include C++ for beginners, C++: what they did not teach you in CS 223, The C++ Standard Template Library (STL), Network sockets programming, Multithread programming, and CVS. We will be done with the C++ for beginners by the time of this publication. If you would like to attend any of these workshops feel free to stop by our weekly meeting at 6:00 on Wednesdays. More detailed information on dates of the workshops will be at our webpage (http://www.acm.uiuc.edu/sigsoft) when it becomes available.

MechMania

The logo has been enlarged,
The teams are registered,
The code is in order,
The API is being printed,
And the sponsor is happy.

Now what am I talking about? MechMania of course, ACMs annual coding competition. No, I don't want you to grab Stroustrup's C++ book and run into a lab. There is another part of the competition that needs you to join in. It only requires one skill, the ability to stare forward. Yes, on October 4th 1320 DCL will be filled with screaming Coders who are trying to rip each other's heads off. Even though this in itself wouldn't be a bad performance, there will also be the **Ultimate C++ Competition** going on in the same room.

We have locked 16 teams of Coders, from all over the United States, in a lab for 17 hours and given them an API (Advanced Programmers Interface) to a game they knew nothing about until they arrived in the lab 3 days prior Even though this may be a sick form of mental torture, they also will be laboring away to produce a Artificial Intelligence program to play in MechMania and try to smash the competition using any means possible.

You are all welcome to view as much of the screaming, the shouting, and applause that you can stand for FREE. All you need to do is come on Sunday October 4, from 3:00 to 4:30 PM in 1320 DCL, and enjoy the show.
http://www.acm.uiuc.edu/mechmania

SIGDave

By Nick Michels

SigDave welcomes students who feel like hanging out, talking about new things in computers, showing demos, and just generally fooling around 'til you start to annoy people. What does SigDave focus on? Nothing. Absolutely nothing. And that is what has allowed us to come up with our project ideas so far:

- Coin Operated MP3 Player
- Instructional VRML Site
- Playing With Linux
- Getting TTY-Quake To Work
- Roving Ping Pong Ball Shooters
- And Much Much More.

If you want to expand on any of these ideas, or just like to pick up great things for a quote page, feel free to stop by in 1225 on Wednesdays at 8pm.
`sigdave@acm.uiuc.edu`
`http://www.acm.uiuc.edu/sigdave`

JAVA 3D:

An Improved Graphics API

By Ray Kaplan

Java3D is the new graphics API from Sun Microsystems using the Java programming language. Since it is a Java API, it allows for platform independent 3D graphics, especially over the internet.

Java3D is not a slow graphics API. It uses a platform specific graphics package such as OpenGL or Direct3D to access the hardware. This means that all of the processing is not done in Java but in the machine code for the computer that it is running on. It also means that Java3D can automatically take advantage of any graphics hardware that accelerates Direct3D or OpenGL.

One of the main benefits of Java3D is that is does not require a special browser to run. This is unlike VRML (Virtual Reality Modeling Language) or other popular formats for 3D graphics on the web. In many cases there are not browsers on many platforms for these formats.

Java3D does work with several 3D graphics file formats, and it works very well with VRML. This is because Java3D has a scene graph where the top node defines a virtual universe, and lower nodes define locales, down to geometries and attributes. Those of you familiar with VRML will find that this structure is very similar to VRML -- leading Sun to create a Java3D VRML loader.

The coordinate system of Java3D uses three signed 256 bit floating point numbers with the decimal point separating two 128 bit numbers. Because of this, Java3D can handle a size of something as large as the universe or as small as a quark.

There are three rendering modes in Java3D, immediate mode, retained mode, and compiled-retained mode. Immediate mode requires the programmer to specify the exact primitives to be used for drawing, and does not allow for optimizations by API. Retained mode requires the application to create a scene graph and specify which objects in the graph will change during rendering. Compiled-retained mode has all of the same requirements as retained mode, but allows for the application to compile parts or all of the graph into an internal format for faster rendering. Compiled-retained mode is the fastest of all the modes.

With Java3D Sun has created a platform independent graphics API that can take advantage of the platform specific graphics capabilities. More on Java3D can be found at Sun's Java3D site as well as the current beta for Java3D.

What Is Java:

An Instrospection

By Joel Jones

For many of you reading this, Java is either a hot tasty beverage or possibly an obscure island on the other side of the world. But if you are in the know, you might also know that Java is a programming language. Since the beginning of what are known as high-level computer languages in the 1950s, there have been literally thousands of different languages and variants. What makes Java different? We could proceed with a list of different language features, but there is little new there if each feature is taken individually. What makes Java unique as a language is the well-engineered combination of features that make for a language which has relatively straight-forward syntax and semantics. The most distinguishing feature of Java though, is the rapid adoption of the language for many different applications. We will see that this rapid adoption is partially due to some of the more distinctive features of the language.

Java is an object-oriented language, much more "pure" than another very popular language, C++. By pure, we mean that there are fewer elements for structuring data, such as the "structs" or "unions" of C++. Java has garbage collection, which is a fancy way of saying the system keep track of any memory you have allocated and reuses it when you don't need it any more. Java also has an ever burgeoning collection of class libraries dealing with everything from collections and date calculations to sophisticated encryption enabled electronic commerce. But the most important aspect of Java's popularity is the particular implementation technology that allows Java programs to be distributed across the network to any computer, regardless of CPU. This technique is called a virtual machine, which has several different meanings in computer science. In this context, a virtual machine is an interpreter for a predefined set of primitive operations which is meant to be implementable on more than one combination of CPU and operating system. By embedding this virtual machine in different contexts, notably web browsers, Sun (the inventors of Java) was able to infiltrate into many areas that would have been very difficult for them to in any other way.

What does this mean for the developing computer scientist? One lesson is that commercial success is dependent upon more than superior technical characteristics. But on a deeper note, Java embodies many of the best ideas of language design of the past thirty years, and deserves a careful look.

A Quick Introduction to PERL

By Romesh Kumbhani

For the past eleven years, I have been bombarded with programming languages I had to learn. Yet for all I have gained, there has only been one language that I truly loved: that, my friend, is PERL. While usually relegated to the bottom of people's lists in terms of a programming language, PERL is one of the most powerful, pragmatic and coolest languages around.

Now, while finding someone who has not heard of PERL is difficult, there nonetheless are those that have not. So for those, listen up. PERL is script-based, interpreted language. It stands for Practical Extraction and Report Language, though for the last few years it's grown far beyond that. What once was an overrated language based on Awk, Sed, Grep and some other basic UNIX utilities, is now a powerful object-oriented language without limitation. However, I might be a little biased.

For the most part, PERL is for the lazy. By this I don't mean it requires little work or planning. Rather it's for those that hate to consistently compile code, especially if you're only tweaking a program. Basically all you need to do is edit source and run your program; what could be simpler? Also since PERL isn't compiled to machine code, it offers true portability between operating systems as well as architectures.

Another asset is what's referred to as "loose-typing" and I don't mean in the sexual sense. Unlike other languages (e.g. C, Java, etc..), in PERL a variable is a variable is a variable. If you want a variable to stand for a number just start using it as a number. If you want to make it a string, use it as a string. It's as simple as that. Sure this might lead to confusion and errors if you started using a string variable (e.g, "BigBadJoe") as a number, but then again you're smart enough not to make that mistake, right? The essential PERL credo is that a program is only as smart as the person that programmed it. Then again, you could also strictly define variables if you wanted (but that would be work, and who has time for that?).

In addition, PERL also offers simple and fast text manipulation. This is primarily why most people use it. Often it's used when designing programs that operate primarily in text-based environment (e.g. WWW, Telnet, IRC, shell scripts, etc.). The main reason for this is that PERL offers a level of text manipulation to which most languages cannit compare. It's been rumored that an entire web server was written in PERL in three lines. While I've yet to see this mythical beast, I'm sure it could be done. In addition, PERL allows for things such as reading in entire documents and assigning each line to an element of an array with only one command. One could easily see why this would be useful in CGI environments. If you really want to see the power of PERL, check out the obfuscated PERL programming contest at http://www.tpj.com/tpj/contest.

Now I guess it wouldn't be fair to list off all these benefits without telling you some of the drawbacks. Well, there aren't any! OK, maybe there is one. Speed. Because PERL is an interpreted language, it's main flaw is that in order for the program to run, the system you are using has to first load the PERL Interpreter that converts your script to machine code. Fortunately the actually "latency" time on most modern computers is so small it's not worth worrying about.

Now before I go, I might as well leave you with a little smackling of some PERL to show it's power. Don't worry if you can't read it, most people can't. It was designed to save space.

```
#!<insert path to Perl5 here>,
# eg, “!/usr/local/bin/perl",
# though omit the quotes>
# the following should be written all on
# one line without break:

for(;print"\e[2J";$d--) {for$i(0..31)
{for(0..7){$|=1;$c=cos$d/($P=4*atan2(1,1));
$s=sin$d/
$P;(1,2,4,8,16,32,64,128)[$_]&(63,127,63,1,65,1,65,1,65,1,65,1,65,1,65,
1,63,31,63,1,1,1,65,1,1,1,65,1,1,127,65,127)[$i]&&print"\e[",25+int$c*($Y=-
4+int$i/4)-$s*($X=($i%4-2)*9+$_), ";",40+
int$s*$Y+$c*$X,"H*"}}select('','','',.1)}
```

Make sure you run this on a curses-based terminal (students cluster is fine), and that your window is at least 80x40. Standard 80x24 will not be large enough. Remember, that entire thing is on one line. Have fun.

Teach Yourself Windows 95 Programming in 21 Seconds:

1) Go to www.acm.uiuc.edu/windevils
2) Click on "Workshops"
3) Go to the folder labeled "Windows Programming Basics"
4) Go to the "First Program" link
5) Click on "Your First Program" link
6) Copy and paste the code in to a Windows C++ compiler.
7) Compile and run it!

There you go! Windows programming in 21 seconds.

If you think that learning Windows Programming in 21 seconds is a little too fast or incomplete (like most other "Teach yourself in 21 days" books), come to the Windows Programming Workshops sponsored by WinDevils. We will teach you how to program Windows the right way! The workshops are on Saturday and Sunday at 11:00 AM in 1225 DCL. Both workshops each week cover the same material -- so if you miss Saturday, just come to the Sunday session.

Hope to see you there!

3D API continued from page 3

Many people would be able to figure out what the screen would look like after only reading the previous code. However, with the ease of use comes a penalty: speed. Scene graph APIs are notoriously slow and not too flexible. This doesn't have to be the case, but it is pretty difficult to build a fast scene graph API. The current project, Fahrenheit (www.sgi.com/fahrenheit), aims to eliminate this speed problem with it's scene graph API. Other APIs that fall under this category are Sun's Java 3D, Microsoft's Direct3D Retained Mode, and Open Inventor. VRML would also fall under the category of a scene graph, but it is just a file format, not an API. When comparing scene graph APIs, one must compare the different object models. Each API specifies a certain group of objects which encapsulate the API's functionality. Listing all the objects and properties of each object would take up way too much space here. Even though scene graphs represent the future, they cannot address the speed and flexibility issues which game developers require.

The second category of 3D APIs is referred to as an immediate mode API. The most popular three immediate mode APIs are OpenGL, Direct3D, and Glide. The oldest of the three is OpenGL. It was adopted from the GL API, developed by SGI for accessing the graphics features of it's workstations. OpenGL is a C/C++ API that exists on many different platforms, which makes it a prime candidate for cross-platform development. OpenGL is a very easy API to work with. To draw a 3D triangle only needs:

```
glBegin( GL_TRIANGLES);
glVertex3f( 1.0, 0.0, 0.0);
glVertex3f( 0.0, 0.0, 0.0);
glVertex3f( 0.0, 1.0, 0.0);
glEnd();
```

And many features are just simply turned on with a call to glEnable() or another "turn on" type function. OpenGL also has a retained mode feature called display lists which increase the speed of drawing. If the previous code didn't run fast enough, and it was going to be called multiple times without changing the parameters, you could call **glNewList(x, GL_COMPILE)**; before the code and glEndList(); after the code. Then, whenever you want to draw that triangle, just call glCallList(x); instead. This ease of use is the primary reason for the popularity of this API, and why it will probably be the first API that I'll teach. OpenGL is also extendable and full-featured. However, OpenGL is full of catches, which is primarily the reason why your 3D game card maker doesn't supply an OpenGL driver for it.I'm not talking about the Quake Mini GL driver, I'm talking about a full OpenGL driver. OpenGL support is something that usually costs someone $1000 to $3000 or more. And even then, the OpenGL driver you paid for always has some glitches.

Direct3D, the API by Microsoft, has had a rocky beginning. However, even with many people vocally against it, there are now many games and cards which support it. On the Wintel PC, it is the most used and supported API. First, Direct3D is built as a direct hardware API. Where as OpenGL is nicely abstracted, Direct3D is very low level and defines hardware data structures known as execute buffers. Microsoft, who previously had to deal with tons of broken video drivers in the 2D era, built Direct3D this way on purpose, so it would be easier to test. Direct3D is a strict API for this reason, and probably will never be extendable. In OpenGL, if you call a feature which isn't supported by the hardware, it is supposed to be supported in software. That means you could be programming a game which was fast, then enable one more thing and the game would be dead slow. Also, you wouldn't know which cards support the feature and which don't. Microsoft's solution to this problem was using capability bits. By calling a function, you figure out what features the hardware supports. This solution is oddly controversial to some people.

Direct3D's immediate mode API is not easy to use. Most people complained about the complicated usage of execute buffers, and got a response from MS a few versions later: DrawPrimitive. DrawPrimitive is another layer in the Direct3D API which allows a triangle to be drawn just like using OpenGL. Now Direct3D supports low-level, mid-level, and a scene graph level API. However, setup is still complicated and sifting through the multitude of versions can be a pain. The nice thing about Direct3D is that it does target all the consumer-level 3D cards out there. I'll probably teach this API if SigGraph does a game type project, so members can get usable industry experience.

Glide is a proprietary API put out by 3Dfx for programming to their Voodoo cards only. As quoted from the documentation, "Glide is not a full featured graphics API... Glide specifically implements only those operations that are natively supported by the Voodoo Graphics hardware." Face it, if you want to produce a commercial title, Glide isn't going to be your primary API. However, if you own a 3Dfx Voodoo and want to experiment and max out the Voodoo's capabilities, Glide hits the spot. Glide has also had a troubled beginning especially after the Rush chip was released. It's primary goal is speed and Voodoo functionality, and has sacrificed backwards compatibility in the past to achieve this goal. However, while MS worked hard to supposedly lower Direct3D's average function overhead from 4000 clock cycles to 2500, Glide has supposedly always been around 7. However, Glide is not threadsafe, backwards compatible, and isn't reentrant (meaning the developer has to worry about this before calling the Glide functions). Again, Glide is not a full featured graphics API. Something like texture memory management

continued on next page

DSP continued from page 3

Just like an audio signal, a filter can be expressed as a sequence of samples in time. (2) This sequence of samples is called the "impulse response" of the filter (3); we will call it filter[k]. We will call the sequence of samples that represents the signal to be filtered - in other words, the numbers stored on the CD - "input[n]," and the output of the filter "output[n]". n and k are simply index variables; k is the index into the filter coefficients, and n is the number of the input sample currently being processed.

Using these definitions, the FIR filter - the key to digital implementations of graphic equalizers, high-pass filters, low-pass filters, and echo - can be expressed as one sum:

output[n] = sigma(k, filter[k] input[n-k])

For FIR filters, filter[k] is zero outside of a finite range; therefore, it is only necessary for sum over the range where filter[k] is nonzero. This sum is taken for each sample in the input sequence by repeating the sum for every value of n where the input samples exist. We usually also assume that filter[k] is zero whenever k is less than zero. This means that the index "n-k" into the input samples is never negative, and we don't have to worry about trying to access an input sample that has not yet arrived.

This equation tells us that to do FIR filtering, we must only take each sample "n" in the input signal, then multiply filter[0] by input[n], filter[1] by input[n-1] (the previous sample), filter[2] by input[n-2], and so on, until we run out of numbers in the sequence filter[k]. The sum of all of these multiplications is the output of the filter that corresponds to input sample input[n].

In other words, to perform digital filtering, we simply execute a bunch of multiply-accumulates.

It's not hard to see how this type of filter can be used to add echo to a digital signal. If filter[k] is a 1, followed by a bunch of zeros, followed by a 0.25, we'd hear the original signal followed by a quieter, delayed copy. For reverb, we'd use more nonzero samples, to simulate the sound bouncing around a large room: a 1, followed by zeros, then 0.25, more zeros, 0.0625, more zeros, followed by a few more echos before the reverb dies off completely. The sum above will add together the original signal and all of the delayed replicas, all based on the impulse response filter[k] of the filter.

For a graphic equalizer, we use another sequence of filter samples filter[k]. This sequence varies as you move the controls, and reflects the emphasis that the controls place on various frequency bands. (If you graph at the strength of the frequency bands in the filter sequence, it will follow a curve through the points set by the controls.) The same theory that shows one can sample an analog signal and then reconstruct it also shows that the summation operation shown above can be used for filtering.

For example, if we used the sequence filter[k] of where filter[0]=0.5, filter[1]=0.5, and filter[k] for all other values of k was 0, the output would always be the average of the current and last input sample. This filter emphasizes low frequencies and cuts out high frequencies. As an example, here is the filter applied to a small section in the middle of two infinite sequences, one low frequency sequence and one high frequency sequence:

input[n]=(..., 1, 1, 1, 1, ...) the lowest possible frequency: 0Hz
output[n]=(..., 1, 1, 1, 1, ...) no attenuation

input[n]=(..., 1, -1, 1, -1, ...) the highest possible frequency: 22.05KHz (4)
output[n]=(..., 0, 0, 0, 0, ...) the signal disappears

Note that this filter serves as a low-pass filter, preserving the low frequency signal and removing the high-frequency signal. And if this filter is applied to a more complex input, the same effects will be seen; the high frequency portions of the signal will be attenuated, and the low frequency portions will be preserved. Using this technique, almost any type of filter can be made.

There are many other types of signal processing techniques that are also used - Fourier transforms, IIR (infinite impulse response) filters, wavelet transforms, filter banks, adaptive filters, and an endless list of other operations. The mathematics behind many of these techniques are much more complex than the basic FIR filtering operation shown above. However, all of these operations are built up from the same basic multiply-accumulate operation upon which the FIR filter is built. Therefore, they run efficiently on the same DSP chips that run the FIR filtering algorithms - not obscure, incomprehensible pieces of silicon that implement esoteric operations.

continued on next page

3D API continued from previous page

is absent and lighting must be done by hand, as the API has no light sources. Glide does allow you to go beyond the functionality of other APIs, all at blistering speeds. Just remember, Voodoo only.

Want to learn more? Come to a SigGraph meeting. Thursdays at 7:00 PM, we meet in front of the ACM office. I'm also interested in appointing an artistic coordinator. So, if you know Photoshop or animate regularly, join up!

`http://www.acm.uiuc.edu/siggraph`

Parallel Game Programming, Part I

By Jason Govig

Developing efficient algorithms to solve many popular games is a difficult and expensive task. Many algorithms and strategies have been developed to reduce the cost of game search trees, such as A*, Minimax, and Alpha-Beta Pruning, but in many games, these are still very computationally expensive. One solution is to reduce the amount of computation one computer has to do by using many computers or processors to help solve the problem simultaneously.

Background

Many solutions to problems are approached by a method referred to as divide and conquer. These problems can be broken down into smaller sub-problems that can individually be solved and later combined to assemble a complete solution. Example: To solve the number of prime numbers in a given range, one can divide the range into smaller ranges and solve each range individually, combining the total result after each range is finished. By introducing the concept of parallel processing, once can solve each sub-problem on a separate processor, whether it is an individual computer on a network or a multi-processor super-computer.

Game Search Trees

Generating trees of all possible moves and searching for a sequence of moves that would eventually lead to a goal state is a similar type of problem. Instead of breaking the problem down into a range of values, we can divide the problem into nodes of a tree. Each processor can be assigned to handle a node, and then assign more processors to handle its successors. With this method, all paths can be traversed simultaneously rather than one at a time as depth-first or breadth-first search does on a single computer.

The field of Artificial Intelligence allows us to determine at a faster rate and with a higher accuracy what paths to take and how much those paths cost. Creating path cost functions help determine how much cost is involved in reaching a node. In the case of chess, the cost of a move might be the number of pieces the computer captured minus the number of pieces the player captured, and each piece may have its own value based on its importance. By searching through a tree, we are looking into the future and guessing what might happen, but we may still want to look farther than limits allow. This can be done using heuristics, an estimate of the cost to reach a goal state. If we are able to expand a complete tree to the finish of game, then this would not be necessary, but in many advanced games, the search space is so large that expanding the full tree would take too long or occupy too much memory. By using heuristics, we can reasonably limit the depth of the search and still obtain a decent move without running out of time or memory.

How To Do It

One popular game search tree algorithm is Minimax. Max nodes represent the computer and min nodes represent its opponent. First, max generates all of its successor nodes, and then min generates all of its successors, which are again max nodes. After a goal state or limit is reached, the costs are evaluated and returned. Min nodes return the minimum value of all its successors and max nodes return the maximum value, assuming higher costs are good for max and bad for min. When the root of the tree is reached, the move is returned with the maximum value. Note that this algorithm assumes that your opponent will make the best possible decision, maximizing its decision at each level.

A similar algorithm was developed that always returns the negative value of the cost to the parent, and parent nodes always chooses the maximum value to return. This is called Negamax, and it simplifies the Minimax algorithm by requiring only one function to do the work instead of two, min

continued on next page

DSP continued from previous page

(1) It turns out that these conditions never actually apply in the real world. The 16-bit samples means that very quiet parts of the sound will be lost in random noise, and it is impossible to design sample recording or playback system that does not introduce certain distortions. However, in practice, the differences are very small for frequencies under 20KHz.

(2) Once again, this is an approximation. An infinite number of samples is required to implement an arbitrary filter exactly; furthermore, the values of all past, present and future samples are also required. However, a finite number of samples can be used to implement that filter with any specified amount of error. In practice, this error can easily be made smaller than the error made by expressing the original samples as 16-bit numbers. Although future samples are still required, one can simply delay the output until all of the needed input samples are available.

(3) The filter coefficients h[x] are referred to as the impulse response because they are defined by the response - the output of a filter - to an "impulse". For digital signals, an impulse is simply the sequence (1, 0, 0, 0...) - a one followed by an infinite number of zeros. The "finite" in finite impulse response filters means that the impulse response is zero after some finite number of output samples.

(4) This assumes that the samples are from a CD, with a 44.1KHz sampling rate.

Open Source Software: A Roundtable Discussion

By Mike Khalili

The idea of Open Source/Free Software is almost as old as the concept of software itself. When much of the newest software came from an academic environment, it was only logical to share the source for that software with one peers to gain input on ones work.

However, times changed. As the computer has become a more important part of our society, more software has come from corporate sources. As the world around it changed Free Software changed. A great number of Open Source applications were developed for UNIX systems. The rapid growth of the Internet led to new applications as well. The bind package, which does name resolution, sendmail, the most used mail transfer agent, and the apache web server, also the most used product of its kind, all were Open Source applications. A number of free operating systems now exist as well, including Linux, OpenBSD, NetBSD, and FreeBSD. In the past year, Netscape made their web browser Open Source, leading to many users who had never used an open source product directly having their first contact with one. Today, Free Software permeates throughout the realm of all software.

The concept of making software Open Source is rather simple. When the number of developers of one product is limited to a single team of software engineers at one company, the room for development and improvement is also limited. If a user desires a new feature or discovers a bug, they must request that the changes be made by those developers. The developers must then prioritize what they believe important, and how the change will effect the profitable of their company. Free software is the antithesis of this. The only limitations upon its development is if at least one person is willing and able to make an improvement upon a piece of software. If a user would like something done, either he or she can do that themselves or attempt to find at least one other person willing to make that improvement. By freeing software from the resource limitations of a single company, Open Source software can be developed much faster, and maintained and improved much better.

At this years ACM Conference, Reflection/Projections we will be having a panel discussing Free Software/Open Source, where it has been, where it stands today, and what its future is. The panel will be comprised of the following four people:

Eric S. Raymond: Eric Raymond has both written a great deal of Free Software, and written a great deal about Free Software itself. He is the author of fetchmail, the most widely used POP client for UNIX. He also has written the New Hacker's Dictionary, and The Cathedral and the Bazaar, a paper on various models of free software development, amongst his many writings.

Theo de Raadt: Theo de Raadt is the the project coordinator of the OpenBSD project. OpenBSD is a free Berkeley Systems Distribution based UNIX-like operating system. Since its inception, it has been a model for the fast and effective development that can occur under an Open Source model.

continued on next page

game programming continued from previous page

and max. After a goal state has been reached, the values trickle up the tree, until eventually the root node evaluates the maximum of all of its successors and returns its move. The result is the same, but the amount of code is reduced and the idea is simplified.

How To Really *Do It*

With some knowledge of common data structures, especially trees, and a basic concept of artificial intelligent searching techniques, the above algorithms can be coded very easily, but they still need to run in parallel. This is where Charm++ and other parallel programming languages come in. Charm++ is an object-oriented parallel programming language that is an extension to C++. With a little extra work and representation for game states, classes, such as the Negamax algorithm, become chares that are executed on separate processors based on load. Other classes become messages that are sent between chares to signal events, the ending of events, or the passing of data.

For a game search tree, the messages can be the states, and new chares can be created as successor nodes with updated states. After a goal state has been reached, a message can be sent back to a parent node, and eventually a message with a move can be returned to the main class that controls the game. It may sound simple, but there are catches. Since each chare may be on a separate computer, data such as pointers, dynamic arrays, and other data types involving memory addresses cannot be passed with messages. Therefore, additional care needs to be taken on the design of parallel algorithms and game state representation.

In part II we will see a better algorithm where lower and upper bounds are maintained while all nodes are constantly updating their costs. This will lead to an algorithm where branches can be pruned based on these bounds because they will never be chosen based on current bounds, speeding up the search.

For more information and to obtain a copy of Charm++ and its manuals, visit the Parallel Programming Laboratory at http://charm.cs.uiuc.edu/

Choosing a Linux

By Mike Khalili

So you've decided to install linux? An excellent choice. Your next step is to choose a distribution to install.

What is a linux distribution? Linux only refers to the kernel, the crux of the operating system that controls hardware and such. To truly have a useful system you have to have a full distribution, including such things as a shell, basic utilities like ls, editors, and optionally a GUI amongst others. The idea of a distribution may seem a bit strange especially for someone coming from a Windows background, since there is but one choice of a windows distribution. In contrast, there are innumerable choices of linux distributions each with their strengths and weaknesses. The diversity of distributions over the years has been one of the great strengths of linux. So the question begs to asked, which distribution is right for you? There is no clear cut answer. Only you can decide which distribution best fits your individual needs. However, I can give you some guide to each distrib.

Slackware: Slackware is the old standby of the linux world. People have been using Slackware for quite a while now. Slackware could be called a relatively vanilla linux distrib. It gives you all the basic tools you need to have a workable linux system. However, it lacks basic things other distributions have, such as a package management system. It also has a bad habit of being a bit behind the times. Slackware isn't a bad distrib, but it seems to be past its glory days, and is slowly looking less and less like a modern linux distribution.

Redhat: Redhat (www.redhat.com) has overtaken Slackware as the standard by which all other linux distributions are judged. Redhat did this by offering a combination of easy installation, the Redhat Package Management system (rpm), and some nice configuration tools. It's install is very straightforward and simple. It's tools generally work pretty well, but occasionally you may encounter a bug in them. Redhat generally stays pretty up to date. In general, Redhat has established itself as a strong distribution. A new linux user would be wise to at least consider Redhat.

Caldera: Caldera (www.caldera.com) attempts to take on Redhat in all of the categories in which Redhat has succeeded. Unfortunately, in general, it fails.Its install method just didn't measure up with Redhat. Once it was installed it tended to have more problems than Redhat. The one nice option Caldera had which Redhat did not was a choice of default installation options. This is similar to Windows' typical, compact, ect. options. This should allow the new user to choose an install that meets their needs without choosing every package they want, one by one. If this is important to you, Caldera might be a good choice.

S.u.S.E.: S.u.S.E. (www.suse.com) was a distribution that began in Germany. From there it spread to the rest of Europe, and recently to America. S.u.S.E is a very impressive distribution. The default S.u.S.E. CD set contains 3 CDs worth of packages, an amount unsurpassed by other distributions, plus an additional CD containing a live filesystem, which lets you boot a full and complete linux distribution off of a single CD, with no hard drive space. S.u.S.E.'s install was extremely nice, and the choice of packages amazing. The install/admin tool YaST (yet another setup tool) puts many distributions' tools to shame. If multilanguage support is important to you, S.u.S.E. leads the way. Like Caldera, S.u.S.E. also has a number of default installs to choose from, in fact far more than Caldera. The one pitfall of the S.u.S.E. distribution is that during the install, it asks you if you'd like to save a record of the packages you install. If you choose no, you may find S.u.S.E. a bit hard to deal with afterwards. However, S.u.S.E. offers a lot for the experienced user, and for the new user, S.u.S.E. is unsurpassed.

Debian: Debian (www.debian.org) is a distribution named for its creators: Ian Murdoch, and his wife Deb (get it, Deb and Ian). Debian is a champion for the free software cause. The Debian Project writes at some length about how free free software should be, authors social contracts, ect. Unfortunately, the Debian Project isn't nearly as good at writing a linux distribution as they are at writing about writing a linux distribution. In comparison to many of the other distributions mentioned here, Debian doesn't match up, especially for new users. It's install is pretty poorly designed and unintuitive, it's package management system mediocre, and the way it installs a base then packages annoying. Debian is a great distribution for people who want to rant and rave about free software, the GNU project, and conforming to standards, but as a usable system it is pretty bad, especially for new users.

Stampede: Stampede Linux (www.stampede.org) is a linux system that focuses on optimization of binaries for Pentium

continued on next page

free software continued from previous page

John E. Davis: John E. Davis has been involved in free software projects since 1990. In that time he has written such programs as the slrn newsreader, the editor jed, most, a more/less paging program, and the s-lang library and language. He has also contributed to the development of rxvt, an X terminal program, the lynx web browser, and DOSEMU, a DOS emulator.

Clark Evans: Clark Evans is a software developer who has taken a six month leave from the industry to study licensing and economics. He is one of the founders of the JOS Project, a project to create a free java-based operating system. Clark Evans is currently working with Mr. Prasanth Nair, a logistics consultant, on a book exploring the political economics of software.

GUI: There *HAS* to Be a Better Way

By Tony Sintes

During a recent interview I was asked, "So which do you prefer, GUIs or command line." Finding this question a bit strange I responded, tongue in cheek, "Well, I prefer GUIs with command-lines." Ya, I haven't heard back from them yet... Of course, coming from a UNIX background with heavy use of Xwindows, this response made perfect sense to me.

Xwindows and Windows 3.1 are archetypal examples of a windowed GUI environment. You type a command or click on an icon and a window pops up. All interaction with the application is then done through this window and various dialogs. Each application is self contained within a window. Each application works differently (though each shares common elements such as controls, menus, and dialogs). This type of GUI basically forces the user to learn a new set of menus each time they learn to use a new application.

So, what benefit have we gained over the command line? Before the GUI, you had to learn and memorize commands to type into a terminal. Now, you have to learn commands and pick them from a menu. It is an improvement, but we can do better.

What if you could make an application more intuitive? Instead of working inside a window and making menu choices, what if you were interacting with objects? When you start your car, do you pull down a menu, choose ignition|start (a '|' 'indicates a submenu), and then press the yes button when it asks you if you're sure? No. So why would you interact with your computer that way?

So, why not take an Object Oriented approach to GUI design. In an OOUI, instead of interacting with an application in a window, you interact with objects.

Take an email program for example. What's wrong with the way we do it today?

Today, to compose an email, you start up your email reader. To compose a new email message, you go up to the menu, pull one down and choose "Compose." Or even better, in another email program you have to sort through a hieroglyphic- like toolbar hoping to click on the icon that will allow you to compose a new email. To digress, you know toolbars have failed when you need text prompts to help you choose the right one. You know what I mean. You want to change the formatting of your letter but you don't know which tool button will allow you to do it. So what do you do? You slowly move your mouse over each one, until the little text prompt that pops up lets you know what each button does.

Okay, now you've done whatever it is you need to do to get a new email started. First thing you do, enter an email address. Then a subject. Finally you can write the email. Once you're done, you go to yet another menu, pull one down, and select send. At this point, you may even be greeted with a little dialog asking if you really want to send the email.

This process isn't very hard. But I bet it wasn't so quick the first time you did it. We've also been doing things this way

continued on next page

Linux continued from previous page

and later processors. A relatively recent split the the GCC (GNU C Compiler) led to two groups forming GCC and EGCS (Expirimental GNU Compilation System). A tangential group PGCC (Pentium GCC) provides patches against egcs to allow for pentium optimizations, a feature not yet supported by GCC. Most distributions compile their packages with GCC, Stampede compiles them with egcs/pgcc. For this reason Stampede may be faster for some users on pentium and later systems. Stampede in many ways is an offshoot of Slackware. It is not a horribly well tested distribution, and lacks some things such as an ftp install, and easy to use tools. For this reason it isn't a great choice for new users. However, for experienced users, Stampede may be a good choice. The optimizations and some fun packages, especially those related to security, may impress some more experienced users.

Nomad: Nomad Linux (www.nomadlinux.com) is what I use. It is a locally developed distribution. Nomad is based upon the encap system of package management, another local creation. The encap system, which causes each package to be in a separate directory, and then linked in, is an extremely intelligent system, and Nomad uses it to its fullest, encapping from the start. For this reason Nomad is a great distribution for users who have experience with the encap system. However, for new users it may not be ideal. The distribution is not complete and from time to time it shows. Simple problems in some of the packages may seem trivial to experienced users, but may pose obstacles for new ones. In addition, Nomad lacks graphical tools, and the install isn't great, and as such will soon be rewritten. Like Stampede, Nomad is a good choice for people who have been running linux for a year or two, but not for those doing their first linux install.

Still confused? Need help? The Linux Users Group will be holding a workshop on October 19 in 1320 DCL on how to install a linux system. People will be there to present a bit more detail on some of this material, and to help you through your first linux experience. In addition, LUG meets every Tuesday at 8pm, and I am directing the Linux Buddies program this year, so if you have any questions mail me, and we'll try to get someone to come and help you.

OOUI continued from previous page

for so long that it's hard to imagine other ways of doing it.

So how does an OOUI improve upon this idea? Easy. To write an email, you go to a templates folder and drag off a "New Email" template icon to your desktop or to another folder. You click on the text part of the new icon and enter the subject. To edit your email, you double click on the email icon. The icon opens up to a text editor where you can compose your message. You write your message, save it, and close the editor. Next, you'll want to send the email. So you double click on your contacts folder. Inside you'll find icons representing people and other subfolders representing email groups. So, if you want to send the email to a person, you drag it and drop it on top of the appropriate person icon. If you want to send it to an entire group, you simply drop it on top of the group folder. If you want to send it to one member of a group, you open the group subfolder and drop it on the appropriate member. Now you're done. You've composed an email without wrestling with menus or making guesses in a toolbar.

A HA! You exclaim. But, if I want to make my letter double spaced or change some formatting option I'll need to go to a menu. Wrong. Since we're dealing with objects, each object has a well defined list of allowed operations. To change the formatting, you would right click on the email icon. This would present you with a context sensitive popup that contains all valid operations. One would be settings or properties. When you open the properties, you would see all the formatting options. Now instead of having to wade through menu after menu, you simply click on the icon to see all of the actions that you can take on the object.

Modern operating systems are getting better in regards to Object Oriented User Interfaces. Windows and OS/2 both have an object oriented user environment (it may take a beating, but believe it or not, OS/2's version is much more advanced). However, these user environments are not being used to their full potential. Sure, we see more and more drag and drop in Windows. But, it seems that this is drag and drop from one application to another. You really don't get the feeling that you're dealing with objects. Instead, you're just opening a file and working with an application.

That, in essence, is the strength of the OOUI. It marginalizes the importance of the application. Instead of using an application to get you're work done, you're just using your computer and its operating environment to get your work done.

Next time, I'll talk more about what goes into the planning and design of GUIs and OOUIs. Later this semester, SigUNIX will also be presenting a language/platform neutral workshop on GUI/OOUI design.

A Brief History of Video Games

By Nick Michels

If there's one thing in life many of us here at the university have in common, besides a love of pizza, it's a love for video and computer games.

It would be impossible to talk about every gaming system in detail without filling a novel, but most fingers point towards three men--Steve Russel at M.I.T., Ralph Baer at Magnavox, and Atari-founder Nolan Bushnell--with making the first modern electronic games in the 1960s and 1970s. Most of today's "major players" were in different fields then. Sega was a jukebox and pinball maker. Namco was a ride manufacturer. Nintendo, founded in 1889, was still dependent on playing cards and simple toys. Game programming was left mainly to university mainframes, like Digital's PDP line, since low-cost microprocessors weren't readily available for the coin-operated and home markets. Hence the emergence of the "light-tennis" style paddle game. Based solely on dedicated circuitry, Atari's Pong and other paddle systems (like those based on General Instruments' popular AY-3-8500 chip) took the world by storm in the '70s, but still only hinted at the future of gaming.

The first generations of microprocessor-based gaming systems still weren't terribly sophisticated by today's standards but still allowed more complex action and sounds. Names like "Vectrex," "Intellivision," "Colecovision," "VCS (a.k.a. 2600)," "Channel F" and many others still reside in attics, closets, and web sites. Game programming became something of an art form, especially with cartridge capacities sometimes less than 4 kilobytes, and third-party companies like Activision and Imagic emerged in what became a burgeoning industry. For a number of reasons, that industry crashed in a period from 1983 to 1984.

Sega's "Master System" and, to a great extent, Nintendo's "Nintendo Entertainment System" (NES), revived the industry. Developed a few years earlier, the NES was released to the American market in late 1985 and early 1986. The system centered around two customized chips: a CPU (basically a customized 1.79 MHz 6502 with some extra sound circuitry) and a picture processing unit (PPU). Each had 2 kilobytes of SRAM and access to its own side of the cartridge port.

The CPU accessed the main program while the PPU could reference screen data to tiles held in its portion of the game pak. What is hardware without software, though? If there's one thing the NES will be remembered for, it was the amazing

continued on next page

games continued from previous page

variety and scope of gameplay it offered. Super Mario Bros., The Legend of Zelda, Metroid...the list of classics goes on and on. Moreover, as technology improved extra cartridge circuitry and creative coding allowed for more complex graphics and action as well as larger games. Near the end of its glory days, the largest NES game, "Kirby's Adventure," reached 6 megabits in size.

The 16-bit era is still pretty recent. With the NES monopolizing the home market, the competition decided to up the ante with newer hardware. These systems, primarily NEC's "Turbo Grafx-16," Sega's "Genesis," and, later, Nintendo's followup "Super NES," promised new levels of gameplay performance. NEC's dual 8-bit CPU (with 16-bit graphics) machine attracted a loyal following, but Sega ended up neck and neck with Nintendo in a fierce system war. Fans chose sides and will still argue the issue of which was "better" to this day!

The Genesis, employing a Motorola 68000 CPU, offered parallax screen scrolling, horizontal bias scrolling, more colors (512 total versus 52 on NES), more on-screen colors, more sprites, and other things. The almost totally customized Super NES, based on a 65C816 CPU, offered even more sprites, colors (15-bit color), translucency, hardware background scaling and rotation (through "Mode 7," one of the system's eight graphics modes), and many other features. One of the Super NES' more interesting feats was that it played host to some digital signal processing. The Super NES' sound system uses a DSP and more complex games added DSPs as coprocessors onto their game paks (e.g. Super Mario Kart, Pilotwings, F1-ROC 2). In 1993, Star Fox was the first of Nintendo's Super FX games. A 16-bit RISC chip with DSP functions, the Super FX was included in over ten million cartridges. The chip's designer, Argonaut Software (http://www.argonaut.com), was so pleased with its success that they formed a separate RISC division, Argonaut RISC Cores (ARC) (http://www.risccores.com).

On August 23, 1993, Nintendo signed a contract with Silicon Graphics, Inc. to develop a next-generation 64-bit system. With ever-present hardware competition on the horizon (3DO, Atari's Jaguar, Sega's forthcoming Saturn, and Sony's Playstation), it was in Nintendo's interest to "up the ante" one more time. The result, of course, was today's "Nintendo64." Now with Sega's new "Dreamcast" on the horizon, the availability of low-cost 3D PC graphics accelerators, and new machines in development by all companies, the bit wars are far from over. The only question is, with all of this hardware, can *software* keep up? Even the best hardware is nothing without new gaming experiences to offer, after all. As new generations of systems and games appear, let's never forget the successes of the past, and let's look forward to the future.

For additional information, check out these sources:

"Game Over" by David Sheff
"Zap, the Rise and Fall of Atari" by Scott Cohen
and any gaming web site such as www.sega.com, www.nintendo.com, or www.playstation.com

acm

student chapter
university of
illinois urbana-
champaign

membership form

Return or mail this form to:
1225 Digital Computer Lab, MC-258
1304 W. Springfield Ave.
Urbana, IL 61801

name:

campus address:

campus phone:

home address:

home phone:

electronic mail:

curriculum:

UNIVERSITY STATUS

- ☐ freshman
- ☐ sophomore
- ☐ junior
- ☐ senior
- ☐ m.a. / m.s.
- ☐ ph.D.
- ☐ faculty / staff
- ☐ postdoc
- ☐ alumni
- ☐ other

special interest groups

lug	linux users group
bug	be users group
sigarch	architecture
sigart	artificial intelligence
sigbio	biocomputing
sigbiz	enterpreneurship
sigcas	computers and society
sigdave	short-term distractions
siggraph	graphics
sigir	information retrieval
sigmicro	microcomputers
sigmusic	music
signet	networking and security
sigops	operating systems
sigsoft	software development
sigunix	unix programming
sigvr	virtual reality

MEMBERSHIP TYPES

return form with check or money order payable to the ACM at UIUC

- ☐ $40 for eight semesters
- ☐ $22 for four semesters
- ☐ $12 for two semesters

ACM NATIONAL MEMBER

- ☐ yes — #
- ☐ no
- ☐ currently applying

for internal use

Received Date: _____ by:_____
Chk #_______ $_______

Enter Date:_______ by: _______