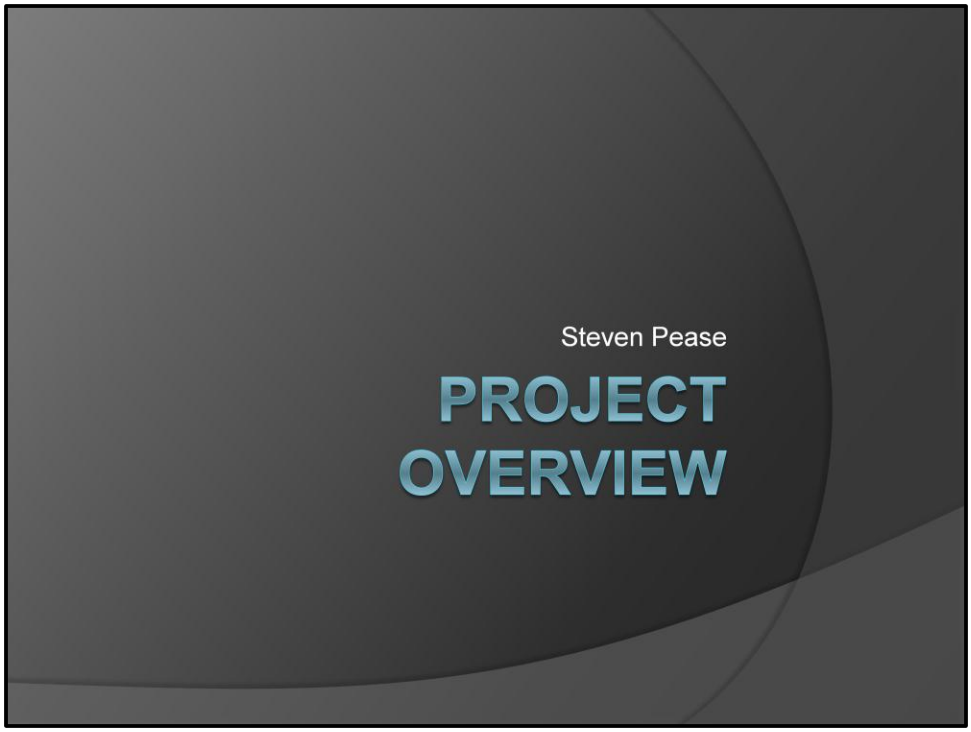
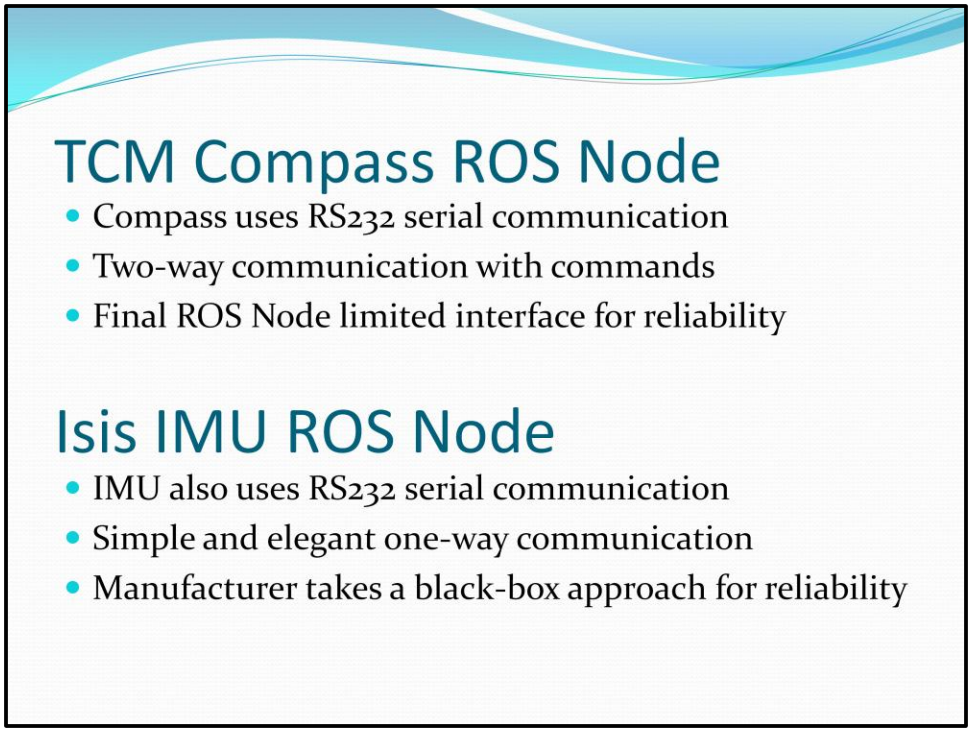
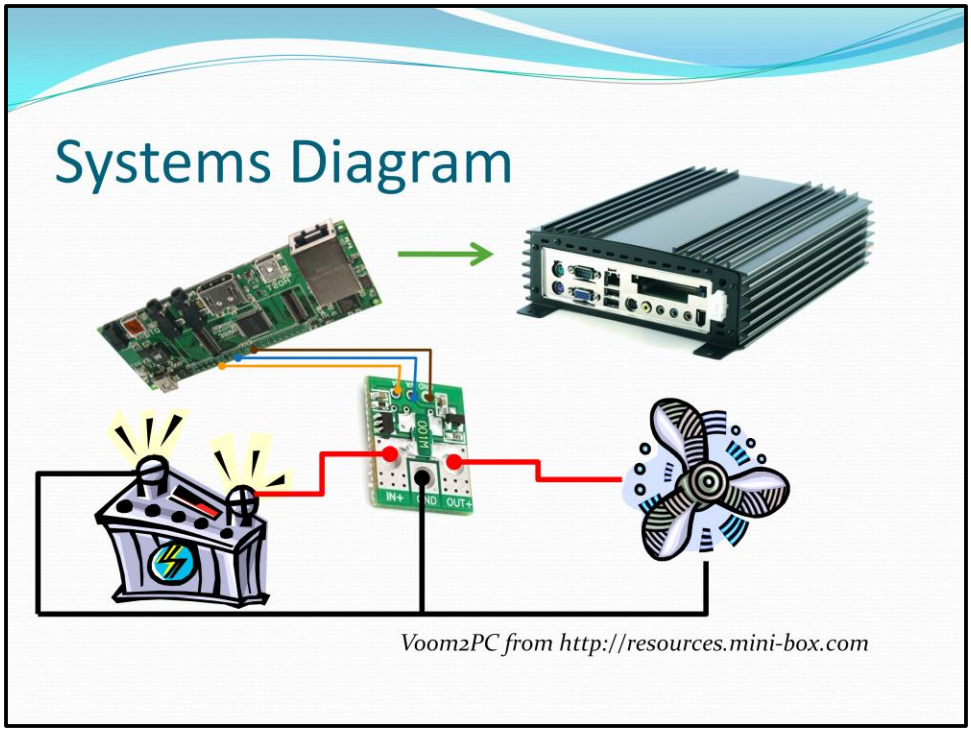

Steven Pease
PROJECT OVERVIEW

The research internship that I did at USC was centered around one of the autonomous boats to be used for aquatic sampling. The boat itself is around 4-6 feet long. Navigation is achieved using a combination of a GPS, compass, and IMU. Obstacle avoidance was to be done using a mounted sonar. The 'intelligence' of the unit was an Intel Atom-based mini-itx computer. Three batteries are used – one for the electronics, one for the sonar, and one for the electric motor/rudder.



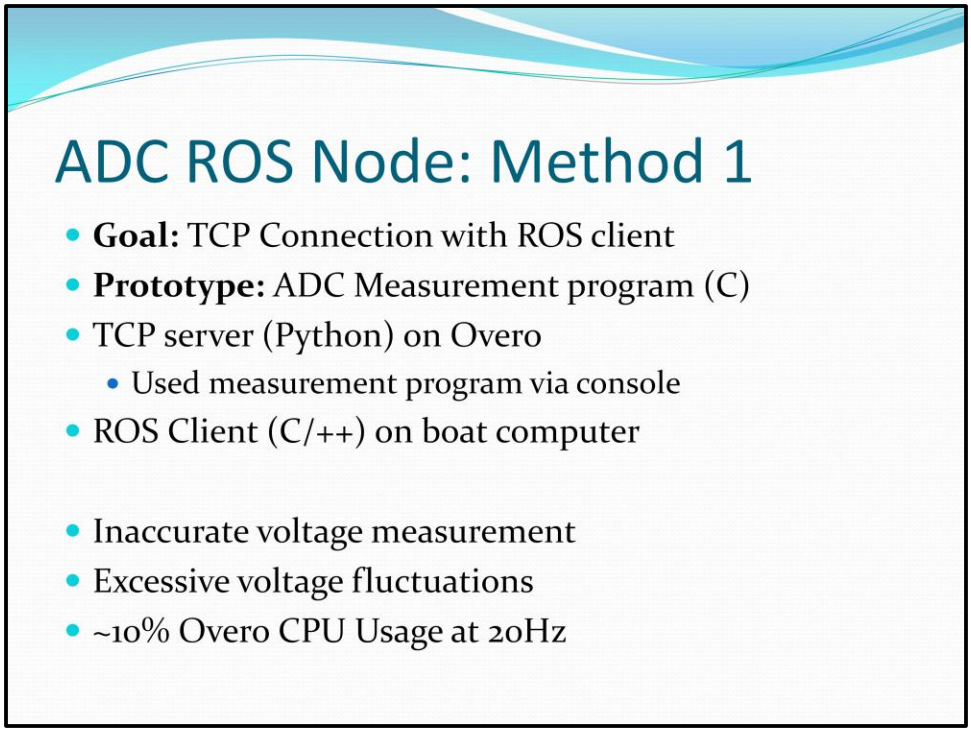
My first couple of projects to get my feet wet were to develop lowlevel serial drivers for the compass and the IMU. All of the development was done under Linux, and I ended up designing a C++ serial library for both using Linux system calls. The drivers were needed because it had been decided to use ROS (Robot Operating System) to pass messages between components. Each component in ROS becomes a 'node', that runs as a separate process. I completed writing ROS nodes for both the IMU and the Compass.



My primary project for the summer was to design a power monitor to determine how much power was being used by certain components. There also existed a sister project to the boats which used solar-powered monitoring buoys for taking measurements from stationary positions. A power monitoring system would also allow the buoys to report how much solar energy they were receiving and how charged their batteries were.

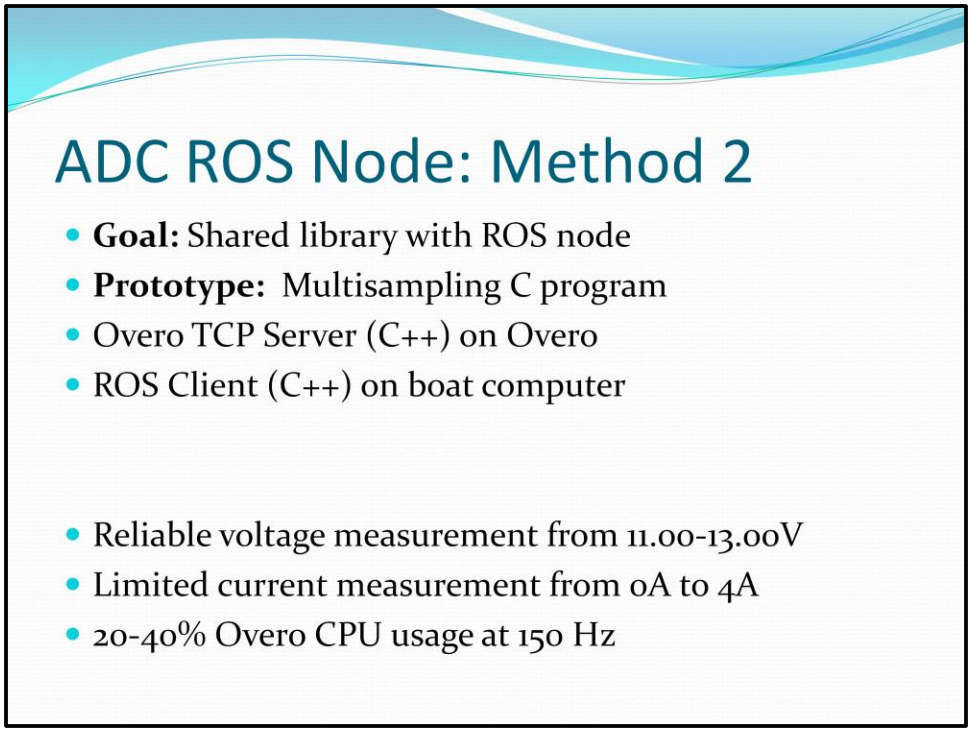
The graduate students in the lab strongly recommended the Gumstix Overo microcomputer, which has 6 ADC inputs. Since many of the voltages and currents were well in excess of the Overo's ADC tolerances, I chose the Attopilot current/voltage converter based on further advice. The graduate students showed me how to do most of the wiring required to physically set up the components.

This diagram depicts the physical flow of data in the system. The Attopilot converts the current/voltage to tolerable voltage levels for the Overo ADC using resistors. The Overo processes the data and then transmits it to the boat's main computer. There the data is received and retransmitted using a ROS node.

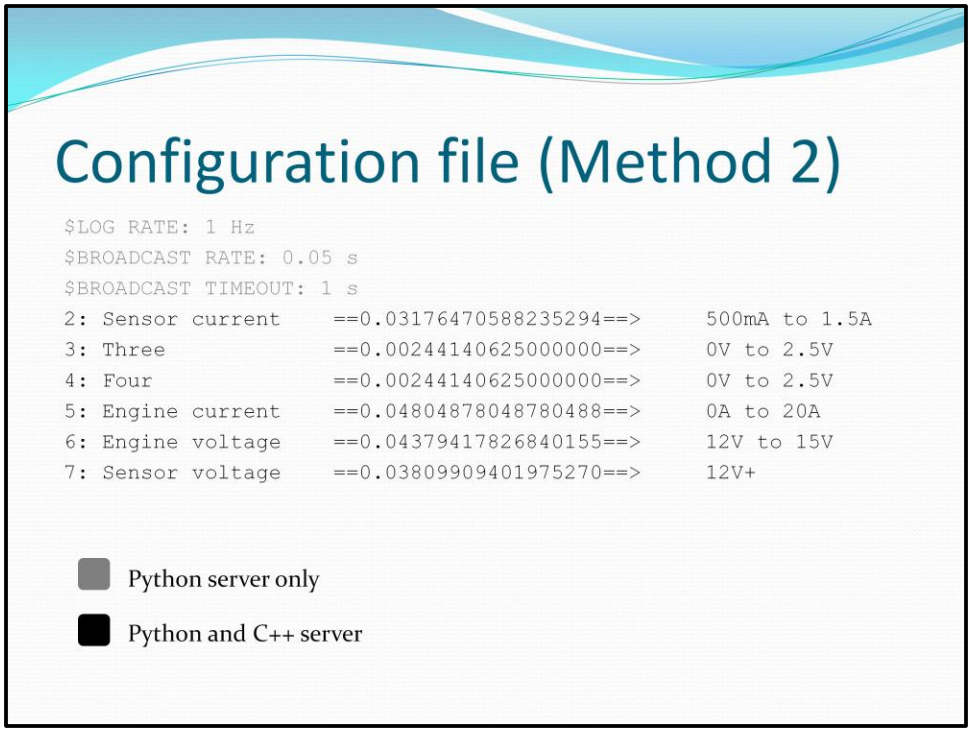


My first attempt used a Python server to read the ADC output from a console program and broadcast it to the boat's main computer. This approach was not very accurate, and seemed very inefficient to me since it required the Python server to start a process every time it took a measurement.

All network communication from the Overo to the main computer used Linux sockets with an application-level protocol that I wrote especially for this task. I tried to use a ROS node directly on the Overo. However, the Overo, ROS, and BOOST (required by ROS) all require their own separate compilation toolchains to build successfully. Designing a network protocol turned out to be easier than getting them to work together.

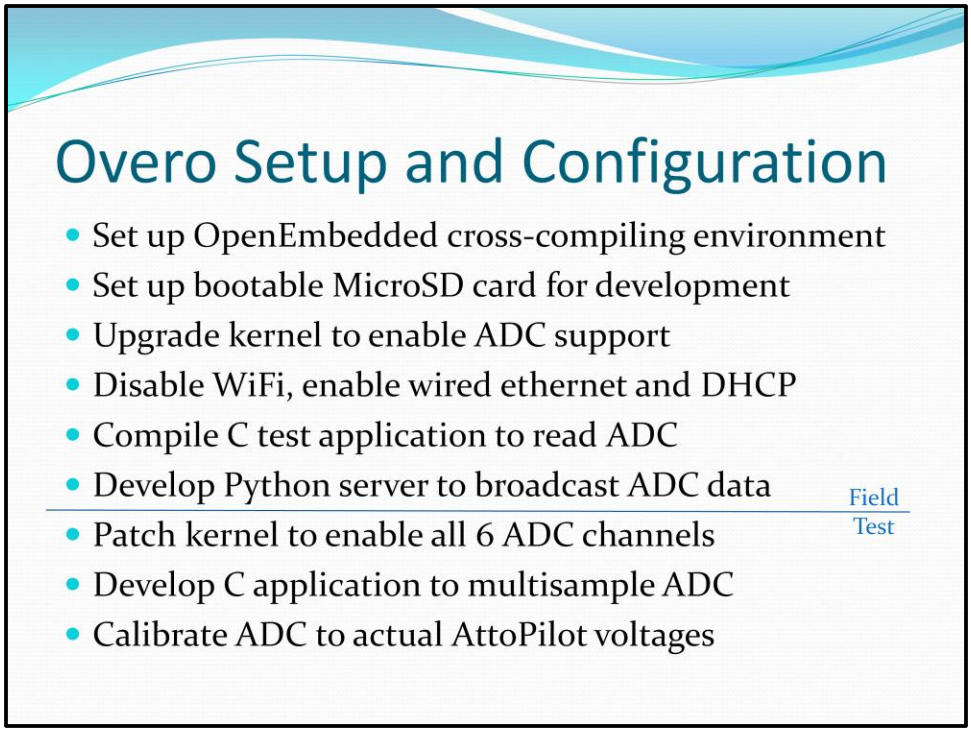


My second attempt was a big improvement. Rather than using an intermediate console program, the server got the data from the ADC directly using ioctl() calls. This allowed the server to sample-and-average the data and get accuracy comparable to the voltmeter I used to calibrate with. The current accuracy was not as impressive but was improved. Finally, I was able to use the same source code library to build the Overo server and the ROS client.



This is the configuration file used by the Overo server to convert the raw 10-bit ADC value to a floating-point current or voltage. The format is:
Channel #: Name ==Calibration Multiplier==> minimum_value to maximum_value

The minimum and maximum value were presently not used, only stored, but were intended to be used for an alert system. The use of some SI prefixes was supported, to make the file as readable as possible.

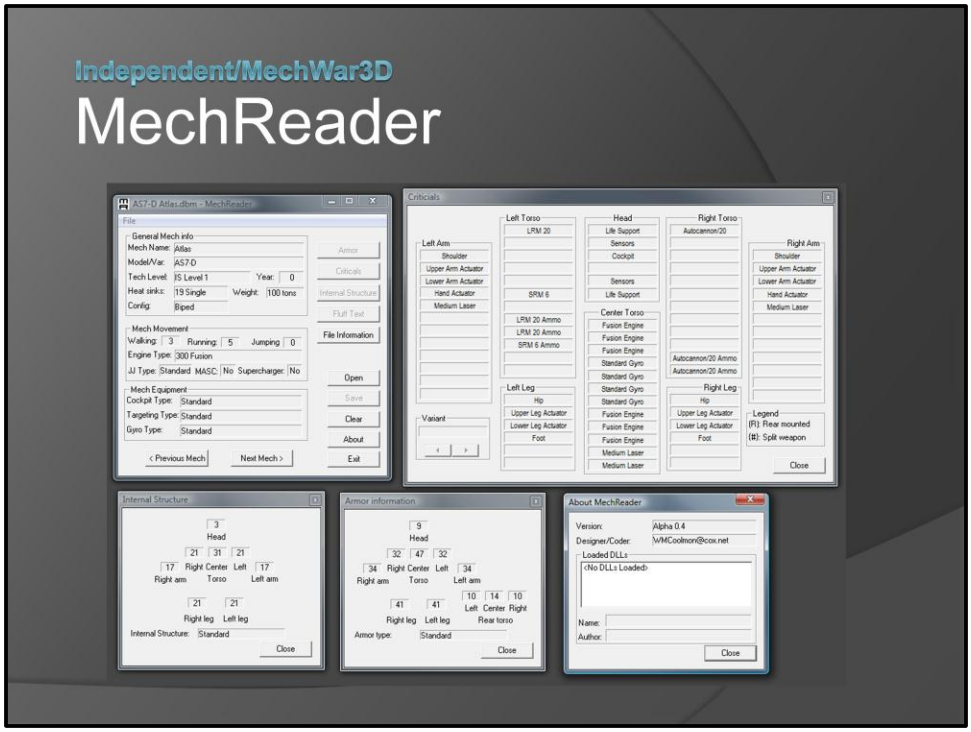


The graduate students showed me how to access the Overo via a USB terminal. From that point on I was pretty much on my own to develop software for it, since the graduate students had only used the previous generation of Gumstix microcomputers.

The initial setup took a lot of steps. (click)
However, to set up additional units' software, all one would need to do would be to create a MicroSD card. (click)
To deploy with different equipment, one would need to calibrate the ADC and field test.

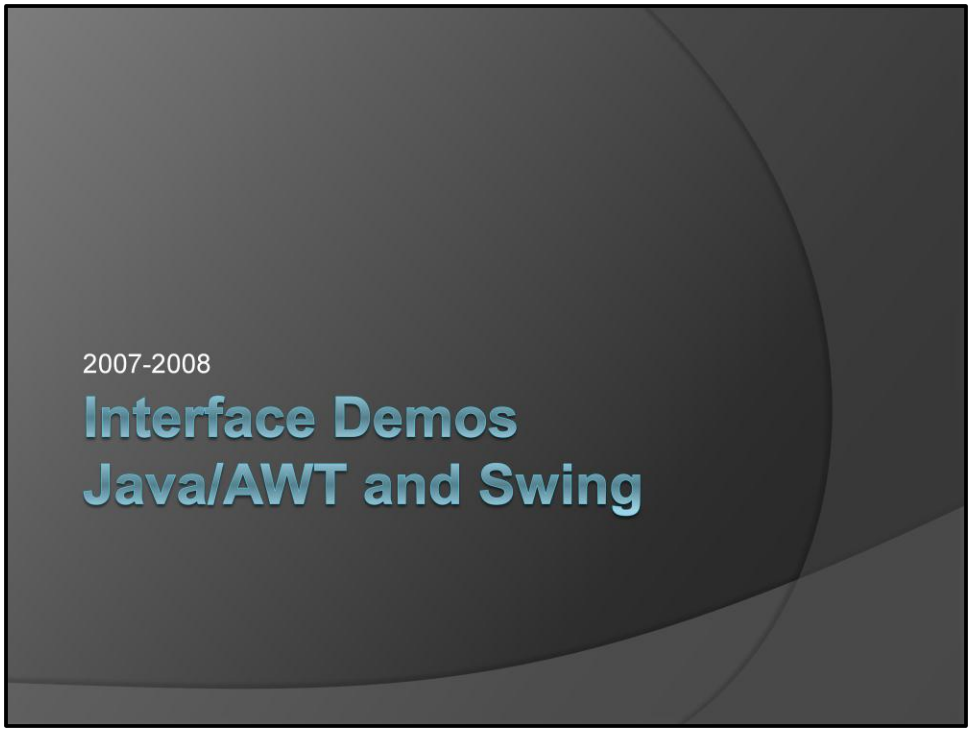
My overall conclusion was that the Overo was not the best tool for the job, but it was cheap and offered a lot more extra capabilities than a dedicated ADC would. At the point that I was working on the boat, it probably could have replaced the boat's main computer and would have consumed much less power.

These are standalone applications I've written in C++ which use the Win32 C API for GUI

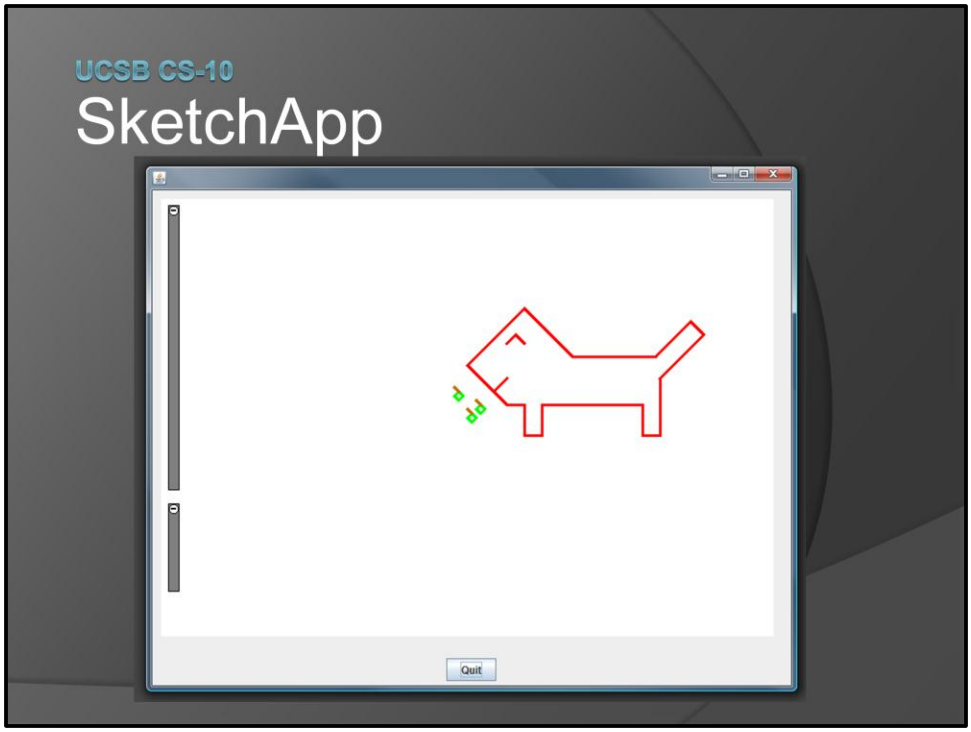


This is the first decent-sized program I wrote. A bit of backstory: There is a board game called Battletech which is based on the idea of combat in the far future between 10-meter-tall piloted robots. It is the inspiration for Mechwarrior. Players of the board game can design new 'Battlemechs' according to weight, space, heat, and technology limitations specified by the rules. A number of different computer-based editors exist to automate the process, and save Battlemech designs to disk. A simulator for the board game called Megamek also uses file-based Battlemech designs. Nearly all of these programs use different file formats to store the same data.

MechReader is capable of reading several of those file formats, then storing and displaying the design in a uniform fashion. It also had support for loading support for additional file formats via DLLs. To decode a file format I would often have to load a design in an editor, tweak a few values, and then see what had changed using a text editor. In other cases the file would be ASCII-based and I only had to write a parser.

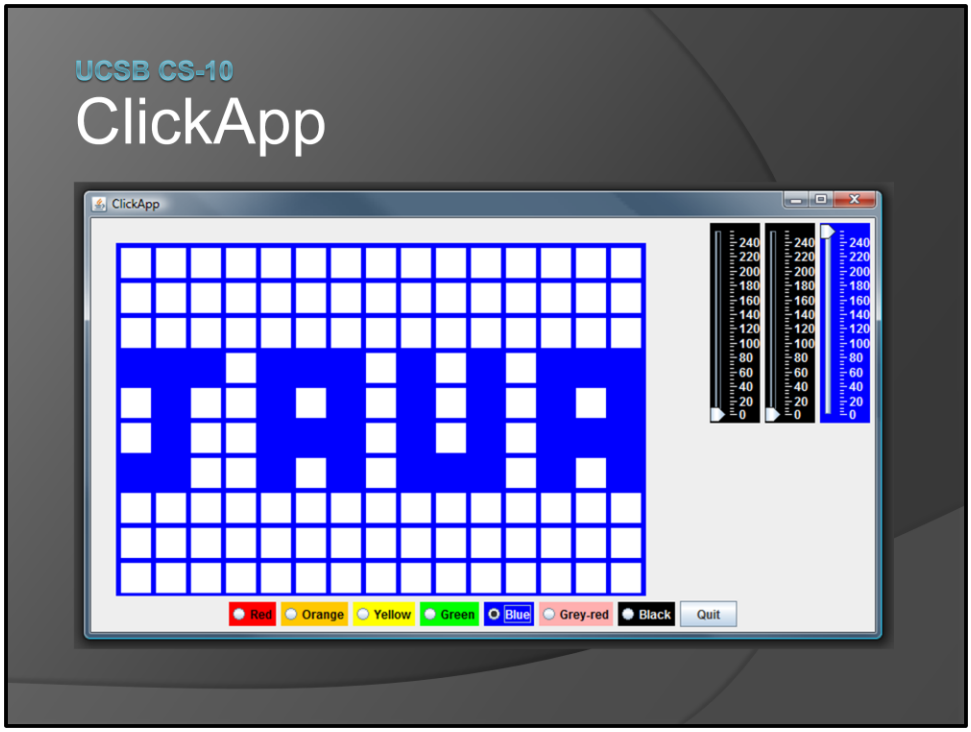


The following programs are all written in Java, using the AWT and Swing sublibraries for drawing and rendering. The first two programs were written for a class at UCSB, the third was inspired by that class but was not written for it.



This is a drawing program for a Computer Science class at UCSB. The eight purple dots in the bottom center move the cursor (a small colored dot) in that direction. If I remember right, the project spec called for a few different buttons to change the cursor and an undo capability. I implemented the color picker on the left, a record functionality (So you could draw an object and then redraw it multiple times, like the green things in the above picture), and put those in what I called 'miniframes'. (click)

As you can see, the miniframes are capable of being moved and collapsed, to give more space to the picture. The minframes were definitely not part of the spec at all.



This was done for a Computer Science class at UCSB. It is a simple box-based drawing program; you click in the squares to ‘draw’, or right-click to erase.

I believe the original project spec only called for having radio buttons for different colors. The bars on the right are for red, green, and blue and are colored as such. They become brighter as the color value is increased. The radio buttons on the bottom change color according to the combination, and also update their names according to what color they are set to (note ‘Grey-red’).

This is probably the most ‘modern’ of the projects, and was fully developed by me (as are the other Java projects). It is an archival utility for .vp files used by the game ‘Freespace’. They are analogous to .tar files, or to uncompressed .zip files. It also features a built-in previewer/editor that can show text files and image files which Java has built-in support for. When I wrote Maja, no graphical editor or viewer for .vp files existed for Mac OS X or Linux. It also became popular on Windows as well.

Note that each type of pane has a number associated with it. Thus if you want two directory tree views, you would click ‘1’ on what is now the output or the editor. If you wanted to swap either of the left-hand panes with the right-hand panes, you would click ‘M’ on the pane to swap. All of the panes and the editor can be resized. It can also open mutiple projects in tabs.

I did eventually add experimental .zip support in a later version of Maja, but found Java’s built-in ZIP support to be very limited.

The rest of the demos are written using the Freespace engine's 2D and 3D graphics functions. These abstracted calls to the lowlevel OpenGL or Direct3D interface (at one point, both were supported). For 2D functions, this would often mean setting a color or bitmap, and then providing coordinates and possibly text for drawing. For 3D functions, additional setup of the renderer and lights was often needed, as well as some basic matrix and vector math in order to orient and position everything properly.

Freespace is a space-fighter video game similar to Wing Commander, or any of the Star Wars 'X-wing' games. It originally shipped in 1998, and became open source in about 2003. Since then it has become the subject of many graphical and gameplay upgrades. The 'Freespace Source Code Project', which I was a programmer for, focused on upgrading the Freespace engine.

Note that I only focused on coding, so all of the models and art here are either original game art, or something created by another contributor.

The first demo in this section is more of an enhancement to an existing game screen. (click)
This is a loadout screen for a space fighter. To the far mid-left are icons for the lasers and guns that can be equipped. In the lower-left are icons for the missiles that can be equipped. At the bottom of the screen are icons to show you what fighters the other wingmen in your squadron are flying.

In the center is the space fighter itself. To the left are the guns selected for your fighter. To the right are the missiles equipped on your fighter. Note that you can equip a different gun or missile in each 'slot'.

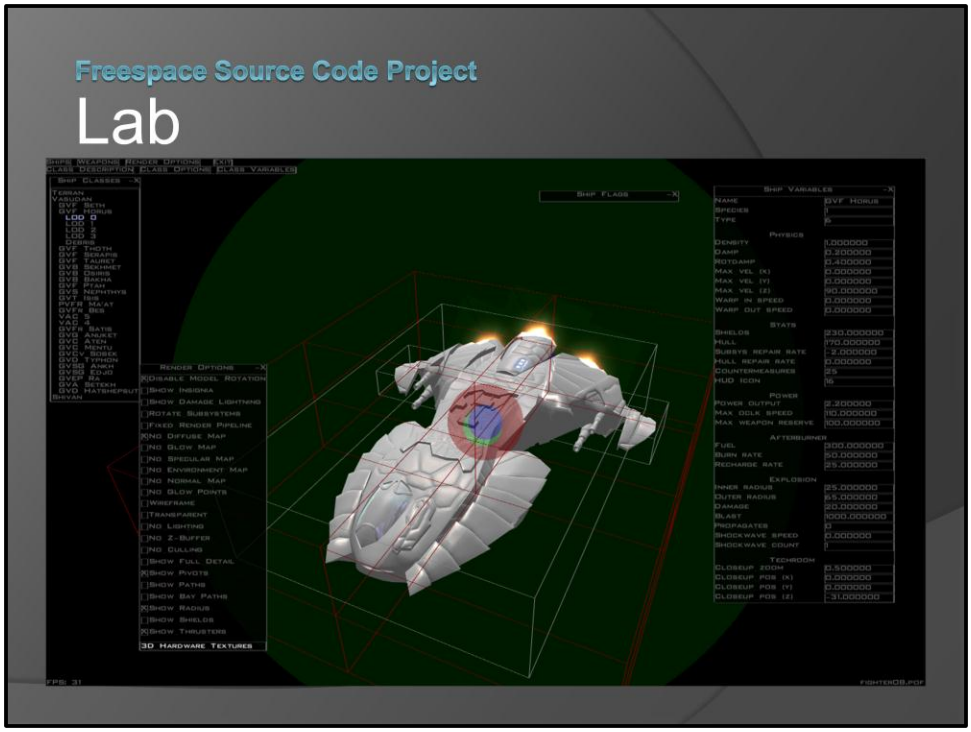
Every single icon and the image of the ship on this screen is prerendered art and had to be created with some kind of paint program or specially rendered for this screen. (click)

The ships and missile in Freespace already had a 3D model associated with them for use during gameplay. I changed these elements…(click)…to use the renderer for the game to draw the icons and the central fighter using the 3D model. This allowed for rotation of the fighter model, decreased the amount of art required to add a new fighter, and also let me show additional info to the player. Now you can see that the bottom slot will fire from twice as many hardpoints as the bottom slot. (click)

Here is the original screen again, except with missiles selected this time instead of guns. Note the animation on the right (click). This was a prerendered animation.

(click) Here it is with a fan-created model.

Here is a side-by-side comparison of the changes to the weapon selection screen.



The final demo I have started out as a proof-of-concept to show that it would be possible to create a level editor within the engine. The decision was made to continue with the level editor as a separate program, and the proof-of-concept became ‘the lab’. (click) The lab is mainly used for previewing new fighter models or textures to see how they will look when rendered by the engine.

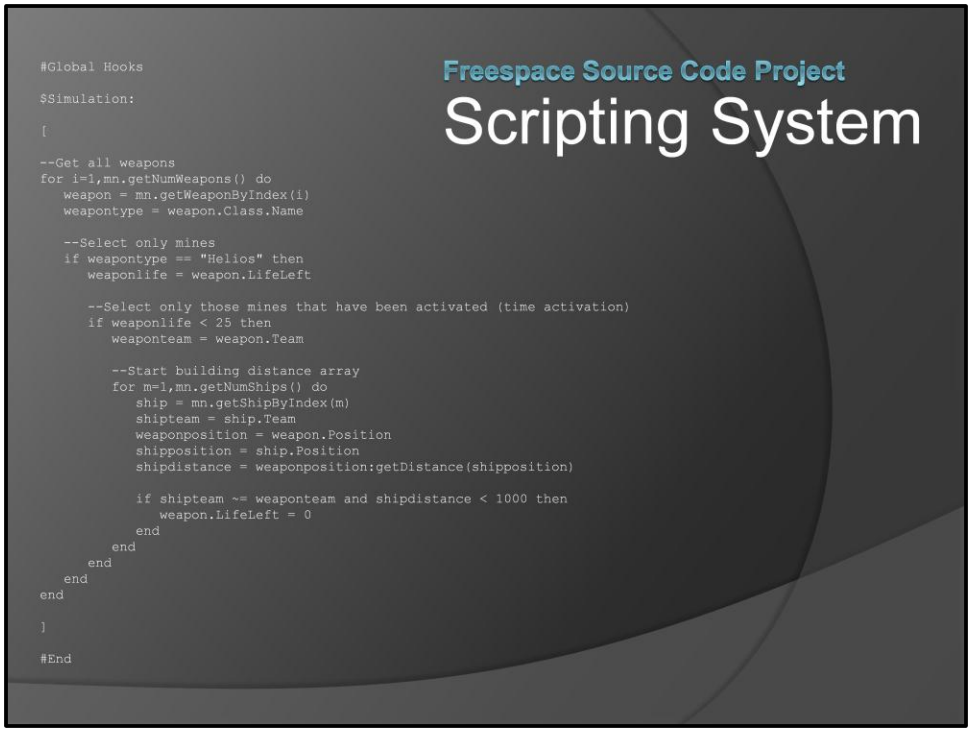
The interface for the lab is based on basic drawing functions for lines, boxes, and text. Input is done by checking whether the mouse button is down or up. Keyboard input is presently disabled, but at one point I implemented it using similar polling.

On the left hand side of the screen is a tree control within a window. On the right side of the screen is a set of text controls. At one point, these could be used to change the stats of the ship from within the game to temporarily test different top speeds, hull strength, etc without the need to edit a configuration file and restart the game.

Finally, the checkboxes in the ‘Render Options’ window control integer flags which are passed along to the model renderer. This lets the user instantaneously see things like the fighter’s shield mesh. (click) Preview the fighter without textures. (click) View the fighter as a wireframe (click), which can be used by the targeting computer on the heads-up display. Or it can be used to show collision data (click).

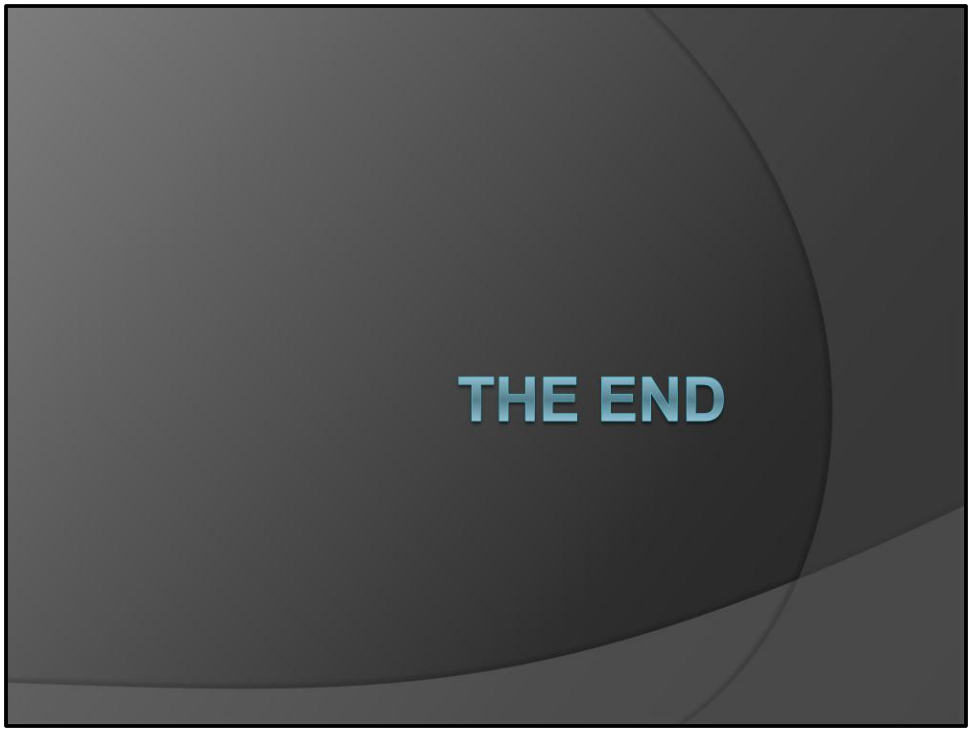
All of the different 3D rendering modes were either there when I did the weapon selection screen, or were added by other coders after I implemented the lab. However no convenient interface existed to use them or to view a ship model

fullscreen.



Though not strictly GUI-related, I also implemented a Lua-based scripting system for the Freespace engine. It was the largest subproject I did for the project. It now has about 400 functions and variables for 40 object types and 9 libraries, about 75% of which was implemented by me.

This particular script was used by someone to create a proximity trigger by detonating the “Helios” bomb (which has a large shockwave) whenever a fighter from a different team strays within 1km of the bomb.


THE END