

# Design of an Upgraded System for Controlling the Trimming of the Sole of a Shoe

Craig Harrison

July 29, 2008

## Abstract

Due to the rapid development of new technology, hardware and software systems quickly become obsolete. The software and hardware controlling this shoe sole trimming machine were originally designed over thirteen years ago to run under the MS-DOS operating system with limited memory, display capabilities, and processing power. The software was modified to run under the GNU/Linux operating system with a new touch screen interface. Concurrently, the old computer system and motion controller cards were replaced with more contemporary equivalents and the software was updated to be compatible with this new hardware. The resulting system is network accessible, easier to maintain, and more user-friendly. These changes will increase operator efficiency, save company money, and allow the sole trimming system to continue functioning in the face of new technology for years to come.

## 1 Introduction

The scour, shave, and trimming (SST) machine was originally designed in two distinct parts: the trimming apparatus and the controlling system [1]. The controlling system consists of a generic desktop computer running software which interfaces with the trimming apparatus via a motion controller card and a digital I/O card. This software was first designed in the mid 1990's to run under the MS-DOS operating system on a 66 MHz computer. Since that time the MS-DOS operating system has become obsolete and the accompanying hardware has become difficult to replace. In order to address these problems the control system was redesigned to utilize recent advances in software and electronics; the ISA-bus motion controller card [2] and digital I/O card have been replaced by two PCI-bus motion controller cards [3], the computer is being upgraded to a multi-GHz machine operating under GNU/Linux, and the keyboard based input system is being replaced with a touchscreen monitor.

Each of these physical improvements rep-

resents a corresponding significant change to the software as well. In order to execute the software under Fedora 9, our GNU/Linux distribution of choice, the MS-DOS specific Borland C++ code was ported to GNU compatible C++. The new motion controller cards also introduce a new Linux driver with an updated API that the software was modified to interface with [2] [3]. All keyboard-specific commands have to be removed from the software and the user interface is in the process of being redesigned to operate naturally with a touchscreen monitor.

## 2 Development

The SST upgrade was broken down into four main phases of development. The first phase involved porting all of the MS-DOS software to GNU/Linux, excluding the hardware components. Phase II was concerned with getting the new software to the point where it worked with the previously installed motion controller and digital IO cards on a GNU/Linux system. In the third phase, where development currently is at, the software transforms from being a port of the MS-DOS software to using entirely new hardware, the largest changes being new motion controller cards and a new touchscreen-based graphical user interface. Phase IV will be concerned with minor modifications, bugfixes, and features to ensure the flawless operation of the new system.

## 3 Compiler Migration

A straight-forward, yet time consuming, portion of development was focused on migrating the Borland C++ (BC4) source code to GNU C++. The majority of the modifications involved specifics of the operating systems or standard libraries, which are addressed in depth elsewhere in this paper. Nonetheless, there were various subtle differences between the Borland and GNU C++ compilers that were addressed in order to successfully port the SST software. One major problem addressed was the compiler dependent sizes of the default data types [7]. On a 32 bit Linux system GNU C++ defines an `int` to be 32 bits and a `long int` to be 64 bits, as determined by use of the `sizeof()` operator. Under version four of the Borland C++ compiler on DOS an `int` is defined as only 16 bits and a `long` is defined as 32 bits. In most areas this was not a problem as the Linux version was simply able to hold larger numbers than were expected. However, there were locations, specifically in communicating with the motion controller card, where assumptions were made about the sizes of the built-in types. This was a difficult bug to detect as the nature of the change caused the code to execute without problems, except for erroneous values being passed to the motion controller hardware. In order to maintain compatibility with the various function calls throughout the software the function definition was altered to correctly cast the variable number of `ints` and `long ints` to the POSIX standard `int16_t` and `int32_t` types, respectively. In the change from using an `...` to a parameter

list in the function call the ISO C standard no longer guarantees that the arguments will be in a continuous block of memory [7]. This resulted in a larger function to implement the same functionality.

Size problems were also experienced with custom data types. The majority of the size changes were handled transparently by the compiler, but, for locations where assumptions were made in source code as to the offsets of member data, fixes had to be implemented by hand. The largest difficulty to overcome was due to optimizations being performed by the GNU C++ compiler wherein custom data types' member data were aligned certain byte boundaries in order to improve performance. Under the Borland compiler this padding did not take place and as such the code did not take any padding into consideration. In order to prevent packing within data types where it was not wanted the GNU C++ `#pragma` directive was used to create a block of code with a tight packing mode enabled specifically around the important data structure, as shown in Figure 1.

```
#pragma pack(1)

struct rpyfmt {
    int16_t cmd;
    int32_t rpy;
};

#pragma pack()
```

Figure 1: Use of a `pragma pack(1)` directive

## 4 Standard Libraries

The Borland C++ compiler and Turbo C for MS-DOS provided an extensive library of C and C++ functions; these libraries supplied such basic functionality as manipulating strings, working with time, and displaying graphics [6] [5]. Many of these functions were never standardized and are exclusive to MS-DOS. Even the so-called Borland standard C++ library was not entirely portable as the language itself had not yet been fully developed when the original SST software was written. Therefore the implementations of many Borland classes, though very similar to the standard implementations, are often available under slightly different names (see Figure 2).

	Borland	C++ Std
Class	<code>strstream</code>	<code>stringstream</code>
Function	<code>read_token()</code>	<code>operator&gt;&gt;()</code>
Header	<code>&lt;cstring.h&gt;</code>	<code>&lt;string&gt;</code>

Figure 2: Selected implementation differences

The Borland compiler was also missing the standard template library (STL) that is now a part of the mostly-standardized C++ library [9], resulting in a variety of data-specific containers in the SST source code. The keyboard polling routines utilized throughout the SST software are, on the other hand, available exclusively in the MS-DOS Turbo C, as are the graphics routines. While the custom containers do not create problems running under GNU/Linux, the lack of MS-

DOS specific functions do.

Despite the fact that a new touchscreen-based user interface was to be designed for the final upgraded software, the first goal was to port the MS-DOS interface as closely as possible to Linux. Implementing the missing collections of Turbo C functions under GNU/Linux running X11 required the use of two external libraries, one for the keyboard input and one for the graphical output. For keyboard functionality the ncurses library was chosen as it is a standard on Unix-based systems and simple to use.

The Allegro library was chosen for emulating the Turbo C graphics. It provides a portable, high level interface to the underlying graphics and windowing systems of many different operating environments [8]. In order to ease the eventual move from this emulation to an entirely new user interface that is currently in development, an intermediate class was written to wrap the initialization and drawing functionality provided by the Allegro library into a generic API that our software can use regardless of the final graphics implementation. In the Allegro implementation a window is created with same resolution as the MS-DOS software and then all drawing functions draw their basic geometric shapes and text directly into this window. The new implementation is likely to be running in a separate process, in which case the wrapper functions will merely send drawing requests to the child process, whose implementation is described elsewhere in this paper.

Under MS-DOS there was no sub-second timing functionality available in either the Borland or Turbo C libraries. The SST soft-

ware implemented a `Timer` class using a hardware timer interrupt every 55 milliseconds. Under Linux this was no longer feasible, instead the POSIX function `gettimeofday()`; was used, which gives much greater resolution on most systems as well as being a far more portable alternative. This functionality is contained in the the standard header file `sys/time.h` as are various non-POSIX standard functions for performing basic arithmetic with the returned `timeval` structure, all of which can be found in the man pages of any Unix-based operating system.

## 5 Operating Systems

MS-DOS, the operating system the SST software was originally designed to run under, is a single threaded operating system, meaning that it can only run one process at a time. The GNU/Linux operating system is a multitasking, multiuser operating system similar to UNIX and conforming to the POSIX family of standards. Code written intelligently to run under GNU/Linux can be easily ported to many other platforms, including BSD, Solaris, as well as most other UNIX-like systems. The GNU/Linux operating system is also under constant development, increasing the likelihood of support for new hardware in years to come.

Under MS-DOS the SST software was limited to 64 kilobytes of memory, which in turn limited the software's design in various ways. Under 32 bit Linux every process has over four gigabytes of address space in which to work, essentially removing all usual memory

limitations. As a combined result of this and the speed of contemporary processors, new code added to the SST software was written with compatibility and style as major foci rather than speed and memory.

The move to GNU/Linux also presented some major technological challenges. The final goal was to use new motion controller cards (MultiFlex PCI 1440) with a proprietary driver and wrap the updated API with functions that emulate the old API. This driver needed minor modifications to work with the most current Linux kernel, but has been found to be a solid driver under the continual support and upkeep of the motion controller manufacturer. For the initial port to Linux, however, the intention was to use the previously installed motion controller card, which is unsupported by this new PCI driver, as well as the the previously installed digital IO card. Under MS-DOS the motion controller card was accessed by the simple process of writing to memory mapped IO. Communication with the digital IO card was done via the `outportb()` and `inportb()` built-in functions.

Communicating with the digital IO card under the more complex GNU/Linux operating system proved to require some modifications. Linux supplies the nonstandard i386 architecture dependent `inb()` and `outb()` commands that behave very similarly to their DOS counterparts. Normally this project tries not to use such nonstandard functions, but in this case their use was acceptable as they would only be used until the final transition to the new motion controller cards. Apart from parameter orders, the

main difference from the MS-DOS versions of these functions is that under Linux permission must first be gained for the specific ports the process will be working with. Jumpers were set on the digital IO card to place it at port `0x300` and initialization code was written to call the Linux-specific `ioperm()` routine to gain access to the two bytes of input and output located at ports `0x300` and `0x301`. In order to be successfully granted port permissions by `ioperm()` the process was also executed with superuser permissions via `sudo`. For a more permanent solution to permission problems the final software will be owned by the superuser and have the setuid bit set so that the program will execute with superuser permissions regardless of the user it is currently executing under.

Communicating with the previously installed DCX-PC 100 motion controller card was the most involved change from MS-DOS to GNU/Linux. Under MS-DOS the card's registers were automatically mapped into the application's memory. Under Linux a driver was required to map the card into the SST process's memory. A driver written for the same motion controller card by University of Maine faculty member Andrew Sheaff was available for our use; however, the driver had undergone very little testing and had been written for the 2.4 version of the Linux kernel. In order to have the driver work under Linux version 2.6 a new Makefile was written for the newly implemented kernel module build mechanism and the driver source was highly modified to work with the 2.6 kernel. Many functions that were part of the 2.4 kernel API no longer exist in the 2.6 kernel

and had to be replaced with rewritten code that utilized new API commands to implement similar functionality. Once the driver compiled for the 2.6 Linux kernel many modifications were then required to fix bugs that existed in the original code. The two major problems encountered were the driver permanently locking itself and the in-core buffer never being synced with the actual device.

Running the SST software on a multitasking operating system such as GNU/Linux introduces the theoretical problem of other processes gaining control of the processor during a critical moment in the operation. The SST software is designed to check inputs, such as the emergency stop button, many times every second. If the software was to ever lose control of the processor for an extended period of time shoes could be destroyed and operators could be injured. To prevent this sort of event from occurring two precautions will be taken with the final control system. First, the SST process will be given a low niceness value, a POSIX standard method of requesting a high priority from the kernel's scheduler. Second, the final control system will use a multi-core processor, allowing other processes to run at the same time without taking away any CPU time from the main SST process.

## 6 Graphical Interface

The MS-DOS based SST software had its own graphical user interface written using the basic Turbo C drawing functions. For the upgraded edition of the software an entirely

new graphical user interface is being written. Comments from over many years of the machine's use will guide the intuitive touchscreen interface to be more efficient and operator friendly. An interface without a keyboard means that all operations must either be button based or use cursor movement to discern the motives of the operator.

Unlike the MS-DOS interface, the Linux interface will be able to run in a separate thread and make use of external libraries for both portability and ease of development. Gtkmm, the official C++ interface to the popular GTK+ library, is being used to implement the user interface. GTK+ is an integral part of the popular gnome desktop, ensuring that it will be maintained for years to come as well as providing a wealth of documentation.

The new application is current developing as a menu and button based interface with the idea that the most commonly used functionality will be quickly accessible via buttons directly available at all times while less used functionality will be available in drop down menus, similar to the layout of many popular word processors. The precise layout of the drop down menus and buttons is contained within a string of XML and is therefore highly modifiable.

The majority of the screen space is taken up by a white drawing area in which the final SST software will display the shape of the shoe sole it is current trimming as well as the shape of the sole after trimming. Eventually the operator may be able to make fine adjustments to the shape of the sole in this area. In order to implement this drawing

area a new C++ class was created which inherits from the `Gdk::DrawingArea` and adds other functionality vital to the useability of the application. This specialized `DrawArea` class maintains a hidden `Gdk::Pixmap` buffer with the required callbacks to redraw portions of the window from this buffer whenever needed. This class also implements the callback required to allow for natural resizing of the window by creating new buffers and copying over the old buffer whenever the screen is increased in size. The end result of this class is a robust, trouble-free drawing area which can be modified via commands very similar to the simple Turbo C functions in MS-DOS.

The actual drawing functions are implemented using the Cairo library, the official C++ wrapper for the Cairo graphics library and the preferred method for drawing in `gtkmm` [?]. Cairo functions allow a huge assortment of different geometric shapes to be drawn with many different styles; the SST `DrawArea` class uses it to draw simple anti-aliased dots, lines, rectangles, and text with modifiable thicknesses in order to simulate the Turbo C graphics routines, albeit with a more polished look.

The fallout of using `gtkmm` rather than allowing the SST software handle the interface and drawing by itself is that the `gtkmm` `Gtk::main::run(window)` function must be running all the time in order for GTK+ to continuously check for input events and redraw the menus and window. For this reason the graphical user interface was designed to be running as a separate process from the core SST software, with all drawing commands sent over a POSIX pipe. This im-

plementation is slow and inefficient as multiple extra function calls must occur for every drawing operation and the graphics process must check the pipe multiple times per second. There can also be up to a tenth of a second delay from the time a drawing function is sent into the pipe to when the graphical process actually draws it to the buffer. Fortunately, the drawing function calls occur relatively infrequently compared to most graphics intensive applications and so a tenth of a second delay does not impact the experience of a human operator in any noticeable way.

## 7 Remote Connection

Developing the SST software under a GNU/Linux environment opened many opportunities to employ networking both during the development process and afterwards. With the closest SST machine a two hour travel away, the ability to test software on the machine remotely was an important aid. The implementation of robust networking software on the final product would also allow knowledgeable individuals to debug the working machines from across the country, saving company time and money.

The software package used to make remote connections was OpenSSH, a free implementation of the SSH secure shell. It is not, however, possible to connect directly to the SST machine, as the factories have their own internal networks, which are unaccessible from outside. In this case a reverse tunnel was created from the SST machine to a readily accessible external server (see Figure 3). Once cre-

ated, this reverse tunnel allows anyone who can connect to the external server to have their connection forwarded over this tunnel to the internal SST machine. The downside to this solution is that it works only so long as the reverse tunnel stays up; as soon as the tunnel goes down it takes physical access to the SST machine to bring it back up.

```
ssh user@server -R 55000:localhost:22
```

Figure 3: reverse tunneling over SSH

In order to implement a more robust solution a bash shell script was written and set to run whenever the machine started. This shell script runs in an infinite loop, trying again to create a reverse tunnel every time the previous one goes down. Unfortunately, after a long enough period the external server will forget about the reverse tunnel, despite the SST machine still being connected. In this event either physical access to the machine is required or the machine must be rebooted. In the interest of time a cron job was set up on the SST machine which initiated a reboot once an hour during the workday. If the remote connection was available the shutdown could be canceled; if not, the longest possible time the connection could go down for was one hour.

In order to work productively on the SST software from a remote location physical feedback from the machine was required. For example, it was meaningless to have the software time out waiting for a Find Index (FI) command to complete without being able to check if the motor was physically moving to-

wards its index. This was addressed by installing a webcam next to the SST machine and connecting it via USB. In order to easily check the webcam output all SSH connections were created with the `-Y` flag, which enables trusted X11 forwarding. This allows graphical applications to open their displays over the remote connection, as shown in Figure 4.



Figure 4: X11 forwarding of a webcam

## 8 Acknowledgements

This work is co-funded by the ASSURE program of Department of Defense in partnership with the National Science Foundation REU Site program under Grant No. 0754951.

The SST machine was originally designed by University of Maine faculty members Dr. Richard Eason and Eric Beenfeldt. The work outlined in the paper was done under

the supervision of Dr. Eason, who initially developed, and is currently upgrading, the control portion of the machine. [10] *gtkmm 2.4 documentation*. gtkmm development team, 2008.

## References

- [1] Beenfeldt, E. and Eason, R. *Apparatus and process for trimming a sole of a shoe*. U.S. Patent 5,485,643. 23 January 1996.
- [2] *DCX-PC100 User's Manual*. Precision Micro Control Corporation. California, 1995.
- [3] *MultiFlex PCI 1000 Series User Manual*. Precision Micro Control Corporation. California, 2007.
- [4] *MultiFlex PCI 1000 Series Command Reference*. Precision Micro Control Corporation. California, 2007.
- [5] *Turbo C Reference Guide*. Borland International, Inc. California, 1988.
- [6] *Borland C++ Programmer's Guide*. Borland International, Inc. California, 1993.
- [7] ISO/IEC 9899:1999, *International standard programming languages - C*. ISO/IEC, 1999.
- [8] Shawn Hargreaves, *Allegro: A game programming library*. 2006.
- [9] SGI, *Introduction to the Standard Template Library*. 1994.