1 System Overview

When the program is launched a window comes up with several menu choices. The "File" menu acts as you would expect. You can have multiple G-code programs open at the same time; each one appears in its own tab. You can edit and save programs from here too, although the editor is nothing fancy.

The "Render" menu generates a picture of the object described by the G-code program. There are two choices: "2D Display" and "3D Display". Either choice generates a picture for the G-code in the foremost tab. The picture will appear in a new window, and the image is not automatically updated when you change the G-code. If you modify the G-code, and you want to see the result, choose "Render" again. Images for both versions of the G-code (new and old) will remain open. These images (and the data used to generate them) take a lot of memory. If you have several of these image windows open, then you could easily run out of memory.

Choosing "2D Display" shows a static image of the object, looking from above with each point shaded by the depth of cut. There is no user interaction (although you can make the window larger or smaller). If you choose "3D Display", then you get an interactive 3D image of the object. You can't use the mouse to rotate the image. Instead, use J and L to rotate left and right, I and M to rotate up and down, and U and O to twist the object in place. Use A and Z to zoom in and out. Also, X will close the window. Upper and lower case letters both work.

The "Settings" menu has several choices. Most of the choices should be self-explanatory, but not everything works. Both "Scale" and "Inches/mm" do work. If memory is tight, or if the image updates uncomfortably slowly, then reducing the scale should help. If the scale is s, then the virtual machine should work exactly as though the tool moves in steps of $1/s$ inches (or millimeters). Put another way, it's as though you multiplied all of the coordinates in your program by a constant. For instance, if you change the scale from $1,000$ to 100, that's like multiplying all coordinates by 0.10. For objects that are only a few inches in size and are of modest complexity, leaving the scale at 1000.0 should be fine, but for large objects, you may need to make this smaller. In fact, unless you're doing something really intricate, a value of 100 (or even less) is probably fine.

The "Work Offsets" dialog does work (although it hasn't been very thoroughly tested), and it should be clear what to do.

Don't try to use the "Uncut Shape" dialog. It's meant as a way for the user to indicate the size of the raw material, but it doesn't work. Right now, the program will automatically use an object that's large enough to encompass all cuts made by the program.

The "Tool Table" works, but only up to a point. The system recognizes D and H values from this table when using cutter comp and tool length offsets, and it knows the diameter of the tool. What it can't do is use anything other than simple endmills. If you change the diameter value in the D-column, then you must hit the return/enter key to tell dialog that you mean it before you click the “OK” button. One last thing: there’s a secret default tool. This tool has a cutting diameter of 0.020 inches, but the cutter comp is 0.250 inches. This is handy sometimes when you’re trying to understand what cutter comp is doing; it makes it easier to visualize the actual path of the tool. If there’s no `M06` in your program, then you’ll get the secret default tool.

Concerning Java and memory...When you launch the program that I wrote, your computer isn’t really running my program. The computer runs Java, which in turn opens up and runs my program. This means that, whenever my program needs more memory, it must ask the Java system for the memory it needs. You may have oodles of memory on your machine, but Java won’t let my program use it unless you tell Java that it’s OK. I provided one way around this problem (a `.bat` file), but there are others.

2 G-codes and M-codes

I tried to implement all of the G-codes that one might care about and that are likely to be consistent across different makes of controller. Here’s a list, with some comments.

`G00`	Rapid travel.
`G01`	Ordinary move. There are no surprises with these two. However, for all practical purposes, my program makes no distinction between `G00` and `G01`. The program does not check for things like whether the spindle is on when you cut, or whether you’re trying to cut in rapid mode. Likewise, the program accepts a feed rate (F-value), but the feed rate has no effect on the output.
`G02`	
`G03`	Circular interpolation. These do what you’d expect, although I’m not very confident that they work correctly if you are not in `G17` mode.
`G04`	Dwell. This is accepted, but ignored. Actually, it might cause an error, depending on how you use it. I’m not sure.
`G15`	Polar coordinates off.
`G16`	Polar coordinates on. Again, using `G18` and `G19` might not work right. I tried to make them work correctly, but it hasn’t been thoroughly tested.
`G17`	
`G18`	
`G19`	See above and below for caveats about `G18` and `G19`.
`G20`	Inches.
`G21`	Millimeters. The abstract machine works internally in either inches of millimeters, but you can freely change from one unit of measure to the other within your G-code. In fact, my program is probably more flexible than many real controllers, perhaps dangerously so. You can sprinkle `G20`’s and `G21`’s all over the place.

`G28`	This is accepted, but ignored. Smid calls it "machine zero return"; I'm not sure exactly what it should do.
`G40`	Cancel cutter comp.
`G41`	Cutter comp, left.
`G42`	Cutter comp, right. These work as they should, but (as usual) they might be wrong under `G18` and `G19`. Also, I do things differently than how most real CNC machines seem to work. As soon as a `G41/42` appears, the virtual machine peeks ahead to see what the next statement will be, and it bumps the tool out by the cutter comp amount to be ready to cut the very first curve with the correct cutter comp. I think this is better than the way it's typically done, but it could confuse people who are trying to test against a particular real-world machine. Anyway, there seems to be so much variation from one machine to another in how this aspect of cutter comp is handled, that some degree of mismatch between my virtual machine and the real world is unavoidable. Dealing with cutter comp is messy and it's the thing most likely to have bugs. Also I chose to handle cutter comp by using what are probably very unorthodox methods. They are powerful, in the sense that I could extend my program to handle cutter comp on (as yet unspecified) curves that are more complex than line segments and arcs of circles, but there could be bugs for reasons that are hard to track down.
`G43`	Tool length offset, in the normal case.
`G44`	Tool length offset, in the negative sense. I doubt that this is used often, but it works.
`G49`	Cancel tool length offset. Tool length offset works as it should. One oddity is due to the fact that, in a virtual machine, tools don't have a natural length. For the time being, treat all tools as though the machine magically knows where the tip of every tool is. If you use `G43` after setting $H = 0$ in the tool table, then it will act like it should. If $H > 0$, then the tool will cut more deeply under `G43` and less deeply under `G44`.
`G52`	Allows you to specify a new local reference frame. This is something like a work offset (`G54`,. . .,`G59`), but the coordinates are specified in the program, and must be given relative to the PRZ.
`G54`	
`G55`	
`G56`	
`G57`	
`G58`	
`G59`	To change work offsets. These work as you would expect, but the coordinates must be given relative to the PRZ in the work offset table. There's no natural way to define "machine coordinates" other than the PRZ.

`G90`	Absolute mode.
`G91`	Incremental mode. These work as expected.

As you can see, none of the canned cycles, like `G81` for drilling, have been implemented. Some of them might manage to get through my system without causing an error, but none of them do anything. I haven't thought very hard about this, but I don't think that they would be hard to add.

Here are the valid M-codes.

`M01`	Smid calls this "compulsory program stop."
`M02`	And he calls this "optional program stop." In both cases, I make the program halt, like an `M30`.
`M03`	Spindle on, clockwise.
`M04`	Spindle on, counter-clockwise. The program notes internally that the spindle is on, but it has no real effect.
`M05`	Spindle off.
`M06`	Tool change.
`M07`	
`M08`	
`M09`	These coolant on/off commands are accepted, but they do nothing.
`M30`	Halts the program.
`M40`	
`M41`	Spindle high/low. Accepted, but ignored.
`M47`	Repeat program. Since I'm not sure exactly what this should do, it generates an error.
`M48`	
`M49`	Feed & speed override on/off. Accepted, but ignored.
`M98`	Call subprogram.
`M99`	Return from subprogram. Both of these work.

The system also accepts line numbers (with `N`), but they don't have any meaning since none of the valid codes can refer to line numbers.

3 Shortcomings

The most obvious shortcoming is that it can be slow to bring up the image. Once the image is up, it's reasonably fast to move around in 3D. The larger the tool is, the slower it is. If the tool is only 0.020 in diameter, then it's plenty fast, but a tool that's 0.250 in diameter takes roughly 150 times as long as a tool that's only 0.020 in diameter. Having to wait 10 seconds, or even a minute for the image to come up is possible. In round numbers, with a quarter inch endmill, you can expect to wait 0.25 seconds for every inch of tool travel (at least that's true on my machine). I have ideas about how to fix this, but it will take some time to make the necessary changes. As a quick fix, you can change the scale value from 1,000 to something like 100.

The other big problem is the fact that the program is a memory hog. There are things I can do to improve matters, but some amount of hoggishness is unavoidable. The program isn't very nice about its memory needs; it may crash if it can't get the memory it wants.

You'll notice that, in 3D mode, the image you get isn't really a solid object at all; it's just the "skin" of the object's top. Making the object appear as a solid is easy, but I find that, given the less than ideal method of shading that I use now, it's easier to visualize what's been cut if the object isn't solid.

Objects that are large and/or intricate will be drawn slowly in 3D mode. You'll notice this for anything larger than about three inches square. I mentioned above that you can reduce the scale value as a crude fix for this problem. Doing that makes the internal representation of the object less detailed. I have ideas about how I can improve the speed, but implementing these idea will take some time.

The way the object is shaded isn't the greatest. Right now, the surface of the object is shaded based on the depth of the cut. This is OK most of the time, but sometimes it would be nice if there were more contrast between adjacent surfaces. I know how to fix this (I think), but it's non-trivial.

If you zoom in too far on objects, the program may crash. I know exactly why this happens, but I'm probably going to totally redo that part of the program anyway, so I don't want to deal with the problem.

Even for very simple objects, there's an internal limit on the permitted size of the object. This limit is roughly a cube 32 inches on each side. I could allow objects that are literally miles to the side by doubling the memory requirements, but, unfortunately, there's no middle ground. In any case, my current less efficient methods of memory usage dictate a limit of roughly a square six inches on a side.

4 Bonus

The "Test" menu is there for my own debugging purposes. It's not meant to be very clean, but there are a few things there that you might think are interesting.

The output of "Test Voxel Frame" looks something like what you see in the 3D window under Mach3. Every tool position gets a dot in three-dimensional space, and this draws those dots. The drawing is cruddy, but it wasn't really meant for public consumption. You can rotate and zoom using the keyboard.

Choosing "Test Pulses" will show the series of pulses (zeros and ones) generated by the G-code. These are the signals that would be sent to the motors of a real-world CNC machine. Again, this was for debugging, and I think that it only shows the first 100 pulses.

The other choices under the "Test" menu have to do with the way that my program converts G-code to pulses. It's a multi-step process that begins with the lexer (in computer science lingo, the “lexical analyzer”). The lexer converts the string of characters that make up the G-code into bite-sized pieces that later stages find easier to digest. For instance, the lexer converts

```
X0.100 Y1.500
```

into two “tokens”, one for the X-value and another for the Y-value. That's probably not so interesting to you, but the other layers of the process might be.

The parser takes the tokens from the lexer and converts them to meaningful statements. Basically it verifies that the G-code isn't absolute gibberish. The output of “Parsify” should look pretty much like the input G-code.

The next five steps, “Simplify 00” through “Simplify 05” take the original G-code and convert it to G-code of a simpler and simpler form. The 00 step eliminates subroutine calls. The output is exactly what you would have to type (over and over again) if there were no such thing as `M98/99`. The 01 step converts everything to the standard unit of the machine (inches or millimeters) so that `G20/21` no longer appear in the G-code. The 02 step removes any work offsets so that `G52` and `G54` through `G59` are eliminated, along with `G43,44,49` for tool length offset. All coordinate values generated by step `01` are adjusted as needed. The 03 step gets rid of polar coordinates. The 04 step changes all coordinate values to be absolute so that `G90/91` is not needed in the output, and step 05 adjusts everything for cutter comp.

The output of step 05 is exactly the G-code that you would have to type to create the same object as the original G-code, but on a machine that understands only `G00, G01, G02 and G03`, along with the `M`-codes for tool change and spindle & coolant control. The ability to see the output of this final step may be useful from a teaching point of view. It would help students to visualize what cutter comp and subroutine calls (for example) really do.