1 What is This Thing?

For the time being, I am calling this program "Ger," which I got by combining "G," as in "G-code," with the "er" sound of "verify." So, the program is a G-code verifier or a "Gerifier" – "Ger" for short. You can pronounce as in the name Gary (or Jerry) when you feel friendly toward the program, or a bear's growl when the program isn't doing what you want it to, which shouldn't be very often.

I thought of calling the program "Gim," as in "G-code simulator," but I'll bet that name is taken for something. If you think of a better name, let me know.

As you use this program, please let me know if you find anything about it that doesn't seem right. Obviously, if it translates your code in a way that doesn't make sense, I want to know about that. In fact, I'd like to hear about anything odd or unexpected that the program does. If you see an inefficiency, inelegance, or just have a good idea related to the program, then I want to hear about that too. If you do find any bugs, it will help me if you can send me the input G-code and/or explain (in detail!) what you did that led to the strange behavior. I can't fix a bug that I can't reproduce. I am at `rsfairman@yahoo.com`.

At this point the program can do two things: translating G-code to a simpler form and rendering a two-dimensional image of the part being cut.

1.1 Translation

You can give Ger some G-code, and have it spit out a simplified version of the code you gave it. Here's what I mean.

Fundamentally, any G-code program boils down to having the tool do one of two things: move linearly or move along a circular arc – there are also helical moves, but these are just a combination of a linear move and an arc move. For example, if the program you input involves cutter compensation (`G42`), work offsets (`G56`), polar coordinates (`G16`), incremental mode (`G91`), and so forth, then the output program will have the same effect as the input program – will move the tool in the same way and cut the same part – but it won't involve any of these more complicated commands. The output program will involve only `G00, G01, G02` and `G03`.

In fact, there are a few codes in addition to these four that can't be eliminated from the output program. These additional codes are `G17/18/19`, which specify which of the XY, ZX or YZ-plane to work in, along with the M-codes for spindle start/stop (`M03/04/05`) and tool change (`M06`). In a section below, there is a list of the G/M-codes that Ger accepts as input, along with a brief description of what each one does.

The translator is useful for several things. It's a good teaching tool. The more esoteric G-codes can be difficult to understand, and once you do understand them, it's easy to forget how they work. By running your G-code through

Ger, and looking at the output, you can check your understanding of particular codes.

Second, not all controllers are very friendly about certain G-codes. For instance, I have had bad luck using cutter comp on Mach3. Maybe it's me, but I've had Mach3 go haywire too many times when I was using cutter comp to trust it. Using Ger, you can write your program with cutter comp, have Ger translate it to a program which does the same thing, but does it without cutter comp, and give that simpler program to the real-life controller.

Third, even experts get confused. If you're writing a long program, particularly one that uses sub-programs and incremental mode, it's easy to get mixed up about where the tool is at a particular point in the program. Ger expresses its output strictly in absolute mode, making it easy to check that the tool is where you think it is as the program proceeds. If the tool is not where you think it should be, then Ger makes it easy to find out where your program went wrong.

1.2 2D Rendering

Ger is able to show a flat (2D) image of the object cut by your G-code (I'm working on 3D rendering, but it's not ready yet). The depth of cut is indicated by the shade of gray used, with darker areas being cut more deeply.

Once the image is visible, you can drag it around with the mouse or zoom in and out by using the mouse wheel. Double-click on the image to bring it back to the standard view showing the entire object centered in the window. You can use the keyboard instead of the mouse: use the arrow keys to pan the image, the plus and minus keys to zoom in and out, and the enter key to return to the standard view.

2 Overview of Menus

When the program is launched a window comes up with several menu choices. The "File" and "Edit" menus act 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. You can adjust how much space each of the left and right panes has by dragging the divider between them.

Use the "Action" menu to perform a translation/simplification of your G-code or to show an image of the object. After you've loaded some G-code, either by opening a file or by typing it in, choose "Translate" and a simplified version of your program will appear in the right panel. The format is basically the same as ordinary G-code, although it's been restated to use only the most fundamental linear and arc moves. The line numbers that appear on the right (the N-values) refer to the line in the input file that lead to that line of output. You can use this to understand where your input program went wrong – why

didn't the simulated controller move the tool in the way I thought my G-code specified?

The "Action" menu is where you can bring up a various dialog boxes that set up the simulated machine. The "Set Geometry" dialog allows you to define the dimensions of the raw material being machined, whether to work in inches or millimeters, the location of the PRZ (*i.e.*, the origin) and the tool turret.

The "Tool Table" dialog allows you to specify up to 20 tools in the turret. Ger is able to cut with endmills, ballmills (cutters with a hemispherical tip), center drills and ordinary drills. In fact, a "drill" is treated as if it were an ordinary pointy-ended mill; you can make linear cuts or plunging cuts with one. Programs change tools in the ordinary way, by using `M06`. For instance, `M06 T3` changes to the third tool in the turret. If a program doesn't specify a particular tool, then it defaults to the first tool in the turret, which is a quarter-inch endmill.

If your program invokes `G55/56/57/58/59`, then use the "Work Offsets" dialog to set the corresponding values.

Finally, the "Scale" dialog allows you to adjust the level of detail at which the program works. For most uses, it's not necessary for rendered images to be calculated to 0.001 in – at that level of detail a square inch of material would fill an unusually large computer monitor. If you do need more detail, the "Scale" dialog allows you to get it, although the rendering process may take up to 100 times longer and use 100 times the memory.

To show an image of the part cut by your G-code, choose "Render – 2D." If you modify the G-code, and you want to see the result, choose "Render – 2D" again.

3 G-codes and M-codes

Different controllers use slightly different variations of the G/M-codes. Ger tries to take a middle-of-the-road approach, although it is more flexible than many real-world controllers about the order and grouping of codes. For instance, something like this is accepted:

```
G01 X1.0 G21 G01 Y250.0
```

and would be interpreted as

```
G01 X1.0
G21
G01 Y250.0
```

Statements may be terminated by either a new-line (*i.e.*, a "carriage return" or "enter"), or by a semi-colon. Thus, `G01 X1.0; Y2.0` is valid and means something entirely different than `G01 X1.0 Y2.0`. If it is possible for *you* to make sense of your program by reading the codes one "chunk" at a time, then the simulator can probably make sense of it too, although the controller on the actual milling machine may be more strict.

Many of the codes that make sense on a real-world machine are ignored by the simulator. For instance, `M08, M09` (coolant on/off) is ignored, and `M47` (repeat program) doesn't really make sense in this context.

To be clear on how this program interprets the different codes, here's a list of how they are seen by the program's virtual controller. The table below is organized by "actionable chunks," meaning a series of inputs that makes sense together. To make the descriptions more concise, `[...]` is used for an optional term, n means a number, possibly with a decimal, and w means a whole number.

Move `[X`n`] [Y`n`] [Z`n`] [F`n`]`
: Notice the use of `[` and `]`: each of these terms is optional.
This command moves the tool along a line segment to the given X, Y, Z, at the given feed rate. If you are in rapid mode (`G00`), then the `F` is not allowed.

`G00`
: Rapid travel mode. The program begins in rapid travel mode. If the very first line is `X1.000 Y1.000`, then the tool will move to that position in rapid mode.

`G01`
: Ordinary (not rapid) travel mode.

`G02`
: Clockwise arc.

`G03`
: Counter-clockwise arc.
`G02/G03 [X`n`] [Y`n`] [Z`n`] [I`n`] [J`n`] [K`n`] [F`n`] or`
`G02/G03 [X`n`] [Y`n`] [Z`n`] [R`n`] [F`n`]`
Circular interpolation. These do what you'd expect. Something to remember: if an arc is specified using `R` rather than `I/J/K`, then a positive `R`-value produces an arc that subtends less than 180°; using a negative `R`-value produces an arc that's more than 180°.
Also, various machines may treat the ZX plane slightly differently, depending on whether they are "standard" or "vertical machining." So, `G02` and `G03` may be treated in opposite fashion. Basically, what does "clockwise" mean? See Smid, *CNC Programming Handbook*, p. 280.
To cut a complete circle, the X/Y/Z values given with the command must be equal to the starting point of the tool, and you must use the I/J/K format, not an R-value.
Geometrically, the distance of the center to each of the two end-points must be the same (that distance is the radius). When using the I/J/K format, it's not unusual for the center of an arc to require many digits to specify exactly, perhaps even an infinite number. So that the programmer only needs to input a reasonable number of digits, a certain amount of leeway must be allowed. Ger will accept a center if the distances to the two end-points are within 1% of each other, and it will assume that the radius of the circle is equal to the larger of these two distances.

`G04`
: Dwell. This is accepted, but everything from `G04` to the end of the line (or a semi-colon) is ignored.

`G15` Polar coordinates off.

`G16` Polar coordinates on.

You may enter polar coordinate mode while in incremental mode (`G91`) or you may enter incremental mode after invoking `G16`, or go back to absolute mode (`G90`). Not every real-world CNC controller allows this. Ger also allows circular interpolation (`G02/G03`) while in polar coordinate mode, another thing that most real-world machines do not permit. Use of `G01` or `G00` while in polar coordinate mode is *not* allowed.

`G17` Work in the XY-plane (the default).

`G18` Work in the ZX-plane.

`G19` Work in the YZ-plane.

While in cutter comp mode, only the standard XY-plane may be used. This is typical for real-world controllers.

`G20` Inches.

`G21` Millimeters.

Ger works internally using either inches of millimeters, but you can freely change from one unit of measure to the other within your G-code. In fact, Ger is more flexible than many real-world controllers; you can sprinkle `G20`'s and `G21`'s all over the place.

`G28` Machine Zero Return. This is accepted, but everyting from `G28` to the end of the line (or a semi-colon) is ignored. On the simulated machine, it has the effect of moving the tool to the position of the tool turret.[1]

`G40` Cancel cutter comp, also known as "tool radius compensation."

`G41` Cutter comp, left.

`G42` Cutter comp, right.

`G41/G42 D`*w* *or*
`G41/G42 H`*w*

The cutter comp is given relative to a value stored in a D-register or an H-register. The D-register typically contains the diameter of the tool and the H-register contains a value for tool wear. These are provided by using the "Tool Turret" dialog. As a reminder, the left/right distinction here refers to the position of the tool relative to its path. Thus, `G41` puts the tool to the left of the path it follows.

[1] In fact, I'm not sure whether I've implemented this.

Ger does things slightly differently than how many real-world controllers 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 tool radius 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. There seems to be so much variation from one controller to another in how this aspect of cutter comp is handled, that some degree of mis-match between Ger and the real world is unavoidable. In any case, the usual advice applies to *all* machines: invoke cutter comp away from the part and make at least one move before contacting the part.

Note that Ger does not allow things like

```
G41 X## Y## Z## D##
```

You can't have any extra stuff between the `G41` and the `D` or `H`-register specification. Many real-world CNC machines do allow this.

One last point about cutter comp. Ger allows cutter comp to be used on interior angles, while many real-world controllers do not. However, Ger is not as smart as some real-world controllers about how it deals with this case. For instance, if the tool path is specified to form a very acute vertex, say 5°, and the tool makes many small moves (less than the tool radius) near this vertex, then the tool path in cutter comp mode might not be what you expect. On the other hand, it's hard to imagine what a person might intend in a case like this, so it seems better to let the tool do oddball things as a way of informing him that he *input* something oddball.

About the only sitation I can imagine where this would arise is if you feed Ger code that was posted by a (rather stupid) CAM program. The way to "fix" this is to have infinite look-ahead in cutter comp mode so that Ger can remove all of those small moves near the apex from the statement stream, but it's not clear to me that this is something that *should* be fixed. It seems better for the controller to behave in a way that makes it apparent that the input G-code is weird, and probably wrong, rather than acting in a way that hides from the user potential problems with his G-code.

`G43` Positive tool length offset (TLO).

`G44` Negative tool length offset.

`G49` Cancel tool length offset.

`G43/G44 [Z`n`] H`w

The simulator considers the `Z`-value to be optional, although it is required by many real-world machines.

As a reminder, in the case of G43, this has the effect of adding the value in the H-register to all Z-moves until a G49 is reached. If the H-value is negative, then the tool will cut more deeply. G44 has the opposite effect.

A program may not enter TLO while in incremental mode, when using polar coordinates, or while in cutter comp mode. However, once TLO is in effect, a program may enter any of these modes. If your program does enter one of these modes, then it must leave these modes before cancelling TLO.

G52 Local coordinate system.

G52 [X*n*] [Y*n*] [Z*n*]

This changes the origin to be the point at the given X, Y, Z, stated relative to the PRZ. All moves after G52 are as though the PRZ is at this new position. Each of the X, Y, and Z values is optional; omitted values are assumed to be zero (no change). This is cancelled with G54.

G54 Return to normal work offset. That is, use the PRZ as the origin. If a tool length offset is in force (G43 or G44), then it remains in force after calling G54.

G55

G56

G57

G58

G59 Codes G54 through G59 are similar to G52, but the values used for X, Y and Z come from the "work offsets" table. So, these codes appear alone, simply as G55 (say), without any additional codes. Just as with TLO mode, none of these codes, including G52, may be entered or exited while in incremental mode, polar coordinate mode, or cutter comp mode.

Normally, the values used for work offsets are the distance from machine zero to program zero (the PRZ), but the virtual machine has no "machine zero." For the purpose of work offset settings, think of the PRZ as the machine zero. Put another way, the work offset values should be the amount in each direction, X, Y and Z, that you want the tool to be offset from the position it would have without the change due to applying one of G55 through G59.

G90 Absolute mode.

G91 Incremental mode.

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.

`M00`

`M01`

`M02` These are different forms of "program stop." The simulator simply halts when it reaches one of these.

`M03` Spindle on, clockwise.
`M03 S`*w*
where *w* is the spindle speed.

`M04` Spindle on, counter-clockwise. Similar to `M03`. The program notes internally that the spindle is on, but it has no real effect.

`M05` Spindle off.

`M06` Tool change.
`M06 T`*w*
changes to the tool in the *w* position of the turret. The simulator moves the tool in a straight line from its curent position to the tool turret, changes to the new tool, and stops at that position.[2]

`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. The simulator halts immediately if it reaches this code.

`M48`

`M49` Feed & speed override on/off. Accepted, but ignored.

`M98` Call subprogram.
`M98 P`*w* `L`*w*
This calls a sub-program, where `P` gives its number, and `L` gives the number of times to call it.

`M99` Return from subprogram.

Two other codes are accepted: the `O` code for a program number, and `N` may be used for line numbers. Any line numbers given with `N` are ignored. Every program is required to start with an O-value.

If there are any errors in the program, then the line number on which the errors are detected are given by counting lines in the input text file, not with reference to any `N`-specified line numbers in the input G-code. Thus, if an error is reported on line 3 (say), then the problem was detected on the third line from the top, not the line that starts with `N3` (assuming there is such a line).

Finally, comments are expressed with parenthesis, `(...)`, as usual. The simulator accepts multi-line comments, but does not accept nested comments.

[2] These motions don't appear in the translator part of the program; they are used to detect tool crashes.