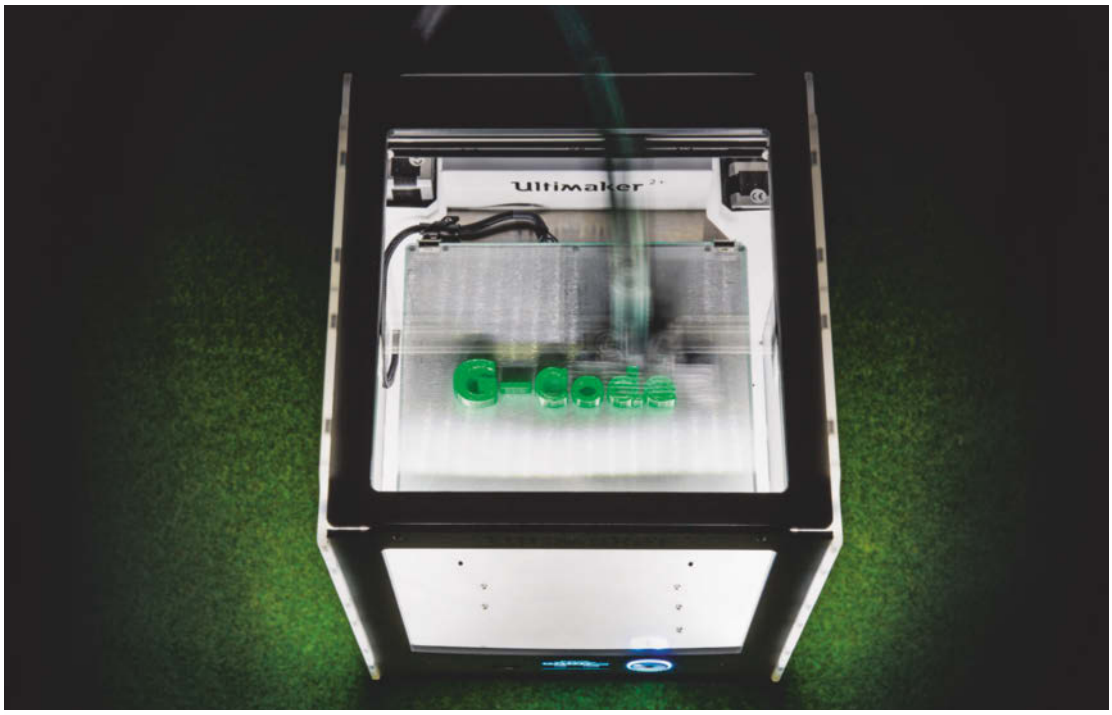


G-Code verstehen und generieren

G-Code, die Muttersprache von CNC-Maschinen wie Fräsen, Drehbänken und 3D-Druckern, ist eine der einfachsten Programmiersprachen überhaupt. Wir erklären die Basics, damit Sie .gcode-Dateien für schnellere Verarbeitung modifizieren und mit Python eigene erzeugen können.

Von **Pina Merkert**

Der Maschinencode für CNC-Fräsen, Drehbänke und 3D-Drucker ist eine für Menschen lesbare Programmiersprache, deren Standardbefehle mit dem Buchstaben `G` anfangen: der G-Code. In der Frühzeit der CNC-Maschinen wurden die Programme von CNC-Programmierern direkt in dieser Sprache geschrieben, ähnlich wie in der Frühzeit der Computer viele Entwickler in Assembler programmiert haben. Inzwischen produzieren CAM-Prozessoren für Fräsen und Slicer für 3D-Drucker

den meisten G-Code, vergleichbar mit Compilern, die Maschinencode für CPUs schreiben. Die G-Code-Generatoren erlauben aber nicht alles, was man mit G-Code programmieren kann. Für einige Spezialfälle lohnt es sich, G-Code per Hand zu programmieren oder mit eigenen Skripten zu erzeugen. Schwer ist das nicht, da G-Code nur aus wenigen Befehlen besteht, die alle ähnlich funktionieren.

Mit dem Wissen aus diesem Artikel verstehen Sie sämtlichen G-Code, den Slicer wie Cura, Skeinforge oder Slic3r erstellen – und können ihn modifizieren, um beispielsweise in einem Druck verschiedene Düsentemperaturen für Ihren 3D-Drucker auszuprobieren und anschließend die beste auszuwählen. Für weitere Teststücke wie ein Kalibrierungskästchen (Einstellen der Schrittzahl für X-, Y- und Z-Motoren und Einstellen der Betthöhe) oder einen Geschwindigkeitsturm (Ermitteln der maximalen Druckgeschwindigkeit Ihres 3D-Druckers) haben wir ein paar Python-Skripte auf GitHub geladen, die Sie über ct.de/wmdg finden. Die Skripte nutzen eine kleine von uns erstellte Bibliothek namens gcodehelpers.py, die Sie nutzen können, um mit wenig Aufwand G-Code für eigene Testdrucke zu generieren. Das Erzeugen von G-Code mit einem Programm ist besonders für 3D-Drucker relevant, da der Code eines aussagekräftigen Teststücks schnell mal ein paar tausend Befehle enthält.

G0 und G1

Im Prinzip funktioniert G-Code wie Turtle-Grafik. Sie haben als Kind kein LOGO programmiert? Die Idee ist, dass Sie einer Schildkröte (deswegen Turtle) Befehle geben, wo sie langlaufen soll. Überall wo sie war, hinterlässt sie einen Strich. G-Code ist weniger albern, da sich dort statt einer Schildkröte das Werkzeug, also der Fräser, Drehstahl oder das Hotend samt Düse bewegt. Die aktuelle Position speichert die Maschine in einem kartesischen Koordinatensystem, selbst wenn sie wie Delta-3D-Drucker keine rechtwinkligen Achsen verfährt. Gehen Sie einfach davon aus, mit X-, Y- und Z-Koordinaten in Millimetern zu tun zu haben.

Die Befehle `G0` und `G1` definieren einen neuen Punkt im Raum, zu dem sich das Werkzeug bewegt. `G0` ist für schnelle Maschinenbewegungen gedacht, bei denen eine Fräse kein Material abnimmt und ein 3D-Drucker nichts extrudiert. `G1` verwenden Sie für alle Bewegungen, bei denen die Maschine Material be- oder verarbeitet. Beide Befehle sorgen für eine lineare Bewegung von der aktuellen zur neuen Position.

Die Koordinaten des Zielpunkts schreibt man ohne Einheit mit Leerzeichen getrennt einfach hinter den Befehl. Da man Koordinaten, die sich nicht ändern, weglassen darf, muss man vor jeder Zahl mit den Buchstaben `X`, `Y` und `Z` die Raumachse angeben. 3D-Drucker nach Fused-Deposition-Modeling-Prinzip (FDM) verwenden eine vierte Dimension namens `E` für die Länge des Filamentfadens in Millimeter, die der Extruder durch die Düse drückt. Zusätzlich darf man hinter `F` angeben, wie schnell die Maschine zur neuen Position fährt. Ein Befehl, zur Position (10 mm, 20 mm, 30 mm) mit einer Geschwindigkeit von 1200 mm/min zu fahren und dabei 5 mm zu extrudieren, sieht so aus:

```
G1 X10 Y20 Z30 E5 F1200
```

Wenn sich die Geschwindigkeit nicht ändert, kann man sie zu Beginn des Programms einmal festlegen und anschließend nur Positionen angeben:

```
G1 F2000
G0 X10 Y10 Z0.3 E0
G1 X20 E0.793
G1 Y20 E1.586
G1 X10 E2.379
G1 Y10 E3.172
```

Ein 3D-Drucker würde mit diesem Programm ein Quadrat mit 10 mm Kantenlänge drucken und dabei 3,172 mm Filament verbrauchen.

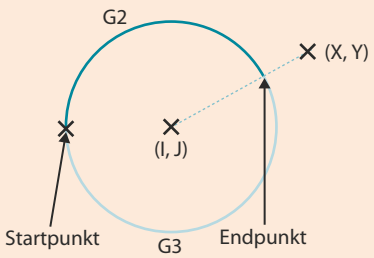
Die Koordinaten gibt man als Gleitkommazahlen mit Punkt statt Komma an. Wie viele Nachkommastellen Sinn ergeben, hängt von der Genauigkeit der Maschine ab. Ordentliche Fräsen weichen von der angegebenen Position weniger als 0,01 mm ab, während 3D-Drucker in X- und Y-Richtung üblicherweise nur auf 0,05 mm genau positionieren.

G2 und G3

G-Code enthält auch Befehle für Kreisbahnen im (`G2`) und gegen (`G3`) den Uhrzeigersinn. Bei denen interpoliert die Maschine Zwischenpositionen nicht linear sondern zirkulär, sodass das Werkzeug einer Kreisbahn folgt, bis es dem mit `X` und `Y` angegebenen Endpunkt am nächsten ist. Über `I` und `J` gibt man dabei die X- und Y-Koordinate des Kreismittelpunkts an. Eine Z-Koordinate gibt es beim Kreisbefehl nicht, da er standardmäßig in der X-Y-Ebene erfolgt. Wer Kreise in der X-Z- oder Y-Z-Ebene fräsen

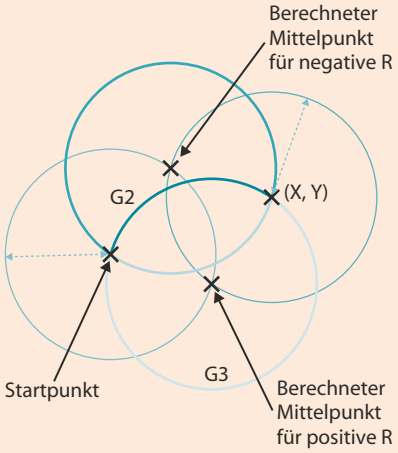
Kreisbefehle mit Angabe des Mittelpunkts

Gibt man bei den Kreisbefehlen G2 und G3 mit den Koordinaten I und J einen Mittelpunkt an, fährt die Maschine eine Kreisbahn um diesen Mittelpunkt. Liegt der mit X und Y angegebene Zielpunkt nicht auf dem Kreis, stoppt die Maschine am nächstgelegenen Punkt auf dem Kreis. Der Radius ergibt sich aus dem Abstand zwischen Startpunkt und Mittelpunkt.



Kreisbefehle mit Angabe des Radius

Definiert man bei G2 und G3 keinen Mittelpunkt, sondern gibt stattdessen mit R einen Radius an, konstruiert sich die Maschine selbst den Kreismittelpunkt. Dieser liegt auf dem Schnittpunkt von zwei Kreisen mit Radius R um Start- und Zielpunkt. Von den zwei Punkten, die infrage kommen, wählt die Maschine bei positiven R denjenigen, um den sich die kürzere Kreisbahn ergibt, bei negativen R den mit der längeren Kreisbahn.



will, muss die Ebene vorher mit `G18` oder `G19` umschalten. Zurück zur X-Y-Ebene schaltet man mit `G17`.

Moderne CNC-Fräsen verstehen `G2` und `G3` auch mit Angabe eines Radius hinter `R`. Bei dieser Variante darf man mit `I` und `J` keinen Mittelpunkt angeben, da die Maschine den anhand des Radius selbst berechnet. Dann fährt die Maschine auch immer den angegebenen Endpunkt an. Da es üblicherweise zwei verschiedene Kreisbahnen gibt, die am Zielpunkt enden würden, fährt die Maschine bei positiven Radien `R` immer die kürzere der beiden Varianten (weniger als 180°), bei negativen Radien die Längere (mehr als 180°).

G-Code für 3D-Drucker enthält aber üblicherweise keine komplizierten Kreisbefehle, da einige

Firmwares diese nicht verarbeiten können. Stattdessen zerlegen die Slicer Rundungen in kleine lineare Bewegungen mit `G1`, was auf jeder Maschine funktioniert.

Maschinen sind Automaten

G-Code spricht die Maschinen als Zustandsautomaten an. Mit dem Befehl `G91` schalten Sie von absoluten auf relative Koordinaten um, zurück zu absoluten Positionsangaben geht es mit `G90`. Derselbe Befehl `G1 X10 Y10` fährt die Maschine also an unterschiedliche Positionen, je nachdem ob vorher `G90` oder `G91` im Programm stand. Ähnlich verhält es sich mit `G20` und `G21`, die bewirken, dass alle nachfolgenden Koordinaten in Zoll oder in Millimetern interpretiert werden.

In der ersten Zeile eines G-Code-Programms hat man keine Ahnung, in welchem Zustand sich die Maschine gerade befindet. Daher sollten Programme immer mit einem Satz an Befehlen starten, die den Modus initialisieren, in dem der folgende Code arbeiten soll:

```
G21 ;metric values
G90 ;absolute positioning
G92 ;set current position as zero
```

Der Befehl `G92` definiert, an welchem Punkt sich die Maschine gerade befindet. Er löst keine Bewegung aus, sondern verschiebt das interne Koordinatensystem der Maschine. Wo der Nullpunkt liegt, hängt nämlich von der Maschine und manchmal sogar vom Werkstück ab. Gibt man keine Koordinaten an, verschiebt die Maschine ihren Nullpunkt auf die aktuelle Position. `G92 X10 Y10 Z10` sagt der Maschine hingegen, dass sie jede ihrer Achsen 10 mm in negativer Richtung bewegen müsste, um den Nullpunkt zu erreichen.

M-Codes

Neben den mit `G` beginnenden Befehlen, die auf allen Maschinen funktionieren sollten, gibt es maschinenabhängige Befehle, die mit `M` anfangen. Beispielsweise schaltet `M82` den Extruder eines 3D-Druckers auf absolute Koordinaten (ähnlich wie `G90` das für X, Y und Z tut) und `M107` den Lüfter zum Kühlen des Drucks aus. Auf einer Fräse ergeben diese Befehle keinen Sinn oder sind sogar mit einer ganz anderen Funktion belegt. Bei G-Code müssen Sie daher immer prüfen, ob der Code auch zu Ihrer Maschine passt. Es ist gut möglich, dass für einen RepRap erstellter G-Code auf einem Ultimaker nicht funktioniert und im schlimmsten Fall die Maschine beschädigt.

Glücklicherweise geben Slicer und Fräsprogramme üblicherweise in Kommentaren (alles, was hinter einem `;` oder in Klammern steht, ist ein Kommentar) an, für welche Maschine der Code erzeugt wurde und welchen G-Code-Dialekt er verwendet. In der Praxis beschränken sich die Unterschiede meist auf einen Block an Befehlen für die Initialisierung und einen, der ausgeführt wird, nachdem die Arbeit getan ist. Außerdem sollte das Werkstück oder Druckobjekt in den Bauraum passen, was Sie leicht sehen, wenn absolute Koordinaten zum Einsatz kommen (`G90`).

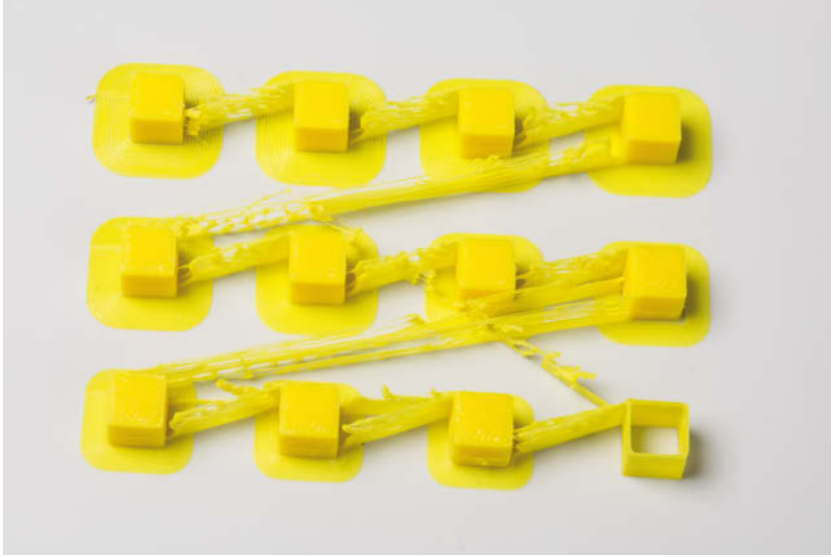
Beim Anpassen von G-Code für Ihren 3D-Drucker werden Sie häufig die Druck- und Betttemperatur ändern wollen. Die Extrudertemperatur setzen die Befehle `M109` und `M104`. Sie unterscheiden sich darin, dass der Drucker beim ersten wartet, bis die hinter `S` angegebene Temperatur wirklich erreicht wurde, während `M104` die Temperatur nur einstellt und der Drucker ohne zu warten den nächsten Befehl ausführt. Genauso verhält es sich mit den Befehlen für die Temperatur des Heizbetts, bei denen `M190` wartet und `M140` nicht.

Mit diesem Wissen können Sie die Aufheizzeit Ihres Druckers verkürzen, indem Sie das Aufheizen des Betts, des Extruders und das Nullen der Achsen parallelisieren. Wir heizen unseren Tevo Tarantula fürs Drucken von ABS beispielsweise mit folgender Sequenz auf:

```
M190 S100 ;set bed to 100°C and wait
M140 S105 ;set bed to 105°C
M109 S220 ;set nozzle to 220°C & wait
M104 S230 ;set nozzle to 230°C
G28 X0 Y0 ;move X/Y to min endstops
G28 Z0    ;move Z to min endstop
```

Während der etwa 10 Minuten, bis das Bett 100 °C erreicht, bleibt das Hotend kalt, damit kein Kunststoff heraustropft. Sobald das Bett 100 °C erreicht, heizt es weiter bis 105°C, während die Düse auf 220 °C aufheizt. Das Aufheizen des Hotends dauert lang genug, damit das Bett bis dahin etwa 104 °C warm wird. Danach beginnt der Drucker schon mal die Achsen zu nullen, während die Düse die letzten 10 °C bis zur eigentlichen Drucktemperatur aufheizt. Anschließend hat das Bett die eigentlich geplanten 105 °C erreicht, während die Düse nicht weniger als 228 °C heiß ist – warm genug, um den Druck zu starten. Cura setzt die Temperaturen mit `M140 S105` und

Bei diesem Teststück hat der Drucker hinten rechts gar kein Filament vor Bewegungen eingezogen und vorne links 9,5 mm³. Optimal ist wenig Einzug, aber auch keine Fäden – hier also 7 mm³.



An den kleinen Würfeln mit Kragen erkennt man, wie stark sich benachbarte Druckbahnen überlappen dürfen, bevor überschüssiger Kunststoff herausquillt. Der Würfel links oben nutzt gar keine Überlappung, während der rechts unten 100 Prozent des Zwischenraums ausfüllt. Die Fäden zwischen den Würfeln entstanden, weil wir vor Bewegungen zu wenig Filament eingezogen hatten.

`M109 S230` und braucht damit ungefähr eine Minute länger zum Aufheizen.

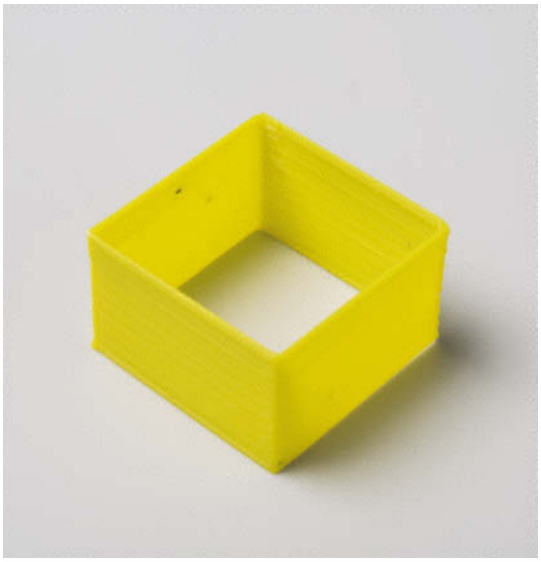
G-Code mit Python

Schön, dass G-Code so leicht zu verstehen ist. Besonders Programme für 3D-Drucker geraten jedoch schnell sehr lang. Daher bietet es sich an, den Code mit einem Skript zu erzeugen, das in Sekunden Tausende von Codezeilen erzeugt. Ein solches Skript schreibt einfach Zeilen mit `G1`-Befehlen in eine Textdatei; die Frage ist nur, welche Koordinaten es jeweils für `X`, `Y`, `Z` und den Extruder `E` einsetzen soll. Für die Antwort muss man sich eine Detailvergrößerung vorstellen, was beim 3D-Drucken genau passiert.

Der Extruder drückt den Filamentfaden in die heiße Düse, wo der Kunststoff zähflüssig wird. Abhängig vom Druck, den der Extruder aufbaut, fließt mehr oder weniger Plastik durch die Düse und verteilt sich auf der Oberfläche darunter, beispielsweise der zuvor gedruckten Schicht. Um zu berechnen, wie viel Filament der Extruder durch die Düse schieben soll, muss man das Volumen des Plastikfadens kennen, der gedruckt wird. Wenn alles funktioniert, ist der so hoch wie die Schicht und drückt sich seitlich neben der Düsenöffnung leicht heraus. Das recht zähflüssige Plastik nimmt dabei zu den Seiten hin eine nahezu runde Form an. In der Mitte ergibt sich ein rechteckiger Bereich, der leicht breiter ist als die Düse (Breite `w`), also beispielsweise 0,42 Millimeter bei einer Düse mit 0,4 Millimeter großem Loch. Der Flächeninhalt dieser Form (ein Rechteck mit je einem Halbkreis links und rechts) beträgt

$$a = w \cdot h + \pi \left(\frac{h}{2}\right)^2$$

Das Volumen ergibt sich, indem man diese Fläche mit der Länge der gedruckten Linie multipliziert.



Dieses Kästchen eignet sich gut als erstes Teststück für einen neu zusammengebauten 3D-Drucker. Mit seinen Maßen stellen Sie die Schritte pro Millimeter für X- und Y-Achse ein.

$$v = l \cdot a = l \cdot \left(w \cdot h + \pi \left(\frac{h}{2} \right)^2 \right)$$

Damit die Düse genau diese Menge Kunststoff extrudiert, muss das Volumen des vom Extrudermotor nachgeschobenen Filaments exakt gleich groß sein.

$$v = e \cdot \pi \left(\frac{1,75mm}{2} \right)^2 \quad \Leftrightarrow$$

$$e = \frac{l \cdot \left(w \cdot h + \pi \left(\frac{h}{2} \right)^2 \right)}{\pi \left(\frac{1,75mm}{2} \right)^2}$$

Damit Sie das nicht per Hand rechnen müssen, haben wir das alles in der Funktion `print_move()` unserer Python-Bibliothek gcodehelpers.py zusammengefasst (siehe ct.de/wmdg). Der Funktion geben Sie einfach einen Startpunkt, einen Endpunkt und die aktuelle Position des Extruders, und sie gibt die neue Position des Extruders und den `G1`-Befehl mit allen Koordinaten zurück. Zusammen mit einer Schleife, die die Schichthöhe jeweils um 0,2 Millimeter erhöht, haben Sie schnell den G-Code für ein hohles rechteckiges Rähmchen generiert (Datei generate_calibration_box.py im Repository). Damit der Code, den dieses Skript erzeugt, problemlos auf Ihrem eigenen 3D-Drucker läuft, müssen Sie die Start- und Stoppsequenz in gcodehelpers.py anpassen. Tragen Sie dort die für Ihr Filament passenden Heizbefehle ein und ändern Sie die Homing-Sequenz, damit Ihr Drucker seinen Nullpunkt findet.

Unsere Bibliothek enthält noch einige weitere Funktionen, die es Ihnen einfach machen, Python-Skripte für eigene Teststücke zu entwickeln. Beispielsweise berechnet `dilate_erode()` mit der Bibliothek `shapely` die Koordinaten für Umrisslinien mit einem Abstand zu einem übergebenen Polygon. Die Funktion `print_brim()` nutzt das, um G-Code für einen Kragen um das Objekt zu erzeugen, der die Betthaftung erhöht. Ganz ähnlich erzeugt `print_wall()` Code für Wände mit mehreren Bahnen Dicke. Die Kombination aus `line_pattern()`, `infill()` und `print_layer()` füllt den Zwischenraum zwischen den Wänden auch aus. Im Skript generate_line_overlap_cubes.py haben wir das genutzt, um herauszufinden, wie stark sich nebeneinanderliegende Druckbahnen überlappen müssen, damit möglichst wenig Luft zwischen ihnen bleibt, aber auch kein Plastik überquillt. Mit ABS auf dem Tevo Tarantula konnten wir nicht mehr als 40 Prozent des Raums zwischen den Bahnen füllen – eine Erkenntnis, für die wir mit einem normalen Slicer tagelang einzelne Drucke hätten starten müssen.

Mit selbst programmierten Tests dieser Art finden Sie für die zahlreichen Parameter eines Slicers die bestmöglichen Werte heraus. Dafür verbrauchen Sie nur wenig Filament, können die Qualität Ihrer Drucke aber eventuell deutlich steigern. Das gilt besonders für billige Drucker, da sich die Hersteller hier meist nicht die Mühe machen, optimierte Presets mitzuliefern.

Schicken Sie uns ein Pull-Request via GitHub, wenn Sie gcodehelpers.py verbessern oder erweitern – beispielsweise um weitere Infill-Muster. Wir freuen uns auch über Ihre Ideen für Teststücke zum Kalibrieren – oder über Skripte, falls Sie gleich zur Tat schreiten. (pmk) c't

Repository, Dokumentation:
www.ct.de/wmdg