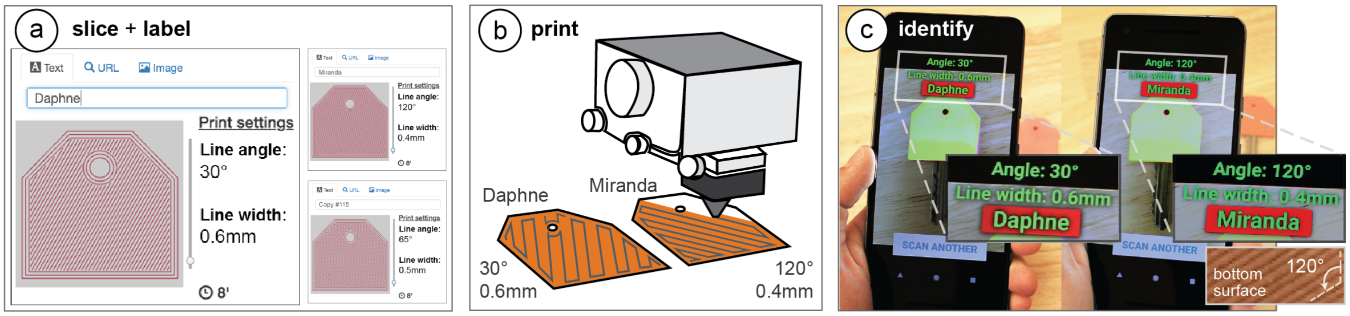
G-ID: Identifying 3D Prints Using Slicing Parameters

Mustafa Doga Dogan[1], Faraz Faruqi[1], Andrew Day Churchill[1],
Kenneth Friedman[1], Leon Cheng[1], Sriram Subramanian[2], Stefanie Mueller[1]

[1]MIT CSAIL, Cambridge, MA, USA
{doga, ffaruqi, adchurch, ksf, leonc, stefanie.mueller}@mit.edu

[2]University of Sussex, Brighton, UK
sriram@sussex.ac.uk



Daphne
Print settings
Line angle: 30°
Line width: 0.6mm
8'

Miranda
Print settings
Line angle: 120°
Line width: 0.4mm
8'

Copy #115
Print settings
Line angle: 65°
Line width: 0.5mm
8'

Angle: 30°
Line width: 0.6mm
Daphne

Angle: 120°
Line width: 0.4mm
Miranda

Figure 1. 3D printed objects inherently possess surface patterns due to the angle of the print path and the thickness of the trace the 3D printer lays down. G-ID exploits such features that would normally go unnoticed to identify unique instances of an object without the need to embed an obtrusive, additional tag. G-ID provides (a) a user interface for slicing individual instances of the same object with different settings and assigning labels to them. After (b) 3D printing, users can (c) identify each instance using the G-ID mobile app that uses image processing techniques to detect the slicing parameters and retrieve the associated labels.

ABSTRACT

We present G-ID, a method that utilizes the subtle patterns left by the 3D printing process to distinguish and identify objects that otherwise look similar to the human eye. The key idea is to mark different instances of a 3D model by varying slicing parameters that do not change the model geometry but can be detected as machine-readable differences in the print. As a result, G-ID does not add anything to the object but exploits the patterns appearing as a byproduct of slicing, an essential step of the 3D printing pipeline.

We introduce the G-ID slicing & labeling interface that varies the settings for each instance, and the G-ID mobile app, which uses image processing techniques to retrieve the parameters and their associated labels from a photo of the 3D printed object. Finally, we evaluate our method's accuracy under different lighting conditions, when objects were printed with different filaments and printers, and with pictures taken from various positions and angles.

CCS Concepts

• Human-centered computing → Human computer interaction (HCI); *Human-centered computing;*

Author Keywords

personal fabrication; 3D printing; identification; making; tags.

INTRODUCTION

Machine-readable tags have many applications ranging from package delivery and tracking [39, 40], to interactive museum exhibits [15, 55], and games [20, 46]. While in the last decades, researchers have mainly focused on developing 2D tags [11], the rise of 3D printing now enables researchers to investigate how to embed tags directly with the geometry of 3D objects. For instance, QR codes are no longer limited to 2D applications and can now also be used as part of a 3D object's surface geometry by modifying the 3D model [23].

One key challenge when using tags is how to make them unobtrusive [4]. To make tags less visible, researchers have investigated how to leave the surface intact and instead change the inside geometry of a model. *InfraStructs* [53], for instance, scans the object's interior with a terahertz scanner, while *AirCode* [26] requires a projector and camera setup to detect internal air pockets using subsurface scattering. Even though these approaches leave the object's surface intact, both need large equipment, which prevents these solutions from being used in everyday scenarios.

To be able to use regular scanning equipment, such as a mobile phone camera, researchers proposed to analyze small imprecisions on the object's surface that are created during the fabrication process. Those imprecisions are unobtrusive yet machine readable and can therefore be used as tags. Such imperfections make it possible to identify which fused deposition modeling (FDM) printer was used to create an object [27]. For the purpose of easier explanation, we will refer to FDM printers as 3D printers for the remainder of the paper.

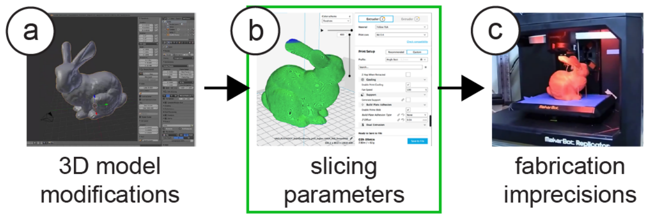


Figure 2. Different ways to embed tags into 3D models while leaving the surface intact: (a) changing the internal geometry (*Infrastructs* [53]), (b) varying slicing parameters (G-ID), (c) relying on fabrication imprecisions (*PrinTracker* [27]).

In this work, we propose a different approach to identification. When a 3D model is prepared for 3D printing, it first undergoes *slicing*, a computational process which converts a 3D model into layers and their corresponding print path (a **G**-code file), which the extruder then follows to create the 3D object. The parameters for the slicing process can be modified for each individual instance, which allows **G**-ID to create unique textures on the surface of objects that can be detected with a commodity camera, such as those available on a mobile phone. Since our approach allows us to be in control over which printed instance has been modified with which slicer settings, we can identify each instance and retrieve associated labels previously assigned by a user.

Our contributions can be summarized as follows:

- A method to utilize the subtle patterns left as an inevitable byproduct of the conventional 3D printing process to identify objects without the need to embed an additional tag.
- A tool for users who want to create multiple instances of an object but intend to give each one a unique identifier.
- A mobile app that helps users take pictures of objects to be identified, as well as a stationary setup to detect finer variations in slicing parameters using image processing.
- An evaluation of the space of slicing parameters that can be varied to generate unique instances of a 3D model and the corresponding detection accuracy under different environmental and hardware-related conditions.

We demonstrate these contributions with a diverse set of interactive applications.

BACKGROUND: TAGGING 3D OBJECTS

There are many factors that influence the design of a tag that can be embedded in a physical object. These range from deciding what purpose the tag is to serve, how to embed it with the 3D model geometry, the fabrication process, and the tag detection mechanism. In this section, we review the literature to highlight factors that aid the development of tags.

Purpose of the Tag (Data Storage vs. Identification)

A tag is a label used to (1) store data or (2) identify items. Inherently, every tag contains some amount of information. If storing data is a tag's main purpose, then it must have a capacity large enough to represent the encoded message. For example, QR codes are a type of matrix barcodes that are able to store more information than 1D barcodes. The more elements the matrix has, the more data can be stored (a 21x21 or 177x177 QR code with high level error correction can hold 72 or 10,208 bits, respectively). This allows for storing larger data such as a URL in the code itself (e.g., [26]).

Not all tags are intended to store large amounts of information. For instance, to identify similar looking items, it is often sufficient to extract a few features from the object. An object's physical appearance can serve this purpose, i.e., color codes function as a tagging mechanism without the intent of storing information; color coding of otherwise similar looking electrical wires helps identify which cable is connected to which voltage. Similarly, 3D objects' visual characteristics can be appropriated for identification [24, 27].

In this work, we use the subtle patterns left by 3D printing to identify objects. Our goal with using different slicing parameters is not to store a lot of information but to create a parameter space large enough to identify individual instances. The identifiable features can then be used to retrieve more information about the object at hand.

Model Geometry (Color, Surface & Internal Geometry)

After deciding which type of tag to use, the tag can be embedded in the model by either changing the 3D model's color, surface geometry, or internal geometry.

Current approaches that change the surface color of the object to embed a tag significantly change the object's visual appearance. For instance, *LayerCode* [28] prints layers in different colors to create barcodes on the surface of the object. Less obtrusive are techniques known from 2D image processing that hide tags in existing images (e.g., using halftoning [6, 36]). While not yet explored, such techniques can likely also be applied to the color textures of 3D printed objects using recent advances in multi-color 3D printing [3].

For 3D printers without multi-color printing capabilities, tags can also be embedded into a 3D model's surface geometry. Embossing tags (3D QR codes [23, 44]), however, is obtrusive. Subtler results can be achieved by either embedding geometric noise on the surface (Aliaga et al. [2]), but the resulting features are harder to detect. To keep the surface unchanged, tags can also be embedded as part of the internal object geometry. *AirCode* [26] and *InfraStructs* [53] change the model to place air pockets beneath the surface of the object. Yet such voids cannot be printed directly, which could jeopardize the object's integrity and makes its fabrication more complicated. *AirCode* addresses this issue by splitting the model into two parts that are printed separately and then assembled together. *InfraStructs* proposes to use multiple materials with different refractive index values, or to add openings to the model that allow for support material to be washed away from small cavities, however, this adds post-processing time. With our approach, the surface and interior stay intact and the model is not split into multiple parts.

Fabrication Process (Filament, Slicing, & Printing)

During the printing process, one can change either the filament, the slicer settings, or exploit features of the fabrication process to create unique identifiers.

Different filaments can be used to create tags by either combining them in two different colors to create a barcode or by printing layers from two types of specialty filaments that look different under NIR light (*LayerCode* [28]). Optimizing the filament choice for tags, however, is not always possible when specific filaments need to be used (e.g., PC filament for extra strength does not exist in different colors) or when the 3D printer only allows for single-material extrusion.

The slicing process can be used to influence how a 3D model is converted into layers and the resulting print path, which can be used as a tag. Thus, it is possible to alternate between different print settings, such as two-layer height values to represent two types of binary bits in a barcode (*LayerCode*). This approach is inspired by Alexa et al. [1], who showed that by adjusting the layer height setting during printing the fabrication time can be reduced.

Finally, the printing process can be used to create tags. Inspired by *physical unclonable functions* (random manufacturing impurities) to generate semiconductor chips with unique identifiers [13, 52], researchers showed that variations in mechanical components of the printer (motors, nozzles) leave a unique "stamp" on the printed object [27, 35]. Such deviations can be used to differentiate between multiple copies due to accidental under and over-extrusion on layers [44]. However, since these signatures are uncontrollable, they first have to be read, learnt, and stored before identifying the object. Similarly, in other fabrication processes, such as metal 3D printing, the inherent randomness present in the fused micro-structures have been used to create tags [8].

Our approach does not require modifying the 3D printer or changing materials during printing.

Detection Mechanism (Camera/Scanner, Microphone)

Tags can be detected using various mechanisms, including different types of cameras and scanners for visual detection, and microphones for audio detection.

To visually detect a tag that is on the surface of the object, users can use a commodity camera [23, 28] or a 2D scanner [27]. To visually detect tags embedded inside an object, the use of large or expensive equipment is necessary. Such equipment can include Terahertz scanners (*InfraStructs* [53]), high-resolution 3D scanners [18], micro-CT scanners [5], or large multi-component setups (*AirCode* [26]). These are not accessible for nontechnical users in everyday use.

For acoustic detection of tags on the surface of objects, researchers investigated how to create ridges and knobs that when strum sound differently (*Acoustic Barcodes* [16], *Tickers and Talkers* [41]). For the detection of tags on the inside of objects, researchers investigated how to design cavities that sound differently when squeezed (*SqueezaPulse* [17]), blown into (*Acoustic Voxels* [25], *Blowhole* [47]) or when a vibration frequency is applied (*FabAuth* [24]).

To enable everyday use of 3D printed tags, the tags need to be detectable using commodity hardware, capturing the tags must follow a simple workflow, and the detection method needs to work across a variety of different environments. G-ID only requires a conventional smartphone camera, i.e., it does not require any large or expensive imaging equipment.

G-ID: LABELING AND IDENTIFYING OBJECTS BY THEIR SLICING PARAMETERS

The main contribution of G-ID is a framework to *label* and *identify* 3D printed objects by their distinct slicer settings.

G-ID *labels* 3D printed objects by intentionally varying the slicing settings of an unmodified 3D model which determine the path the extruder will follow. This allows G-ID to produce multiple instances that all have a unique artifact, e.g., the small grooves on the surfaces of the object that can be shaped differently when the print path is laid down.

G-ID then *identifies* the 3D printed object by such textures, i.e. after users take a picture of the object with a commodity camera, G-ID applies image processing techniques to first extract and then correlate the features with their initial slicing settings to retrieve the identifying label.

Main Benefits of Using Different Slicing Parameters

Slicing parameters reveal themselves on any printed object as a fabrication byproduct that is normally ignored. One may make use of these inevitable textures that come for free due to 3D printing. G-ID combines a wide range of slicing parameters to create a sufficiently large parameter space. For each slicing parameter for surface and infill, there is a variety of values available (see section "Spacing of Slicing Parameters"). The use of so many values is enabled by G-ID's recognition algorithm, which uses a Fourier-based method for precise measurements. Detecting these values precisely in turn enables new applications such as "Finding optimal print settings" (see "Application Scenarios").

G-ID Workflow for an Identification Application

In the following section, we describe how we use (1) *the G-ID labeling interface* to assign each instance of a 3D printed object a unique tag prior to 3D printing, and how we use (2) *the G-ID identification app* that runs on a mobile device to detect each object's tag after 3D printing.

We explain the workflow of G-ID using an application scenario, in which we will create a set of forty key covers—each with an unobtrusive feature that identifies its owner. We use these key covers in our research lab: At the end of the semester when members depart, we often forget to write down who has returned their keys. Using G-ID, we can quickly identify whom the previously returned keys used to belong to and then send a reminder to those who have outstanding keys.

Labeling Interface (Slicer)

To assign each key cover a unique tag, we open G-ID's labeling interface (Figure 3) on our computer and load the 3D model of the key cover by dragging it onto the canvas. Since we want to create 40 key covers, we enter 40 instances into the left-hand panel of the interface.

#1 Generate Instances: We select "Mobile phone" as the desired detection setup. Next, we click the "Generate previews" button, which slices each instance of the key cover with a unique set of slicing settings. In this case, since we only request 40 instances, G-ID only varies the slicing parameters *initial bottom line angle* and *initial bottom line width* (more on this in section "Slicing parameters used for labeling"). After slicing is completed, G-ID previews each instance as shown in Figure 1a.

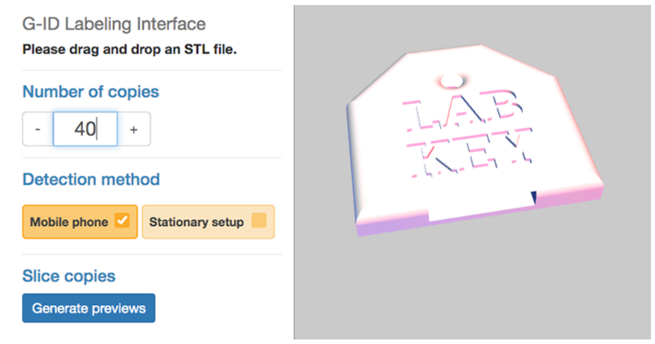

G-ID Labeling Interface
Please drag and drop an STL file.
Number of copies
40
Detection method
Mobile phone ✓ Stationary setup
Slice copies
Generate previews

Figure 3. G-ID labeling interface: load a 3D model and enter number of unique instances needed.

#2 Enter Labels: We can now enter a label in the form of a text, an image, or a URL next to the preview of each instance. Since we want to give each key cover the name of one of our lab members, we enter one name per instance. We can update these labels later any time, for instance, when a new team member joins our lab and we need to reassign the key.

#3 3D Printing: Upon clicking the "Export" button, each instance's G-code file is saved, as well as a digital file (XML) that stores the object information and the entered label corresponding to each instance. We can now send the G-code files to our FDM printer to obtain the printed instances. We also transfer the digital file that stores the object information to our smartphone to be used later for identification.

Identification Interface (Mobile App + Object Alignment)

At the end of the semester when we update our key inventory, we use the G-ID mobile app on our phone to identify which of the returned keys belonged to whom. After launching the app, we first select the model we would like to scan, i.e., the key cover, from our object library (Figure 4a). The app then helps us to align the camera image with the object by showing an outline of the object on the screen, similar to how check cashing or document scanning apps work (Figure 4b). When the outlines are aligned in this human-in-the-loop setting, the app automatically captures and processes the image (Figure 4c). It then identifies the features in the photo associated with the surface-related slicing parameter settings, retrieves the user-assigned label, and shows it on the screen (Figure 1c). We check off the lab members who returned their keys and send a reminder to everyone else.

Surface & Interior: Detecting Infill Using a Light Source

In the above described scenario, G-ID was able to successfully label each instance using only slicing parameters that

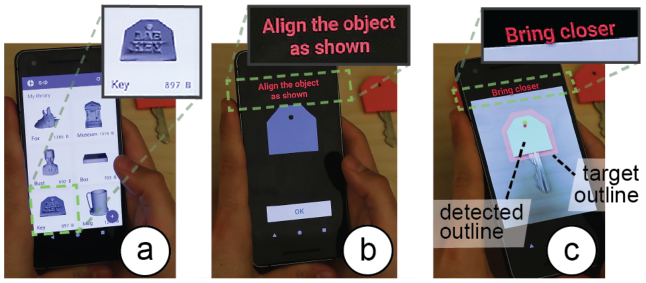


Figure 5. G-ID mobile app for identifying instances: (a) select model from library, (b) face the object, (c) once outlines are aligned, the app automatically takes the image.

affect the object's surface, such as the initial bottom line width and angle because the number of instances was small. However, for scenarios requiring more instances, G-ID can also sense the interior of objects (*infill*) at the expense of adding a small light source as described in the next scenario.

For our department's annual celebration, we are asked to print a set of 300 coffee mugs as a giveaway. Each coffee mug, when inserted into a smart coffee machine (camera and light source below the tray table), automatically fills the mug with the user's preferred drink. Similar to the previous scenario, we use G-ID's labeling interface to generate the instances, but this time G-ID also varies the parameters *infill angle*, *infill pattern*, and *infill density* once it used up the parameter combinations available for the surface. As users insert their mug into the smart coffee machine, which has a stationary setup, the integrated light makes the infill visible due to the translucent nature of regular PLA 3D printing filament (Figure 5). G-ID takes a picture, extracts the infill angle, pattern, and density in addition to the previously mentioned bottom surface parameters, and after identification, pours the user's favorite drink.

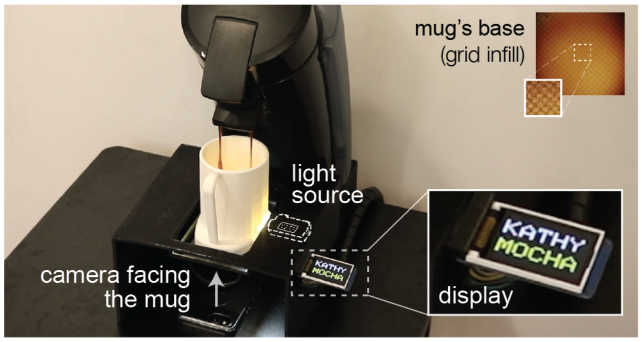


Figure 4. By adding a light, we can also detect variations in infill, such as different infill angles, patterns, and densities, which allow for a larger number of instances. Here, the coffee maker recognizes the mug's infill and pours the user's preferred drink.

SLICING PARAMETERS USED FOR LABELING

In the next section of the paper, we report on the types of slicing parameters for surface and infill that can be used for creating unique identifiers.

Surface Parameters

Bottom Surface: Resolution & Angle

When the bottom layer is printed by moving the print head along a path, two parameters influence how the path on this first layer is laid out. *Initial bottom line width* defines the width of a single line on the bottom surface and thus the resulting resolution. *Initial bottom line angle* sets the direction when drawing the lines to construct the surface. Combinations of these two parameters are shown in Figure 6.

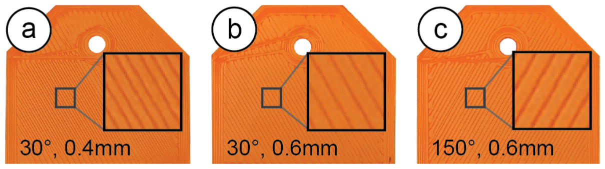


Figure 6. Combinations of different line widths and angles.

Intermediate Layers: Resolution & Angle

It is possible to vary the slicing parameters for the intermediate layers in the same way as for the bottom surface. *Layer height* when varied leads to different layer thicknesses across the printed object and thus affects the overall print resolution. *Rotating the 3D model* on the build plate leads to different layer angles across the side surface. Combinations of these two parameters can be seen in Figure 7.

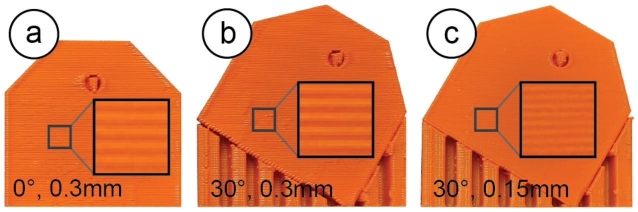


Figure 7. Different layer angles achieved by rotating the model at the expense of additional support material and print time (a vs. b), and various print qualities (b vs. c).

However, using the slicing parameters for layers comes at several drawbacks. As can be seen, changing the layer orientation results in a significant increase in print time due to the extra support material required. Further, changing the layer resolution can result in a notable difference in print quality across different instances. We still include it here to provide a complete overview of all available parameters.

Infill Parameters

Next, we review slicing parameters that change an object's internal print path.

Infill: Resolution, Angle, & Pattern

Three parameters influence how the infill is laid out. *Infill line distance* determines how much the lines of the infill are spaced out and thus determines the internal resolution. The denser the infill lines, the higher the infill density. *Infill angle* rotates the infill lines according to the direction specified in degrees. Different combinations of these two parameters are shown in Figure 8a. *Infill pattern* allows for different layouts of the print path (Figure 8b), such as grid or triangle shapes.

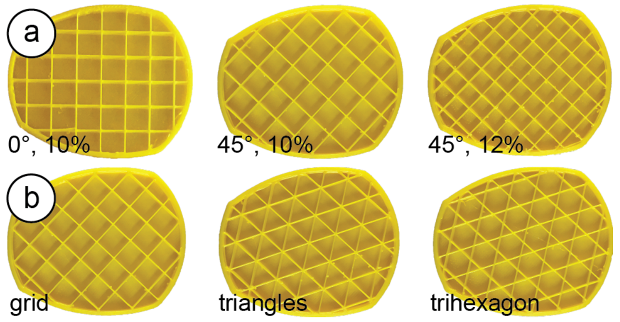


Figure 8. Cross-sections of the mug model show (a) different infill angles and densities, (b) different infill patterns.

Selecting Parameters that Minimize Print Time & Material

When the user enters the number of instances, G-ID varies those slicing settings first that have the least amount of impact on print time and required material. For instance, the bottom line angle does not add any print time, does not change the resolution, and does not require additional material, whereas changing the bottom width changes the resolution slightly. Infill parameters are varied afterwards since they affect most layers of the model, starting with the infill angle, followed by infill pattern and width.

DETECTING & IDENTIFYING SLICING PARAMETERS

To detect these slicing parameters on a 3D print, we apply common image processing techniques. Our pipeline is implemented using *OpenCV* and uses *SimpleElastix,* a state-of-the-art image registration library [25].

Aligning the Object's Base in Handheld Camera Images

In the first processing step, G-ID needs to further refine the position and orientation of the object in the photo the user has taken to match the outline of the 3D model that was shown on the screen. For such alignment, most existing tagging approaches include specific shapes with the tags. For example, QR codes [17] have three square finder patterns and *AirCodes* [22] have four circles that are used to align the image. We did not want to add such markers and therefore decided to infer the position and orientation of the object based on the contour of its surface, which G-ID can extract from the 3D model.

Images Used for Alignment

When the user processes the 3D model in the G-ID labeling interface, it automatically saves the outline of its base in the XML file as a binary image (stored as a Base64 string). When the user loads the XML file in the app, the object appears in the user's model library. After choosing the desired model, the app shows the user the stored outline to assist them with facing the object from the right angle. The app automatically captures the image when the contours are matched (i.e., the bitwise XOR error between the detected contour and target contour is below an acceptable value). G-ID then applies several pre-processing steps, such as applying a bilateral filter to smooth different color and shade regions without blurring their boundaries.

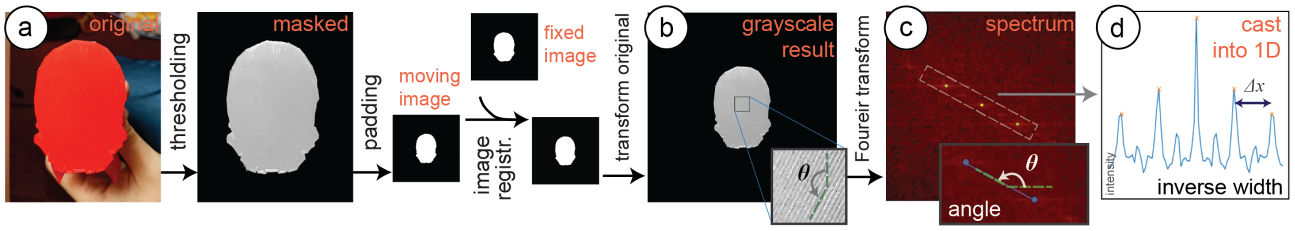


Figure 9. Image registration & processing pipeline. (a) The captured outline is registered with that of the 3D model model for (b) improved alignment. (c) Its Fourier transform is used to infer line angle *θ* and width *d*. (d) The distance Δ*x* between intensity peaks on the inclined line is inversely proportional to *d*.

Removing Overhangs

In the next processing step, G-ID removes overhangs, i.e., filters out object geometry that is not part of the bottom surface but appears in the image due to the camera angle. G-ID filters these parts out based on their shading in the camera image: Since the bottom surface is flat and located at a different height than an overhang, the two surfaces have different reflection properties, and thus appear brighter or darker in the image (Figure 9a). To find the shading of the bottom surface, G-ID samples a small region of interest to extract the HSV values to get the corresponding threshold range. G-ID then uses the threshold to mask the contour.

Reducing Perspective from the Image (Undistorting)

After extracting the contour, G-ID applies an affine transformation (translation, rotation, scaling, shearing) to compute the deformation field required for alignment. As input to this image registration, we use the masked image from the previous step (converted to a binary outline) as the moving image. The fixed image is the outline of the 3D model at a constant scale. While a projective transformation would best rectify the perspective, it can be approximated by an affine transform since the perspective deviation is minimized due to human-in-the-loop camera image capturing.

To find the best affine transformation, we use adaptive stochastic gradient descent as the optimizer. As the objective function, we use mean squares since we work with binary images, which have little gradient information. The computed parameter map of the affine transformation is then applied to the image the user has taken to align it with the digital 3D model outline (Figure 9b).

Detecting Bottom Line Angle and Width

Since the traces of the 3D printed surface have a periodic layout, we are able to detect their orientation and widths by looking at the frequency spectrum, i.e., we take the 2D Fourier transform of the image. From this spectrum, we can determine the bottom line angle *θ* by extracting the slope of the line on which the periodic peaks lie (peaks are marked yellow in Figure 9c). We can determine the bottom line width *d* by casting the intensity values on this inclined line into a 1D array and computing the distance *Δx* between the maxima, which is inversely proportional to *d*. This approach is more robust than looking at the original image itself because in case the lines have irregularities, their distances may be inconsistent, whereas the Fourier transform acts as a smoothing filter and provides an averaged value.

Error Checking

If the picture the user has taken is of poor quality (out-of-focus, poor lighting, or accidental shaking of camera), the lines on the object surface will not be clear enough to extract correct measurements of parameters. Fortunately, these false readings can be avoided due to the nature of the 2D Fourier transform. In the Fourier spectra of digital photos, there is a strong intensity along the x and y-axis since real-world objects have many horizontal or vertical features and symmetries [38, 48]. If surface lines are not distinguishable, peak intensities appear on the x and y-axis and therefore erroneously result in detection of 0° or 90°. Thus, the detection of either of these two angles indicates a false reading. Therefore, we exclude these two values from our allowed slicing parameter space. Whenever our algorithm detects these two angles, we notify the user that the image has to be retaken for correct measurement.

Detecting Infill Angle, Width, and Pattern

To detect the infill parameters, we first remove noise from the image that is caused by the bottom lines on the surface (Figure 10). We remove them by (a) increasing the contrast of the image, (b) blurring the image with a 2D Gaussian smoothing kernel, and (c) applying adaptive thresholding.

To detect the infill *pattern,* we compare the resulting shapes after thresholding to the known infill pattern templates shown in Figure 10d/e. To determine infill *density,* we compare the infill templates at different scales to the size of the shapes in the image, the matching template then indicates the size of the pattern. Similarly, the infill *angle* is detected by rotating the template and finding the angle that gives the smallest sum of squared difference to shapes in the image. Since infill is detected in the stationary setup, alignment of

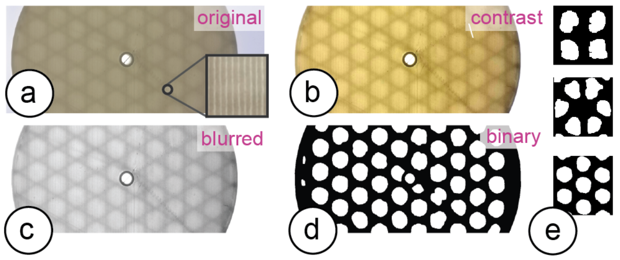


Figure 10. Image processing to extract the components of the infill pattern: (a) photo, (b) contrast increased, (c) blurred, (d) binarized, and (e) matched to the template of the respective infill type: grid, triangles, trihexagon from top to bottom.

the base is less of a concern and image registration can either be simplified or ignored for shorter processing times.

SPACING OF SLICING PARAMETERS

We conducted an experiment to determine which parameter spacing can be reliably identified using our detection method. For the experiment, we printed a number of instances with different slicing settings and used our detection method to identify each pattern. Table 1 summarizes the results of this experiment under regular office light conditions. We focus this analysis on the parameters related to bottom surface and infill, which are seen from the object's base, and do not consider those related to the side (intermediate layers).

	Min	Max	Spacing	Variations
Bottom:				
Angle (°)	0°	180°	5°	36-2=34*
Width (mm)	0.35	0.6	0.05	6
Bottom Total:				= 204
Infill:				
Angle (°)	0°	60°/90°	5°	12 / 18
Width (mm)	2.6	3.2	0.6	2
Pattern (type)	-	-	-	3
Infill Total:				= 84
Total:				**= 17,136**

Table 1: Each slicing parameter's range (min, max) and incremental spacing values as determined by our experiments. The last column shows the number of variations that can be realized. (*2 angles reserved for error checking).

Using the object's base for identification has many advantages: (1) it is easier for users to take aligned pictures of the base since it is flat, (2) it is more time and material efficient to manipulate base-related features since they only affect a single layer and have no influence on the print quality of the main surface of the object, (3) there are more combinations of identifiable features since both bottom surface and infill can be seen just from the base, (4) it is convenient to computationally process a flat surface.

We therefore decided to first focus on the bottom layer and infill parameters, however, further analysis can be done concerning side surfaces by repeating our experiment.

Selecting 3D Models to Evaluate Parameter Spacings

How finely differences in slicing parameters can be detected depends on the size of the area of the bottom surface. The larger the surface, the more features can be used by the algorithm for classification. To determine a spacing of parameters that works well across different 3D models, we used objects with varying surface areas for our experiment.

To select these objects, we downloaded the top 50 3D models from Thingiverse [48] and ranked them by their bottom surface area (i.e., the largest square one may inscribe in the contour of the first slice, determined by an automated MATLAB script). We found that 25 models had a large surface area (>6cm^2), 7 models had a medium surface area (1.2-6cm^2), and 18 models had small surface areas (<1.2cm^2). These ranges were determined empirically based on our initial tests. We randomly picked one object representing each of these three categories and printed multiple instances using the parameters below.

Determining the Range for Each Slicer Setting

Before slicing each of the models with different settings, we first determined the min and max values for each setting.

Bottom: For the bottom angle, we can use 0°-180°. Going beyond 180° would cause instances to be non-distinguishable (i.e., 90° looks the same as 270°). We took the min value for initial bottom line width as 0.35mm, the default value recommended in the slicer *Cura*. Although this parameter can be as large as twice the nozzle size (2*0.4mm), we limit the max value to 0.6mm to avoid disconnected lines. For the pattern settings, which do not have min and max values, we considered the "line" pattern for the bottom surface.

Infill: As for the infill angle, we can use a range of 0°-60° for the *trihexagon* and *triangular* patterns, and 0°-90° for the *grid* pattern, as their layouts are periodic with period 60° and 90°, respectively. For infill line distance (density), we determined that having infill units smaller than 2.6mm makes the pattern unrecognizable — we thus used it as the min value. The max value is 3.2mm for objects with medium base area but may go up to 8.0mm for larger objects. Going beyond this value would imply an infill density of less than 10%, and thus fragile, less stable objects. The three infill pattern (type) settings do not have min or max values.

Slicing with Different Spacings and Capturing Photos

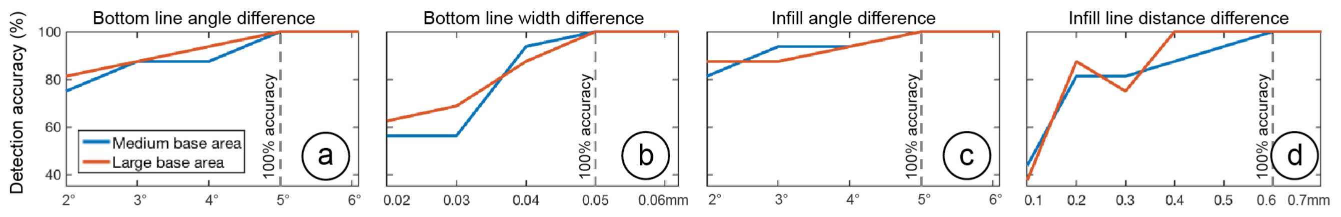
Next, we used our three selected objects (small, medium, large), and printed them with different slicing settings. Our goal was to determine how finely we can subdivide the given parameter ranges for accurate detection. To find the optimal spacing for each parameter, we made pairwise comparisons of two values (e.g., for angles, instance #1: 8° - instance #2: 5°; difference: 3°), while keeping all other parameters constant. Based on 16 pictures taken for each pairwise comparison, we report the accuracy at which we can distinguish the two instances. For the prints with infill variations, we held a small light source *(Nitecore Tini* [34]) against the side of the 3D printed object before taking the image.

Results of the Experiments

As expected, objects from the "small" category did not have sufficient base area to fit in enough infill units and thus not give satisfactory results. We therefore conclude that G-ID cannot be used for very small bases and excluded them from the rest of the analysis. We next discuss the results for each slicing parameter for the medium and large object category.

Bottom Line Angle & Width

The dashed lines in Figure 11a,b indicate that a spacing of 5° and 0.05 mm provides a classification accuracy of 100% for both the medium and large base area categories, respectively. Thus, a range of 0°-180° would give us 36 variations for the angle. However, we exclude the two degrees 0° and



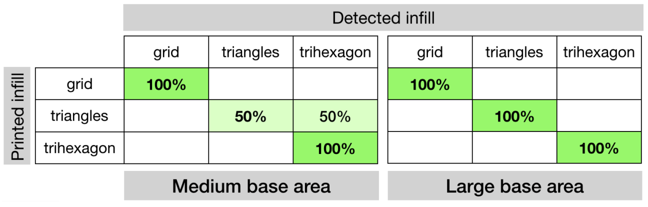
90° from the bottom line angle range since these are reserved for error checking (as described in section "Detecting Slicing Parameters"), therefore we have 34 variations. A range of 0.35-0.6mm allows 36 variations for the width.

Infill Angle & Line Distance (Width)

The dashed lines in Figure 11c,d show that a spacing of 5° and 0.6mm provides a 100% detection accuracy for the two categories, respectively. Thus, for the ranges of 0°-60° and 0°-90°, we can use 12 or 18 variations, respectively. For the width, we can use, for medium objects, a range of 2.6-3.2mm (2 variations); and for large objects 2.6-8.0mm (10 variations). We report the smaller number in Table 1.

Infill Pattern

The confusion matrix in Figure 12 shows that the "grid" and "trihexagon" work for both medium and large classes. For large objects, we can also use all three different patterns.



	Detected infill					
	Medium base area			Large base area		
Printed infill	grid	triangles	trihexagon	grid	triangles	trihexagon
grid	100%			100%		
triangles		50%	50%		100%	
trihexagon			100%			100%

Figure 12. Sliced infill patterns vs. detected infill patterns.

Total Number of Instances Possible

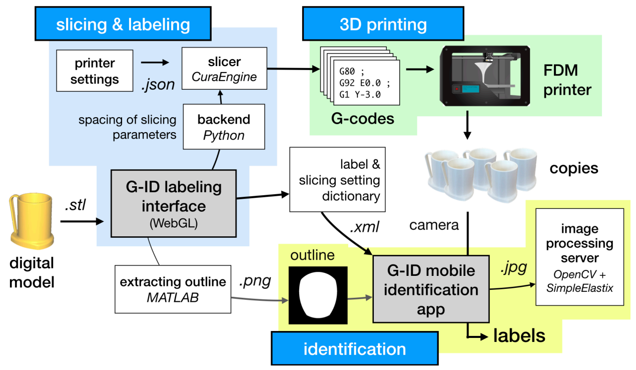
Based on the results, using a parameter spacing that works for both medium and large objects, we can achieve a total of 204 instances if we only use the camera, or 17,136 instances if we use both the camera and light (see Table 1). In comparison to other types of tags, we can say that these parameter spaces have a larger code capacity than a 1D barcode with 7 or 14 bits, respectively.

Cross-Validation

To cross-validate our parameter spacing, we printed another random set of 16 objects (10 large, 6 medium) from the same top 50 model list from Thingiverse (excluding the ones we used in the previous experiment) with white filament. Before printing, we randomly assigned each model a combination of slicing settings from the available parameter space and then tested if G-ID can identify them. Our second goal was to see if the parameter spacing still applies when multiple parameters are varied at the same time (our previous experiment only varied one slicing parameter per instance at a time). Among the 10 objects with large bottom area, all slicing parameters were correctly identified. Among the 6 objects from the medium category, 5 were correctly identified. In total, the detection accuracy is 93.75%. The falsely identified model had the smallest bottom surface area (1.4cm^2), which confirms the fact that objects without sufficient surface area cannot be recognized.

SYSTEM IMPLEMENTATION

An overview of the system is shown in Figure 13. G-ID's labeling interface for creating multiple instances runs on the browser and is based on WebGL. Once the user drags a 3D model (.stl) onto the canvas and enters the number of instances, the interface calls its backend written in Python, which is responsible for the distribution of the slicing parameters. Once the slicing parameters are determined for each individual instance, the backend calls the slicer *CuraEngine* [43] to compute the G-code for each instance.


slicing & labeling
3D printing
printer settings
.json
slicer CuraEngine
G80 ; G92 E0.0 ; G1 Y-3.0
G-codes
FDM printer
spacing of slicing parameters
backend Python
label & slicing setting dictionary
copies
digital model
.stl
G-ID labeling interface (WebGL)
.xml
camera
image processing server OpenCV + SimpleElastix
outline
extracting outline MATLAB
.png
G-ID mobile identification app
.jpg
labels
identification

After the instances are sliced with their individual slicer settings, the user is shown the 2D sliced layers as well as the 3D model. For rendering these previews, we use the JavaScript library *Three.js*. Finally, G-ID saves an .XML file with the slicing parameters, labels and the contour of the object's base as an image (created using an automated MATLAB script) for future identification with the G-ID mobile app.

EVALUATION

Different Materials, Lighting Conditions, Thicknesses

Surface: To see how the filament's color and different lighting affect the detection of surface parameters, we printed six instances of the key cover with eight different colors of Ultimaker PLA filaments (total of 48 instances) and using a dimmable LED lamp varied the light in the room from 0 – 500 lux (measured with lux meter Tacklife LM01). We picked the filament colors to be the primary and secondary colors of the RYB model, as well as white and black filament. We selected the light intensities to represent different low-light conditions. For slicing, we used surface parameters distributed evenly along bottom line angle and width within the allowed range from Table 1. The results are shown in Figure 14. All colors worked well for lighting conditions above 250 lux. This shows that our approach works well in classrooms (recommended light level: 250 lux) [32] and offices/laboratories (recommended light level: 500 lux). The results also show that the camera needs more light to resolve the lines for lighter colors (i.e., white and yellow) than for darker colors. Since we used white filament for our parameter spacing evaluation, the results in Table 1 work even for worst-case scenarios. This also means that for other filament colors, an even smaller surface area would suffice for correct detection since the surface details can be better resolved.

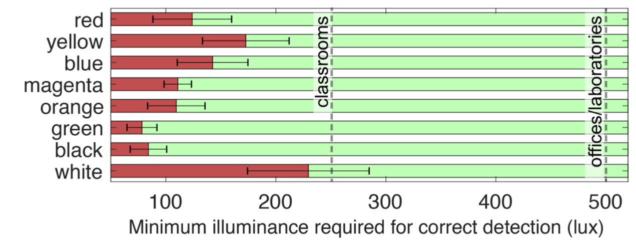


Figure 14. Different filament colors vs. minimum illuminance required to correctly detect the traces on the bottom surface.

Infill: We were able to detect the patterns for all of the aforementioned filament colors except for black due to its opaque nature. Further, the brighter the light, the thicker the object base may be: 145 lumens suffice for a base of 1mm (suggested thickness in Cura), 380 lumens suffice for 1.75 mm.

Different 3D Printers

Since G-ID takes as input universal units like millimeters (width) and degrees (angle), our method extends to FDM printers other than the one we used for the experiments. To confirm this, we fabricated six test prints on the 3D printers Prusa i3 MK3S and Creality CR-10S Pro in addition to the Ultimaker 3 that we used for our experiments, and inspected the traces laid down with a microscope. The line widths of the prints had an average deviation of 9.6μm (Prusa) and 10.7μm (Creality) from the Ultimaker 3 prints. The line angles had an average deviation of 0.5° (Prusa) and 0.25° (Creality). These deviations are insignificant for our detection method since the spacing values chosen for the parameters are much larger than these values. To verify this, we used our mobile app to take pictures of these samples and ran our algorithm, which correctly detected the unique identifiers.

Camera Distance and Angle

Distance: If the phone is held too far away from the object, the camera cannot detect the detailed grooves on the surface. To determine how far the camera can be held from the object, we took pictures with smartphones of different camera resolutions. We found that the iPhone 5s (8MP) can resolve the slicing parameter features up to 26cm, Pixel 2 (12.2MP) up to 36cm, and OnePlus 6 (16MP) up to 40cm. Taking into account the cameras' field of view, this means that users can fit in an object with one dimension as large as 29cm, 45cm, and 52cm, respectively. G-ID guides users into the correct camera distance by varying the size of the object outline displayed on the user's phone for alignment.

Angle: Since our image registration technique uses affine transformation, it is not able to fully remove the perspective distortion if the camera angle varies strongly. However, since our app guides users to align the object, the distortion is negligible. To show our algorithm can robustly read bottom surface parameters on a variety of shapes, we created a virtual dataset similar to [28]. We downloaded the first 600 models from the Thingi10K database [57] that have a rotationally non-symmetric base of appropriate size, sliced them with a random set of slicing settings, and rendered the G-codes using 3D editor *Blender.* We placed the virtual camera at points located on a spherical cap above the object base, with 8 evenly sampled azimuthal angles for each of the 5 polar angles. The percentage of shapes read correctly for each polar angle value θ is given in Table 2. The spacing values chosen for the parameters act as a buffer to prevent false readings. The objects for which detection failed at small angles had rather rounded bases, which makes alignment challenging.

θ	4°	6°	8°	10°	12°
Accuracy	98.50%	94.50%	86.67%	75.00%	64.50%

Table 2: Polar angle θ vs. the percentage of identified objects.

APPLICATION SCENARIOS

Below, we outline three further example scenarios in addition to the two previously explained applications.

Finding Optimal Print Settings: G-ID can be used to identify which slicing parameters a particular 3D print was created with. In Figure 15, a maker is trying to find the best angle for optimizing mechanical strength, and prints the model multiple times with varying settings. Rather than carefully writing down which settings were used for which one, the maker can retrieve a particular setting from a print simply by taking a picture of it. It is unlikely that they users estimate these settings by eye correctly as seeing tiny differences is not trivial.

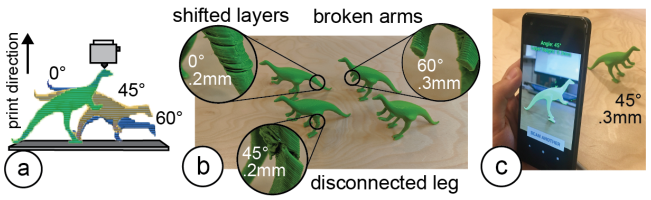


Figure 15. Identifying particular print settings with G-ID: (a) exploring different slicing parameters and (b) printing them, (c) retrieving the best settings using G-ID.

User Identification: G-ID can be used to create identifiable objects that belong to a specific user cheaply and rapidly as it doesn't require the user to embed, e.g., an RFID/NFC tag in a separate step. For instance, in *toys-to-life* video games, physical character figurines that carry a G-ID label can be used to load a player's identity/score. When users insert their figurine into the G-ID reader, it communicates the user's ID to the game to display their name and score (Figure 16).

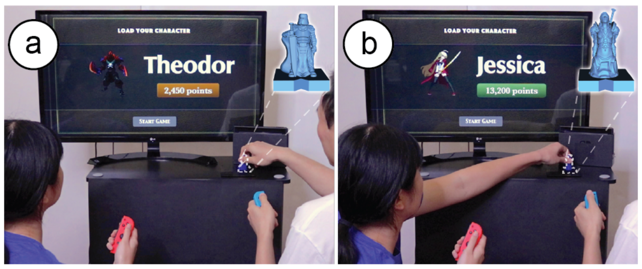


Figure 16. (a) Toys-to-life figurines are used to identify the player and display their info in the video game.

Labeling a Commercial 3D Print for Anti-Counterfeiting: Online 3D printing services such as *3D Hubs* [21] or *makexyz* [29] ensure to refund customers if they can show a 3D model was not printed according to the user's specifications. By slicing a model using certain settings and storing this information, they can verify that a returned object was indeed fabricated by them before a refund is approved. Let us assume a 3D model is leaked and frauds attempt to print a copy themselves and then return that to get a refund (although they never bought it in the first place). Businesses can use G-ID to cross-verify the print setting used to create the original.

DISCUSSION & LIMITATIONS

Next, we discuss limitations and future work for labeling and identifying objects by their slicing parameters.

Other Slicing Parameters

In this work, we focused on surface and infill parameters, which offer a large number of unique identifiable features. Other slicing parameters, such as those that create geometry that is removed after fabrication (e.g., those related to *support material* or *build plate adhesion*) are less suitable.

However, for special materials, such as wood filaments, the shade of the object's color can be altered by varying the speed and temperature, which could be used to create differences among instances: the hotter/slower the extruder, the more the wood particles burn and the darker the resulting surface (Figure 17). However, since the changes affect the objects appearance, we do not consider them for our work.

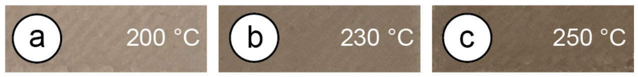


Figure 17. The hotter the nozzle, the darker the print's finish.

Rotational Symmetry of Outlines

Since we use the outline of objects to extract the orientation of features, objects whose bases are rotationally symmetric are less suitable for our approach. Thus, the number of identifiable angles is reduced for certain shapes, e.g., for a square base, the range narrows down from 0°-180° to 0°-90°.

Non-Flat Side Surfaces

When a camera's optical axis is parallel to the object's axis of rotation on the build plate, the layer traces on a curved surface appear as straight, parallel lines and the features can be extracted similarly using Fourier transforms. For that, we instruct the user to take a picture from the correct viewpoint

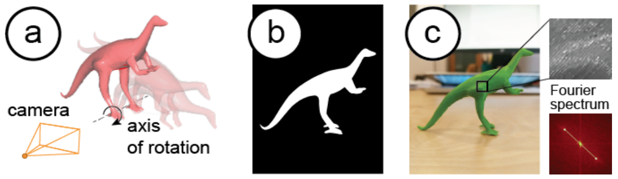


Figure 18. (a) Model's axis of rotation is orthogonal to the image plane of the virtual camera, (b) rendered target outline for the user to align model, (c) image has parallel traces.

(as shown in Figure 18) by generating a silhouette guide of the object taking into account the camera's focal length.

Applicability Beyond FDM Printing

FDM and SLA printing have been the most accessible consumer techniques for the last decade. Although SLA achieves better resolutions, individual layers on objects can still be distinct (Figure 19). As for DLP printing, the projected pattern creates rectangular voxels that cause the edges to look stepped; different voxel sizes thus lead to different appearances [13]. Most printing methods use infill, so the general idea of varying infill still applies. We thus think our method will stay relevant. Even if printing imperfections become smaller in the future, there are two factors to consider: As 3D printers improve in resolution, camera resolution improves over time (see Samsung's latest 108MP sensor [43]). Computer vision gets better too: Neural networks can now pick up details beyond what the human eye or traditional image processing can detect, e.g., they have been used to identify tiny defects in solar cells [6] or dents in cars [56].

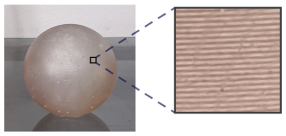


Figure 19. Formlabs SLA print captured by a phone camera.

CONCLUSION & FUTURE WORK

We presented G-ID, a method that utilizes the subtle patterns left by the 3D printing process to identify objects and that provides both a labeling and identification interface for creating and detecting surface and infill parameters using only a commodity camera. We reported on the code capacity enabled by an off-the-shelf slicer. In the future, this can be scaled up by building a custom slicer for creating unique print paths that go beyond what current slicers offer, e.g., spatial pattern tiling could expand the number of encodings (2 different tiles would give more than 204^2=41,616 combinations). Also, our current implementation uses optimization-based image registration, which takes a few seconds. In the future, we can enable continuous detection for faster image capturing using optimization-free contour matching methods [9, 10].

ACKNOWLEDGMENTS

We thank Alexandre Kaspar, Liane Makatura, Danielle Pace, and Jack Forman for the fruitful discussions. This work was supported in part by NSF Award IIS-1716413. Sriram Subramanian is grateful for the ERC Advanced Grant (#787413) and the RAEng Chairs in Emerging Technology Program.

REFERENCES

[1] Marc Alexa, Kristian Hildebrand, and Sylvain Lefebvre. 2017. Optimal discrete slicing. *ACM Trans. Graph.* 36, 1. https://doi.org/10.1145/2999536

[2] Daniel G. Aliaga and Mikhail J. Atallah. 2009. Genuinity signatures: Designing signatures for verifying 3D object genuinity. *Computer Graphics Forum* 28, 2: 437–446. https://doi.org/10.1111/j.1467-8659.2009.01383.x

[3] Vahid Babaei, Kiril Vidimče, Michael Foshey, Alexandre Kaspar, Piotr Didyk, and Wojciech Matusik. 2017. Color contoning for 3D printing. *ACM Trans. Graph.* 36, 4: 124:1–124:15. https://doi.org/10.1145/3072959.3073605

[4] Patrick Baudisch and Stefanie Mueller. 2017. Personal fabrication. *Foundations and Trends in Human-Computer Interaction* 10, 3-4: 165–293. https://doi.org/10.1561/1100000055

[5] Fei Chen, Yuxi Luo, Nektarios Georgios Tsoutsos, Michail Maniatakos, Khaled Shahin, and Nikhil Gupta. 2018. Embedding tracking codes in additive manufactured parts for product authentication. *Advanced Engineering Materials* 0, 0: 1800495. https://doi.org/10.1002/adem.201800495

[6] Haiyong Chen, Yue Pang, Qidi Hu, and Kun Liu. 2018b. Solar Cell Surface Defect Inspection Based on Multispectral Convolutional Neural Network. *Journal of Intelligent Manufacturing* (12 2018), 1–16.

[7] Hung-Kuo Chu, Chia-Sheng Chang, Ruen-Rone Lee, and Niloy J. Mitra. 2013. Halftone QR codes. *ACM Trans. Graph.* 32, 6: 217:1–217:8. https://doi.org/10.1145/2508363.2508408

[8] Adam Dachowicz, Siva Chaitanya Chaduvula, Mikhail Atallah, and Jitesh H. Panchal. 2017. Microstructure-based counterfeit detection in metal part manufacturing. *JOM* 69, 11: 2390–2396. https://doi.org/10.1007/s11837-017-2502-8

[9] Csaba Domokos and Zoltan Kato. 2010. Parametric estimation of affine deformations of planar shapes. *Pattern Recogn.* 43, 3 (March 2010), 569-578. http://dx.doi.org/10.1016/j.patcog.2009.08.013

[10] Csaba Domokos and Zoltan Kato. 2012. Simultaneous affine registration of multiple shapes. In *21st International Conference on Pattern Recognition (ICPR).* IEEE Computer Society, Los Alamitos, CA, 9–12

[11] M. Fiala. 2005. ARTag, a fiducial marker system using digital techniques. In *2005 IEEE computer society conference on computer vision and pattern recognition (cvpr'05)*, 590–596 vol. 2. https://doi.org/10.1109/CVPR.2005.74

[12] Paul Filiatrault. 2009. Does colour-coded labelling reduce the risk of medication errors? *Canadian Journal of Hospital Pharmacy* 62, 2: 154–155. https://doi.org/10.4212/cjhp.v62i2.446

[13] Formlabs. SLA vs. DLP. https://formlabs.com/blog/3d-printing-technology-comparison-sla-dlp/

[14] Blaise Gassend, Dwaine Clarke, Marten van Dijk, and Srinivas Devadas. 2002. Silicon physical random functions. In *Proceedings of the 9th ACM conference on computer and communications security* (CCS '02), 148–160. https://doi.org/10.1145/586110.586132

[15] Tony Hall and Liam Bannon. 2005. Designing ubiquitous computing to enhance children's interaction in museums. In *Proceedings of the 2005 conference on interaction design and children* (IDC '05), 62–69. https://doi.org/10.1145/1109540.1109549

[16] Chris Harrison, Robert Xiao, and Scott Hudson. 2012. Acoustic barcodes: Passive, durable and inexpensive notched identification tags. In *Proceedings of the 25th annual ACM symposium on user interface software and technology* (UIST '12), 563–568. https://doi.org/10.1145/2380116.2380187

[17] Liang He, Gierad Laput, Eric Brockmeyer, and Jon E. Froehlich. 2017. SqueezaPulse: Adding interactive input to fabricated objects using corrugated tubes and air pulses. In *Proceedings of the eleventh international conference on tangible, embedded, and embodied interaction* (TEI '17), 341–350. https://doi.org/10.1145/3024969.3024976

[18] J. Hou, D. Kim, and H. Lee. 2017. Blind 3D mesh watermarking for 3D printed model by analyzing layering artifact. *IEEE Transactions on Information Forensics and Security* 12, 11: 2712–2725. https://doi.org/10.1109/TIFS.2017.2718482

[19] Jong-Uk Hou, Do-Gon Kim, Sunghee Choi, and Heung-Kyu Lee. 2015. 3D print-scan resilient watermarking using a histogram-based circular shift coding structure. In *Proceedings of the 3rd ACM workshop on information hiding and multimedia security*,115–121. https://doi.org/10.1145/2756601.2756607

[20] Meng-Ju Hsieh, Rong-Hao Liang, Da-Yuan Huang, Jheng-You Ke, and Bing-Yu Chen. 2018. RFIBricks: Interactive building blocks based on RFID. In *Proceedings of the 2018 CHI conference on human factors in computing systems* (CHI '18), 189:1–189:10. https://doi.org/10.1145/3173574.3173763

[21] 3D Hubs. https://www.3dhubs.com/

[22] Z. C. Kennedy, D. E. Stephenson, J. F. Christ, T. R. Pope, B. W. Arey, C. A. Barrett, and M. G. Warner. 2017. Enhanced anti-counterfeiting measures for additive manufacturing: Coupling lanthanide nanomaterial chemical signatures with blockchain technology. *J. Mater. Chem. C* 5, 37: 9570–9578. https://doi.org/10.1039/C7TC03348F

[23] Ryosuke Kikuchi, Sora Yoshikawa, Pradeep Kumar Jayaraman, Jianmin Zheng, and Takashi Maekawa. 2018. Embedding QR codes onto b-spline surfaces for 3D printing. *Computer-Aided Design* 102: 215–223. https://doi.org/10.1016/j.cad.2018.04.025

[24] Yuki Kubo, Kana Eguchi, Ryosuke Aoki, Shigekuni Kondo, Shozo Azuma, and Takuya Indo. 2019. FabAuth: Printed objects identification using resonant properties of their inner structures. In *Extended Abstracts of the 2019 CHI conference on human factors in computing systems* (CHI EA '19), LBW2215:1–LBW2215:6. https://doi.org/10.1145/3290607.3313005

[25] Dingzeyu Li, David I. W. Levin, Wojciech Matusik, and Changxi Zheng. 2016. Acoustic voxels: Computational optimization of modular acoustic filters. *ACM Trans. Graph.* 35, 4: 88:1–88:12. https://doi.org/10.1145/2897824.2925960

[26] Dingzeyu Li, Avinash S. Nair, Shree K. Nayar, and Changxi Zheng. 2017. AirCode: Unobtrusive physical tags for digital fabrication. In *Proceedings of the 30th annual ACM symposium on user interface software and technology* (UIST '17), 449–460. https://doi.org/10.1145/3126594.3126635

[27] Zhengxiong Li, Aditya Singh Rathore, Chen Song, Sheng Wei, Yanzhi Wang, and Wenyao Xu. 2018. PrinTracker: Fingerprinting 3D printers using commodity scanners. In *Proceedings of the 2018 ACM sigsac conference on computer and communications security* (CCS '18), 1306–1323. https://doi.org/10.1145/3243734.3243735

[28] Henrique Teles Maia, Dingzeyu Li, Yuan Yang, and Changxi Zheng. 2019. LayerCode: Optical barcodes for 3D printed shapes. *ACM Trans. Graph.* 38, 4: 112:1–112:14. https://doi.org/10.1145/3306346.3322960

[29] Makexyz, LLC. https://www.makexyz.com/

[30] Kasper Marstal, Floris F. Berendsen, Marius Staring, and Stefan Klein. 2016. SimpleElastix: A user-friendly, multi-lingual library for medical image registration. *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*: 574–582.

[31] A. Mittal. 2017. Generating visually appealing QR codes using colour image embedding. *The Imaging Science Journal* 65, 1: 1–13. https://doi.org/10.1080/13682199.2016.1241941

[32] National Optical Astronomy Observatory (NOAO). *Recommended light levels (illuminance) for outdoor and indoor venues*. Association of Universities for Research in Astronomy. Retrieved from https://www.noao.edu/education/QLT-kit/ACTIVITY_Documents/Safety/LightLevels_outdoor indoor.pdf

[33] Nintendo Amiibo. Retrieved from https://www.nintendo.com/amiibo/

[34] NiteCore Tini. Retrieved from https://flashlight.nitecore.com/product/tini

[35] Fei Peng, Jing Yang, Zi-Xing Lin, and Min Long. 2019. Source identification of 3D printed objects based on inherent equipment distortion. *Computers & Security* 82: 173–183. https://doi.org/http://doi.org/10.1016/j.cose.2018.12.015

[36] Siyuan Qiao, Xiaoxin Fang, Bin Sheng, Wen Wu, and Enhua Wu. 2015. Structure-aware QR code abstraction. *Vis. Comput.* 31, 6-8: 1123–1133. https://doi.org/10.1007/s00371-015-1107-x.

[37] Tim Reiner, Nathan Carr, Radomír Měch, Ondřej Šťava, Carsten Dachsbacher, and Gavin Miller. 2014. Dual-color mixing for fused deposition modeling printers. *Comput. Graph. Forum* 33, 2: 479–486. https://doi.org/10.1111/cgf.12319

[38] M. S. Rzeszotarski, F. L. Royer, and G. C. Gilmore. 1983. Introduction to two-dimensional Fourier analysis. *Behavior Research Methods & Instrumentation* 15, 2: 308–318. https://doi.org/10.3758/BF03203566

[39] Martin Schmitz, Martin Herbers, Niloofar Dezfuli, Sebastian Günther, and Max Mühlhäuser. 2018. Off-line sensing: Memorizing interactions in passive 3D-printed objects. In *Proceedings of the 2018 chi conference on human factors in computing systems* (CHI '18), 182:1–182:8. https://doi.org/10.1145/3173574.3173756

[40] Ruchir Y Shah, Prajesh N Prajapati, and Y K Agrawal. 2010. Anticounterfeit packaging technologies. *Journal of Advanced Pharmaceutical Technology & Research* 1, 4: 368–373. https://doi.org/10.4103/0110-5558.76434

[41] Lei Shi, Idan Zelzer, Catherine Feng, and Shiri Azenkot. 2016. Tickers and talker: An accessible labeling toolkit for 3D printed models. In *Proceedings of the 2016 chi conference on human factors in computing systems* (CHI '16), 4896–4907. https://doi.org/10.1145/2858036.2858507

[42] Piyarat Silapasuphakornwong, Masahiro Suzuki, Hiroshi Unno, Hideyuki Torii, Kazutake Uehira, and Youichi Takashima. 2016. Nondestructive readout of copyright information embedded in objects fabricated with 3-d printers. In *Digital-forensics and watermarking*, 232–238.

[43] Samsung ISOCELL Bright HMX. https://www.samsung.com/semiconductor/image-sensor/mobile-image-sensor/S5KHMX/

[44] Chen Song, Zhengxiong Li, Wenyao Xu, Chi Zhou, Zhanpeng Jin, and Kui Ren. 2018. My smartphone recognizes genuine QR codes!: Practical unclonable qr

code via 3D printing. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 2, 2: 83:1–83:20. https://doi.org/10.1145/3214286

[45] Hai-Chuan Song and Sylvain Lefebvre. 2017. Colored fused filament fabrication. *CoRR* abs/1709.09689. Retrieved from http://arxiv.org/abs/1709.09689

[46] Andrew Spielberg, Alanson Sample, Scott E. Hudson, Jennifer Mankoff, and James McCann. 2016. RapID: A framework for fabricating low-latency interactive objects with RFID tags. In *Proceedings of the 2016 chi conference on human factors in computing systems* (CHI '16), 5897–5908. https://doi.org/10.1145/2858036.2858243

[47] Carlos Tejada, Osamu Fujimoto, Zhiyuan Li, and Daniel Ashbrook. 2018. Blowhole: Blowing-activated tags for interactive 3D-printed models. In *Proceedings of graphics interface 2018* (GI 2018), 131–137. https://doi.org/10.20380/GI2018.18

[48] Thingiverse. https://www.thingiverse.com/

[49] Rachel Thomas. 2017. Fourier transforms of images. *Plus*. Retrieved from https://plus.maths.org/content/fourier-transforms-images

[50] Kazutake Uehira, Masahiro Suzuki, Piyarat Silapasuphakornwong, Hideyuki Torii, and Youichi Takashima. 2017. Copyright protection for 3D printing by embedding information inside 3D-printed objects. In *Digital forensics and watermarking*, 370–378.

[51] Ultimaker. 2019. CuraEngine. *GitHub*. Retrieved from https://github.com/Ultimaker/CuraEngine

[52] Ingrid Verbauwhede and Roel Maes. 2011. Physically unclonable functions: Manufacturing variability as an unclonable device identifier. In *Proceedings of the 21st edition of the great lakes symposium on great lakes symposium on VLSI* (GLSVLSI '11), 455–460. https://doi.org/10.1145/1973009.1973111

[53] Karl D. D. Willis and Andrew D. Wilson. 2013. InfraStructs: Fabricating information inside physical objects for imaging in the terahertz region. *ACM Trans. Graph.* 32, 4: 138:1–138:10. https://doi.org/10.1145/2461912.2461936

[54] Zhe Yang, Yuting Bao, Chuhao Luo, Xingya Zhao, Siyu Zhu, Chunyi Peng, Yunxin Liu, and Xinbing Wang. 2016. ARTcode: Preserve art and code in any image. In *Proceedings of the 2016 ACM international joint conference on pervasive and ubiquitous computing* (UbiComp '16), 904–915. https://doi.org/10.1145/2971648.2971733

[55] Nikoleta Yiannoutsou, Ioanna Papadimitriou, Vassilis Komis, and Nikolaos Avouris. 2009. "Playing with" museum exhibits: Designing educational games mediated by mobile technology. In *Proceedings of the 8th international conference on interaction design and children* (IDC '09), 230–233. https://doi.org/10.1145/1551788.1551837

[56] Qinbang Zhou, Renwen Chen, Huang Bin, Chuan Liu, Jie Yu, and Xiaoqing Yu. 2019. An Automatic Surface Defect Inspection System for Automobiles Using Machine Vision Methods. *Sensors* 19 (02 2019), 644.

[57] Qingnan Zhou and Alec Jacobson. 2016. Thingi10K: A Dataset of 10,000 3D-Printing Models. *arXiv preprint arXiv:1605.04797* (2016).