LayerCode: Optical Barcodes for 3D Printed Shapes

HENRIQUE TELES MAIA, Columbia University
DINGZEU LI, Adobe Research
YUAN YANG, Columbia University
CHANGXI ZHENG, Columbia University

With the advance of personal and customized fabrication techniques, the capability to embed information in physical objects becomes evermore crucial. We present LayerCode, a tagging scheme that embeds a carefully designed barcode pattern in 3D printed objects as a deliberate byproduct of the 3D printing process. The LayerCode concept is inspired by the structural resemblance between the parallel black and white bars of the standard barcode and the universal layer-by-layer approach of 3D printing. We introduce an encoding algorithm that enables the 3D printing layers to carry information without altering the object geometry. We also introduce a decoding algorithm that reads the LayerCode tag of a physical object by just taking a photo. The physical deployment of LayerCode tags is realized on various types of 3D printers, including Fused Deposition Modeling printers as well as Stereolithography based printers. Each offers its own advantages and tradeoffs. We show that LayerCode tags can work on complex, nontrivial shapes, on which all previous tagging mechanisms may fail. To evaluate LayerCode thoroughly, we further stress test it with a large dataset of complex shapes using virtual rendering. Among 4,835 tested shapes, we successfully encode and decode on more than 99% of the shapes.

ACM Reference Format:

1 INTRODUCTION

Invented 45 years ago, the optical barcode has become an indispensable minutiae in today’s digital era. The design is simple, e.g. black and white bars printed on a flat surface, but its use is ubiquitous. From package delivery and airplane boarding to inventory management and patient identification, the barcode serves as a link that bridges physical artifacts to modern digital systems.

In this work, we rethink barcodes in the context of additive manufacturing, popularly known as 3D printing. 3D printing offers a quick way of making customized, complex shaped objects. Unlike a mass-produced product which by design has a reserved flat surface region to host barcodes, 3D printed shapes are often complex and curved: thin features, slender threads, and holes are not uncommon. As a result, traditional barcodes cannot be placed on such objects.

Recent years have seen a few approaches proposed toward embedding optical tags in 3D printed objects, on the surface [Kikuchi et al. 2018], beneath the surface [Li et al. 2017] and inside the objects [Willis and Wilson 2013]. However, these approaches either require specialized (and expensive) hardware to read the tags or
only work on a limited set of simple shapes (i.e., those with a flat or smooth surface). This limitation, in stark contrast to the complexity of shapes that current 3D printers commonly produce, remains a significant open problem.

We introduce LayerCode, to bring the concept of optical barcodes into 3D printed objects, especially those with curved shapes and fine structures. Our key idea is inspired by the structural resemblance between optical barcodes and 3D printed objects: essential in a barcode are its black and white bars arranged in parallel; universal in all 3D printed objects are printing layers introduced in a parallel fashion. In fact, virtually all additive manufacturing uses a layer-by-layer printing process [Livesu et al. 2017; Redwood et al. 2017]. Thus, if we could interleaver two “types” of layers in a 3D printing process, we would be able to embed a barcode everywhere along a 3D printed object.

Materializing this idea faces two challenges. The first is algorithmic. Due to an object’s complex shape, its layering structure may appear curved, disconnected, or shadowed when captured by a camera. We therefore seek a robust encoding and decoding algorithm that embeds information in printing layers and later retrieves this information from the images of a conventional camera. The second challenge rests in practical realization. In various types of 3D printers, including those that support only a single material, we need to introduce two distinguishable layer types.

We address the first challenge by introducing a new coding algorithm. Unlike the standard barcode that maps every bit to a bar thickness, we encode individual bits based on the local change of layer thickness, which, as we will show, is invariant under different surface orientations and curvatures. At decoding time, we exploit a key observation that each layer spans the entire cross-section of the object. This suggests that there exist many image-plane paths along which we can decode. The rich set of decoding paths is advantageous, enabling us to sidestep shadows, highlights, and uncertain image regions to decode robustly.

We address the second challenge by developing software and hardware updates for printers. For printers that support two materials (such as the Makerbot Replicator 2 and PolyJet), distinct layer types are naturally introduced by assigning different materials. For fused deposition modeling (FDM) printers with only a single material (such as the Ultimaker 2), we propose to change the filament deposition height during printing to indicate different layer types. Last but not least, for stereolithography printers (such as an Autodesk Ember), we propose to mix near infrared (NIR) dye in the printing resin to create the second type of layers. This unobtrusive and machine-readable tagging is similar in spirit to [Li et al. 2017] and finds many applications.

Our proposed LayerCode approach features a number of attributes desired for tagging 3D printed objects:

**Robustness on complex shapes.** LayerCode tags can be applied to objects with complex shapes (e.g., see Figure 3), and are significantly more versatile than existing approaches. Besides demonstrating our algorithm with real-world examples, we also test it exhaustively using rendered images on Thingi10k [Zhou and Jacobson 2016], a dataset with 4,835 printable meshes across a wide range of shapes.

**Ease with a conventional camera.** LayerCode tags can be read by a conventional camera, without resorting to expensive hardware (Figure 2). Even for the NIR tags, the only additional hardware needed for decoding is a NIR filter and a NIR light source (e.g., TV remote); both are low-cost and easily accessible.

**Compatibility with 3D printers.** LayerCode tags can be used in various types of 3D printers, whether they are single material or multi-material FDM or stereolithography printers. Additionally, we show how even a single-material stereolithography printer like the Autodesk Ember can support the requisite two types of layers.

**Structural preservation.** Since LayerCode tags are built upon the layer-by-layer 3D printing process without modifying the original shapes, they have a minimal, if not negligible, impact on the print’s mechanical properties. This feature contrasts starkly to previous approaches, as they all alter the shapes to a certain extent.

**Ubiquitous tagging of an object.** Embedded in 3D printed layers, LayerCode tags span over the entire object body, both inside and on the surface. Such ubiquity of a tag is beneficial: tags can be decoded along many surface paths, which makes the decoding process robust. This redundancy also renders the tag readable from multiple camera view angles or within a broken or damaged object (Figure 17).

**Depth information for free.** The interleaving parallel layers of a LayerCode tag can be reinterpreted as an ideal parallel light pattern projected on the object. Thus, using the structured light technique of computer vision, even from a single image of the tagged object, we are able to estimate the depth of the object from the camera (Figure 10). In other words, every LayerCode tag automatically conveys shape information of its carrier object for free.

In summary, we highlight the following contributions:

- A new feature-rich tagging mechanism that exploits the layering structures employed in additive manufacturing processes.
- A decoding algorithm that is robust against high curvatures, rough surfaces, thin features, occlusions, and other factors that limit the use of previous approaches.
- We propose three distinct methods that achieve LayerCode tags in various types of 3D printing processes.
- A comprehensive evaluation of 4,835 rendered images as well as over 20 physical objects across three 3D printers.
2 RELATED WORK

The rich features of LayerCode set it apart from existing tagging mechanisms. We now elaborate its differences from existing approaches, and a summary is shown in Table 1.

Although traditional barcodes are robust, they are privy to certain assumptions. Convention dictates that in order for a barcode to work, it must be laid on a flat surface, surrounded by two quiet zones of empty space, and consist of evenly-spaced encoded digits composed of fixed length modules [Woodland and Bernard 1952]. The code requires preservation of the ratios of these modules even when scanned at an off-axis angle, which in turn demands a flat surface for tags in order to read correctly. The strict requirement on flatness significantly limits the adoption of barcodes to arbitrary geometries.

In one of the early efforts to tag mass-produced printed objects, [Weigelt et al. 2010] extensively developed and discussed printing electronics in the object’s interior. Since then, much of the research along this direction has focused on embedding specialized hardware inside the 3D printed objects. For example, magnets, Radio-Frequency IDentification (RFID) chips, optical elements, circuits, and extra support materials have since been utilized for tagging purposes [Iyer et al. 2018; Kao et al. 2016; Willis et al. 2012; Willis and Wilson 2013; Yoon et al. 2016]. However, these hardware components not only lead to additional costs, but also require highly specialized and usually expensive equipment for accessing the embedded information. In comparison, LayerCode is a natural and cost-free byproduct of the printing process that only needs a camera for decoding.

Table 1. A comparison of challenging design considerations and features across several tagging techniques.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsmooth rough surfaces</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Thin shell or rod</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Accessible hardware decoding</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Tested on multiple 3D printers</td>
<td>N/A</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Structural preservation</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Appearance preservation</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Ubiquitous tagging</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Depth estimation for free</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>

In computer graphics and HCI, advances in fast and accurate sound simulation enable acoustic sensing and tagging. Early pioneering work includes the appearance-altering Acoustic Barcodes and Lamello [Harrison et al. 2012; Savage et al. 2015]. To better maintain the exterior appearance, various methods were proposed to optimize internal resonant chambers to achieve robust tagging performance, including BlowHole [Tejada et al. 2018], Acoustic Voxels [Li et al. 2016], and SqueezaPulse [He et al. 2017]. Although acoustic tagging approaches have shown promise, they share an inherent limitation: they cannot handle arbitrary shapes, like thin rod structures and thin shell objects, because of physical size constraints from the resonant chamber. LayerCode, on the other hand, is capable of working on a wide range complex shapes, as shown in Figures 3, 11, and 23.

Most related to our proposed method are AirCode and Optimal Discrete Slicing. AirCode uses unnoticeable subsurface scattering to embed a QR code-like pattern to preserve superficial appearances [Li et al. 2017]. One key limitation is that the control of subsurface scattering requires high-precision resin-based printers, which precludes an application to consumer-level filament-based printers. This is because the layered nature of the printing process was not accounted for when designing the subsurface tags. Alexa et al. [2017] proposed variable layer deposition thickness (i.e. layer height) slicing to optimize printing time. Instead of optimizing time, we leverage variable layer slicing to encode information, making LayerCode available to a wide range of 3D printers (see §5). Another advantage from considering the printing process is the lack of compromise in cost or fabrication/cleaning time, which introduce significant tradeoffs in hardware or acoustic-based methods.

We are not the first to utilize controllable layer heights in 3D printing. Pioneer work from two decades ago focused on slicing speed while producing coherent slices friendly to printers [McMains and Séquin 1999]. More recently, the focus has shifted to more high-level design-related goals. Wang et al. [2015] optimized layer heights to preserve salient regions on printed meshes. Starly et al. [2005] designed a novel slicing algorithm for CAD NURBS models to overcome the accuracy issue in precision manufacturing. Similar to [Alexa et al. 2017], VarSlice [Crayons 2016] and others [Zuechel et al. 2016] manipulate layers based on curvature to speed up printing where possible. A comprehensive review can be found in [Nadiyapara and Pande 2017]. Inspired by the long line of work on layer slicing, we explore the layering nature to develop a robust tagging scheme.
The layer information we embed not only encapsulates the tag, but also conveys depth information. To estimate depth from the images, there is abundant literature on leveraging structured light for 3D reconstruction [Taubin et al. 2014]. Most previous methods use active and controlled light sources with multiple images to help decode the depth [Hall-Holt and Rusinkiewicz 2001; Zhang et al. 2002]. For depth estimation from a single image, due to its ill-posed nature, most prior work has resorted to data-driven methods [Chen et al. 2016; Saxena et al. 2006]. Relying on structured layers, we demonstrate it is possible to estimate depth from a single image and can be complementary to existing data-driven approaches.

3 ENCODING

Conceptually, the encoding process decomposes a 3D printed shape into two sets of interleaving layers, which we refer as the black and white layers, respectively (Figure 4), to echo the black and white bars in standard barcodes. We also refer the black and white layers generally as the coding layers to distinguish from the 3D printing layers made in the 3D printing process. Each black or white coding layer consists of multiple consecutive 3D printing layers, and thus has a variable thickness.

In practice, we need to assign each 3D printing layer different properties (such as colors) so that at decoding time, the black and white layers can be recognized from a camera image. These practical details are deferred until §5. In this section, our goal is to assign each coding layer a thickness to encode a piece of information.

The input to our encoding algorithm is a 3D shape, the tag information represented as a bit string, as well as the 3D printing direction with respect to the printed object (i.e., the direction along which 3D printing layers will be grown). Unlike other tagging methods, there is no restriction on the 3D printed shape. We leave the flexibility of choosing a printing direction to the user, because the printing direction may depend on the specific shape, printing software, support materials, and perhaps subjective preferences. The output of the encoding algorithm is a series of slices along the printing direction to specify the thickness of each coding layer.

**Challenges and insights.** In a standard optical barcode, the black and white colors are used to label individual bars, and a bit (0/1) is encoded in the thickness of each bar. Unfortunately, it would be problematic to simply transfer this design to curved surfaces. As shown in Figure 4, a layer’s thickness on a curved surface will appear spatially variant after being projected on an image. Thus, a new coding scheme is needed.

**Fig. 4. Distorted thickness.** A sphere is coded with black and white layers of equal thickness. On the captured image, curvature and perspective cause layers to appear spatially varying in size.

A key insight comes from noticing the fact that if the coding layers are thin (relative to the inverse of the surface curvature along the printing direction), the thickness ratio of two consecutive layers measured in a local region of the image plane is invariant. This is because in a small local region, two nearby coding layers share approximately the same surface tangent plane, and the projection from the tangent plane to the image plane follows an affine transformation which preserves the layer thickness ratio.

Using local thickness ratios also favors the decoding step. As will be discussed in §4, it allows us to sample the thickness ratio of two layers at many local regions on the image, and collectively estimate a thickness ratio that is robust against imaging noise and artifacts.

**Coding scheme.** We propose the following scheme to encode every bit in a bitstring. A bit “1” is encoded if the thickness ratio of two consecutive layers is either $1/M$ or $M$, where $M$ is a constant larger than 1 that we will discuss shortly, and a bit “0” is represented by a unitary thickness ratio (i.e., the same thickness). The representation of a bit string always starts from a layer with a baseline thickness $h$. The next layer thickness $a_{n+1}$ is either $h$ or $Mh$ according to the current bit $b_{n+1}$ and the previous layer thickness $a_n$, namely,

$$a_{n+1} = \begin{cases} a_n & \text{if } b_{n+1} = 0, \\ Mh & \text{if } b_{n+1} = 1 \text{ and } a_n = h, \\ h & \text{if } b_{n+1} = 1 \text{ and } a_n = Mh. \end{cases}$$  

(1)

At decoding time, we recover the bit string sequentially, using the inverse map

$$b_{n+1} = \begin{cases} 1 & \text{if } \log a_n - \log a_{n+1} = \pm \log M, \\ 0 & \text{if } \log a_n - \log a_{n+1} = 0. \end{cases}$$  

(2)

In practice, the value of $\log a_n - \log a_{n+1}$ will never be precisely $\pm \log M$ or 0 due to the image estimation errors. But a nice property of this coding scheme is that the estimated values of $\log a_n - \log a_{n+1}$, when viewed as a random variable, will form three distribution modes symmetrically centered at $\pm \log M$ and 0. In §4.1, we will return to this property for robust decoding. Figure 5 illustrates this scheme for $M = 2$.

In theory, $M$ can be any value larger than 1. It offers the user the flexibility of trading off the total number of bits of a shape for the robustness of decoding. A larger $M$ sets $\pm \log M$ further away from 0, so at decoding time the estimated $\log a_n - \log a_{n+1}$ is more distinctive; but the layers are thicker, and thus the shape can store less information. If $M$ becomes too large, a coding layer may occupy a large surface area, where the surface curvature starts to vary considerably. Then, the local layer thickness ratio also becomes spatially varying. In all our examples, we used $M = 2$.

---

Henrique Teles Maia, Dingzeyu Li, Yuan Yang, and Changxi Zheng

ACM Trans. Graph., Vol. 38, No. 4, Article 1. Publication date: July 2019.
Among the coding layers, we also need to label where a bit string starts and ends. And we use the following simple rules. We start a bit string from a layer with a thickness $N h$, where $N$ is considerably larger than $M$ (in practice, $N = 4$), followed by two layers (one black and one white) with a thickness $h$ each. This appends a bit “0” to the beginning of the message. Then, after encoding the full bit string, a single layer is added to encode a bit “1” followed by another layer of thickness $N h$. This additional structure isolates a tag and disambiguates the bit string direction on the image plane.

Lastly, we reach a lemma about the total thickness of a bit string.

**Lemma 1.** Provided a bit string of length $T$ (bits), the 3D printing thickness $H$ needed to host this bit string is bounded by

$$2N + 2 + T + M h \leq H \leq (2N + 3 + T \cdot M)h.$$  

**Proof.** In addition to the beginning 3 layers of a total thickness $(N + 2)h$, and the end layer of thickness $N h$, there are $T + 1$ layers in-between corresponding to the $T$ bits followed by the ending bit “1”. Recall that whenever a bit “1” appears, the layer thickness changes across two layers. Therefore, among the $T + 1$ layers, there is at least one layer whose thickness is different from others. If that layer is a thick layer (of thickness $M h$), we obtain the lower bound. If that layer is thin (of thickness $h$), we reach the upper bound.  

Conversely, this lemma shows that if a 3D shape has a size $D$ along the printing direction and $D \gg h$, then its information capacity (i.e., total bits) is at least \( \frac{D}{2M} - \frac{2N^2 + 3}{2M} \).

**Repetition.** The user needs to choose the layer’s baseline thickness $h$ at encoding time, although as presented in §4, our decoding algorithm is agnostic to $h$. From Lemma 1, we know that if a 3D printed object has a size $D$ along the printing direction, and if we need to store $T$ bits, $h$ should be at most $D/(2N^2 + 2 + T + M)$. Often times, $h$ is much smaller than this bound. Then, we repeat the same bit string (and thus the layer thickness pattern) multiple times, occupying the entire printing distance. Effectively, we embed multiple copies of the bit string in the entire object (Figure 6 & Figure 23).

This repetition introduces no additional printing cost, and is beneficial in practice. It allows the barcode to be read from a wider range of camera angles, and thereby eases camera alignment at decoding time. Additionally, the redundant bit strings allow for a robust voting scheme at decoding time (see §4.2).

**Bit capacity.** LayerCode design supports a flexible length integer base-2 bit-length encoding, which may be adapted to design and application constraints. In order to encapsulate the unique object IDs of our database, all of our virtual evaluations (§6.1) use a 24 bit-length base-2 encoding, leading to an entropy over 16 million. For reference, a traditional UPC-E barcode uses base-10 encodings and supports an entropy of 2 million. Similarly, our real world two-color and layer-height examples (see §5.1 and §5.2 respectively) also use 24 bit-length encodings. The near-infrared prints discussed in §5.3 employ 12 bit binary encodings due to the printer’s smaller build volume and subsequently smaller prints.

**Error correction coding.** Our coding scheme is about encoding a bit string in a physical representation (e.g. layer thickness) and decoding from a tangible form (e.g. 3D printed objects). Thus, it is able to carry any error-correction code. In our experiments, we choose not to add any error-correction redundancy to study the pure performance of our method. Our coding scheme can support various error-correction coding schemes such as the Reed-Solomon codes [Reed and Solomon 1960]. These coding schemes add redundant bits to a bit string for correcting errors at decoding time.

### 4 DECODING

We now describe our core algorithm of decoding LayerCode tags from a camera image. To start decoding, we expect the black and white layers of the object to appear distinctively on the image. This is guaranteed through our fabrication methods specific to different types of printers. Focusing on the core decoding algorithm here, we defer those fabrication details in §5.

To motivate the overarching idea of our algorithm, we start by considering a few increasingly challenging situations (see Figure 6). First, on a curved surface, the thickness of a coding layer varies spatially on the image (Fig 4), making the decoding (e.g., using (2)) easily fallible. If the surface curvature is relatively small, previously existing rectifications include decoding along multiple projection lines of pixels [England 1996] and along curved paths [Liu et al. 1998]. Nevertheless, the concept of an image-space decoding path is flawed once a more complex shape is considered. As illustrated in Figure 7, if a shape zigzags or branches, it is almost impossible to find a path along which the entire encoded bit string is covered. This might suggest that a more reasonable approach is to instead segment individual coding layers and somehow measure the layer thickness. Yet, such a layer-centric approach is also vulnerable as highlights, shadows, and image noise may “shatter” a layer into disjoint regions (Figure 6-right).

We propose a *graph-based* algorithm. We treat each coding layer region, which may not include an entire layer, as a graph node. Two nodes are connected if they are from different but neighboring layers. As we will show, a robust decoding algorithm can be realized by strategically traversing this graph.

**Image Preprocessing.** Before delving into the decoding details, we preprocess the camera image to separate the object from its background and remove highlights and shadows, which are regions where pixel intensities are too high or too low. The preprocessing step depends on specific types of 3D printed objects—whether
they are bi-material objects, objects with variable layer deposition heights, or objects containing NIR resin; they exhibit different image features. We defer this fabrication-specific preprocessing step in §5. Afterward, we binarize the remaining pixels, labeling them as in either black layers or white layers (see Appendix ?? for details). Later in Figure 14-e and -h, we show an example of images before and after this preprocessing step. Images resultant from this step are ready for decoding (see Algorithm 1 for an outline of major steps).

4.1 Graph Construction

First, we construct a graph to represent the layer structure. Through a flood-fill process, we identify individual pixel regions where all the pixels are labeled black or white at the end of the preprocessing step. Each region is represented as a graph node, and two nodes are connected if their regions are adjacent to each other (Figure 8-a,b).

Next, we associate every edge e with two quantities, a 2D vector v in image space and a binary label r. Consider an edge e that connects nodes A and B. Its vector v represents the general direction along which we can move from the image region A to the region B. As will become clear shortly (§4.2), this direction will guide us in traversing the graph without getting trapped in a loop. To compute v, we first identify boundary pixels in each region. These are the pixels within δ pixels away from another region (δ = 3 in practice). At each boundary pixel, we estimate a boundary normal direction as the direction along which we can enter into a different region by moving the shortest distance. v is then defined as the average normal direction over all boundary pixels between region A and region B. When computing the average, we use the normal direction np for pixel p in region A, and the opposite normal direction −np for p in B. Thus, the average direction v is in fact associated to the directed edge from A to B, and for clarity we denote it as vA→B. The direction for the opposite edge is just vB→A = −vA→B.

The binary label r is associated to the undirected edge, and is denoted as rA→B for clarity. We compute rA→B as follows. First, from each boundary pixel p between A and B, we estimate the layer thickness hA(p) of the region A by first finding the shortest image-plane vector dp between p and another region that is not A or B but connected to A. hA(p) is then set to be the length of dp projected on the normal direction np (see Figure 7). Symmetrically, from p, we also estimate the layer thickness hB(p) of B using a similar step. Then, pixel p contributes a vote for rA→B. It votes for label ”0” if

Algorithm 1 Decoding Steps

1. procedure Decode
2. Process the image and segment the image pixels.
3. Build Connectivity Graph → § 4.1
4. while Traverse every path on the graph do → § 4.2
5. if Decodes on paths seen are in agreement then
6. Terminate Traversal Early → § 4.3
7. Vote on path candidates → § 4.2

Fig. 7. Layer thickness estimation. (Left) Looking at local pixel regions is not sufficient to guarantee that neighboring layers can be found. (Right) Computing the shortest path between neighbors as a vector or traced path will often not measure an accurate layer thickness. Distances must be projected along the printing direction (or boundary normal direction).

Fig. 8. Graph construction and traversal. (left) We identify individual pixel regions (A–E) through flood filling. (middle) We create a graph, where each node represents a pixel region, and two nodes are connected if their regions are adjacent to each other. Since the layers are added along the printing direction, it makes no sense to traverse back and forth along the printing direction for decoding—for example, A→B→C does not produce a valid bit string, while A→B→D is reasonable (right).
We terminate the graph traversal if we have surveyed a sufficient number of paths, and most of them are already in agreement. As shown in Figure 9, this is particularly useful when dealing with objects where the number of paths grows exponentially due to holes and other fine features.

We begin by imposing a lower bound on how many decoding paths to consider before checking for early termination. Once at least $K$ unique paths have successfully been decoded, we begin to tally the agreement across votes for each individual bit ($K = 64$ in practice). If all bits individually concur by 80% or more, the graph traversal terminates and outputs the agreed upon decoding. Otherwise, it continues. It is important not to vote on entire bit strings, but rather to vote on individual bits, as this way best allows early termination when there are only a few bits in discord.

### 4.3 Early Termination

We terminate the graph traversal if we have surveyed a sufficient number of paths, and most of them are already in agreement. As shown in Figure 9, this is particularly useful when dealing with objects where the number of paths grows exponentially due to holes and other fine features.

We begin by imposing a lower bound on how many decoding paths to consider before checking for early termination. Once at least $K$ unique paths have successfully been decoded, we begin to tally the agreement across votes for each individual bit ($K = 64$ in practice). If all bits individually concur by 80% or more, the graph traversal terminates and outputs the agreed upon decoding. Otherwise, it continues. It is important not to vote on entire bit strings, but rather to vote on individual bits, as this way best allows early termination when there are only a few bits in discord.

### 4.4 Extension: Depth Recovery

The black and white layers not only encode a bit string but impose a geometric structure that brings additional advantages. The appearance of these layers can be reinterpreted as a parallel light pattern projected to the object, and this interpretation has an interesting connection to the depth recovery using structured light techniques from computer vision [Taubin et al. 2014].

On the object surface, the boundary curves between any two consecutive layers by construction must be on a series of parallel planes. The distances between these planes depend on individual layer thicknesses. If we know the orientation of those planes with respect to the camera, we can recover the depth of every point on layer boundaries by intersecting a camera ray with the plane where the point resides. In this way, from a single image, we can recover the object’s depth. Unlike traditional structured light approaches, we require no active projector emitting light patterns. A graphic depiction of this idea is provided later in Figure ?? of Appendix ??.

In practice, when we place an object and photograph it, the coding layers are all parallel to the table surface (because of the way the layers are 3D printed). Thus, the layer plane’s orientation aligns with the table’s surface orientation, which can be inferred using a standard camera calibration process (e.g., with a checkerboard placed on the table). By decoding the bit string, we obtain every layer thickness, being it $h$, $Mh$, or $Nh$ (recall §3), and in turn the distances between the layer planes. The baseline thickness $h$ can be either set a priori or retrieved from the object that encodes the $h$ value. More details of this extension are provided in Appendix ??.

The estimated depth is useful in many ways, such as direct 2.5D image manipulation and image re-synthesis from novel viewpoints, as shown in Figure 10. In §5.4, we also demonstrate the use of depth information for virtual recovery of damaged objects.

### 5 FABRICATION

Since its inception, LayerCode has been designed to work with a wide variety of layered manufacturing methods. This section describes three different embodiments of LayerCode adapted to various types of 3D printers: Stratasys PolyJet, Ultimaker 2, and Autodesk Ember. Fabricating LayerCode objects on these 3D printers carries advantages and tradeoffs for each: varying from ease of implementation to visual concealment of the barcodes. Recognizing that some of the following approaches require augmentation of 3D
Although not all 3D printers support multi-material fabrication, the most direct way of making a LayerCode object is by using a multi-material 3D printer. Many 3D printers (e.g., MakerBot, MakerGear, and PolyJet) now support multi-material fabrication with decreasing costs. By mapping the black and white layers of a LayerCode tag to two colors of materials, these printers can produce LayerCode objects without any modifications to software or hardware.

As a demonstration, we use Stratasys PolyJet to fabricate two-color LayerCode objects (see Figure 12). Decoding these types of LayerCode objects is straightforward, as their surface textures are already in two colors. Simple thresholding in image space suffices to binarize the input camera image and prepare for decoding (as described in §4). While simple for fabrication, this type of LayerCode tags would change the object appearance. In certain applications (e.g., see [Li et al. 2017]), appearance preservation is desired, so unobtrusive or completely invisible barcodes are preferable.

The next two fabrication approaches aim to offer this feature.

5.1 Two-Color Fabrication

The direct way of making a LayerCode object is by using a multi-material 3D printer. Many 3D printers (e.g., MakerBot, MakerGear, and PolyJet) now support multi-material fabrication with decreasing costs. By mapping the black and white layers of a LayerCode tag to two colors of materials, these printers can produce LayerCode objects without any modifications to software or hardware.

As a demonstration, we use Stratasys PolyJet to fabricate two-color LayerCode objects (see Figure 12). Decoding these types of LayerCode objects is straightforward, as their surface textures are already in two colors. Simple thresholding in image space suffices to binarize the input camera image and prepare for decoding (as described in §4). While simple for fabrication, this type of LayerCode tags would change the object appearance. In certain applications (e.g., see [Li et al. 2017]), appearance preservation is desired, so unobtrusive or completely invisible barcodes are preferable.

The next two fabrication approaches aim to offer this feature.

5.2 Fabrication with Variable Layer Heights

Although not all 3D printers support multi-material fabrication, virtually all printers are able to print at a range of resolutions. Here the resolution indicates the height of a single layer deposited during the 3D printing process. We refer to it as the layer height (to avoid confusion with the aforementioned coding layer thickness).

Noticing printers’ ubiquitous ability of layer height control, we propose to use distinct 3D printing layer heights for each type of coding layers. When fabricating black layers, we use a small layer height \( h_0 \) (i.e., a high printing resolution); we switch to a larger layer height \( h_1 \) for making white layers (see Figure 14). This approach requires only a single material, and introduces little change to the surface geometry. Under environment lighting, the resulting LayerCode tags are barely noticeable to our eyes (Figure 14-e).

Interestingly, the small and large layer heights cause the two types of coding layers to have distinctive distributions of specular highlights. This can be understood by the illustration in Figure 13.
we provide a low-cost solution by augmenting an Autodesk Ember. Words, they appear nearly transparent in the visible light range but (SLA) resins—the materials commonly used in Stereolithography printers, when printing an object, Ember lowers its bottom window and can hold only one type of resin. Like most stereolithography printers, when printing an object, Ember lowers its build platform (which faces downward) to almost touch the bottom of the resin-filled tray. To grow a printing layer, a UV laser shines through the transparent tray bottom, and solidifies the part of resin between the build platform and tray bottom. We found that in this process the tray is fixed on a build plate which is the limiting factor if we wanted to add another 180° tray. Noticing this limitation, we custom designed a new build plate (Figure 15-d) which fits in the printer and supports two resin trays; each will be used to hold a different resin. To make use of both trays, we modified Ember’s firmware such that whenever a different coding layer is started, the printer 1) lifts its build platform, 2) switches the tray by rotating the build plate, 3) lowers down its build platform again, and 4) resumes the printing. More details of this augmentation are provided in Appendix ??, and we will open source the computer-aided design (CAD) models of the build plate, the firmware update code, and all instructions for amending the printer.

Figure 1-g shows a LayerCode object printed by our double-material Ember. To our eyes, the LayerCode tag is completely invisible, so the object’s appearance is fully preserved. To decode a tag, we image its carrier object in the NIR range by mounting a longpass filter with a cut-on wavelength at 850nm1 in front of the camera. No special camera is needed, as the conventional image sensor is capable of capturing NIR light.

When imaging the object indoors, we need to illuminate the object with a NIR light source (such as the 850nm and 950nm LEDs commonly used on TV remote controls). However, since sunlight has NIR wavelengths2, it can be exploited to expose LayerCode tags without resorting to additional lighting. Figure 16 shows a pair of images taken under uncontrolled natural daylight on a partly transparent vase (left) and the same object imaged with a NIR filter in front of the camera (right) to decode the NIR LayerCode tag.

1We use the filter from Thorlabs Inc. under this link.
2In fact, nearly all the infrared radiation in sunlight is near infrared.
A remarkable strength of LayerCode tags is the ability to decode even when the object is damaged, thanks to the layer-by-layer printing process that spreads the tag over the entire body of the object. For example, Figure 17-b shows an angel model with a broken wing. Nevertheless, we can still traverse the remaining part of the LayerCode graph and successfully decode the tag.

Since LayerCode tags are present inside prints, scuffing and moderate fractures do not inhibit use. However, similar to barcodes, one cannot recover from missing entire layers at any point along the code. This limitation is partially addressed by repetition of the codes. Resilient to physical changes, the LayerCode may still be recovered as long as one copy of the code remains present, even if damaged.

The ability to read tags from damaged objects opens the door to many applications. For example, through an embedded LayerCode tag, one can recover the original model to patch a broken piece [Teibrich et al. 2015]. Another scenario can apply to augmented reality. From a single image of the object (damaged or not), we can extract the interior of the object. Decoding the damaged piece reveals the embedded tag, from which we know 1) the original 3D model, and 2) its 3D depth and position with respect to the camera. (Right) This information enables a virtual repair of the angel displayed in an augmented reality fashion.

sunny day. The embedded LayerCode tag is discernible and can be successfully decoded.

We see LayerCode tags with great potential not just as a tag, but as a means of intellectual property (IP) protection and anti-counterfeit detection while preserving aesthetics.

5.4 Discussion on Implementation and Application
Across the three approaches described, as we make the LayerCode tags to better preserve object appearance, increasingly more software and hardware modifications are needed—there is no free lunch.

All three approaches have a minimal impact on the object’s mechanical strength, since they fully preserve the object’s volumetric shape (up to the printing resolution). In contrast, previous tagging approaches (such as [Kikuchi et al. 2018; Li et al. 2017; Willis and Wilson 2013]) all alter the object shape inside or near the surface.

In terms of printing time, all print jobs took from 40 minutes to several hours, depending on the model complexity. When printing with two colors and with variable layer heights, the time costs are comparable to printing without LayerCode tags. For Ember printing with NIR resin, we observed a minor overhead (about 15% to 20% slow-down) because of the extra tray swaps.

The decoding success rate across the entire database of each of the 30 viewing directions is color-mapped on a sphere, whose equator is aligned to the plane perpendicular to the printing direction. This mapping is unrolled in the Mercator projection, with representative views of a tagged bust shown (on top) for the red selected points of interest.

Moreover, we envision LayerCode tags being used for 3D printer steganography, similar in spirit to Machine Identification Codes (also known as the Yellow dots), a watermark that many paper printers and copiers leave on every single printed page for device identification. Our LayerCode tags (especially the unobtrusive ones) allow 3D printers to introduce similar watermarks for the same purposes (e.g., counterfeit detection) as those that have motivated paper printers.

6 EVALUATION ON VIRTUAL DATASET
To understand the performance of our decoding algorithm more thoroughly, we also test our algorithm on a large dataset of shapes using synthetic images generated by a photorealistic renderer. A glimpse of the tested shapes is shown in Figure 23. This evaluation over such a virtual dataset is justified by several considerations:

i. Cost and time. In regards to both cost and time, it is unaffordable to 3D print all the shapes in the dataset. 3D printing of a single object is usually an hour-long process, barring failures. Virtual rendering of 3D printed objects, on the other hand, can be finished in a short time, and the resulting images are photorealistic.

ii. Feasibility. For many complex shapes that we use in this evaluation, it is hard, if not impossible, to fabricate them via current commodity 3D printers. But 3D printing technology is constantly and rapidly improving. Therefore, it is desirable to test our algorithm on those complex shapes to prepare for the future.

iii. Thoroughness. In a virtual environment, we can test our algorithm using a large number of objects viewed from many camera angles. Thoroughly testing over all these variances provides us statistical insights which in turn guide our use of LayerCode tags in practice. This thoroughness is made possible only through virtual experiments.

6.1 Database Construction
We tested our algorithm over a set of shape meshes from the Thingi10k dataset [Zhou and Jacobson 2016]. The testing shapes are selected through the following “printability” criteria: 1) They must be watertight 2-manifolds (i.e., no self-intersections), and 2) have only a single connected component. 3) They should also have consistent surface normals without degenerate faces. Following these criteria, we obtain 4,835 meshes.

Each of these meshes is processed to embed a LayerCode tag indicating the mesh’s database ID. When we encode the tag (using the procedure in §3), the printing direction is chosen to be the longest dimension of an axis-aligned bounding-box containing the mesh, and the baseline layer thickness $h$ is set to repeat the tag three times. The output of the encoding step is a shape with two sets of coding layers ready for rendering. Each type of layer is assigned a different material color (i.e., red and blue). We then use the physics-based renderer Mitsuba [Jakob 2010] to generate a photorealistic image from a chosen camera angle.

To understand how the view angles affect the decoding, we uniformly sample 30 viewing directions on a sphere co-centered with the object. Sampled views near the poles aligned with the printing direction are discarded, since looking along the printing direction is unlikely to reveal the entire tag. Figure 23 shows 18 representative shapes and the rendered images from multiple view angles. The decoding algorithm takes as input only a single image, and so each given view is decoded independently.

6.2 Results Statistics

Camera angle dependency. Only one photo from a single view is needed for decoding. However, due to surface curvature and local occlusions, the coding layers are better captured from certain angles. A natural question is what camera angles are more suitable for decoding the tag. Figure 18 reports our experiment results, suggesting that view directions just north or south of the equator is statistically the most promising for decoding tags. This is somewhat counter-intuitive, as one might expect the directions at equator to be the most promising.

Our hypothesis is that these slightly tilted view angles allow the coding layers to be viewed by avoiding occlusions introduced by bulges at one end or the center of the shape. For example, the bust in Figure 18-top has its head and shoulders protrude from its center axis, making decoding hard from above but much easier from below.

Figure 19 shows that some shapes can accommodate a wider range of view angles than others for successful decoding. For example, one shape is readable from all 30 views, whereas 44 other shapes are not decodable at all (which account for only 0.9% of the shapes in the dataset). On average, for any given shape, its tag is readable from 51% of the viewing directions we sampled. Overall, 78.0% of the shapes can be decoded in 10 view directions, 49.5% can be decoded in 15 directions, and 21.7% can succeed in 20 directions.

Timings. Decoding time varies from seconds and up to 5 minutes, depending on specific shapes and view angles. Much of the time complexity is derived from our graph based approach, which consists of image and graph processing steps that are slower than simpler approaches, but allow LayerCode to handle a significantly broader diversity of shapes with one consistent algorithm. Notably, profiling reveals the majority of capture time is spent on pixel-wise graph building operations including morphological image processing, computing neighboring region distances, and masking. Each graph node may be treated in parallel for time-critical applications.

Image resolution also impacts decoding time, since resolution will vary the size of the many image processing operations. Our experiments use a fixed resolution 2048 × 2048 px in all rendered images, on par with modern smartphone cameras. Similarly the complexity of the extracted graph will also factor into decoding time. Simpler shapes, curvy or flat, lead to smaller number of graph nodes, and thus are faster to decode. On the other hand, holes or occlusions tend to split coding layers on the image plane into separate graph nodes and result in a larger graph. Thus, shapes with many holes and fine structures take longer to decode.

Lower bound of $h$. In §3, we derive the upper bound of $h$ from Lemma 1. A smaller $h$ allows the object to host more copies of the tag. But if $h$ is too small, the coding layers will become hardly discernible on the image. In an experiment, we progressively reduce $h$ and encode only a single copy of an ID in the object. In this process, we keep the camera angle and image resolution unchanged, and check at what $h$ value the decoding would fail. Not surprisingly, the
At each iteration, we encode an ID in the current shape and check if the resulting shape can hold a readable LayerCode tag. 324 holes of random radii are added before decoding is no longer possible. Decoding is possible even when the final wedge is 10.81% of its original volume and has many fine features.

**Stress test.** To further gain some insights on to what extent the shape may have fine features while remaining decodable, we designed a stress test, inspired by the shape of Swiss cheese. Starting with a wedge shape, we iteratively add holes with a random radii at random locations (see Figure 21). As more holes are hollowed, parts of the shape become thinner and more fine features emerge. At each iteration, we encode an ID in the current shape and check if it can be decoded. Eventually, decoding algorithm fails when the shape is hollowed out until only 10% of its original volume is left.

**Failure cases.** Out of the 4,835 tested shapes, 44 cannot be decoded at all. We discuss three of them shown in Figure 22. The Escher-like staircase exhibits highly complex topology at most camera angles, which makes it hard to find a complete path on the decoding graph. The spiral appears simple at first glance, but each spiral always occludes some coding layers, and so the entire tag is hardly seen from any given angle. Similarly, the bumpy blob occludes itself all over, and the heavy shadows spread over the surface, causing ambiguities and making the image processing prone to errors.

7 LIMITATIONS & CONCLUDING REMARKS

We have presented LayerCode, a tagging scheme that embeds carefully designed optical barcodes as a deliberate byproduct of the existing layer-by-layer 3D printing process. At its core, a LayerCode tag is an optical barcode readable by a conventional camera. For this reason, it also retains a few limitations of standard optical barcodes.

Foremost, LayerCode tags are agnostic to choices regarding viewing angles, printing orientations, and application semantics, yet requires a direct line of sight for decoding. If an object is completely occluded or poorly illuminated, decoding will fail. The ability to decode also depends on the camera view angle. While as shown in our experiments, LayerCode tags can be correctly read from a wide range of camera angles, there are other view angles (such as those nearly aligned with the printing direction) from which the decoding is prone to failure. Therefore, optimizing for how a shape might be held, seen standing, or made less visible would certainly improve robustness. Similarly, since not all angles are equally easy to decode, processing multiple views in parallel to achieve more robust decoding also serves as an exciting avenue for future work.

Our decoding algorithm runs for up to tens of seconds, slower than decoding a regular barcode. This is partly because our current implementation uses Matlab, and partly because we wish to explore a sufficient number of decoding paths for the sake of robustness. Then, an interesting future direction is how to speed up the decoding algorithm. If we can significantly shorten the decoding time, it would be possible to decode from a multi-frame capture or a short video clip, and further improve the robustness.

In our NIR Ember printing process, a small detail might cause a practical concern. Every time the printer starts a new coding layer, the build platform switches from the tray holding one resin to another tray filled with the second type of resin. As a result, every such switch brings a small amount of resin in one tray to another. In our experiments, though this cross mixing of resins causes no negative effects on the tag's readability. But if we were to print for an extended period, resin contamination would be accumulated, and might become a practical issue to consider.

Despite LayerCode's potential for IP protection and counterfeit detection, it is not a physically one-way tag (meaning one that is "easy to compute, but hard to invert" [Rompel 1990]). With a high-resolution camera for measuring coding layer thicknesses and a spectrometer analysis of NIR resin formula, it is possible to reverse engineer and counterfeit a LayerCode tag on another 3D printer. To achieve truly unclonable tags, we might have to consider a fusion of optical codes, RFIDs, and other new modalities. This remains as an interesting future direction.
Fig. 23. Successfully decoded shapes. A peek into the diversity of tested shapes within our database. Each view presented is correctly decoded by our graph-based algorithm. Shapes with bumpy, shell, thin, curvy, and other challenging properties showcased here are still subject to encoding and decoding by our LayerCode approach. Three shapes indicated by the stars are discussed in the main text.
Last but not least, LayerCode tags use only two types of coding layers to encode a bit string, corresponding to the black and white codes in standard barcodes. In theory, there is no limitation on how many coding “colors” can be used. As 3D printers become more precise and robust, it is possible to extend LayerCode to use ternary or quaternary coding layers for higher information capacity. One can also explore a generalized version where a pairwise ratio may go beyond 1 and even $M^2$ or even $M^3$ to support thicker layers near challenging regions (see inset). For all that, LayerCode is the first step toward robustly tagging complex, 3D printed shapes, and it is our hope that our open-sourced code, hardware, and benchmark database can help the research community develop more robust and ubiquitous physical tagging mechanisms.

ACKNOWLEDGMENTS

We would like to thank Qingnan Zhou for sharing code to generate the database mosaic, as well as Joni Mici, Bill Miller, and Mohamed Haroun for their assistance with printing. We thank Eitan Grinspun and Oded Stein for their helpful discussions, along with Anne Fleming for proofreading. The authors would also like to thank the anonymous referees for their valuable comments and helpful suggestions. The work is supported by the National Science Foundation anonymous referees for their valuable comments and helpful suggestions.

REFERENCES


