# LayerCode: Optical Barcodes for 3D Printed Shapes

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

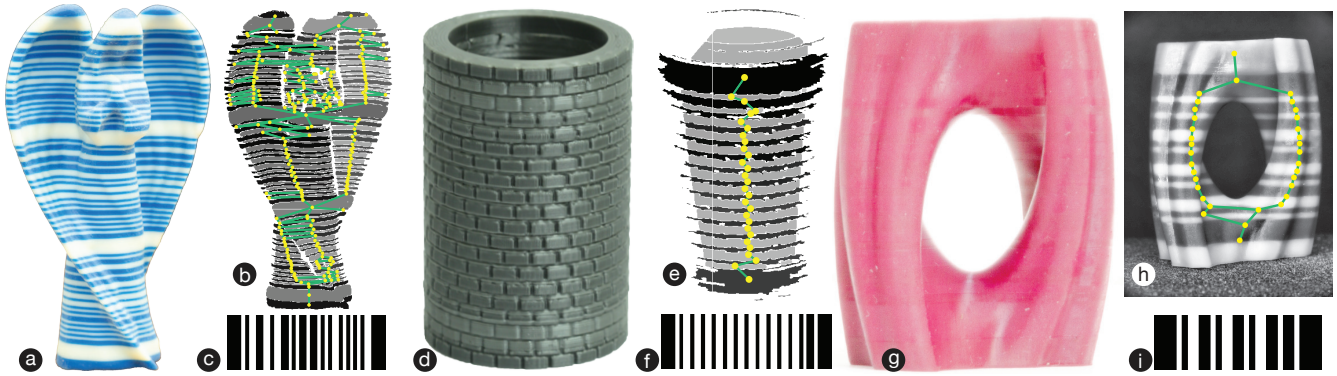


Fig. 1. **LayerCode tags** are deployed in 3D printed objects through two-color printing (a), variable layer heights (d), and near-infrared steganography (g). In the first case (a), the LayerCode tag is visible; in the second (d), the tag is less visible; and in the third (g) it is completely invisible, but still machine-readable. Just like reading a barcode, we capture an image of each object, and our decoding algorithm processes the image to create a decoding graph (b, e, h), from which a linear barcode is recovered (c, f, i). In this case, the corresponding LayerCode bit string reveals a 24-bit code repeated 3 times in (a), a 24-bit code repeated once in (d), and a 12-bit code repeated once in (g).

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.

Authors' addresses: Henrique Teles Maia, Columbia University, henrique@cs.columbia.edu; Dingzeyu Li, Adobe Research, dinli@adobe.com; Yuan Yang, Columbia University, yy2664@columbia.edu; Changxi Zheng, Columbia University, cxz@cs.columbia.edu.


0730-0301/2019/7-ART1 $15.00
https://doi.org/10.1145/3306346.3322960

CCS Concepts: • **Hardware → Emerging interfaces**; • **Mathematics of computing → Graph algorithms**;

Additional Key Words and Phrases: 3D printing, information embedding, fabrication, physical hyperlinks

**ACM Reference Format:**
Henrique Teles Maia, Dingzeyu Li, Yuan Yang, and Changxi Zheng. 2019. LayerCode: Optical Barcodes for 3D Printed Shapes. *ACM Trans. Graph.* 38, 4, Article 1 (July 2019), 17 pages. https://doi.org/10.1145/3306346.3322960

## 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 interleave 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.

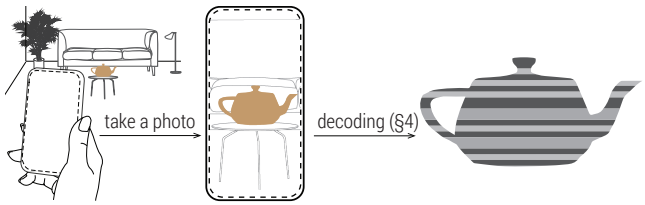


Fig. 2. **Use scenario.** A LayerCode-tagged object is captured by a conventional camera. Our graph-based algorithm then decodes the embedded information from the image.

*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.

*Appearance preservation.* LayerCode tags, when fabricated using two materials of different colors, change the appearance of the object. However the object appearance is preserved in the other two 3D printing approaches, namely by changing the FDM deposition thickness and using resins mixed with NIR dyes (Figure 14 and 16).

*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.

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

| | barcode | Printed Optics [Willis et al. 2012] | InfraStructs [Willis and Wilson 2013] | Acoustic Barcodes [Harrison et al. 2012] | Acoustic Voxels [Li et al. 2016] | Lamello [Savage et al. 2015] | RFID based [Iyer et al. 2018, 2017] | BlowHole [Tejada et al 2018] | AirCode [Li et al. 2017] | LayerCode |
|---|---|---|---|---|---|---|---|---|---|---|
| unsmooth rough surfaces | ✗ | ✓ | ✓ | ✗ | ✓ | ✗ | ✓ | ✓ | ✗ | ✓ |
| thin shell or rod | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✓ |
| accessible hardware decoding | ✓ | ✗ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| tested on multiple 3D printers | N/A | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ |
| structural preservation | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ |
| appearance preservation | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✓ | ✓ |
| ubiquitous tagging | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ | ✓ | ✗ | ✗ | ✓ |
| depth estimation for free | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ |

## 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.

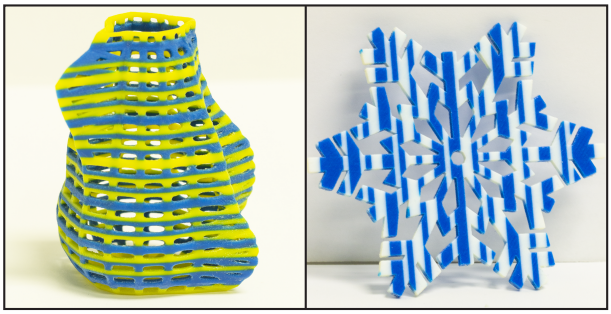

Fig. 3. **Challenging shapes.** LayerCode tags can be embedded and decoded successfully in challenging shapes such as those with holes, thin features, curved surfaces, and branching threads. To our knowledge, no previous optical tagging mechanism can handle these challenging shapes.

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 [Zucheul 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.

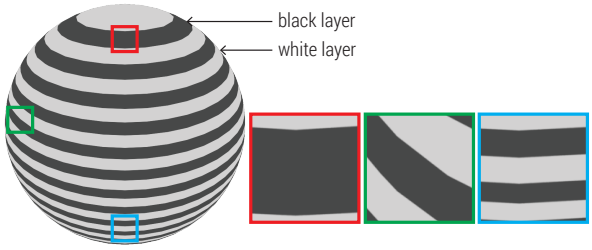


Fig. 4. **Distorted thickness.** A sphere is coded with black and white layers of equal thickness. On the captured image, curvature and perspective cause layers to appear spatially varying in size.

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 this 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.

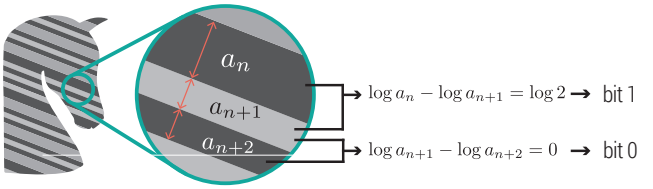


Fig. 5. **Encoding scheme.** Each pair of layers encodes a single bit. A bitwise 0 or 1 can be determined by computing the ratio of adjacent layer thicknesses.

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,

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

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

$$\mathrm{b}_{n+1} = \begin{cases} 1 & \text{if } \log a_n - \log a_{n+1} = \pm \log M, \\ 0 & \text{if } \log a_n - \log a_{n+1} = 0. \end{cases} \tag{2}$$

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 $M = 2$.

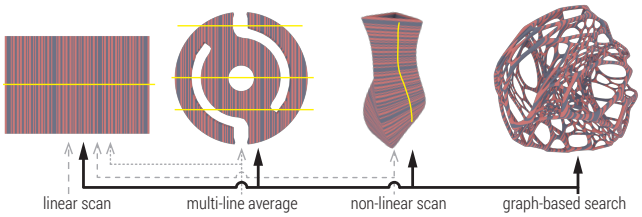


Fig. 6. **Barcode challenges.** Simple input shapes (Left) are followed by more challenging geometry (Right). Linear scan works with simple flat surfaces, but cannot generalize to flat pieces with holes. These might be decoded by projecting all pixels to one dimension; however, globally projecting fails to handle curved objects. Instead, locally tracing across layers is effective, until subsequent layers are too far to trace (e.g., see Figure 7-left). Lastly, our graph-based method can be used to handle highly complex shapes, and is backwards compatible with all previous challenging shapes.

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 $Nh$, 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 $Nh$. 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 $T$ ($T$ 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 ending layer of thickness $Nh$, 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 $Mh$), 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 $\left\lceil \frac{D}{h \cdot M} - \frac{2N+3}{M} \right\rceil$.

*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 $D/(2N+2+T+M)$. Oftentimes, $h$ 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

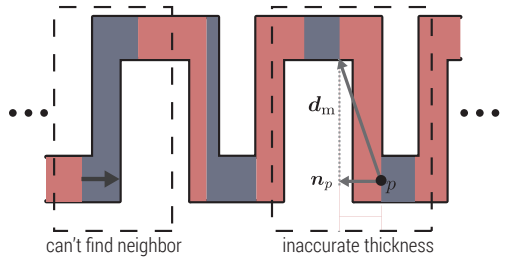


Fig. 7. **Layer thickness estimation.** (Left) Looking at local pixel regions is not sufficient to guarantee that neighboring layers can be found. (Right) Computing the shortest path between neighbors as a vector or traced path will often not measure an accurate layer thickness. Distances must be projected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

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 A 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 $\boldsymbol{v}$ in image space and a binary label $r$. Consider an edge $e$ that connects nodes $A$ and $B$. Its vector $\boldsymbol{v}$ 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 $\boldsymbol{v}$, we first identify boundary pixels in each region. These are the pixels within $\delta$ pixels away from another region ($\delta = 3$ 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. $\boldsymbol{v}$ 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 $\boldsymbol{n}_p$ for pixel $p$ in region $A$, and the opposite normal direction $-\boldsymbol{n}_p$ for $p$ in $B$. Thus, the average direction $\boldsymbol{v}$ is in fact associated to the *directed edge* from $A$ to $B$, and for clarity we denote it as $\boldsymbol{v}_{A\to B}$. The direction for the opposite edge is just $\boldsymbol{v}_{B\to A} = -\boldsymbol{v}_{B\to A}$.

The binary label $r$ is associated to the undirected edge, and is denoted as $r_{A\leftrightarrow B}$ for clarity. We compute $r_{A\leftrightarrow B}$ as follows. First, from each boundary pixel $p$ between $A$ and $B$, we estimate the layer thickness $h_A(p)$ of the region $A$ by first finding the shortest image-plane vector $\boldsymbol{d}_m$ between $p$ and another region that is not $A$ or $B$ but connected to $A$. $h_A(p)$ is then set to be the length of $\boldsymbol{d}_m$ projected on the normal direction $\boldsymbol{n}_p$ (see Figure 7). Symmetrically, from $p$, we also estimate the layer thickness $h_B(p)$ of $B$ using a similar step. Then, pixel $p$ contributes a vote for $r_{A\leftrightarrow B}$. It votes for label "0" if

**Algorithm 1** Decoding Steps

```
procedure DECODE
    Process the image and segment the image pixels.
    Build Connectivity Graph                                  ▷ § 4.1
    while Traverse every path on the graph do                 ▷ § 4.2
        if Decodes on paths seen are in agreement then
            Terminate Traversal Early                         ▷ § 4.3
    Vote on path candidates                                   ▷ § 4.2
```

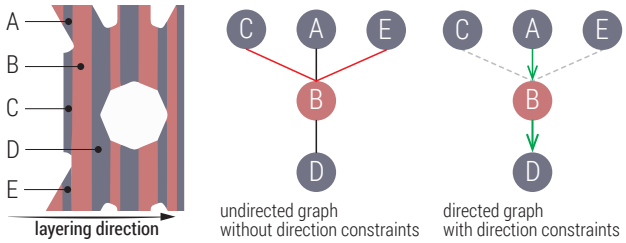


Fig. 8. **Graph construction and traversal.** (left) We identify individual pixel regions (A-E) through flood filling. (middle) We create a graph, where each node represents a pixel region, and two nodes are connected if their regions are adjacent to each other. Since the layers are added along the printing 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).

$|\log h_A(p) - \log h_B(p)| < \frac{1}{2}\log M$ (i.e., closer to 0), indicating the second case in (2) and suggesting a bit "0" encoded between $A$ and $B$. On the other hand, if $|\log h_A(p) - \log h_B(p)| \geq \frac{1}{2}\log M$, it votes for label "1", suggesting the first case in (2) and hence a bit "1". The final label $r_{A\leftrightarrow B}$ is taken as the majority vote over all boundary pixels.

At first glance, assigning the label $r_{A\leftrightarrow B}$ requires a prior knowledge of $M$, which is not known from the image. Fortunately, our coding scheme presented in §3 enables an easy and robust way of estimating $\log M$. In the above process, we collect all $|\log h_A(p) - \log h_B(p)|$ values for all boundary pixels on the image. From (2), we know that these values are expected to be either $\log M$ or 0, although we do not know what $M$ is. If we think of each $|\log h_A(p) - \log h_B(p)|$ value as a random variable, these random variables must be generated through a mixture of two Gaussians (in 1D): one is centered at 0, and another center (i.e., $\log M$) is unknown but can be estimated using maximum likelihood estimation [Nasrabadi 2007].

This Gaussian mixture estimation also enables us to identify the starting nodes, which correspond to the starting layers (of thickness $Nh$) described in §3. If node $A$ corresponds to a starting layer, then the estimated $|\log h_A(p) - \log h_B(p)|$ values from its boundary pixels will appear as outliers of the Gaussian mixture model, as they are considerably larger than $\log M$. If this case is encountered, we label $A$ as a potential starting node and include it in a set $\mathcal{S}$.

## 4.2 Decoding through Graph Traversal

We now decode the bit string by traversing the graph. Our traversal repeatedly starts from each node in the set $\mathcal{S}$, and moves to the next node through a depth-first search (DFS). Because the object is always 3D printed in a layer-by-layer fashion, we must avoid looping back to earlier layers during the traversal. To this end, the direction vector associated to each edge is helpful. As illustrated

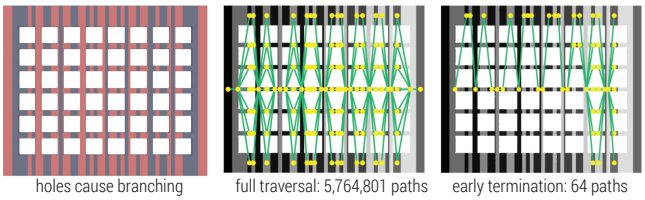


Fig. 9. **Early termination.** (left) Holes and fine features leads to a large decoding graph with many branches. (middle) As a result, a naïve graph traversal unnecessarily explores too many decoding paths. (right) Early termination allows us to declare a tag with confidence after processing just a small fraction of the available paths.

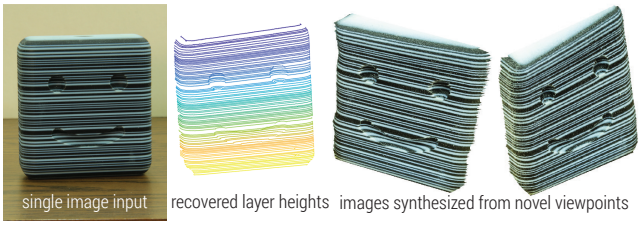


Fig. 10. **2.5D image re-synthesis.** An object carrying a LayerCode tag also carries depth information for free. From a single image, we can estimate the object's 3D coordinates with respect to the camera (middle), which in turn allows us to re-synthesize images from other viewpoints (right).

in Figure 8-c, consider a traversal that reaches a node $B$ from a node $A$. In the DFS, we visit the next node $D$, only when the moving direction from $A$ to $B$ is approximately consistent with the moving direction from $B$ to $D$. In other words, we require $\boldsymbol{v}_{A\to B}\cdot\boldsymbol{v}_{B\to D} \geq \Delta$ ($\Delta = 0.35$ in all our examples).

The traversal stops when a node in $\mathcal{S}$ is reached or when DFS runs out of unvisited nodes. In the latter case, the current traversal path is simply discarded, as we expect a valid bit string to always end with a thick ending layer (recall §3), which must have been included in $\mathcal{S}$. In the former case, we decode a bit string by concatenating the binary labels of all edges on the path. It is worth noting that this path might traverse a bit string backwards. If that happens, we would decode a bit string starting with "1" and ending with "0". From our coding scheme in §3, it is easy to see that we can just reverse the bit string to obtain the original one.

This graph traversal process generates many paths and thus many bit strings. Some of them might be erroneous due to image noise. But collectively, they are robust. Therefore, we finalize the bit string by taking a bit-wise majority vote over all decoded bit strings.

*Remark.* The majority voting, albeit simple, is a fundamental philosophy behind many modern error-resilient systems, from peer-to-peer networks, to Byzantine fault tolerance, to the current emergence of blockchain technology (e.g., see [Lamport et al. 1982; Nakamoto 2008]). Here, we exploit the voting scheme in both assigning the edge labels and decoding the traversal paths. From this very perspective, the aforementioned condition $\boldsymbol{v}_{A\to B}\cdot\boldsymbol{v}_{B\to D} \geq \Delta$ should be seen as a way of culling votes that are likely rejected. It is meant to accelerate the graph traversal but it is not necessary to ensure correctness. Thus, the choice of $\Delta$ is not sensitive.

### 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 individual bits, as this way best allows early termination when there are only a few bits in disaccord.

### 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 on 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 B.

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 B.

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


Fig. 11. **Fabricated pieces** carry LayerCode tags made by two-color printing, variable layer heights, and near-infrared resins. LayerCode tags are successfully tested on bumpy, shell, curvy, and otherwise complex geometry.

printer firmware and/or hardware, we open source all our firmware codes and hardware modifications.

We use a Canon DSLR camera with $5184 \times 3456$px to take photos. To read NIR LayerCode tags, we used a Grasshopper3 camera from Point Grey with a resolution $2048 \times 1536$px for its easy adoption of the NIR filter. We also tried an iPhone camera with a resolution $4032 \times 3024$px and found the results similar.

### 5.1 Two-Color 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.

### 5.2 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.

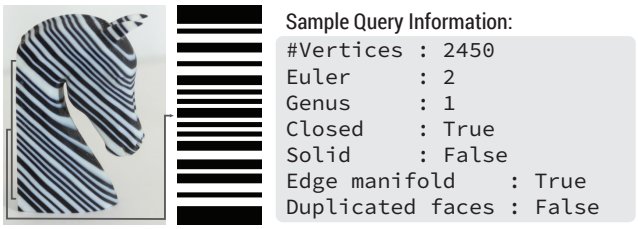


Fig. 12. The 24 bit LayerCode tag embedded in this Zebra-shaped object is repeated twice and reveals shape and related mesh information.

Exploiting this difference, the decoding algorithm is able to segment black coding layers from white coding layers from a camera image. The image processing steps (for producing the input of §4) are outlined in Figure 14 and detailed in Appendix A.

In practice, although the layer height is adjustable, almost all existing 3D printing software use a single layer height for printing an object. We overcome this limitation by carefully constructing a `G-code` program that runs on the 3D printer and instructs when to switch the layer height. Figure 14 depicts this implementation. We use the first-party slicer to generate a `G-code` program that prints the object with the small layer height $h_0$ (Figure 14-b), and another `G-code` program that prints at the larger layer height $h_1$ (Figure 14-b). We then interweave these two at specific locations to construct alternating printing heights (Figure 14-b). In our approach, we always set $h_1$ as an integer multiple of $h_0$ to ensure seamless switches across layer heights.

Since these simple `G-code` manipulations require no hardware changes, we envision that this type of LayerCode tags can be readily incorporated into existing 3D printers with an over-the-air software update. On the other hand, while its impact on object appearance is minimal, this impact is not completely invisible. If stringent appearance preservation is a priority, we recommend the next approach, one that embraces near infrared (NIR) optical properties for LayerCode embodiment.

### 5.3 Fabrication with Invisible Near-Infrared Dye

Inspired by ColorMod [Punpongsanon et al. 2018] which uses photochromic inks to recolor objects after their printing, we propose to control the NIR optical properties of Stereolithography Apparatus

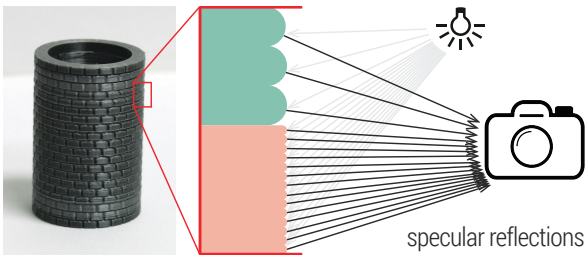


Fig. 13. **Distinctive highlight distributions.** The black layers (orange color) are made of 3D printing layers each with a small height, while the white layers (green color) have a much larger 3D printing layer height. As a result, the specular highlights in black layers appear sparser and more granular, while the highlights in white layers are denser and more uniform. The difference of highlight distributions allows the decoding algorithm to discern the two types of coding layers when processing a camera image.

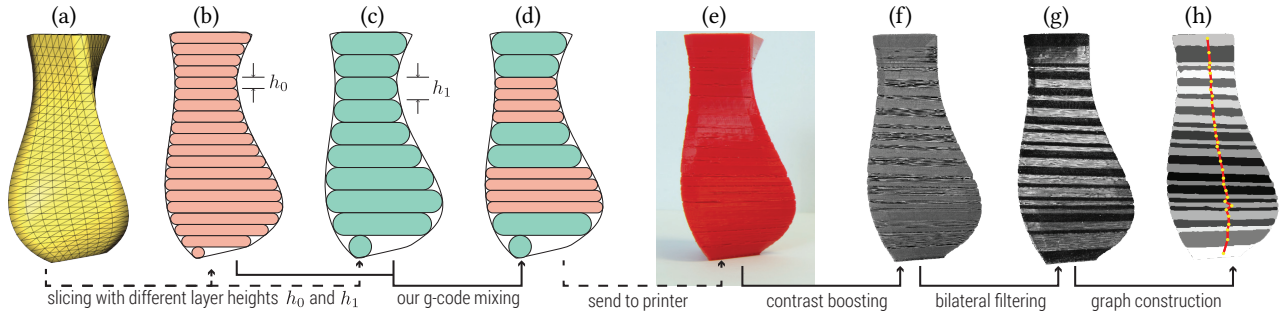


Fig. 14. **Variable layer heights.** A twisted vase is encoded with a variable height LayerCode (a-d), printed (e), and then decoded (f-h). At the decoding time, a camera image (e) is converted into grayscale, followed by contrast boosting (f), bilateral filtering (g), and a Gaussian-mixture-based clustering to binarize the image (h), which is in turn supplied to the decoding algorithm for graph construction and decoding along paths (red curve on (h)).

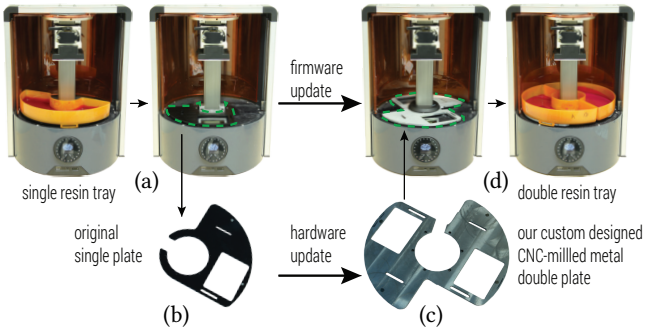


Fig. 15. **Hardware augmentation of Autodesk Ember.** (a) The Autodesk Ember has only one resin tray, and thus cannot support two types of coding layers. We replace the build plate (b) on its rotational platform with a new CNC-milled plate (c) that supports two trays (d). This hardware augmentation together with a firmware modification allows us to deploy NIR LayerCode tags in 3D printed objects.

(SLA) resins—the materials commonly used in Stereolithography printers—using NIR dyes.

NIR dyes are granular substances based on small organic molecules, commonly used in chemical biology and industrial applications [Escobedo et al. 2010; Falkenstern et al. 2018]. They have strong optical absorption in the NIR range (i.e., 700 ~ 1100 nanometers in wavelength), but weak absorption in the visible light range. In other words, they appear nearly transparent in the visible light range but dark in the NIR range. Thanks to this property, we can darken the NIR "color" of a 3D printing material while leaving its visible appearance unchanged by mixing a certain amount of NIR dye in the 3D printing resin. In practice, we mix 35mg of 828nm dye into every 100ml of PR-57 CMYK+W resin. Mechanical stirrers are employed for a day to ensure an even mixture of the dye in the resin.

This procedure creates the resin for one type of our coding layers, and for the other type, we use the original, untouched resin. The challenge is how to use both resins for a single print and switch one resin tray to another for every coding layer. For high-end, expensive 3D printers, it is possible to use both resins simultaneously. Here, we provide a low-cost solution by augmenting an Autodesk Ember.

Autodesk Ember is a stereolithography printer. As shown in Figure 15, it comes with one 180° tray (in orange) with a transparent 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

Fig. 16. **NIR LayerCode tags in sunlight.** The NIR LayerCode tags remain invisible in sunlight (left), but become visible when imaged with a NIR filter in front of the camera (right). No additional light source is needed.

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 D, 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 850nm[1] 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 wavelengths[2], 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

[1]We use the filter from Thorlabs Inc. under this link.
[2]In fact, nearly all the infrared radiation in sunlight is near infrared.

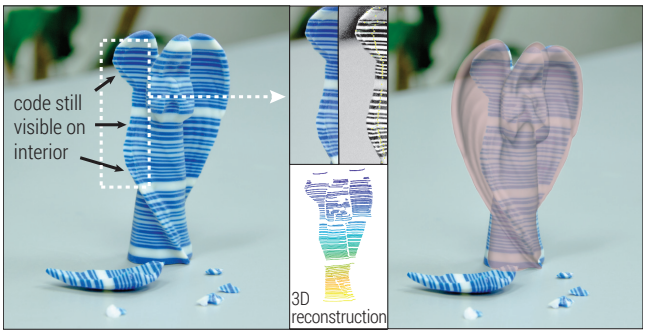


Fig. 17. **LayerCode tag and augmented reality.** (Left) A fallen angel damages its wing. However, the LayerCode tag can still be read from the interior of the object. Decoding the damaged piece reveals the embedded tag, from which we know 1) the original 3D model, and 2) its 3D depth and position with 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.

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

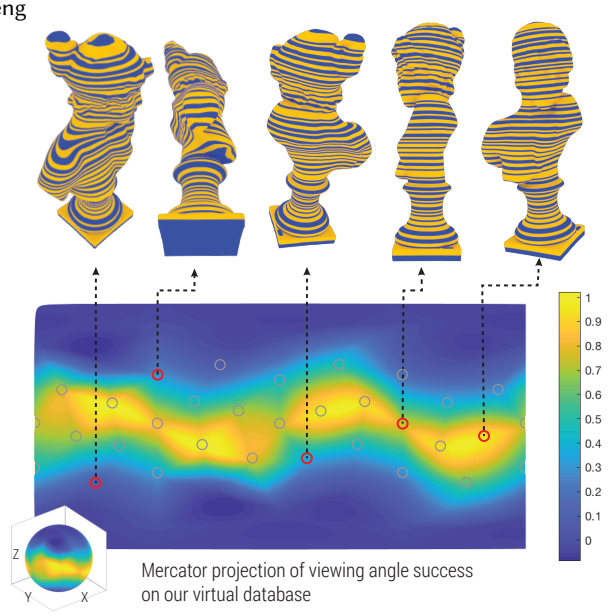


Fig. 18. **The decoding success rate** across the entire database of each of the 30 viewing directions is color-mapped on a sphere, whose equator is aligned to the plane perpendicular to the printing direction. This mapping is unrolled in the Mercator projection, with representative views of a tagged bust shown (on top) for the red selected points of interest.

the tag and estimate its 3D position with respect to the camera (through depth recovery in §4.4), and display an augmented object (e.g., the original model of a currently damaged one) or animate the static object in 3D (e.g., similar to previous efforts on animating books [Billinghurst et al. 2001; Cimen et al. 2018]).

Moreover, we envision LayerCode tags being used for 3D printer steganography, similar in spirit to Machine Identification Codes[3] (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.

[3] https://en.wikipedia.org/wiki/Machine_Identification_Code

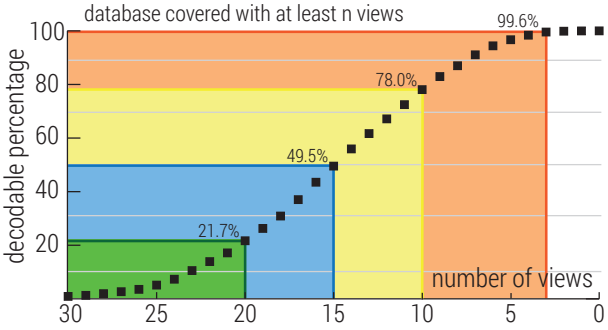


Fig. 19. We plot the distribution of all 4,835 tested shapes with respect to the number of view angles from which they can be decoded successfully. 99.6% of the shapes can be decoded from at least three sampled view directions.

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.

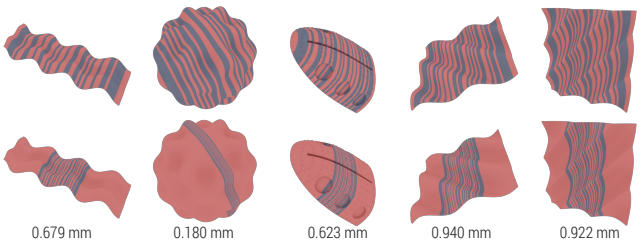


Fig. 20. **Lower bound of $h$.** The decoding becomes challenging if the coding layers are made too thin. Here we show the smallest baseline layer thickness $h$ still readable under different views for shapes normalized to 10cm in length along the printing direction.

Our hypothesis is that these slightly titled 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 $2048 \times 2048$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

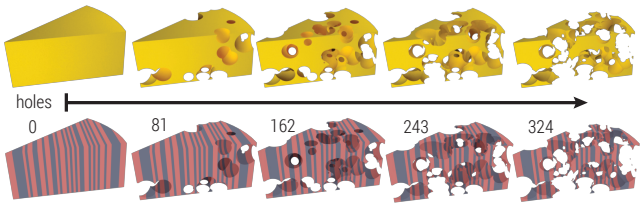


Fig. 21. **Stress test.** From left to right, we keep adding holes to the wedge and check if the resulting shape can hold a readable LayerCode tag. 324 holes of random radii are added before decoding is no longer possible. Decoding is possible even when the final wedge is 10.81% of its original volume and has many fine features.

lower bound of $h$ depends on the object shape. Figure 20 reports the results.

*Shape complexity.* Figure 23 lists some of the nontrivial shapes from our dataset, all of which can be successfully decoded. These shapes all possess a mix of the following challenging features: bumpy surface, thin shell, thin rods, sharp corners, highly occluded surfaces, holes, and so forth. A complete set of shapes including 4,791 successful shapes and 44 failure cases, is provided in the supplementary file. Here we highlight and discuss three of the shapes.

WaterSplash is highly irregular. LayerCode tag manages to survive the fine and thin features by leveraging the solid external base and internal regions. BunnyStripe is a typical thin shell model with stripe-like surface patches. Despite of the holes and discontinuity showing on the images, our graph-based decoding algorithm is able to find valid paths leading to correct decoding. StratumVase is yet another extremely challenging shape with more than 180,000 faces. The shape is similar to Spiral (Figure 22-a) on which our algorithm fails. But a key difference is that the thin slices here do not occlude or shadow the neighboring regions because of the orientation alignment with the printing direction.

*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 an 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.

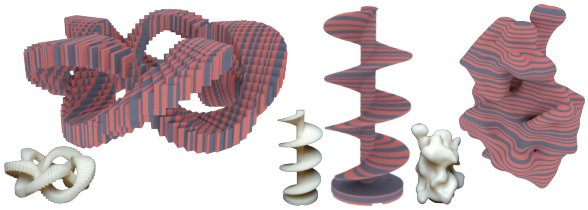

Fig. 22. **Failure cases.** Among the 4,835 shapes, 44 shapes cannot be decoded. Here are three challenging failed shapes.

*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 colors 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 $M$ to $M^2$ 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 in part by the National Science Foundation under Grant No. 1816041 and 1644869.

## REFERENCES

Marc Alexa, Kristian Hildebrand, and Sylvain Lefebvre. 2017. Optimal Discrete Slicing. *ACM Trans. Graph.* 36, 1 (2017), 12:1–12:16.

Mark Billinghurst, Hirokazu Kato, and Ivan Poupyrev. 2001. The Magicbook - moving seamlessly between reality and virtuality. *IEEE Computer Graphics and applications* 21, 3 (2001), 6–8.

Weifeng Chen, Zhao Fu, Dawei Yang, and Jia Deng. 2016. Single-image depth perception in the wild. In *Proc. NIPS*. 730–738.

Gokcen Cimen, Ye Yuan, Robert W Sumner, Stelian Coros, and Martin Guay. 2018. Interacting with Intelligent Characters in AR. *International SERIES on Information Systems and Management in Creative eMedia (CreMedia)* 2017/2 (2018), 24–29.

Steve Crayons. 2016. Variable Slicing for 3D Printing on Autodesk Ember. https://www.instructables.com/id/Variable-Slicing-for-3D-Printing-on-Autodesk-Ember/. [Online; accessed 30-December-2018].

Gary A England. 1996. Method of reading a barcode representing encoded data and disposed on an article and an apparatus therefor. US Patent 5,510,604.

Jorge O Escobedo, Oleksandr Rusin, Soojin Lim, and Robert M Strongin. 2010. NIR dyes for bioimaging applications. *Current opinion in chemical biology* 14, 1 (2010), 64–70.

Kristyn R Falkenstern, Alastair M Reed, Vojtech Holub, and Tony F Rodriguez. 2018. Digital watermarking and data hiding with narrow-band absorption materials. US Patent App. 15/669,103.

Olaf Hall-Holt and Szymon Rusinkiewicz. 2001. Stripe boundary codes for real-time structured-light range scanning of moving objects. In *Proc. ICCV*, Vol. 2. IEEE, 359–366.

Chris Harrison, Robert Xiao, and Scott E. Hudson. 2012. Acoustic barcodes: passive, durable and inexpensive notched identification tags. In *UIST 2012*.

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 *Proc. TEI*. ACM, 341–350.

Vikram Iyer, Justin Chan, Ian Culhane, Jennifer Mankoff, and Shyamnath Gollakota. 2018. Wireless Analytics for 3D Printed Objects. In *Proc. UIST 2018*. 141–152.

Wenzel Jakob. 2010. Mitsuba renderer. http://mitsuba-renderer.org.

Hsin-Liu Cindy Kao, Paul Johns, Asta Roseway, and Mary Czerwinski. 2016. Tattio: Fabrication of Aesthetic and Functional Temporary Tattoos. In *Proc. CHI*. 3699–3702.

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 (2018), 215–223.

Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4, 3 (1982), 382–401.

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 (2016).

Dingzeyu Li, Avinash S. Nair, Shree K. Nayar, and Changxi Zheng. 2017. AirCode: Unobtrusive Physical Tags for Digital Fabrication. In *Proc. UIST*.

Lingnan Liu, Mark Y Shimizu, and Lisa M Vartanian. 1998. Method and apparatus for reading machine-readable symbols having surface or optical distortions. US Patent 5,854,478.

Marco Livesu, Stefano Ellero, Jonàs Martínez, Sylvain Lefebvre, and Marco Attene. 2017. From 3D models to 3D prints: an overview of the processing pipeline. *Comput. Graph. Forum* 36, 2 (2017), 537–564.

Sara McMains and Carlo H. Séquin. 1999. A coherent sweep plane slicer for layered manufacturing. In *Fifth ACM Symposium on Solid Modeling and Applications, Ann Arbor, Michigan, USA, June 9-11, 1999*. 285–295.

Hitesh Hirjibhai Nadiyapara and Sarang Pande. 2017. A review of variable slicing in fused deposition modeling. *Journal of The Institution of Engineers (India): Series C* 98, 3 (2017), 387–393.

Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).

Nasser M Nasrabadi. 2007. Pattern recognition and machine learning. *Journal of electronic imaging* 16, 4 (2007), 049901.

Sylvain Paris, Pierre Kornprobst, Jack Tumblin, Frédo Durand, et al. 2009. Bilateral filtering: Theory and applications. *Foundations and Trends® in Computer Graphics and Vision* 4, 1 (2009), 1–73.

Parinya Punpongsanon, Xin Wen, David S. Kim, and Stefanie Mueller. 2018. ColorMod: Recoloring 3D Printed Objects using Photochromic Inks. In *Proc. CHI 2018*.

Ben Redwood, Filemon Schffer, and Brian Garret. 2017. The 3D Printing Handbook: Technologies, design and applications. (2017).

Irving S Reed and Gustave Solomon. 1960. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics* 8, 2 (1960), 300–304.

John Rompel. 1990. One-way functions are necessary and sufficient for secure signatures. In *Proc. ACM Symposium on Theory of Computing*. ACM, 387–394.

Valkyrie Savage, Andrew Head, Björn Hartmann, Dan B. Goldman, Gautham J. Mysore, and Wilmot Li. 2015. Lamello: Passive Acoustic Sensing for Tangible Input Components. In *CHI 2015*.

Ashutosh Saxena, Sung H Chung, and Andrew Y Ng. 2006. Learning depth from single monocular images. In *Advances in neural information processing systems*. 1161–1168.

Binil Starly, Alan Lau, Wei Sun, Wing Lau, and Tom Bradbury. 2005. Direct slicing of STEP based NURBS models for layered manufacturing. *Computer-Aided Design* 37, 4 (2005), 387–397.

Gabriel Taubin, Daniel Moreno, and Douglas Lanman. 2014. 3d scanning for personal 3d printing: build your own desktop 3d scanner. In *ACM SIGGRAPH 2014 Studio*. ACM, 27.

Alexander Teibrich, Stefanie Mueller, François Guimbretière, Robert Kovacs, Stefan Neubert, and Patrick Baudisch. 2015. Patching physical objects. In *Proc. UIST 2015*. ACM, 83–91.

Carlos Tejada, Osamu Fujimoto, Zhiyuan Li, and Daniel Ashbrook. 2018. Blowhole: Blowing-Activated Tags for Interactive 3D-Printed Models. In *Proc. Graphics Interface 2018*. 131 – 137.

Weiming Wang, Haiyuan Chao, Jing Tong, Zhouwang Yang, Xin Tong, Hang Li, Xiuping Liu, and Ligang Liu. 2015. Saliency-Preserving Slicing Optimization for Effective 3D Printing. *Comput. Graph. Forum* 34, 6 (2015), 148–160.

Karin Weigelt, Mike Hambsch, Gabor Karacs, Tino Zillger, and Arved C. Hübler. 2010. Labeling the World: Tagging Mass Products with Printing Processes. *IEEE Pervasive Computing* 9, 2 (2010), 59–63.

Karl D. D. Willis, Eric Brockmeyer, Scott E. Hudson, and Ivan Poupyrev. 2012. Printed optics: 3D printing of embedded optical elements for interactive devices. In *Proc. UIST 2012*.

Karl D. D. Willis and Andrew D. Wilson. 2013. InfraStructs: fabricating information inside physical objects for imaging in the terahertz region. *ACM Trans. Graph.* (2013).

Norman J Woodland and Silver Bernard. 1952. Classifying apparatus and method. US Patent 2,612,994.

Sang Ho Yoon, Yunbo Zhang, Ke Huo, and Karthik Ramani. 2016. TRing: Instant and Customizable Interactions with Objects Using an Embedded Magnet and a Finger-Worn Device. In *Proc. UIST 16*.

Li Zhang, Brian Curless, and Steven M Seitz. 2002. Rapid shape acquisition using color structured light and multi-pass dynamic programming. In *Proc. 3D Data Processing Visualization and Transmission*. IEEE, 24–36.

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

Lee Zucheul, Daehwan Kim, and Yeong-il Seo. 2016. Variable slicing for 3d modeling. US Patent App. 14/964,916.

LayerCode: Optical Barcodes for 3D Printed Shapes
Henrique Teles Maia, Columbia University
Dingzeyu Li, Adobe Research
Yuan Yang and Changxi Zheng, Columbia University

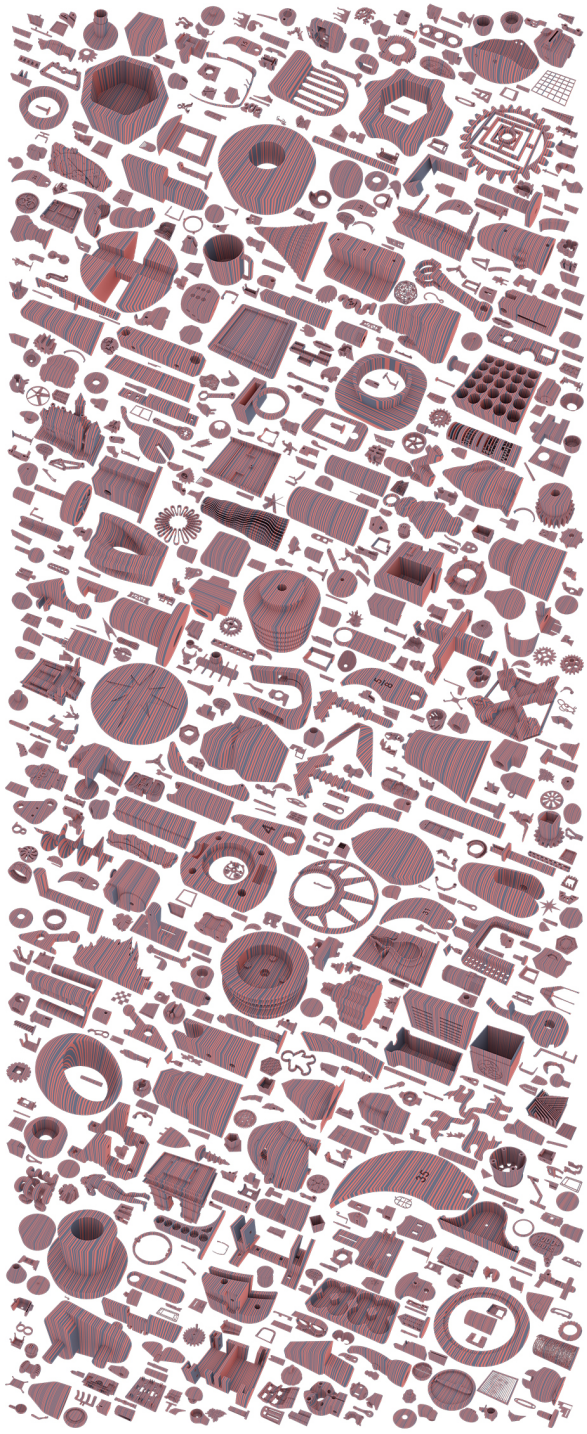

Fig. 24. **A visualization of tagged meshes tested in our database.**

## A IMAGE PREPROCESSING FOR DECODING

At decoding time, we image a LayerCode carrier object using a conventional camera. The image is preprocessed (as outlined in §4) before feeding into the decoding algorithm. Here, we describe the image preprocessing details.

For an object made by a two-color printer, we use intensity thresholding to clean highlights and shadows. The remaining pixels are those on the black and white coding layers, and clearly visible from the camera. To label what type of coding layers each pixel is in, we perform a two-way clustering through a Gaussian mixture model in a sliding window; each window produces the label of its central pixel. We found that a better labeling quality can be achieved by repeating this clustering step three times, each with different sliding window sizes ($50 \times 50$px,$100 \times 100$px, and $200 \times 200$px), followed by a pixel-wise majority vote of the labels. It is also helpful to convert images to the LAB/HSV color space to focus solely on the color channels of AB when clustering, removing some of the uncertainty added by lighting.

Labeling images of objects with varying layer heights but a uniform color (Figure 14-e) requires an altogether different approach. Once the background is removed, the image is effectively grayscale, since the object is 3D printed with a single-color material. In this case, a series of morphological operations are applied on the image, including bottom and top hat filtering, in order to first increase the contrast of the image. Once contrast is improved, we use a bilateral filter [Paris et al. 2009] to blur the black and white layer regions without blurring their boundaries. We rescale, threshold, and clean the image, labeling of regions ready for decoding. Figure 14 illustrates these steps.

Lastly, labeling images of objects with NIR materials is to a large extent similar to processing bi-colored object images. NIR images come in grayscale, and our first step is to increase its contrast, followed by removal of highlights and shadows. Afterward, we classify the remaining pixels into two types corresponding to the black and white coding layers, and this step is similar to bi-colored object images.

## B DETAILS ON DEPTH RECOVERY

Each selected image pixel from layer boundaries forms a ray $\boldsymbol{u} = [X_c, Y_c, 1]^\intercal$ in the camera reference frame. The origin of the camera reference frame is the focal point of the camera, and each pixel in the camera can be represented by the ray denoted above. Assuming this coordinate system, we next calibrate and solve for the 3×3 intrinsic matrix K of the camera. With this in hand, we can invert camera specific parameters (including focal length, pixel size, lens characteristics) to determine where pixels project in space.

Next we must take into account the extrinsic parameters that relate the camera to the world coordinate frame. By looking at a marker in the scene with known properties, such as a checkerboard, we can extract the rotation $R$ and translation $t$ between the world coordinate frame and the camera coordinate frame.

Thus, if the camera lives at

$$\boldsymbol{q}_w = -\mathrm{R}\boldsymbol{t}^\intercal \tag{3}$$

in the world reference frame, we can then similarly write the camera ray incident on our pixel as:

$$\boldsymbol{v}_w = \mathrm{R}\mathrm{K}^{-1}\boldsymbol{u}, \tag{4}$$

This leaves us only to solve for where along this ray $\lambda$, from the camera and through the pixel, does an intersection occur with a given plane height in the scene. We can estimate each plane height by inverting our LayerCode decoding to approximate lengths. Concretely, if the world coordinate frame and our printing layers align in direction, these planes can be described by the normal vector

$$\boldsymbol{n} = \left(0, 0, 1\right) \tag{5}$$

and the point in space

$$\boldsymbol{p}_w = \left(0, 0, \text{height}\right), \tag{6}$$

which, when combined with the ray emanating from the camera position $\boldsymbol{q}_w$, is known:

$$\lambda = \frac{\boldsymbol{n}^\intercal(\boldsymbol{p}_w - \boldsymbol{q}_w)}{\boldsymbol{n}^\intercal \boldsymbol{v}_w}, \tag{7}$$

and thus we solve for the world coordinate of the pixel at:

$$\boldsymbol{p}_p = \boldsymbol{q}_w + \lambda \boldsymbol{v}_w. \tag{8}$$

In any case, the position of the camera needs to be known in space, relative to the world coordinates (or table). A marker is needed in order to compute this extrinsic camera calibration, although not needed for reconstructing the points otherwise. If the camera's position relative to the table is known through a calibration, or computed and fixed for the images used, then the marker is not necessary for recovering the LayerCode shape.

## C VARIABLE LAYER HEIGHT PRINTING

Although some 3D printing platforms support the use of varying layer heights, it is often limited in capacity if available to the user. However, across all 3D printer models, instructions are conveyed from staging software to printer hardware through print files composed of G-code commands. In practice, we achieve alternating layer heights by directly manipulating the underlying G-code.

For a given print, the G-code files are similar in structure regardless of layer height settings, and so files directing fine and coarse versions of a print may be spliced together to achieve a print which alternates for each coding layer. Depending on the printer specifications, other settings might be adjusted accordingly: when working with Fused Deposit Modeling (FDM) printers, it may also be necessary to adjust the nozzle temperature when switching layer heights in order to ensure a successful print.

For FDM printers, G-code instructs the path of the nozzle head throughout printing, specifying different paths and height adjustments for every slice per the layer-height specific settings. This makes it possible to interweave G-code files at the termination of each layer, when the printer nozzle lifts to the next layer. Splicing together the two G-code files at this intermission will cause them to continue each other's print, allowing for seamless alternating print properties, so long as the files are combined at the appropriate layers heights. Due to their thickness differences, each G-code file will require a different number of layers to achieve a certain print height, and thus ensuring these heights match is crucial for print continuity. In practice, this is guaranteed by choosing the thick layer height as an integer multiple of the thin layer height.

This ensures the printer deposits at the appropriate height from the nozzle onto the piece. If care is not taken to align layer-heights directly, or too much space is given when layers get deposited, then visible gaps and artifacts at the swaps partitions may appear. Alternatively, if not enough space is given, the head will sink into printed material, which can roughen the look of the printed material and encourages jamming of the print head.

## D INVISIBLE NEAR-INFRARED PRINTING

### D.1 Dye Mixing

Exact measurements between NIR-dye and resin depend on the mixing properties and spectrographic fingerprint of the NIR-dye. In our case, we found 25-50 milligrams of dye dissolved and mixed evenly (after stirred for one day using a magnetic stirrer) per 100ml of PR-57 CMYK+W resin, and this formula generates a strong enough disparity in the printed coding layers. In experimenting with different levels of NIR signatures, we explored the use of three types of dyes that peak their absorption coefficients at 828nm, 912nm, and 1031nm, respectively. Although NIR-dyes peak in the Near-Infrared electromagnetic range, they may still express some weak signal in the visible light range, and thus discolor the resin they mix with. Therefore, once the dye is mixed with the resin, achieving an *invisible* LayerCode requires only the color of the resin without dye to match that of the NIR-dye resin mixture.

### D.2 Firmware & Hardware Modifications

In order to automate the printing and swapping process, a multi-material resin based printer is required. This presents a significant (and often prohibitive) financial and technical barrier to experimentation and exploration. Here, we show how to upgrade a low-cost and simple-to-use SLA printer to exhibit multi-resin functionality. Primarily, this involves replacing the rotating tray platform of the Autodesk Ember printer, along with updating its firmware to integrate novel swap commands, as shown in 15.

Careful disassembly of the printer allows the removal of the existing tray platform. A substitute extended platform should then be milled from aluminum, or other similarly heavy composite, so as to avoid altering the load on the motors. Notably, replacement platforms that were printed in ABS plastic were found to deflect under the weight of the resin trays, leading to numerous difficulties and failed prints. If the two resin tray windows, which are supported by the resin tray platform, are not level with one another, then between tray swaps the print bed may differ in height, and miss layers which attempt to print. This is similar to misaligned G-code heights in variable-layer-height printing, but far more difficult to recover, since in this case gravity fights against the print progression.

Finally, a firmware update is needed to introduce a swap command, which lifts the build platform to a safe height prior to rotating to another tray. This command concludes the swap by lowering down the build platform to its previous height on the complementary resin tray, where the print may continue to progress as if no swap had occurred. Once printer modifications are completed, two resin trays may be retrofit within the printer chamber on the new

rotation plate, each accessible via swap commands enabled in the updated firmware. Then, during the printing, one may trigger a desired swap at the desired slices via a secure shell connection or by depositing a formatted CSV file along with the model images for printing.

Please refer to our video for a quick demonstration of this hardware modification process. Given the precision required to modify the Autodesk Ember for multi-resin use, we plan to release all the CAD models, designs, instructions, and code involved in amending the printer.

### D.3 Observing NIR LayerCode Tags

Once printed, the final piece appears smooth and uniformly colored to the naked eye in visible light. Since most light bulbs are tuned to illuminate the visible light range, turning lights on or off has little effect on the visibility of NIR LayerCode. However, this does not guarantee the NIR layering will become apparent when captured through a NIR filter, since there must be some appropriate NIR light illumination to introduce the contrast. In order to best expose the LayerCode while indoors, the use of LED NIR lights is recommended to illuminate the scene and printed object. For optimal conditions, choose LEDs that are similar in wavelength to the peak of the NIR. 850nm LED for 828nm Dye, 950nm LED for 912nm Dye, etc. Under a NIR camera, the layers printed with NIR-Dye will appear darker, since they are designed to absorb NIR radiation at given frequencies. In outdoors, natural sunlight can produce enough NIR excitation to reveal the LayerCode pattern, without resort to NIR lights.