Toward a Universal Model for Shape from Texture

Dor Verbin and Todd Zickler
Harvard University
{dorverbin,zickler}@seas.harvard.edu

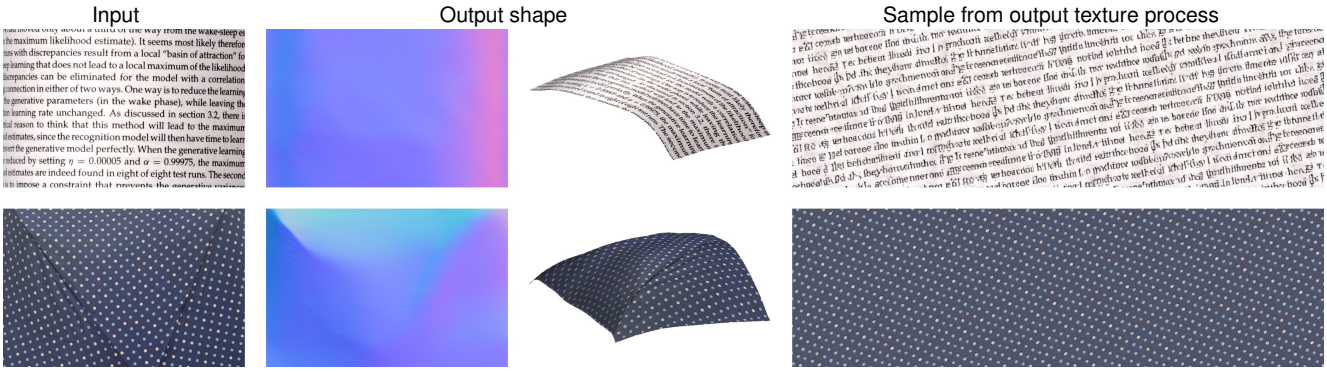

Figure 1: Shape and texture from a single image. The output is a 2.5D shape and a flat-texture generative process. The approach succeeds for a wide variety of textures, including those for which previous methods break down.

Abstract

We consider the shape from texture problem, where the input is a single image of a curved, textured surface, and the texture and shape are both a priori *unknown. We formulate this task as a three-player game between a shape process, a texture process, and a discriminator. The discriminator adapts a set of non-linear filters to try to distinguish image patches created by the texture process from those created by the shape process, while the shape and texture processes try to create image patches that are indistinguishable from those of the other. An equilibrium of this game yields two things: an estimate of the 2.5D surface from the shape process, and a stochastic texture synthesis model from the texture process. Experiments show that this approach is robust to common non-idealities such as shading, gloss, and clutter. We also find that it succeeds for a wide variety of texture types, including both periodic textures and those composed of isolated textons, which have previously required distinct and specialized processing.*

1. Introduction

Texture is the repetition of appearance across local spatial neighborhoods of a surface. When a surface is curved, foreshortening causes spatial compression of local appearance across an image, and this gives a perception of surface shape. Computer vision systems that can exploit this phenomenon will be able to factor images into their underlying shape and texture components, which could serve as useful intermediate representations for shape understanding, material recognition, and other tasks.

Previous approaches to shape from texture use a variety of models for encoding the relevant aspects of local appearance, and they each target a particular class of textures. Some approaches focus on textures that are composed of isolated texture elements. These approaches operate by detecting the elements and inferring the relative distortion between them, or the distortion of each one separately in cases where prior information about the texture is known. Other approaches apply to textures that are stationary stochastic processes, and they estimate shape locally, by measuring how the local frequency spectrum changes between nearby image patches.

This paper explores the possibility of a single shape from texture model that can recover shape from all texture types, including those that are neither perfectly stationary nor comprising perfectly discernible textons. Our model is a three-player game between a discriminator, an unwarper, and a generator. The generator is a texture process that outputs synthetic flat-texture patches computed from samples of a pre-defined stochastic process, whereas the unwarper

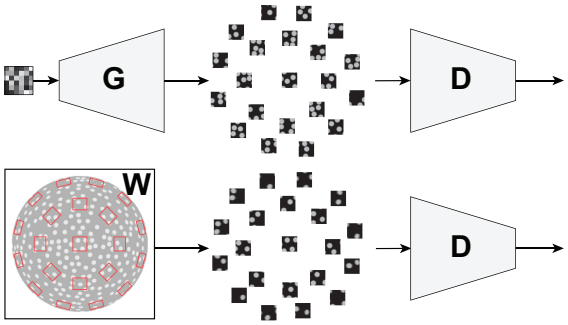

Figure 2: Shape from texture as a three-player game. A discriminator D works to distinguish synthetic texture patches created by the generator G from ones extracted from the image and unwarped by the unwarper W. The generator and unwarper try to produce patch distributions that are indistinguishable, while the discriminator tries to tell them apart.

is a shape process that outputs patches from the input image that are unwarped by a field of 2D affine spatial transformations. The generator and unwarper both feed their output patches to the discriminator, which assigns each patch a scalar value. During the game, the generator and unwarper adapt to produce distributions of patches that are indistinguishable from those of the other, while the discriminator adapts to tell them apart. At an equilibrium, the generator provides a learned synthesis model for the latent flat texture (right column of Figure 1), while the unwarper provides the 2.5D surface shape.

The game can succeed because the generator and unwarper are designed to have intrinsic limitations. The generator can only create patches from a (learnable) cyclostationary stochastic process, while the unwarper can only apply smooth affine warp fields. Intuitively, this means that whenever the two processes are able to discover patch distributions that are similar to one another, these distributions are likely to correspond to the true latent shape. This intuition is supported theoretically by a companion paper [20], which shows that an image of a surface that is texture-mapped with a perfectly cyclostationary texture cannot be explained by any other shape and cyclostationary texture. We also support it experimentally by evaluating its accuracy on a variety of synthetic images and captured photographs.

2. Related Work

Our approach is inspired by two bodies of work that, until now, have developed rather separately.

Parametric Texture Synthesis. We build upon decades of work on statistical models for (flat) visual textures, including foundational work by Julesz [13], Heeger and Bergen [9], and many others (*e.g.*, [19, 22, 3]). In this body of work, 3D shape is avoided, and the task is to distinguish one flat texture from others by finding the parameters of a probability distribution $p(I)$, over texture patches I, that concentrates on patches that are perceptually similar to a training set of examples. This work is often associated with texture synthesis because that is the most convenient way to evaluate it: once the statistical parameters are learned from a training set, new samples can be drawn from $p(I)$ for humans to evaluate as "looking the same". Of these methods, our approach is closest to the minimax entropy approach of Zhu *et al*. [22] and to spatial GANs [12, 3], which use minimax (adversarial) objectives to learn a model for the distribution $p(I)$.

The key difference between all of this work and ours is that we aim to learn $p(I)$ from a set of training samples that, instead of being flat, are each affected by an unknown spatial affine transformation. In addition to finding a distribution that concentrates on our input set of samples, we also find affine transformations of these samples (subject to surface continuity constraints) that allow the learned distribution to concentrate even further. This capability is useful: It means that one can learn the appearance of textured objects without having to flatten them first.

Shape from Texture. Among the rich collection of work in shape from texture—see [18, 16] for deeper reviews—we are most influenced by two classes of techniques. The first was developed by Gårding [6], Malik and Rosenholtz [18], and Clerc and Mallat [4]. The central idea is to encode the appearance of an image patch by its spatial frequency spectrum, as determined by the magnitude responses to filters that are localized in space and frequency, and then to reason about local surface orientation and curvature by analyzing how that spectrum differs from the spectra of nearby patches. One great advantage of this approach is that it provides local normal and curvature estimates without having to rely on an orthographic camera model or on surfaces being continuous over extended areas. A significant disadvantage is that it only applies to a restricted class of textures that are periodic and that do not rotate across the surface.

The second class of techniques was created by Forsyth [5], Lobay and Forsyth [16], and Loh and Hartley [17]. It applies to textures that comprise a limited number of repeated texture elements (textons) that can be independently localized in the image. The basic strategy is to first detect the elements and then to find a set of affine transformations that align each pair of them. With appropriate spatial regularization, the pairwise affine transformations are sufficient to determine shape. The advantages of this strategy are that it can succeed where the first class of methods fails, and that it applies even when the textons rotate or are distributed irregularly across the surface [17]. The disadvantage, of course, is that it is designed for a specific class of textures, excluding those for which repeating isolated elements are not well defined.

Our model borrows from both classes of techniques. Inspired by the first class, we include stochastic periodic inputs to our texture generator, which allows the texture process to exploit non-rotating periodicity when it exists while ignoring it when it does not. Inspired by the second class of methods, we represent shape using a field of affine transformations that are linked by spatial regularization.

3. Model Overview

We assume the input RGB image $\mathcal{I}\colon \Omega \to \mathbb{R}^3$ is captured by an orthographic camera, and we model the latent 2.5D surface shape as the graph of a continuous function $z\colon \Omega \to \mathbb{R}$, where $\Omega = \{0, ..., w-1\} \times \{0, ..., h-1\}$ is the image domain. We assume that the latent flat texture $\mathcal{T}$ was generated by a cyclostationary process, meaning that its statistics are spatially periodic. We denote by $p(I)$ the distribution of flat texture patches I cropped from $\mathcal{T}$. Our goal is to infer both the surface at each pixel $z(x, y)$ and the flat texture distribution $p(I)$ from the input image $\mathcal{I}$, without observing the flat texture beforehand.

When a surface is planar and slanted relative to the orthographic viewpoint, a square patch on the surface projects to a non-square smaller image patch. This map is a restricted type of 2D affine transformation that has three degrees of freedom and is invertible [16]. We use the term *warp* to refer to this type of transformation, and *unwarp* for its inverse. A patch from the input image around the pixel location (x, y) (*e.g.*, a red quadrilateral in Figure 2) corresponds to a flat patch from $p(I)$ that has been warped by some (unknown) 2×2 warp matrix $W(x, y)$. Estimating these latent warps $W(x, y)$ at every pixel is sufficient for estimating the latent surface $z(x, y)$ (up to an inconsequential constant offset and 2-fold ambiguity which we discuss in Section 5).

For now, let us assume there is a single pre-determined square patch size, denoted $Q \times Q$, that is appropriate for the input image. That is, for most image locations (x, y), the area spanned by the warp $W(x, y)$ applied to a $Q \times Q$ square is small enough for the latent surface to be nearly planar within it (*i.e.*, with negligible curvature) and yet large enough to contain useful statistics of the texture's appearance required for shape estimation. An extension to multiple patch sizes and its importance are discussed in Section 5.

We denote by W the dense field of warps comprising a warp $W(x, y)$ for each pixel (x, y) in the input image. We denote by $q(I; W)$ the distribution of square $Q \times Q$ patches obtained by applying each unwarp $W^{-1}(x, y)$ to the non-square image patch centered at (x, y) (bottom of Figure 2). We say that a warp field is "good" if its inverse implies a distribution $q(I; W)$ that is close to the flat texture distribution $p(I)$. Since neither the true warp field nor the true flat texture distribution are known *a priori*, we jointly estimate them by creating a three-player game between a generator, a discriminator, and an unwarper. Broadly speaking, the generator and discriminator behave as a generative adversarial network [8] to learn a model of $p(I)$, while the unwarper learns a warp field W whose inverse implies a distribution $q(I; W)$ that is similar to $p(I)$.

In this game, the parameters of the generator G, discriminator D, and unwarper W are alternately updated by:

$$G^{(t)}, D^{(t)} = \arg\min_G \max_D \mathbb{E}_{I \sim q(I; W^{(t-1)})}[\log D(I)] + \mathbb{E}_{Z \sim p(Z)}[\log(1 - D(G(Z)))], \quad (1)$$

$$W^{(t)} = \arg\max_W \mathbb{E}_{I \sim q(I; W)}[\log D^{(t)}(I)] - C(W), \quad (2)$$

where $p(Z)$ is a predetermined stochastic process, defined in Section 4, whose samples are input to the generator; and $C(W)$ is function, defined in Section 5, that encourages smoothness in the estimated warp field.

The first term in the objective in Equation 1 quantifies the ability of the discriminator to detect patches that come from the unwarper (bottom of Figure 2), while the second term quantifies its ability to detect patches $G(Z)$ generated by the generator (top of Figure 2). We alternate between updating G and D (Equation 1), and updating W (Equation 2). Updating G and D aims to parametrize $q(I; W)$, whereas updating W is meant to find the warps which are given a high score by D, while also yielding a smooth surface as defined by $C(W)$.

The effectiveness of this game relies on designing the players to have some intrinsic limitations. Without limitations, the generator and discriminator could learn a model for the distribution of input image patches themselves, without any unwarping, so the output could always be the trivial "postcard" explanation consisting of a flat shape (planar $z(x, y)$). Similarly, if the unwarper were able to apply unlimited transformations to the image patches, the game could converge to a wide variety of distributions $q(I; W)$ corresponding to a wide variety of unrealistic shapes.

We avoid these scenarios by designing the generator to only produce patches from a (learnable) cyclostationary flat-texture process (Section 4), and by designing the unwarper to only produce warp fields that correspond to smooth continuous surfaces (Section 5). As shown in [20], given an input image of a cyclostationary-textured surface, the only unwarper W for which the distribution $q(I; W)$ represents patches taken from a cyclostationary process, is the unwarper that corresponds to the true shape (up to rotations in the tangent plane, which do not affect shape). This means that when the unwarper discovers a distribution $q(I; W)$ that is similar to the one modeled by the generator and discriminator, this is likely to correspond to the true flat texture distribution $p(I)$ and true shape.

4. Texture Model

Our texture model consists of a generator and discriminator that are trained adversarially. The architecture is inspired by the Periodic Spatial GAN [3], which is similarly designed to generate cyclostationary textures. Our generator learns a deterministic map from samples of a predefined stochastic process $p(Z)$ to patches of a cyclostationary texture process. During the 3-player game the generator creates patches that are at least $Q \times Q$, so its output can be compared by the discriminator to $Q \times Q$ patches sampled from $q(I; W)$. After the game converges, the convolutional generator can be used to create arbitrarily large texture samples (see examples in Figures 1, 4, and 5).

4.1. Generator

Our generator is composed of a sequence of four stride-2 deconvolutions (often referred to as stride-$\frac{1}{2}$ convolutions) with a kernel size of 5×5, each followed by a ReLU nonlinearity and a batch normalization layer [11]. The architecture is shown in Figure 3.

Our generator takes in a collection of input maps, shown in the figure. In order to avoid windowing effects in the generated texture, we use the fact that these input maps can be made arbitrarily large. For the first stride-2 deconvolution in our network, instead of generating an input map of spatial size $M \times M$, we generate a larger input of shape $(M + 3) \times (M + 3)$. Applying a stride-2 deconvolution then results in an output of shape $(2M + 9) \times (2M + 9)$, which can then be cropped into $(2M + 3) \times (2M + 3)$. By cropping 3 pixels from each side of both dimensions, we eliminate any windowing effect. Repeating this process for all four deconvolution layers results in an output of size $(16M+3)\times(16M+3)$, which can finally be cropped to get an output of shape $16M \times 16M$ with no windowing effects. This padding scheme reduces constraints on the deconvolution kernels, and we found it to significantly stabilize our results.

There are three types of stochastic inputs to the generator: local maps, periodic maps, and global maps. Local maps are 2D spatial arrays whose elements are drawn independently from $[-1, 1]$. We concatenate two such maps to the input of each of the last two deconvolutions in the network. The size of each local map needs to match with the input it is concatenated to, and is therefore defined by the architecture of the network (see Figure 3)

The local maps serve as a means of encoding variability in the texture. Local maps injected in the first layers of the network are processed by a relatively large number of nonlinear filters, and impact a large part of the output texture patch due to the structure of the generator. In contrast, local maps added towards the end of the generator undergo a smaller number of transformations, and therefore can only model small-scale variations in the output.

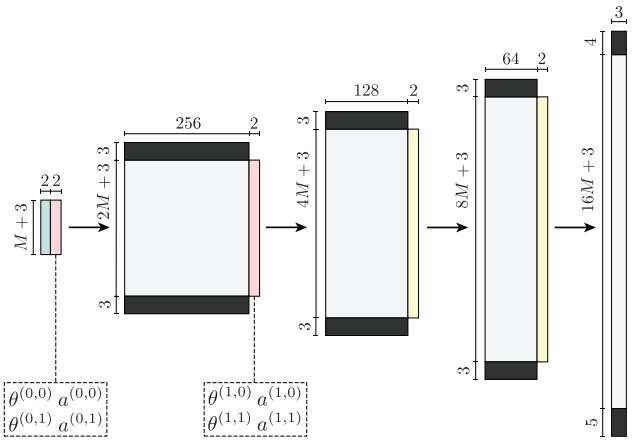

Figure 3: Side view of generator, showing only one spatial dimension (spatial operations are square). Arrows denote 5×5 stride-2 deconvolution followed by a ReLU and a batch normalization layer. Each arrow's output is cropped (darkly shaded) to avoid windowing effects. Adjacent blocks denote concatenation, and colored blocks are input samples from predefined stochastic processes: green are spatially-constant random samples ("global maps"); yellow are spatially i.i.d. random samples ("local maps"); and pink are randomly-shifted sinusoids with learnable frequencies ("periodic maps"). See text for details.

When dealing with real images, adding local maps later in the network allows for better handling of noise and changes in the appearance of the input texture. Adding local maps earlier in the network may also have this effect, but we found that it results in less accurate shape estimation.

Periodic maps are spatial arrays with values determined by sinusoidal functions whose parameters are learned. We concatenate two periodic maps to the input of the first and second deconvolution layers. For the jth periodic map input to the ith deconvolution layer we learn two parameters: the amplitude $a^{(i,j)}$ and orientation $\theta^{(i,j)}$ of the wave vector of the map. We can then write the periodic maps as:

$$\begin{aligned} Z_p^{(i,j)}(\lambda, \mu) &= \sin\left(2\pi k_1^{(i,j)}\lambda + 2\pi k_2^{(i,j)}\mu + \phi^{(i,j)}\right), \\ k_1^{(i,j)} &= a^{(i,j)}\cos(\theta^{(i,j)}), \\ k_2^{(i,j)} &= a^{(i,j)}\sin(\theta^{(i,j)}), \end{aligned} \tag{3}$$

where $\lambda, \mu \in \{0, ..., 2^i M + 2\}$ are the spatial coordinates in the ith scale. The magnitude of each wave vector $a^{(i,j)}$ is learned but limited by a sigmoid function to $[0, 0.5]$ to prevent aliasing in the input sinusoids. The phase $\phi^{(i,j)}$ is drawn uniformly from $[0, 2\pi]$.

The periodic maps are used to capture the cyclostationary statistics of the input texture. We found that adding periodic inputs across multiple scales in the network helps the generator encode these statistics. The later a periodic input

is injected into the network, the higher its frequency can be in the output. By using periodic maps across multiple scales we allow the network to learn a multiscale representation of the periodic statistics of the texture.

In addition to the local and periodic maps, we also add two global maps to the input of the first deconvolution layer. Each global map is constant spatially, and its value is drawn uniformly from $[-1, 1]$. The global maps are used to encode larger variations between patches in an image. For example, we find that using global maps helps deal with non-idealities in the input images, such as shading, by encoding those variations and obtaining a texture model that is more robust to illumination effects.

4.2. Discriminator

The architecture of our discriminator is designed symmetrically to the generator, as suggested in [3]. It is composed of a sequence of four stride-2 convolutions, each followed by a leaky-ReLU activation with slope 0.2. Similar to the cropping in our generator, we avoid windowing effects in the discriminator by not padding the input to any of the convolution layers (*i.e.*, we use "valid" convolution).

The discriminator architecture defines a receptive field of size $R \times R$ in the input. An input patch I is transformed by the discriminator to a probability map $D(I) \in [0, 1]^{S \times S}$. The output of the discriminator at location (λ, μ), $D_{\lambda\mu}(I)$, only depends on a receptive field of size $R \times R$ in the input. In order to estimate the log-probability that the discriminator assigns to an entire patch I (for the objectives in Equations 1 and 2), we define:

$$\log D(I) = \frac{1}{S^2} \sum_{\lambda=1}^{S} \sum_{\mu=1}^{S} \log D_{\lambda\mu}(I). \tag{4}$$

As suggested in [12, 8], when optimizing the discriminator according to Equation 1 we replace $\log(1 - D(G(Z)))$ by $\frac{1}{S^2} \sum_{\lambda=1}^{S} \sum_{\mu=1}^{S} \log(1 - D_{\lambda\mu}(G(Z)))$, and when training the generator instead of minimizing this term we minimize $-\log D(G(Z))$ using the definition in Equation 4.

Since the discriminator is convolutional, we can use it to evaluate how real a patch looks for any input size, as long as it is no smaller than $R \times R$. This enables us to use multiple patch sizes $Q_i \times Q_i$ from the unwarper, as well as using a different sized generator output. We discuss the importance of being able to evaluate the discriminator on different patch sizes in Section 5.

5. Shape Process

The field of affine warps learned by the unwarper should correspond to a continuous surface. This places considerable constraints on them, which can be enforced in many ways. Our strategy is to explicitly interpret each affine transformation as relating to a 3D surface normal vector and a 3D tangent vector, and to add two penalization terms to the objective: one for the degree to which the normal field is not integrable, and one for the degree to which the normal and tangent fields are not smooth.

As mentioned in Section 4.2, we can use the discriminator to evaluate multiple patch sizes. We want the patch size $Q \times Q$ output by the unwarper to be small enough to be approximately planar, yet large enough to contain sufficient information to facilitate shape estimation. Since the characteristic texture sizes are not known *a priori*, we simply use multiple patch sizes $Q_i \times Q_i$ for a few different values of Q_i. We find that our system automatically makes use of the appropriate values of Q_i: Values that are too small are ignored by the system since they do not contain enough information to contribute to shape estimation, while values that are too large to be approximately planar can contribute to shape estimation without preventing the smaller patches from capturing finer details in the shape.

The remainder of this section provides the details of our warp parametrization. Our approach is heavily inspired by [16] and [17]. In order to avoid local minima in shape we find it useful to incorporate multiscale processing similar to [2].

We parametrize the field of warps W by defining a set of four auxiliary variables at each pixel that represent the 3D normal and tangent vectors at that pixel. We relate the normal and tangent vectors to the warp matrix at each point, and use our auxiliary variables to define the smoothness cost $C(W)$ in Equation 2.

We parametrize the surface normals using two scalars $p(x, y)$ and $q(x, y)$ at each pixel location (x, y). At each pixel the normal is:

$$\hat{n} = \frac{-p\hat{x} - q\hat{y} + \hat{z}}{\sqrt{p^2 + q^2 + 1}}, \tag{5}$$

where $\hat{x}$, $\hat{y}$ and $\hat{z}$ are the standard basis of $\mathbb{R}^3$.

In addition to the normal vectors, we wish to parametrize the tangent vector at each pixel in the image. The two vectors $n_z\hat{x} - n_x\hat{z}$ and $n_z\hat{y} - n_y\hat{z}$ are orthogonal to $\hat{n}$ (whose $\hat{x}$, $\hat{y}$ and $\hat{z}$ components we denote by n_x, n_y and n_z respectively), and they span the tangent plane. Therefore we can define two more scalars $s(x, y)$ and $c(x, y)$ for each pixel, and set the tangent vector to be a (normalized) linear combination:

$$\hat{t} = \frac{cn_z\hat{x} + sn_z\hat{y} - (cn_x + sn_y)\hat{z}}{\sqrt{c^2n_z^2 + s^2n_z^2 + (cn_x + sn_y)^2}}. \tag{6}$$

We found that this over-parametrization of the tangent's one degree of freedom works well in practice.

We assume an orthographic camera with viewing direction $-\hat{z}$. Consider the local 3D surface frame as the standard coordinate system $\hat{x}$, $\hat{y}$ and $\hat{z}$ being rotated by the 3×3

matrix that maps $\hat{x} \mapsto \hat{t}$, $\hat{z} \mapsto \hat{n}$, and $\hat{y} \mapsto \hat{n} \times \hat{t} \triangleq \hat{b}$. The columns of this rotation matrix are, from left to right, $\hat{t}$, $\hat{b}$ and $\hat{n}$. The camera projection is in the $\hat{z}$ direction and so the local 2D warp $W(x, y)$ is simply the top-left 2×2 block of the rotation matrix:

$$W(x,y) = \begin{bmatrix} t_x(x,y) & b_x(x,y) \\ t_y(x,y) & b_y(x,y) \end{bmatrix}. \quad (7)$$

The matrix $W(x, y)$ determines the local foreshortening in the vicinity of the location (x, y). Note that the matrix indeed corresponds to foreshortening as $\det(W(x, y)) = n_z(x, y) \leq 1$. We can also assume that $n_z(x, y) \geq 0$, which corresponds to the texture facing the camera.

Given a texture mapped onto a surface we use our three-player game to estimate the normal vectors $\hat{n}$ and tangent vectors $\hat{t}$, which define the field of warps of the unwarper. However, given a specific warp $W(x, y)$ there is a 2-fold ambiguity in shape, since W is invariant under $(n_x, n_y, t_z) \mapsto (-n_x, -n_y, -t_z)$ (see for example [14]). Therefore the two surface normals $\hat{n} = \pm n_x \hat{x} \pm n_y \hat{y} + n_z \hat{z}$ define the same exact warp.

Since this 2-fold ambiguity occurs independently for each pixel in the input image, the normal and tangent vector fields have many solutions corresponding to the same field of warps. By adding a smoothness penalty term $C(W)$, we encourage smooth and integrable solutions (Section 5.1), but then our problem has multiple local optima. In order to avoid these local optima, we use a multiscale representation of the surface normals (Section 5.2).

5.1. Shape Constraints

We define the shape cost $C(W)$ using an integrability term and a smoothness term:

$$C(W) = C^{(I)}(W) + C^{(S)}(W). \quad (8)$$

The integrability term is that introduced by Horn and Brooks for shape from shading problem [10] (a similar approach was used for shape from texture in [17]). The integral of $\nabla z = (p, q)$ over a closed loop should be small. Choosing the integration path C to be a loop around a single pixel and taking the average of the square of this discretized integral over the entire image gives:

$$C^{(I)} = \frac{\alpha^{(I)}}{hw} \sum_{i,j} [p_{i,j+1} - p_{i+1,j+1} + p_{i,j} - p_{i+1,j} + q_{i,j+1} + q_{i+1,j+1} - q_{i,j} - q_{i+1,j}]^2, \quad (9)$$

where h and w are the height and width of the input image, respectively, and $\alpha^{(I)}$ is a scalar weight parameter.

We additionally enforce spatial smoothness in both $\hat{n}$ and $\hat{t}$. We found that using the ℓ_2 norm of the spatial gradients

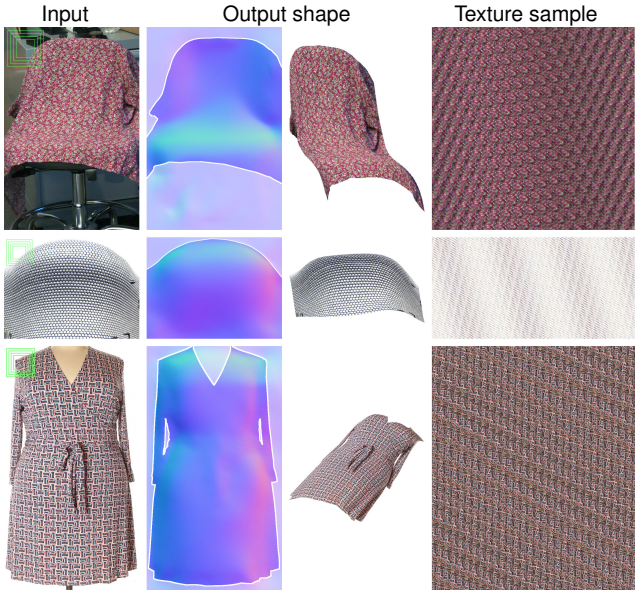

Figure 4: Shape and texture results for photographs. From left to right: input image; output surface normals; output shape; and sample from the learned texture process. Green inset squares show patch sizes used by the unwarper. For visualization purposes, regions of clutter in the input were manually cropped *after* convergence of the game.

of $\hat{n}$ and $\hat{t}$ works well:

$$C^{(S)} = \alpha_n^{(S)} \frac{1}{hw} \|\nabla \hat{n}\|_2^2 + \alpha_t^{(S)} \frac{1}{hw} \|\nabla \hat{t}\|_2^2, \quad (10)$$

where $\|\nabla \hat{n}\|_2^2$ denotes summing all squared elements of the spatial gradient of $\hat{n}$ (and similarly for $\hat{t}$). The weights $\alpha_n^{(S)}$ and $\alpha_t^{(S)}$ control the amount of smoothness we require from the surface normals and tangent vectors.

5.2. Multiscale Optimization

Instead of directly optimizing the per-pixel variables p and q, we adopt the optimization scheme suggested by Barron and Malik in [2] and optimize a set of N components $\{p^{(i)}\}_{i=0}^{N-1}$, where $p^{(i)}$ has size $h/2^i \times w/2^i$ and corresponds to the ith spatial scale. Per-pixel p can be written $p = \mathcal{G}^\top \left(p^{(0)} \circ ... \circ p^{(N-1)}\right)$ with $\mathcal{G}$ a fixed matrix that generates a Gaussian pyramid, and $\left(p^{(0)} \circ ... \circ p^{(N-1)}\right)$ a vector of the multiscale components. The same method is also used for representing q.

In order to create the Gaussian pyramid we use a 2D kernel $k^\top \cdot k$ where $k = m \cdot \frac{1}{16}[1\ 4\ 6\ 4\ 1]$. As noted in [2], choosing $m > 1$ gives coarse scales a larger weight, which helps in obtaining globally consistent normal maps. We found that $m = 1.4$ works well in our experiments.

We found that the pyramid is critical to getting the right surface normals. However, using a pyramid for the tangent

Figure 5: Shape and texture results for synthetic images. From left to right: surface normals of the true shape; input image; output surface normals (with MAE); and sample from output texture generator. Green inset squares show patch sizes used by the unwarper.

vectors did not seem to improve our results. This is probably due to the fact that we do not require any global properties from the tangent vector field, yet we do want the normal vector field to vary smoothly over the entire image, and for it to correspond to a continuous surface.

When training W according to Equation 2, the parameters being updated are the multiscale representations of the normal vectors $\{p^{(i)}\}_{i=0}^{N-1}$ and $\{q^{(i)}\}_{i=0}^{N-1}$, as well as the spatial maps c and s representing the tangent vectors.

6. Experiments

We test our system on photographs as well as synthetic images created with Blender.[1] The same architecture and parameters are used in every case. Our code and dataset are available on the project page [1]. Additional results and an ablation study testing the effect of each component of our model are available in the supplement.

For each input image, the game is initialized with nearly-frontal surface normals ($p^{(i)}$ and $q^{(i)}$ uniformly sampled in $[-5 \cdot 10^{-5}, 5 \cdot 10^{-5}]$) and nearly horizontal tangents (c and s respectively sampled uniformly in $[0.9, 1.1]$ and $[-0.1, 0.1]$). The weights of the generator and discriminator are initialized using the Xavier initialization [7].

All variables are optimized using Adam [15]. We alternate between updating the generator and discriminator weights for 20 iterations using a learning rate of $2 \cdot 10^{-4}$ and minibatch size of 25 and updating the shape parameters for 200 iterations, using a learning rate of 10^{-4} for the surface normal parameters and $5 \cdot 10^{-2}$ for the tangent vector parameters. We use generator output of size 192×192 ($M = 12$) and unwarper patch size selected randomly and independently for each iteration from $\{96, 128, 160, 192\}$. We use integrability and smoothness weights of $\alpha^{(I)} = 10^7$ and $\alpha_n^{(S)} = \alpha_t^{(S)} = 10^2$. We run the game for 25000 iterations which takes approximately 110 minutes for a 640×640 input image, using an NVIDIA Tesla V100 GPU.

We begin by testing the system on synthetic images. The absence of shading, boundaries, or any other monocular shape cues makes these images appear quite unrealistic (Figure 5) but they allow us to evaluate the texture cue in isolation. We quantify the accuracy of the recovered shape using mean angular error (MAE):

$$\text{MAE}(\hat{n}, \hat{n}^{(\text{gt})}) = \frac{1}{hw} \sum_{i=1}^{h} \sum_{j=1}^{w} |\cos^{-1}(\hat{n}_{ij} \cdot \hat{n}_{ij}^{(\text{gt})})|, \quad (11)$$

where $\hat{n}_{ij}$ and $\hat{n}_{ij}^{(\text{gt})}$ are the estimated and ground-truth surface normals at pixel (i, j). The results for three different synthetic input images are shown in Figure 5, and the accuracies for these textures and others, each averaged over four different shapes, are summarized in Table 1. The full visualizations are available in the supplement.

Overall we find that the system reliably recovers good qualitative shape, even in cases where the unnatural, texture-only rendering makes shape hard to perceive for human observers. Quantitatively, the accuracy of the recovered shape is comparable in terms of MAE to that of modern shape from shading algorithms that, like us, do not use boundaries (*e.g.* [21]). We see two main types of artifacts in the output shape. In places where the true surface orientation is close to fronto-parallel, our choice of regularizer tends to create flattened explanations; and in some cases, like the bottom of Figure 5, there are crease-like artifacts caused by the algorithm getting trapped in a local minimum and failing to properly resolve some of the 2-fold ambiguities. Note that our system manages to extract shape even when texture periodicity is very large relative to the unwarper's patch sizes, and even when the texture is entirely aperiodic, as in the middle row of Figure 5. In terms of the learned texture process (right column of Figure 5), we find that it captures statistics over scales that are comparable or smaller than the patch sizes, but not those at significantly larger scales.

For comparison, Table 1 also includes the accuracy of shapes recovered by previous shape from texture algorithms based on isolated textons [16, 17] and stationarity [4].[2] Some visual comparisons are shown in Figure 6. Stationarity [4] works well for periodic textures but breaks for general cyclostationary ones. Isolated textons [17] works well for non-overlapping texture elements but breaks when

[1] http://www.blender.org, accessed 03/28/2020.

[2] Results for [16, 17] were obtained using our own implementation, those for [4] used the original implementation published by the author.

											
[4]	30.5	41.2	45.4	35.6	32.7	37.8	36.1	38.7	29.5	29.0	24.4
[17]	-	-	12.9	35.9	21.5	23.7	-	19.8	22.6	21.6	**6.9**
[16]	27.7	29.8	38.2	40.4	41.7	23.5	33.1	17.6	32.0	47.6	9.2
ours	**15.2**	**16.8**	**12.6**	**15.0**	**18.6**	**17.4**	**19.5**	**17.1**	**14.0**	**20.1**	14.9

Table 1: Shape accuracy (MAE, in degrees) of our and other algorithms for various textures, including those of Figs. 5 & 6 and four additional ones. Each entry is average error over four images with the same texture and four different shapes (see supplement for complete visualizations). Missing entries for [17] are failures to identify a frontal textons.

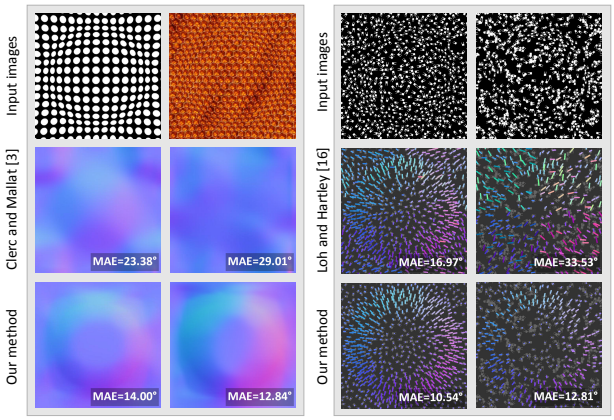

Figure 6: Comparison to two previous methods for shape from texture, in each case including an image inspired by the original paper. *Left panel*: the method based on stationarity [4] recovers qualitative shape when the assumption is valid but breaks for more general cyclostationary textures. *Right panel*: Similarly, the method based on isolated textons [17] breaks when the textons are not well separated. In contrast, ours approach works equally well across all four images. The true shape is the sphere in top of Fig. 5.

they overlap. In contrast, our method performs equally well across all of these texture types.

Figures 7 and 8 evaluate our system's performance for different texture scales and for more realistic synthetic images that contain other common visual phenomena. We find that our system is quite robust to the presence of shading, gloss, and visual clutter. We also find that it obtains good shape estimates across a variety of different texture scales, unless the texture scale is significantly larger than the unwarper's largest patch size (right of Figure 7).

Finally, Figures 1 and 4 show the output of our system for some captured photographs. Our method succeeds in estimating shape across multiple textures and shapes, despite the presence of significant shading and deviations from texture cyclostationarity. The book in the first row of Figure 1 is an especially challenging case: estimating vertical foreshortening is easy due to the white space between the lines, but the horizontal statistics are not even approximately periodic. Furthermore, since the book is mainly foreshortened horizontally, the horizontal statistics are the critical ones for shape estimation, which our model seems to handle well.

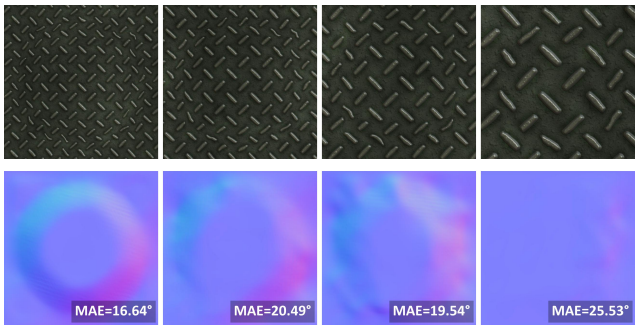

Figure 7: Shape accuracy for images with increasing texture scales. The true shape is the sphere in top of Fig. 5.

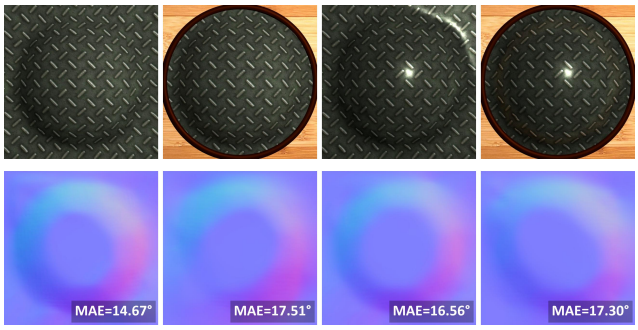

Figure 8: Shape accuracy for images with common non-idealities. The corresponding ideal, texture-only image is in the left column of Fig. 7, and here we show the same scene rendered with (left to right): shading; shading and clutter; shading and gloss; and shading, gloss, and clutter. The true shape is the sphere in top of Fig. 5.

7. Conclusion

We introduced a three-player game for processing an input image into a 2.5D shape and a flat-texture synthesis model. Compared to previous approaches to shape from texture, it has the advantage of working for a much wider variety of textures. It exhibits robustness in the presence of clutter and other visual phenomena and is able to recover qualitative shape from natural photographs that are dominated by a single texture region. Our results suggest that it is worth considering how multi-player games might be used to address other types of intrinsic image tasks, and how they might be combined with perceptual grouping for higher-level vision tasks in real-world environments.

References

[1] Project page: Toward a univeral model for shape from texture, http://vision.seas.harvard.edu/sft/. 7

[2] Jonathan T. Barron and Jitendra Malik. Shape, illumination, and reflectance from shading. *TPAMI*, 2015. 5, 6

[3] Urs Bergmann, Nikolay Jetchev, and Roland Vollgraf. Learning texture manifolds with the periodic spatial GAN. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 469–477. JMLR. org, 2017. 2, 4, 5

[4] Maureen Clerc and Stéphane Mallat. The texture gradient equation for recovering shape from texture. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(4):536–549, 2002. 2, 7, 8

[5] David A Forsyth. Shape from texture without boundaries. In *European Conference on Computer Vision*, pages 225–239. Springer, 2002. 2

[6] Jonas Gårding. Shape from texture for smooth curved surfaces in perspective projection. *Journal of Mathematical Imaging and Vision*, 2(4):327–350, 1992. 2

[7] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010. 7

[8] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 3, 5

[9] David J. Heeger and James R. Bergen. Pyramid-based texture analysis/synthesis. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, 1995. 2

[10] Berthold KP Horn and Michael J Brooks. The variational approach to shape from shading. *Computer Vision, Graphics, and Image Processing*, 33(2):174–208, 1986. 6

[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 4

[12] Nikolay Jetchev, Urs Bergmann, and Roland Vollgraf. Texture synthesis with spatial generative adversarial networks. *arXiv preprint arXiv:1611.08207*, 2016. 2, 5

[13] Bela Julesz. Visual pattern discrimination. *IRE transactions on Information Theory*, 8(2):84–92, 1962. 2

[14] Kenichi Kanatani and Tsai-Chia Chou. Shape from texture: general principle. *Artificial Intelligence*, 38(1):1–48, 1989. 6

[15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 7

[16] Anthony Lobay and David A Forsyth. Shape from texture without boundaries. *International Journal of Computer Vision*, 67(1):71–91, 2006. 2, 3, 5, 7, 8

[17] Angeline M. Loh and Richard Hartley. Shape from non-homogeneous, non-stationary, anisotropic, perspective texture. In *BMVC*, volume 5, pages 69–78. Citeseer, 2005. 2, 5, 6, 7, 8

[18] Jitendra Malik and Ruth Rosenholtz. Computing local surface orientation and shape from texture for curved surfaces. *International Journal of Computer Vision*, 23(2):149–168, 1997. 2

[19] Javier Portilla and Eero P. Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *International Journal of Computer Vision*, 2000. 2

[20] Dor Verbin, Steven J Gortler, and Todd Zickler. Unique geometry and texture from corresponding image patches. *arXiv preprint arXiv:2003.08885*, 2020. 2, 3

[21] Ying Xiong, Ayan Chakrabarti, Ronen Basri, Steven J Gortler, David W Jacobs, and Todd Zickler. From shading to local shape. *IEEE transactions on pattern analysis and machine intelligence*, 37(1):67–79, 2014. 7

[22] Song Chun Zhu, Yingnian Wu, and David Mumford. Filters, random fields and maximum entropy (FRAME): Towards a unified theory for texture modeling. *International Journal of Computer Vision*, 27(2):107–126, 1998. 2