Paparazzi: Surface Editing by way of Multi-View Image Processing

HSUEH-TI DEREK LIU, MICHAEL TAO, ALEC JACOBSON, University of Toronto, Canada

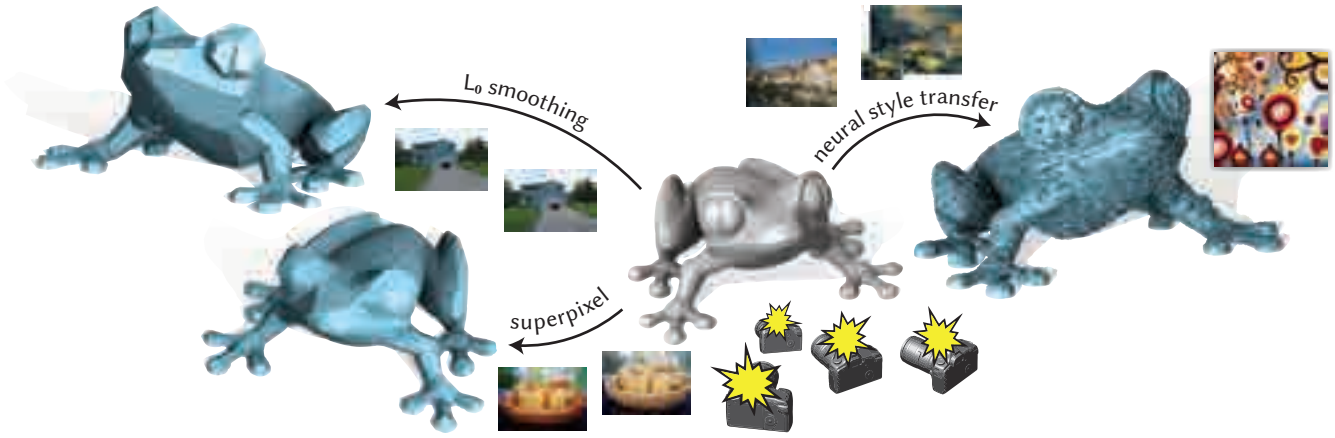


Fig. 1. Paparazzi enables plug-and-play image processing algorithms on 3D shapes. For instance, superpixel produces a Mosaic style shape; L_0 norm makes the shape piecewise planar but preserves features such as the noses; style transfer synthesizes the artistic style from a painting to the geometry. Note that the images are just for showing the 2D effects, they are not in the Paparazzi optimization loop.

The image processing pipeline boasts a wide variety of complex filters and effects. Translating an individual effect to operate on 3D surface geometry inevitably results in a bespoke algorithm. Instead, we propose a general-purpose back-end optimization that allows users to edit an input 3D surface by simply selecting an off-the-shelf image processing filter. We achieve this by constructing a differentiable triangle mesh renderer, with which we can *back propagate* changes in the image domain to the 3D mesh vertex positions. The given image processing technique is applied to the entire shape via stochastic snapshots of the shape: hence, we call our method *Paparazzi*. We provide simple yet important design considerations to construct the *Paparazzi* renderer and optimization algorithms. The power of this rendering-based surface editing is demonstrated via the variety of image processing filters we apply. Each application uses an off-the-shelf implementation of an image processing method without requiring modification to the core *Paparazzi* algorithm.

CCS Concepts: • **Computing methodologies** → **Rendering**; **Mesh geometry models**;

Additional Key Words and Phrases: geometry processing, surface editing, inverse graphics, image-based modeling, geometric deformation

ACM Reference Format:
Hsueh-Ti Derek Liu, Michael Tao, Alec Jacobson. 2018. Paparazzi: Surface Editing by way of Multi-View Image Processing. 1, 1 (September 2018), 11 pages. https://doi.org/10.1145/8888888.7777777

Author's address: Hsueh-Ti Derek Liu, Michael Tao, Alec Jacobson, University of Toronto, 40 St. George Street, Toronto, ON, M5S 2E4, Canada, hsuehtil@cs.toronto.edu.

ACM acknowledges that this contribution was authored or co-authored by an employee, contractor, or affiliate of the United States government. As such, the United States government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for government purposes only.
© 2018 Association for Computing Machinery.
XXXX-XXXX/2018/9-ART $15.00
https://doi.org/10.1145/8888888.7777777

1 INTRODUCTION

Decades of digital image processing research has culminated in a wealth of complex filters and effects. These filters are not only important as pre- and post-processes to other techniques in the image-processing pipeline, but also as useful tools for graphic designers and satisfying effects for consumers and social media users. Many of these filters depend heavily on the regular structure of pixel grids. For example, convolutional neural networks leverage this regularity to enable high-level, advanced filtering operations, such as neural style transfer.

While some simple *image* processing filters (e.g., Laplacian smoothing) have direct analogs for 3D *geometry* processing, building analogs of more complex filters often requires special case handling to accommodate arbitrary topologies, curved metrics, and irregular triangle mesh combinatorics found in 3D surface data. Moreover, many image processing methods are difficult to redefine for 3D geometry. For instance, artistic styles of paintings can be effectively captured and transferred across images, but it is not immediately clear how to transfer the *style* of a 2D painting to a 3D surface.

In this paper, we develop a novel machinery, *Paparazzi*, to simultaneously generalize a large family of image editing techniques to 3D shapes. The key idea is to modify an input 3D surface mesh by applying a desired image processing technique on many rendered snapshots of the shape (hence, *Paparazzi*). Essential to the core of *Paparazzi* is our differentiable rendering process that allows the propagation of changes in the image domain to changes in the mesh vertex positions. We first construct a stochastic multi-view optimization for generalizing energy-minimization-based image processing techniques. We then generalize this algorithm further to accommodate generic iterative image-processing filters. The renderer and its

parameters are carefully constructed to consider view sampling, occlusion, and shading ambiguities. The intermediary and output triangle meshes of our optimization are filtered to ensure watertightness, facilitating downstream geometry processing applications, such as 3D printing (see inset for a 3D-printed L_0-smoothing Frog from Fig. 1). We demonstrate the versatility and plug-and-play nature of *Paparazzi* by generalizing a handful of image filtering techniques to 3D shapes, including guided filters, quantization, superpixel, L_0-smoothing, and neural style transfer. With *Paparazzi*, we generalize these image filters to geometry by simply plugging existing implementations in.

2 RELATED WORK

Our work touches topics across visual computing, including rendering, computer vision, and geometry processing. We focus our discussion on methods similar in methodology or application.

Differential rendering. Rendering is the *forward* process of synthesizing an image given information about the scene geometry, materials, lighting and viewing conditions. Solving the *inverse* problem is tantamount to solving computer vision. Loper and Black [2014] propose a fully differentiable rendering engine, OpenDR, using automatic differentiation. Their renderer is differentiable to any input parameter – not just geometry, thus more general than ours. Though they demonstrate considerable speedup over naive finite differences when differentiating with respect to many mesh vertices, our analytical derivative results in orders of magnitude speedups over their method for the case we consider in this paper (see Fig. 2). Liu et al. [2017] propose a neural network architecture to *approximate* the forward image formation process, and predicts intrinsic parameters, shape, illumination, and material, from a single image. This neural network approach is differentiable and utilizes existing data to achieve plausible material editing results. However, it is approximate and requires a large training effort for each task and for each rendering parameter. Many other differentiable or invertible renderers have been constructed for estimating materials/microgeometry [Gkioulekas et al. 2013; Zhao 2014] or lighting conditions [Marschner and Greenberg 1997; Ramamoorthi and Hanrahan 2001]. While our lighting conditions and materials are quite tame (flat shading with three directional lights), we differentiate the entire image with respect to all mesh vertex positions.

Our analytic derivatives are faster and scale better than existing automatic differentiation frameworks: OpenDR (forward mode) [Loper and Black 2014] and the TensorFlow 3D Mesh Renderer (reverse mode, a.k.a., back propagation) [Genova et al. 2018]. On a single machine, *Paparazzi* can handle problems with more than 100,000 variables, but OpenDR and TensorFlow run out of memory on problems with few thousand and few hundreds variables respectively. In Fig. 2, our runtime (on a 256×256 image) is orders of magnitude faster.

Image-based Surface Editing. In geometric modeling, a number of previous methods have proposed interactive or automatic methods to edit a shape by directly specifying its rendered appearance

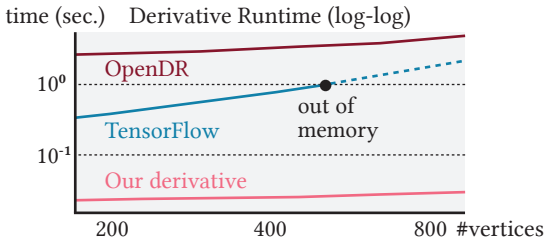


Fig. 2. We compare runtime per iteration of our approach with two autodiff-based approaches (256×256 image). Our approach is faster and scales better.

[Kerautret et al. 2005; Tosun et al. 2007; Van Overveld 1996]. For example, Gingold and Zorin [2008] allow a user to paint darkening and brightening strokes on a surface rendered with a single light source. To overcome the shading ambiguity — a thorny issue shared by all shape-from-shading methods — they choose the deformation that would minimally change the existing surface. Instead, we overcome this ambiguity by increasing the lighting complexity. Schüller et al. [2014] take advantage of the *bas*-relief ambiguity for Lambertian surfaces to create bounded thickness surfaces that have the same appearance as a given surface from a particular view. A unique contribution of our work is that we optimize for desired appearance over all views of a surface using stochastic gradient descent.

Image-based methods are widely used for mesh simplification and accelerated rendering [Weier et al. 2017]. These methods reduce polygon meshes for rendering efficiency, but preserve their perceptual appearance [El-Sana et al. 1999; Hoppe 1997; Lindstrom and Turk 2000; Luebke and Erikson 1997; Luebke and Hallen 2001; Williams et al. 2003; Xia and Varshney 1996]. The success of image-driven simplification demonstrates the power of image-driven methods, but only as a metric. Our method goes one step further, and utilize rendering similarity to generalize a large family of image processing techniques to surface editing.

Kato et al. [2017] generalize neural style transfer to 3D meshes by applying image style transfer on renderings. At a high level, their approach is similar to *Paparazzi* insofar as they propagate the image gradient to the geometry, but their derivatives are *approximate* while ours are *analytical*. In particular, they consider whether a pixel is covered by a certain triangle, which requires approximating a non-differentiable step function with respect to motions of mesh vertices in 3D. In contrast, we consider how a triangle's orientation (per-face normal) changes under an infinitesimal perturbation of a mesh vertex. This captures the continuous change of each pixel's color and enables analytical derivatives. Kato et al. [2017] do not prevent self-intersections (see red above) that inevitably arise during large deformations. Self-intersections may lead to diverging or sub-par optimization results. These differences make *Paparazzi* a more general image-driven method for creating high-quality, even fabricable 3D objects.

Fig. 3. The Bunny is optimized so the gradient of its rendered image for a single view is minimal in an L_0 sense.

Shape from shading. Recovering geometry from photographed (or rendered) images is known as the "shape from shading" problem. This subfield itself is quite broad, so we defer to existing surveys [Prados and Faugeras 2006; Zhang et al. 1999] and focus on the most related methods. Given insufficient or unreliable data, shape from shading algorithms typically fall back on assumptions for regularization, such as surface smoothness [Barron and Malik 2015], and consequently produce less detailed models. The single view shape from shading problem is made easier in the presence of per-pixel depth information, where inverse rendering can be used to *refine* the depth geometry to match the shading image [Or-El et al. 2016; Wu et al. 2014]. Shading-based depth refinement can be extended to full-shape *reconstruction* refinement if given multiple depth and shading images from different views [Choe et al. 2017; Robertini et al. 2017; Wu et al. 2011]. Gargallo et al. [2007] exactly differentiate the reprojection error function with respect to the unknown surface. Delaunoy and Prados [2011] extend this to minimize image-based regularization terms to aid in multi-view surface reconstruction. All such shape-from-shading methods are built around assumptions that the input data is captured – however unreliably – from an underlying fully realized physical shape. Our problem is similar to multi-view shading-based geometry refinement but there is a major difference – we have access to an general underlying geometric representation. We utilize this access to develop a more powerful framework that generalizes various image processing directly to 3D, rather than just geometry refinement.

Single-purpose Geometry Processing Filters. In our results, we show examples of various filters being applied to geometry by simply attaching *Paparazzi* to existing image-processing code (e.g., Skimage [Van der Walt et al. 2014]). Our results demonstrate that we successfully transfer these effects: for example, [Xu et al. 2011] creates piecewise-constant images, and via *Paparazzi* we use their method to create piecewise-constant appearance geometry (see Fig. 3 for a single-view example). Some of the image-processing filters we use for demonstration purposes have previously been *translated* to triangle meshes as single-purpose filters. For example, He and Schaefer [2013] introduce a novel edge-based Laplacian to apply L_0 regularization to meshes. Similarly, in order to create a 3D analogy of guided filters [He et al. 2010] for meshes, Zhang et al. [2015] design a specific triangle-clustering method tailored to guided filtering. Extending texture synthesis to 3D geometry has been an active research area [Dumas et al. 2015; Gu et al. 2002; Knöppel et al. 2015; Lai et al. 2005; Landreneau and Schaefer 2010; Turk 1991; Wei and Levoy 2001], the typical challenges lie in accounting for curvature and irregular mesh discretizations.

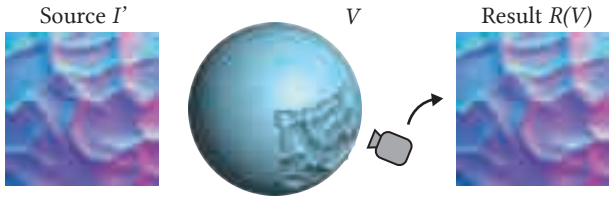


Fig. 4. A sphere is deformed to match a source image I' from the perspective of a single camera.

Rather than increment the state of the art for any specific mesh filter or application (e.g., denoising), our technical contribution is a suite of algorithms to provide a general-purpose, plug-and-play machinery for directly applying a large family of image processing filters to 3D. We evaluate our results in terms of how well *Paparazzi* correctly applies the image-processing effect to the input geometry.

3 OVERVIEW

Paparazzi is a general-purpose machinery that allows a user to apply an image-processing filter to 3D geometry, without needing to redefine that filter for surfaces or even implement new code for triangle meshes. The input to our method is a non-self-intersecting, manifold triangle mesh and a specified image-processing technique. The output is a non-self-intersecting deformation of this mesh, whose appearance has undergone the given processing. The core insight is that if we can pull gradients back from rendered images to vertices then we can apply gradient-descent-based optimization with respect to vertex positions. We first describe the well-posed scenario where the specified image-processing technique is described as an energy optimization in the image domain. Subsequently, we show that a slight modification to our energy-based method enables us to generalize to the class of iterative image-processing techniques.

3.1 Energy-Based Image Filters

Many image editing algorithms can be formulated as the minimization of a differentiable, image-domain energy E. In the ideal setting, we extend any such energy minimization to surfaces by considering the integral of this energy applied to *all possible* rendered images for a space of camera "views":

$$\min_V \int_{i \in \text{views}} E(R_i(V)),$$

where R_i is a function mapping a mesh with vertices V to an image.

Minimization is straightforward to write as a gradient descent with respect to vertices using the chain rule:

$$V \leftarrow V - \int_{i \in \text{views}} \frac{\partial E}{\partial R_i} \frac{\partial R_i}{\partial V}. \tag{1}$$

The space of views can be tuned to satisfy problem-specific needs. For instance, it could be as small as a single front-on camera or as large as the space of all cameras where some amount of geometry is visible. We defer discussion of a good default choice until Section 6.

Consider the toy example of an energy-based image editing algorithm visible in Figure 4, where the energy E is simply the L_2

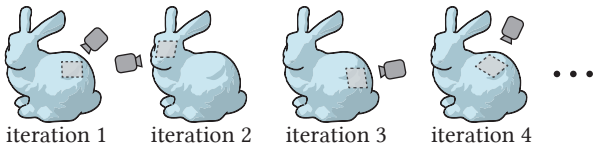


Fig. 5. We sample a view per iteration in the multi-view optimization.

distance to a rendering of another shape. In the optimization, we consider only a single view of a sphere. After gradient descent, the sphere's geometry is deformed so that this one view is imperceptibly similar to the source image. For a single view, our method only changes vertices that affect that view's rendering, which makes this result appear like a decal.

The presence of the Jacobian $\partial R_i/\partial V$ exposes the main requirement of the renderer R: differentiability with respect to vertex positions. In this paper, we present an as-simple-as-possible renderer that is *analytically* differentiable with mild assumptions and is effective for generating high-quality geometries (see Section 5).

3.2 Stochastic Multi-view Optimization

When we look at a single view, the analytical derivative $\partial R/\partial V$ enables the direct generalization of image processing algorithms to geometries via Equation (1), but evaluating this integral over a continuous space or distribution of views is challenging.

We handle this issue by borrowing tools from the machine learning community, who have extensively applied gradient descent to energies involving integrals or large summations when training deep networks [Bottou et al. 2016; Ruder 2016]. Rather than attempt to compute the integrated gradient exactly, we leverage stochastic gradient descent (SGD) and update the geometry with the gradient of a small subset of the views, as few as one. As is common in machine learning literature, we apply momentum both to regularize the noise introduced by using a stochastic gradient and to improve general performance with the *Nesterov-Adam* method [Dozat 2016], a variant of gradient descent that combines momentum and Nesterov's accelerated gradient. Our stochastic multi-view optimization is summarized in Algorithm 1.

As the mesh deforms according to the optimization, the quality of triangles may degrade and self-intersections may (and inevitably will) occur. We discuss the importance of interleaving a mesh quality improvement stage in our optimization loop in Section 4.3.

3.3 Iterative Image Filters

With only minor modifications we may generalize our method from energy-based deformations to the realm of iterative filters, generically defined as an iterative process acting in the image domain:

$$I_{j+1} \leftarrow f(I_j).$$

We replace the energy gradient with the update induced by a single iteration of the filter by replacing the derivative $\partial E/\partial R$ with the difference $\Delta R := f(R) - R$. The update to the mesh vertex positions becomes:

$$V \leftarrow V - \int_{i \in \text{views}} (f(R_i(V)) - R_i(V)) \frac{\partial R_i}{\partial V}.$$

Algorithm 1: Energy-Based Method

Input:
V_0, F_0 triangle mesh
R renderer
E image energy
Output:
V, F optimized triangle mesh
begin
  $V, F \leftarrow V_0, F_0$;
  $O \leftarrow$ offset surface from V_0, F_0;
  while *E not converge* **do**
    $i \leftarrow$ sample camera from O;
    $\partial E/\partial R_i \leftarrow$ compute image derivative for camera i ;
    $\partial R_i/\partial V \leftarrow$ compute shape derivative for camera i;
    $V \leftarrow V - \gamma \frac{\partial E}{\partial R_i} \frac{\partial R_i}{\partial V}$;
    $V, F \leftarrow CleanMesh(V, F)$;

Algorithm 2: Iterative Filter Method

Input:
V_0, F_0 triangle mesh
R renderer
f image filter
Output:
V, F optimized triangle mesh
begin
  $V, F \leftarrow V_0, F_0$;
  $O \leftarrow$ offset surface from V_0, F_0;
  for *iteration < max. iterations* **do**
    $i \leftarrow$ sample camera from O;
    $\Delta R_i \leftarrow f(R_i(V)) - R_i(V)$;
    $\partial R_i/\partial V \leftarrow$ compute shape derivative for camera i;
    $V \leftarrow V - \gamma \Delta R_i \frac{\partial R_i}{\partial V}$;
    $V, F \leftarrow CleanMesh(V, F)$;

This generalization works when no individual application of the iterative filter modifies the image by too much. In our experiments this is close enough to smoothness to allow for our method to converge to a geometry that matches the result of the filter from different views[1]. If a single application of the filter creates a dramatic effect, then our optimization can accommodate by using a smaller step size γ. The resulting algorithm is therefore Algorithm 2.

Before demonstrating results (Section 7), we describe the considerations we made when designing our renderer and parameters.

4 DESIGN CONSIDERATIONS

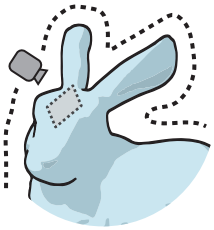
By working with renderings of geometry, image processing techniques may be applied in their native form on the rendered images of pixels. This allows *Paparazzi* to immediately generalize to a large

[1]When f is an first order explicit Euler over an energy's gradient this method is precisely our original method.

family of image processing techniques, but shifts the burden to designing a rendering setup that faithfully captures the geometry and presents it in a meaningful way to the image processing technique. Where and how we render the geometry will have a large impact on the quality of the results.

4.1 Camera Sampling

A good camera placement strategy should "see" every part of the surface with equal probability. A surface patch that is never seen by any camera will not be changed. On the other hand, a surface patch that is visible to too many cameras will be updated faster than other surface regions and result in discontinuity between surface patches.



According to these two criteria: full coverage and uniform sampling, *Paparazzi* uniformly samples cameras on an *offset surface* at distance σ, whose points are at a fixed distance from the given shape, facing along inward vertex normals. This placement ensures that we are less biased to certain views, have smooth camera views around sharp edges, and have full coverage for most of the shapes. Increasing and decreasing σ only affects the near plane because we use an orthographic camera. We set σ to 5% of the shape's bounding box diameter for small-scale deformation and 25% for large-scale deformation (see Section 6 for values of each experiment).

4.2 Lighting & Shading

Our image-driven surface editor is designed so that inputs and outputs are both 3D shapes, thus the quality of intermediate renderings are only important insofar as we achieve the desired output geometry. We propose an as-simple-as-possible default renderer for *Paparazzi*, but take special care to avoid lighting and shading ambiguities that would cause artifacts during optimization.

Shading Ambiguity. It is well-known that a single directional light is not sufficient to disambiguate convex and concave shapes and slope directions (see, e.g., [Liu and Todd 2004]). Just as this *shading ambiguity* can confuse human observes, it will confuse *Paparazzi*'s optimization. One reason is that a single directional light is not sufficient to disambiguate convex/concave shapes and slope directions (see Fig. 6).

Our simple solution, inspired by *photometric stereo* [Woodham 1980], is to increase the complexity of the lighting. By specifying three axis-aligned direction lights with R, G, B colors respectively, we effectively render an image of the surface normal vectors. This avoids the shading ambiguity.

Gouraud Ambiguity. A more subtle, but nonetheless critical ambiguity appears if we follow the common computer graphics practice of smoothing and interpolating per-vertex lighting/normals within a triangle. When rendering a triangle mesh (especially if low-resolution), Gouraud shading [1971] or Phong shading [1975] will make the shape appear smoother than the actual piecewise-linear geometry. While this illusion is convenient for efficient rendering, its inherent averaging causes an ambiguity. A surface with rough

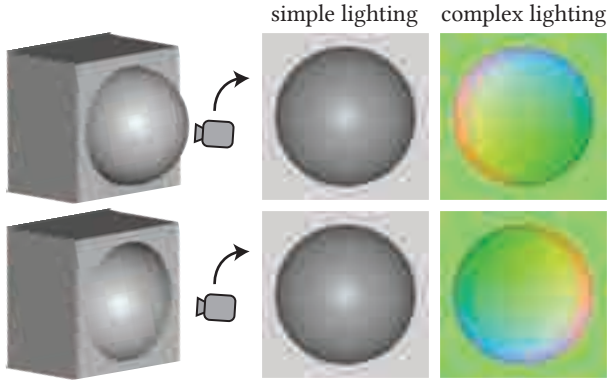


Fig. 6. Shading ambiguity. Convex/concave shapes may result in the same image under a single directional light (middle). Adding complexity to the lighting could resolve the shading ambiguity.

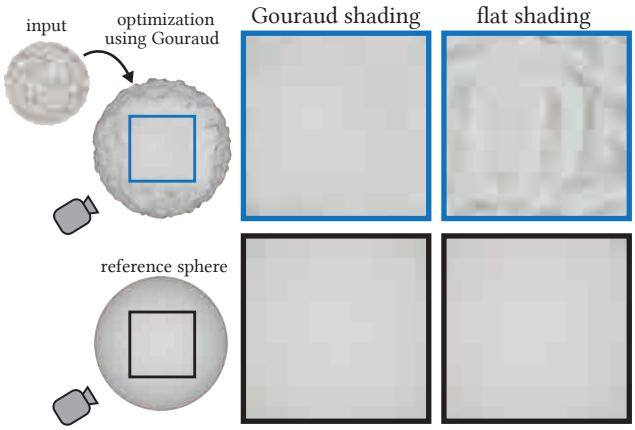


Fig. 7. Gouraud ambiguity. Given a bumpy sphere (left), we minimize image Dirichlet energy under Gouraud shading to get the smoothed sphere (middle). Comparing the renderings of the smoothed region, we observe Gouraud ambiguity which the rendering of a non-smoothed sphere is very similar to the rendered smooth sphere (left column), but flat shading reveals the difference (right column).

geometry can still produce a smooth rendering under Gouraud shading. We refer to this the *Gouraud ambiguity*. Using Gouraud shading during optimization in *Paparazzi* would immediately introduce a null space, leading to numerical issues and undesirable bumpy geometries[2] (see Fig. 7). Instead, we propose using flat shading. This is, in a sense, the most *honest* rendering of the triangle mesh's piecewise-linear geometry.

4.3 Mesh Quality

So far, our rendering choices are sufficient for *Paparazzi* to make small changes to the surface geometry but as the mesh continues to deform, the quality of individual triangles will degrade and even degenerate. Furthermore, local and global self-intersections may occur. For many of the image processing filters we consider, we

[2]We use a fast finite difference approach for the optimization under Gouraud shading.

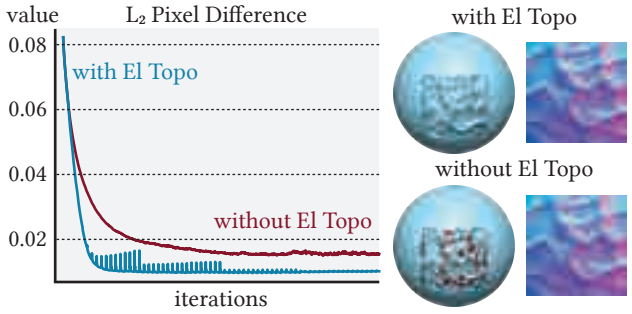


Fig. 8. Shape optimization without El Topo (bottom) may cause self-intersections (red), despite the fact that the rendering is similar to the one with El Topo (top). Besides, optimization with El Topo results in lower error (left). Note that the peaks in the plot are where we perform El Topo.

would like sharp creases and corners to emerge during surface deformation which may not be possible without re-meshing.

These challenges and specifications bear resemblance to re-meshing needed during surface-tracking fluid simulation. We borrow a state-of-the-art tool from that community, El Topo [Brochu and Bridson 2009], a package for robust explicit surface tracking with triangle meshes. It enables *Paparazzi* to generate manifold watertight meshes without self-intersections and refines the mesh in areas of high curvature which allows us to introduce sharp features without worrying about mesh locking. In Fig. 8, we can see that shape optimization requires El Topo to abate issues with bad mesh quality and self-intersections, even though the rendered results are comparable.

El Topo handles two types of operations essential to the success of *Paparazzi*: those that are critical for maintaining manifold, non-intersecting meshes and those associated with triangle quality. We feed El Topo the current non-self-intersecting mesh vertices and faces as well as the new desired vertex locations. El Topo checks for whether triangles will collide or become too close together throughout the continuous motion from the current positions to the desired positions. This can either result in repulsive forces or in topological changes depending on user-defined thresholds. In order to improve the quality of the mesh, which improves the robustness of collision detection, El Topo does standard mesh improvement operations such as edge splitting and edge flipping, which improve the aspect ratios of triangles without affecting the overall topology of the mesh. El Topo also subdivides and decimates meshes to improve the quality of the mesh near high and low curvature regions respectively by keeping edge angles between a user-defined interval.

Remeshing and collision handling are essential for maintaining high-quality meshes during and after optimization, but it is also time-consuming — especially compared to our derivative computation. This is visible in Fig. 9, where we can see that El Topo dominates the total runtime. Because deformations between any individual iteration are generally small, in practice we invoke El Topo every 30 iterations, providing an empirically determined balance between computation time and optimization performance.

5 DIFFERENTIABLE RENDERER

So far, we have discussed the design considerations of *Paparazzi*. As intermediate renderings are not the outputs, we have the flexibility

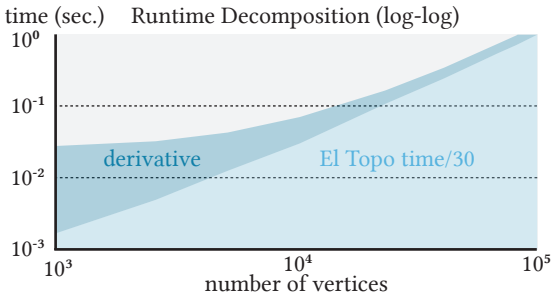


Fig. 9. We show the decomposition of our total runtime, excluding the image processing part. The top half is the time for the derivative computation; The bottom half shows 1/30 of the El Topo runtime.

in designing a suitable renderer which addresses the aforementioned challenges and, more importantly, is differentiable. In particular we present a differentiable renderer that allows us to analytically compute $\partial R/\partial V$, and to generalize image processing to 3D geometry.

5.1 Visibility

The rendering of a triangle mesh with flat shading is continuous away from silhouettes and occluding contours. It is differentiable *almost everywhere*: at all points on the image plane lying inside a triangle, but not across triangle edges or vertices (a set with measure zero). Therefore, we assume infinitesimal changes of surface points will not change the visibility because in practice we only have finite image resolution on computers.

Visibility may change under large vertex perturbations eventually incurred in our optimization loop. Fortunately, due to the efficiency of z-buffering in the real-time rendering engine OpenGL, updating visibility can be handled efficiently by re-rendering the shape once every iteration.

5.2 Analytic derivative

Given our design choices of local illumination and Lambertian surface, we render m directional lights pointed by directions $\hat{\boldsymbol{\ell}}_i$ (a unit vector in $\mathbb{R}^3$) with corresponding RGB colors $\{c_i^R, c_i^G, c_i^B\} \in [0,1]$, the output color $\{r_p^R, r_p^G, r_p^B\} \in [0,1]$ at a pixel p is computed by

$$r_p^R = \hat{\mathbf{n}}_j \cdot \sum_{i=1}^{m} \hat{\boldsymbol{\ell}}_i c_i^R, \quad r_p^G = \hat{\mathbf{n}}_j \cdot \sum_{i=1}^{m} \hat{\boldsymbol{\ell}}_i c_i^G, \quad r_p^B = \hat{\mathbf{n}}_j \cdot \sum_{i=1}^{m} \hat{\boldsymbol{\ell}}_i c_i^B,$$

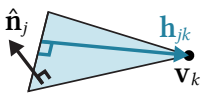
where $\hat{\mathbf{n}}_j$ is a unit vector in $\mathbb{R}^3$ representing the normal of the jth face of the triangle mesh V, and the jth face is the nearest face under pixel p.

Without the loss of generality, we only write the red component r_p^R in our derivation as r_p^G, r_p^B share the same formulation. We can analytically differentiate this formula with respect to the vertex positions to form each row $\partial r_p^R/\partial V \in \mathbb{R}^{3|V|}$ of the *sparse* Jacobian matrix. Only the position of each vertex $\mathbf{v}_k \in \mathbb{R}^3$ at a corner of the jth triangle contributes:

$$\frac{\partial r_p^R}{\partial \mathbf{v}_k} = \begin{cases} \frac{\partial \hat{\mathbf{n}}_j}{\partial \mathbf{v}_k} \sum_{i=1}^{m} \hat{\boldsymbol{\ell}}_i c_i^R & \text{if vertex } k \text{ is corner of triangle } j, \\ 0 & \text{otherwise.} \end{cases}$$

Finally, the 3×3 Jacobian of face normals $\hat{\mathbf{n}}_j$ over triangle vertices $\mathbf{v}_k$,

$\partial\hat{\mathbf{n}}_j/\partial\mathbf{v}_k$, can be computed *analytically*. Note that moving $\mathbf{v}_k$ in the triangle's plane will not change $\hat{\mathbf{n}}_j$. Also, in the limit, moving along $\hat{\mathbf{n}}_j$ only changes $\hat{\mathbf{n}}_j$ in the $\mathbf{h}_{jk}$ direction where $\mathbf{h}_{jk} \in \mathbb{R}^3$ is the "height" vector: the shortest vector to the corner $\mathbf{v}_k$ from the line of the opposite edge. This implies that the Jacobian must be some scalar multiple of $\mathbf{h}_{jk}\hat{\mathbf{n}}_j^\mathsf{T}$. This change is inversely proportional to $\|\mathbf{h}_{jk}\|$, the distance of $\mathbf{v}_k$ to the opposite edge, which means:



$$\frac{\partial\hat{\mathbf{n}}_j}{\partial\mathbf{v}_k} = \frac{\mathbf{h}_{jk}\hat{\mathbf{n}}_j^\mathsf{T}}{\|\mathbf{h}_{jk}\|^2}.$$

6 IMPLEMENTATION

In our experiments, we normalize shapes to fit in a unit-radius cube centered at the origin and upsample shapes to reach 10^5-10^6 vertices in order to capture geometric detail. By default, we use a square, orthographic camera with a 0.5-wide field of view placed at an offset of $\sigma = 0.1$, where the units are those of the OpenGL canonical view volume. The offset surface meshes have 10^3-10^4 vertices. By default we use three directional lights in red, green, and blue colors respectively along each local camera axis, which is equivalent to rendering the surface normals in the camera frame.

We implement our derivative computation in Python using vectorized Numpy operations and calls to OpenGL for rendering and rasterization. We use libigl [Jacobson et al. 2016] and the Meshmixer [Schmidt and Singh 2010] for mesh upsampling and offset surface computation. We test our implementation on a Linux workstation with an Intel Xeon 3.5GHz CPU, 64GB of RAM, and an NVIDIA GeForce GTX 1080 GPU.

6.1 Off-the-Shelf Image Processing Filters

We have designed *Paparazzi* to plug-and-play with existing image processing filters. We are able to use open, readily available, implementations of image-space filters with minimal effort. To evaluate our method we used a handful of off-the-shelf image filters. We use a Python implementation of the fast guided filter [He and Sun 2015], found at `github.com/swehrwein/python-guided-filter`. For SLIC superpixels [Achanta et al. 2012] we use the implementation in the popular Python image processing library, Skimage. We translated a Matlab implementation of image smoothing with L_0-smoothing [Xu et al. 2011] from `github.com/soundsilence/ImageSmoothing` into Python. For neural style transfer [Gatys et al. 2016], we followed the corresponding PyTorch [Paszke et al. 2017] tutorial with minor modification to extract the $\partial E/\partial R_i$ gradient. We implemented simple image quantization with a fixed palette ourselves (see review in [Ozturk et al. 2014]).

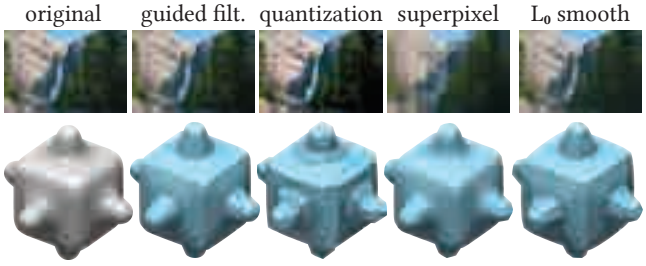
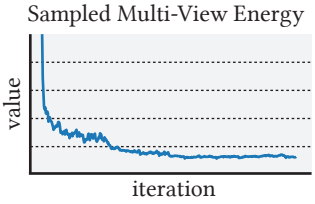


Fig. 10. *Paparazzi* allows direct generalization of image processing to 3D, thus different image editing effects can be directly transfered to 3D shapes.

Applying these filters to 3D geometry requires no modification of the *Paparazzi* algorithms. The caller either provides the $\partial E/\partial R_i$ gradient to use Algorithm 1 for energy-based methods or provides the filter f as a function handle to use Algorithm 2 for iterative methods. From a user's prospective, trying various filters is quite effortless. In Fig. 10 we demonstrate how *Paparazzi* produces different results for the various smoothing-type filters we tested. Each result respects the intent of the particular image processing filter, but now applied to a 3D surface.

7 EVALUATION & DISCUSSION

In Table 1, we decompose our runtime in terms of subroutines: derivative computation, image processing, and mesh cleaning using El Topo. Our differentiation is orders of magnitude faster than previous methods (see Fig. 2). Mesh cleaning is the bottleneck for high-resolution meshes (see Fig. 9). Because our multi-view optimization processes the rendering of local patches multiple times, the runtime performance of the particular input image processing method is amplified by our approach (e.g., simple quantization is much faster than neural style transfer).



For energy-based filters, evaluating the *integrated* multi-view energy would require rendering and evaluating from all possible camera views. Even approximating this with a finite number of views every iteration would be too expensive. Instead, to evaluate convergence behavior we can set up a camera at a fixed view and evaluate its visible energy as the multi-view optimization stochastically decreases the (unmeasured) integrated energy. The energy of the particular rendering does not represent the value of multi-view energy, but the convergence behavior implies the convergence of multi-view energy. In the inset, we show the convergence of the neural style transfer energy.

Example	$\|V\|$	$\|R\|$	*Iters.*	ΔR	El Topo	$\partial R/\partial \mathbf{v}$	*Tot.*
Guided	26K	128	3K	1.03*s*	0.13*s*	**0.08s**	64*m*
L_0	40K	256	3K	0.45*s*	0.17*s*	**0.12s**	40*m*
SLIC	43K	256	3K	0.10*s*	0.28*s*	**0.14s**	28*m*
Quant.	48K	256	1K	0.12*s*	0.27*s*	**0.16s**	11*m*
Neural	143K	128	10K	0.03*s*	1.73*s*	**0.48s**	390*m*

Table 1. For each *Example* image-processing filter on a mesh with $|V|$ vertices, rendering $|R|^2$ pixels for *Iters.* iterations, we list average seconds per iteration to call the image processing filter or gather its gradient ΔR, invoking El Topo (slowest), and computing $\partial R/\partial V$ (bold; fastest). Finally, we report *Total* time to make a result in minutes.

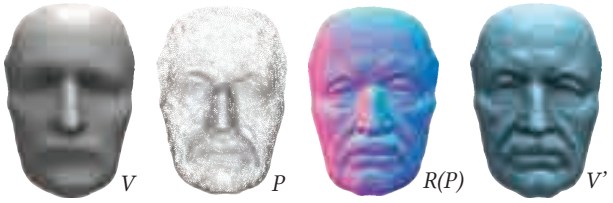


Fig. 11. We transfer geometric details from the input point cloud P to the input shape V through rendering $R(P)$ the point cloud.

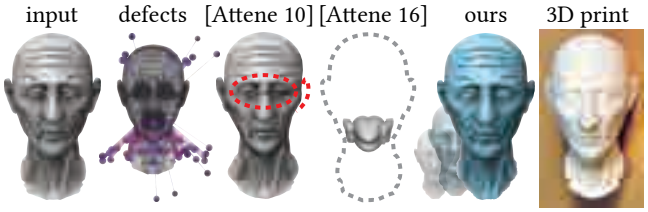


Fig. 12. We can repair a mesh with disconnected parts and self-intersections by creating a coarse proxy and then applying detail transfer. These defects are visualized by MeshmMixer [Schmidt and Singh 2010] and proved challenging for dedicated mesh cleaning methods.

7.1 Evaluation on Image Filters

We evaluate *Paparazzi* according to its ability to reproduce the effect of 2D filters on 3D shapes, instead of its domain-specific success for any specific application (e.g., denoising). In Fig. 10 we see that changing the image processing filter indeed changes the resulting edited shape. Guided filter correctly achieves an edge-preserving smoothing effect; quantization makes surface patches align with predefined normals[3]; superpixel creates super-faces; and L_0-smoothing results in piecewise planar geometry. We can see that these filters are correctly transferred to 3D geometry in a plug-and-play fashion. All the while, our remeshing ensures the output mesh is watertight.

We start by considering a simple yet powerful differentiable energy – the L_2 pixel difference. Because its derivatives $\partial E/\partial R$ are known, we apply Algorithm 1 to generalize this energy to 3D shapes. By caching the rendering of one geometry we can use this energy minimization to transfer its appearance to another geometry. Compared to dedicated mesh transfer tools (e.g., [Takayama et al. 2011]) we do not require the source geometry to be another triangle mesh: simply anything we can render. In Fig. 11 we can transfer details from point cloud P to a triangle mesh V by minimizing the L_2 image difference $\|R(P) - R(V)\|^2$. We use a simple splat rendering but this example would immediately fit from more advanced point cloud rendering (see, e.g., [Kobbelt and Botsch 2004]).

The source geometry could be a mesh with defects such as self-intersections and holes. In Fig. 12, we transfer a triangle soup's appearance on top of a smooth surface reconstruction created using robust signed distance offsetting [Barill et al. 2018]. The result is a new watertight mesh with the appearance of the messy input, which previous mesh repairing methods have difficulty preserving [Attene 2010, 2016]. This mesh is now fit for 3D printing.

In the following Figures 13–17, the images on the left are given as a reference to show the corresponding image processing and are not used to make the surface editing results. By construction our 3D inputs and outputs mirror the "analogies" of Hertzmann et al. [2001], but unlike that method we have direct access to the underlying image processing algorithm.

We now explore a more complicated energy — the Neural Style energy. Recently, inspired by the power of Convolutional Neural Network (CNN) [Krizhevsky et al. 2012], *Neural Style Transfer* has been a popular tool for transferring artistic styles from painting to other images [Gatys et al. 2016]. The goal is to generate a stylized image given a content image and a reference style image. Gatys et al. [2016] define the total energy to be the summation of content and style energies, where the content energy encourages the stylized output image to have similar image structure with the content image and the style energy encourages the output to have similar features with the reference style image. Note that the features are defined using the filter responses of a CNN across different layers.

Transferring artistic styles to 3D geometries is challenging because the redefinition of 2D painting styles on 3D is unclear. With *Paparazzi*, we can generalize it by applying the image neural style transfer on the renderings. Because the image gradient can be computed by differentiating the CNN, we can use Algorithm 1 to generate stylized shapes. In Fig. 13, *Paparazzi* transfers the style of 2D paintings to 3D via growing geometric textures (we provide implementation detail about image neural style in Appendix A).

A large portion of image processing algorithms are not based on energy minimization but applying iterative procedures. These algorithms may not have a well-defined energy or, even if they do, may not have an easily computable gradient. Fortunately, *Paparazzi* provides an effortless way to generalize a variety of iterative image filters using Algorithm 2. The high-level idea is to perform image update on the rendering once, and update the shape once based on how the image change due to the image update.

Guided filters [He et al. 2010] compute the filtered image output by considering the content of a guidance image, the guidance image can be another image or the input itself. He et al. [2010] shows

Fig. 13. We generalize the neural style transfer to 3D by minimizing the style energy of local renderings through manipulating vertex positions.

[3] A *palette* containing the edge and face normals of a cube.

Fig. 14. We generalize the fast guided filter to 3D and achieve edge-preserving smoothing effect.

Fig. 15. Image quantization is applied to geometries and make surface patches facing toward pre-defined color palettes.

Fig. 16. The SLIC superpixel method is applied to 3D objects, results in small surface patches appearing on the shape

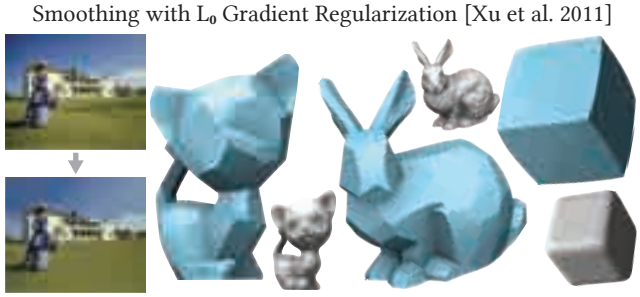


Fig. 17. We minimize the L_0 norm of image gradients and encourage the output shapes (blue) to be piece-wise planar.

that the guided filter is effective in a variety of image processing applications including edge-aware smoothing, detail enhancement, image feathering, and so on. In Fig. 14 we apply the edge-aware smoothing guided filter with the acceleration proposed in [He and Sun 2015]. We set the guidance image to be the input and the filter parameters to be $r = 4, \epsilon = 0.02$. By plugging this filter to the filter function f in Algorithm 2, we can see that the guided filter smooths 3D shapes and preserves sharp features.

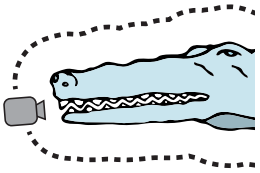
In addition to edge-preserving smoothing, we are interested in utilizing image filters to create different visual effects on geometry. A simple by stylistic choice is image *quantization*, an image compression technique compressing a range of color values to a single value and representing an image with only a small set of colors [Ozturk et al. 2014]. Again, by changing the filter f in Algorithm 2, we can quantize 3D shapes with pre-define color set[4] (see Fig. 15). Note that these colors are encoded in the world coordinate, thus the shape quantization is orientation dependent and requires rendering normals in world coordinates, which is different from other filters that render normals in local coordinates.

Another pixel segmentation approach, but based on both color and spatial information, is the *superpixel*. In Fig. 16, we use Simple Linear Iterative Clustering (SLIC) [Achanta et al. 2012] which adapts k-means to segment pixels to create "super-faces" on shapes.

Last but not least, we consider a filter that minimizes the L_0 norm of image gradient [Xu et al. 2011]. L_0 norm has been a popular tool for image and signal processing for decades because it is a direct measure of signal sparsity. However, L_0 norm can be difficult to optimize due to its discrete, combinatorial nature. Xu et al. [2011] present an iterative image optimization method to minimize L_0 gradient and generate edge-preserving, piecewise constant filtering effects. With Algorithm 2 we can simply apply such iterative procedures to generalize the effect of L_0 norm to a 3D shape and make it piecewise planar which is the 3D analogue of piecewise constant in images (see Fig. 17).

[4]The pre-defined color set are the color of 20 face normals of a icosahedron

8 LIMITATIONS & FUTURE WORK



Paparazzi samples a precomputed offset surface for camera locations. This means heavily occluded or tight inner cavities of a surface will not receive edits (e.g., inside an alligator's mouth). It also means the shape is implicitly trapped inside its original offset surface *cage*. Removing this cage constraint and predicting the change of visibility would aid to creating large shape deformations. For a stricter and more deterministic image energy, it would be important to align the cameras' orientation to encourage consistency across views in the overlapped regions. Meanwhile, we only present analytic derivatives for flat-shaded triangle meshes; similar derivatives could be derived for other representations such as subdivision surfaces or NURBS models. *Paparazzi*'s differentiation is orders of magnitude faster than

previous work. In future work, we would like to further improve the performance of *Paparazzi* by exploiting the parallelism of the stochastic multi-view optimization and improving the collision-detection needed for dynamic meshing (currently, El Topo — used as a black-box — dominates our runtime, see Fig. 9).

At its core, *Paparazzi* is a differentiable renderer with a stochastic multiview gradient-descent procedure that can *back propagate* image changes to a 3D surface. *Paparazzi* imposes a 3D interpretation of a 2D filter, but could be an useful tool for studying other filters that have no straightforward 3D interpretation. Extending *Paparazzi* to operate with global illumination and textures as well as more sophisticated lighting models could be beneficial for applications that require realistic renderings, such as image classification. In our neural style transfer examples, we show only a small indication of the larger possibility for *Paparazzi*'s usefulness to transfer the success of image-based deep learning to 3D surface geometry. *Paparazzi* demonstrates the utility of rendering not just for visualization, but also as a method for *editing* of 3D shapes. It is exciting to consider other ways *Paparazzi* can influence and interact with the geometry processing pipeline.

ACKNOWLEDGMENTS

This work is funded in part by NSERC Discovery Grants (RGPIN2017-05235 & RGPAS-2017-507938 & RGPIN-2017-05524), NSERC DAS (RGPAS-2017-507909), Connaught Funds (NR2016-17), the Canada Research Chairs Program, and gifts by Adobe Systems Inc, Autodesk Inc, and Fields Institute. We thank members of Dynamic Graphics Project, including R. Abdrashitov, R. Arora, G. Barill, E. Corman, L. Fulton, T. Jeruzalski, J. Kenji, S. Kushner, D. Levin, J. Li, V. Modi, D. Moore, R. Schmidt, M. Wei, for early feedback and draft reviews; D. Nowrouzezahrai and M. McGuire for discussions about differentiable renderers and their applications.

REFERENCES

Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. 2012. SLIC superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence* 34, 11 (2012), 2274–2282.

Marco Attene. 2010. A lightweight approach to repairing digitized polygon meshes. *The visual computer* 26, 11 (2010), 1393–1406.

Marco Attene. 2016. As-exact-as-possible repair of unprintable STL files. *arXiv preprint arXiv:1605.07829* (2016).

Gavin Barill, Neil Dickson, Ryan Schmidt, David I.W. Levin, and Alec Jacobson. 2018. Fast Winding Numbers for Soups and Clouds. *ACM Transactions on Graphics* (2018).

Jonathan T Barron and Jitendra Malik. 2015. Shape, illumination, and reflectance from shading. *IEEE transactions on pattern analysis and machine intelligence* 37, 8 (2015), 1670–1687.

Léon Bottou, Frank E Curtis, and Jorge Nocedal. 2016. Optimization methods for large-scale machine learning. *arXiv preprint arXiv:1606.04838* (2016).

Tyson Brochu and Robert Bridson. 2009. Robust topological operations for dynamic explicit surfaces. *SIAM Journal on Scientific Computing* 31, 4 (2009), 2472–2493.

Gyeongmin Choe, Jaesik Park, Yu-Wing Tai, and In So Kweon. 2017. Refining Geometry from Depth Sensors using IR Shading Images. *Inter. Journal of Computer Vision* (2017).

Amaël Delaunoy and Emmanuel Prados. 2011. Gradient flows for optimizing triangular mesh-based surfaces: Applications to 3d reconstruction problems dealing with visibility. *International journal of computer vision* 95, 2 (2011), 100–123.

Timothy Dozat. 2016. Incorporating nesterov momentum into adam. (2016).

Jérémie Dumas, An Lu, Sylvain Lefebvre, Jun Wu, and Christian Dick. 2015. By-example synthesis of structurally sound patterns. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 137.

Jihad El-Sana, Elvir Azanli, and Amitabh Varshney. 1999. Skip strips: maintaining triangle strips for view-dependent rendering. In *Proceedings of the conference on Visualization'99: celebrating ten years*. IEEE Computer Society Press, 131–138.

Pau Gargallo, Emmanuel Prados, and Peter Sturm. 2007. Minimizing the reprojection error in surface reconstruction from images. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*. IEEE, 1–8.

Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. 2016. Image Style Transfer Using Convolutional Neural Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. 2414–2423.

Kyle Genova, Forrester Cole, Aaron Maschinot, Aaron Sarna, Daniel Vlasic, and William T. Freeman. 2018. Unsupervised Training for 3D Morphable Model Regression. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Yotam Gingold and Denis Zorin. 2008. Shading-based surface editing. *ACM Transactions on Graphics (TOG)* 27, 3 (2008), 95.

Ioannis Gkioulekas, Shuang Zhao, Kavita Bala, Todd Zickler, and Anat Levin. 2013. Inverse volume rendering with material dictionaries. *ACM Transactions on Graphics (TOG)* 32, 6 (2013), 162.

Henri Gouraud. 1971. Continuous shading of curved surfaces. *IEEE transactions on computers* 100, 6 (1971), 623–629.

Xianfeng Gu, Steven J Gortler, and Hugues Hoppe. 2002. Geometry images. *ACM Transactions on Graphics (TOG)* 21, 3 (2002), 355–361.

Kaiming He and Jian Sun. 2015. Fast Guided Filter. *CoRR* abs/1505.00996 (2015). http://arxiv.org/abs/1505.00996

Kaiming He, Jian Sun, and Xiaoou Tang. 2010. Guided image filtering. In *European conference on computer vision*. Springer, 1–14.

Lei He and Scott Schaefer. 2013. Mesh denoising via L 0 minimization. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 64.

Aaron Hertzmann, Charles E Jacobs, Nuria Oliver, Brian Curless, and David H Salesin. 2001. Image analogies. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 327–340.

Hugues Hoppe. 1997. View-dependent refinement of progressive meshes. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 189–198.

Alec Jacobson, Daniele Panozzo, et al. 2016. libigl: A simple C++ geometry processing library. (2016). http://libigl.github.io/libigl/.

Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. 2017. Neural 3D Mesh Renderer. *CoRR* abs/1711.07566 (2017). arXiv:1711.07566 http://arxiv.org/abs/1711.07566

Bertrand Kerautret, Xavier Granier, and Achille Braquelaire. 2005. Intuitive shape modeling by shading design. In *International Symposium on Smart Graphics*. Springer, 163–174.

Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. 2015. Stripe Patterns on Surfaces. *ACM Trans. Graph.* (2015).

Leif Kobbelt and Mario Botsch. 2004. A survey of point-based techniques in computer graphics. *Computers & Graphics* 28, 6 (2004), 801–814.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

Y-K Lai, S-M Hu, DX Gu, and Ralph R Martin. 2005. Geometric texture synthesis and transfer via geometry images. In *Proc. SPM*.

Eric Landreneau and Scott Schaefer. 2010. Scales and Scale-like Structures. *Comput. Graph. Forum* (2010).

Peter Lindstrom and Greg Turk. 2000. Image-driven simplification. *ACM Transactions on Graphics (ToG)* 19, 3 (2000), 204–241.

Baoxia Liu and James T Todd. 2004. Perceptual biases in the interpretation of 3D shape from shading. *Vision research* (2004).

Guilin Liu, Duygu Ceylan, Ersin Yumer, Jimei Yang, and Jyh-Ming Lien. 2017. Material Editing Using a Physically Based Rendering Network. In *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2280–2288.

Matthew M Loper and Michael J Black. 2014. OpenDR: An approximate differentiable renderer. In *European Conference on Computer Vision*. Springer, 154–169.

David Luebke and Carl Erikson. 1997. View-dependent simplification of arbitrary polygonal environments. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 199–208.

David Luebke and Benjamin Hallen. 2001. Perceptually driven simplification for interactive rendering. In *Rendering Techniques 2001*. Springer, 223–234.

Stephen R. Marschner and Donald P. Greenberg. 1997. Inverse Lighting for Photography. In *Color and Imaging Conference*.

Roy Or-El, Rom Hershkovitz, Aaron Wetzler, Guy Rosman, Alfred M. Bruckstein, and Ron Kimmel. 2016. Real-Time Depth Refinement for Specular Objects. In *Proc. CVPR*.

Celal Ozturk, Emrah Hancer, and Dervis Karaboga. 2014. Color image quantization: a short review and an application with artificial bee colony algorithm. *Informatica* 25, 3 (2014), 485–503.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. (2017).

Bui Tuong Phong. 1975. Illumination for computer generated pictures. *Commun. ACM* 18, 6 (1975), 311–317.

Emmanuel Prados and Olivier Faugeras. 2006. Shape from shading. *Handbook of mathematical models in computer vision* (2006), 375–388.
Ravi Ramamoorthi and Pat Hanrahan. 2001. A Signal-processing Framework for Inverse Rendering. In *Proc. SIGGRAPH*.
Nadia Robertini, Dan Casas, Edilson Aguiar, and Christian Theobalt. 2017. Multi-view Performance Capture of Surface Details. *Int. J. Comput. Vision* (2017).
Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).
Ryan Schmidt and Karan Singh. 2010. Meshmixer: an interface for rapid mesh composition. In *ACM SIGGRAPH 2010 Talks*. ACM, 6.
Christian Schüller, Daniele Panozzo, and Olga Sorkine-Hornung. 2014. Appearance-mimicking Surfaces. *ACM Trans. Graph.* (2014).
Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
Kenshi Takayama, Ryan Schmidt, Karan Singh, Takeo Igarashi, Tamy Boubekeur, and Olga Sorkine. 2011. Geobrush: Interactive mesh geometry cloning. In *Computer Graphics Forum*, Vol. 30. Wiley Online Library, 613–622.
Elif Tosun, Yotam I Gingold, Jason Reisman, and Denis Zorin. 2007. Shape optimization using reflection lines. In *Proc. SGP*.
Greg Turk. 1991. Generating textures on arbitrary surfaces using reaction-diffusion. *Siggraph* (1991).
Stefan Van der Walt, Johannes L Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. 2014. scikit-image: image processing in Python. *PeerJ* 2 (2014), e453.
CWAM Van Overveld. 1996. Painting gradients: Free-form surface design using shading patterns.. In *Graphics Interface*, Vol. 96. 151–158.
Li-Yi Wei and Marc Levoy. 2001. Texture synthesis over arbitrary manifold surfaces. *Siggraph* (2001).
Martin Weier, Michael Stengel, Thorsten Roth, Piotr Didyk, Elmar Eisemann, Martin Eisemann, Steve Grogorick, André Hinkenjann, Ernst Kruijff, Marcus Magnor, et al. 2017. Perception-driven Accelerated Rendering. In *Computer Graphics Forum*, Vol. 36. Wiley Online Library, 611–643.
Nathaniel Williams, David Luebke, Jonathan D Cohen, Michael Kelley, and Brenden Schubert. 2003. Perceptually guided simplification of lit, textured meshes. In *Proceedings of the 2003 symposium on Interactive 3D graphics*. ACM, 113–121.
Robert J Woodham. 1980. Photometric method for determining surface orientation from multiple images. *Optical engineering* 19, 1 (1980), 191139.
Chenglei Wu, Kiran Varanasi, Yebin Liu, Hans-Peter Seidel, and Christian Theobalt. 2011. Shading-based Dynamic Shape Refinement from Multi-view Video Under General Illumination. In *Proc. ICCV*.
Chenglei Wu, Michael Zollhöfer, Matthias Nießner, Marc Stamminger, Shahram Izadi, and Christian Theobalt. 2014. Real-time shading-based refinement for consumer depth cameras. *ACM Transactions on Graphics (TOG)* 33, 6 (2014), 200.
Julie C Xia and Amitabh Varshney. 1996. Dynamic view-dependent simplification for polygonal models. In *Proceedings of the 7th conference on Visualization'96*. IEEE Computer Society Press, 327–ff.
Li Xu, Cewu Lu, Yi Xu, and Jiaya Jia. 2011. Image smoothing via L 0 gradient minimization. In *ACM Transactions on Graphics (TOG)*, Vol. 30. ACM, 174.
Ruo Zhang, Ping-Sing Tsai, James Edwin Cryer, and Mubarak Shah. 1999. Shape-from-shading: a survey. *IEEE transactions on pattern analysis and machine intelligence* 21, 8 (1999), 690–706.
Wangyu Zhang, Bailin Deng, Juyong Zhang, Sofien Bouaziz, and Ligang Liu. 2015. Guided mesh normal filtering. In *Computer Graphics Forum*, Vol. 34. Wiley Online Library, 23–34.
Shuang Zhao. 2014. *Modeling and rendering fabrics at micron-resolution.* Cornell University.

A NEURAL STYLE IMPLEMENTATION DETAILS

In this section, for the purpose of describing the parameters we used for the image neural style transfer[Gatys et al. 2016], we briefly summarize its key ingredients. Because we use this application to introduce surface details rather than change the large-scale geometric appearance of an input we have to change some of the lighting and camera parameters used in *Paparazzi*.

The goal of image style transfer is to, given a content image I_c and a reference style image I_s, generate an I_t that both looks similar to I_c in the style of I_s. This is performed by finding the image that minimizes an energy E_{NSE}, which is defined as a weighted sum of a content energy and style energy, which are in turn defined on the feature maps (i.e activations) in each layer of a convolutional neural network (CNN):

$$E_{\text{NSE}} = w \sum_{\ell} \alpha_\ell E^{\ell}_{\text{content}} + (1-w) \sum_{\ell} \beta_\ell E^{\ell}_{\text{style}},$$

where w controls the weighting between content and style energies, α_ℓ, β_ℓ correspond to content and style weights for layer ℓ. The feature maps of I_t, I_c, I_s at layer ℓ of the CNN are denoted by $\mathbf{T}^\ell, \mathbf{C}^\ell, \mathbf{S}^\ell \in \mathbb{R}^{N_\ell \times M_\ell}$ respectively, where N_ℓ is the number of features in the layer ℓ and M_ℓ is the size of a feature image on the ℓ^{th} layer.

$E^{\ell}_{\text{content}}$ is the squared difference between the feature maps of the target and content images at layer ℓ:

$$E^{\ell}_{\text{content}} = \frac{1}{2} \sum_{i=1}^{N_\ell} \sum_{j=1}^{M_\ell} (T^\ell_{ij} - C^\ell_{ij})^2.$$

E^{ℓ}_{style} is squared error between the features correlations expressed by the *Gram* matrices over features G of content and style images at layer ℓ:

$$E^{\ell}_{\text{style}} = \frac{1}{4N_\ell^2 M_\ell^2} \sum_{i=1}^{N_\ell} \sum_{j=1}^{N_\ell} \left((G_t)^\ell_{ij} - (G_s)^\ell_{ij} \right)^2,$$

$G^\ell \in \mathbb{R}^{N_\ell \times N_\ell}$ for a feature map F^ℓ is explicitly written as

$$G^\ell_{ij} = \sum_{k=1}^{M_\ell} F^\ell_{ik} F^\ell_{jk}.$$

Both the style of strokes and colors contribute to the image style energy E_{NSE} but we omit the color style transfer because our focus is on the 3D geometry, instead of texture. In particular, we capture the style of strokes from a painting by perturbing vertex positions in order to create the shading changes required to decrease the style energy.

To eliminate the influence of the color component, we make both the style image and the rendered image gray-scale, we use one white directional light along the y-axis of the camera space to render the geometry. We also use offset $\sigma = 0.02$ and field of view of 0.1, OpenGL canonical view volume, to *zoom in* the mesh in order to generate surface details rather than affect the large-scale geometric features. We set α_ℓ, β_1 to be 1 for layer *conv1_1, conv2_1, conv3_1, conv4_1, conv5_1* and 0 for the other layers in the VGG network [Simonyan and Zisserman 2014]. We omit the content energy by setting $w = 0$. In our experiments, we can still transfer the style without losing the original appearance.