Seamless:
Seam erasure and seam-aware decoupling of shape from mesh resolution

SONGRUN LIU* and ZACHARY FERGUSON*, George Mason University
ALEC JACOBSON, University of Toronto
YOTAM GINGOLD, George Mason University

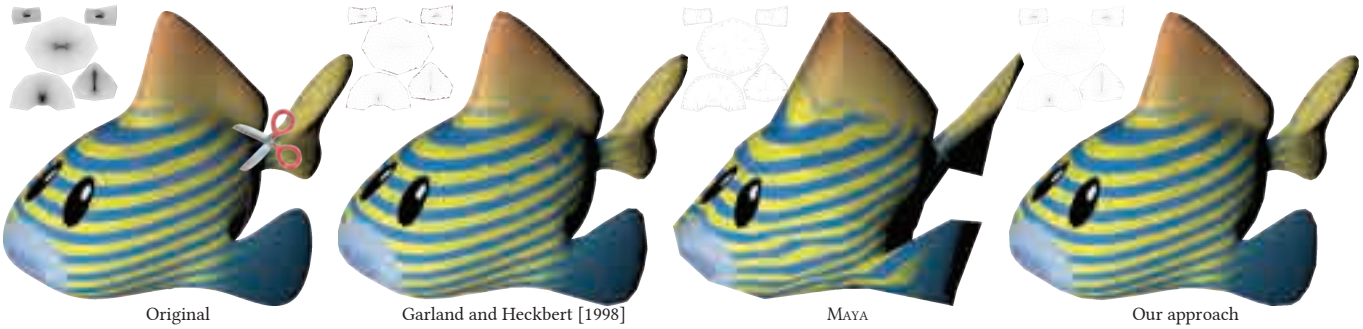


Fig. 1. Our seam-aware decimation allows seamless texture reuse at all decimation levels (here, approximately 1%). Seams on the original model are shown in purple. Parameterizations are shown inset. Garland and Heckbert [1998] (implemented by MeshLab [Cignoni et al. 2008]) do not preserve seams precisely, leading to artifacts in the texture. Red areas near seams in the inset parameterization indicate this deviation in the parametric domain. Maya [2017] prevents decimation of seams entirely, leading to suboptimal allocation of mesh vertices.

A parameterization decouples the resolution of a signal on a surface from the resolution of the surface geometry. In practice, parameterized signals are conveniently and efficiently stored as texture images. Unfortunately, *seams* are inevitable when parametrizing most surfaces. Their visual artifacts are well known for color signals, but become even more egregious when geometry or displacement signals are used: cracks or gaps may appear in the surface. To make matters worse, parameterizations and their seams are frequently ignored during mesh processing. Carefully accounting for seams in one phase may be nullified by the next. The existing literature on seam-elimination requires non-standard rendering algorithms or else overly restricts the parameterization and signal.

We present seam-aware mesh processing techniques. For a given fixed mesh, we analytically characterize the space of seam-free textures as the null space of a linear operator. Assuming seam-free textures, we describe topological and geometric conditions for seam-free edge-collapse operations. Our algorithms eliminate seam artifacts in parameterized signals and decimate a mesh—including its seams—while preserving its parameterization and seam-free appearance. This allows the artifact-free display of surface signals—color, normals, positions, displacements, linear blend skinning weights—with the standard GPU rendering pipeline. In particular, our techniques enable crack-free use of the tessellation stage of modern GPU's for dynamic level-of-detail. This decouples the shape signal from mesh resolution in a manner compatible with existing workflows.

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

Additional Key Words and Phrases: seams, texture mapping, mesh, decimation, simplification, detail, deformation, skinning, animation

ACM Reference Format:
Songrun Liu, Zachary Ferguson, Alec Jacobson, and Yotam Gingold. 2017. Seamless: Seam erasure and seam-aware decoupling of shape from mesh resolution. *ACM Trans. Graph.* 36, 6, Article 216 (November 2017), 15 pages. https://doi.org/0000001.0000001_2

1 INTRODUCTION

A 3D surface's parametrization by two coordinates naturally affords defining functions or *signals* that map points on the surface to scalar or vector values. In practice, a triangle mesh parameterized into the unit square can define a high-resolution signal (e.g., color values) at points on its 3D surface by reading the pixel values of an image at corresponding parametrization coordinates. Parameterization not only decouples the triangle mesh resolution from the signal image resolution, but also provides a bridge between levels of detail approximating the same underlying surface (see Figure 1). If two different resolution meshes of the same surface agree on a parameterization to the plane, then a signal designed for one mesh is readily defined upon other.

To parameterize most interesting 3D surfaces, it is necessary to make *cuts* to avoid foldovers or high distortion. The cuts are discontinuities in the parameterization. They unnecessarily create discontinuities in the surface signal. The *seams* where cuts meet on the 3D surface cause well known visual artifacts for color signals.

*Joint first authors

Authors' addresses: Songrun Liu and Zachary Ferguson and Yotam Gingold, George Mason University; Alec Jacobson, University of Toronto.

0730-0301/2017/11-ART216
https://doi.org/0000001.0000001_2

Indeed, 3D modeling professionals will attempt to hide seams along sharp signal features or in low-visibility regions. Seam artifacts are especially noticeable for geometric signals, where discontinuities produce cracks, gaps, or self-intersections in the 3D surface (see Figure 2).

Seam-related difficulties limit the use of geometric signals. Meanwhile, many (if not most) mesh processing techniques do not account for or track a mesh's parameterization much less its parametric seams. Due to this vicious cycle, downstream phases typically do not reap the potential benefits of parametrized geometric signals.

We propose a virtuous circle: a suite of seam-aware mesh processing techniques. Our suite of algorithms decouple mesh resolution from the complexity of the surface's signals. Our **contributions**, each of which can be used independently, are:

- **Seam erasure** (Section 4), an algorithm to analytically define the space of seam-free textures for a given parameterized mesh. This space is always the non-empty null space of a linear operator. We apply this as a constraint to erase seams from existing textures by solving a sparse, linear system of equations. We do not modify the parameterization or mesh to obtain a seamless textures.
- **Seam-aware decimation** (Section 5): We describe the space of seam-free edge collapse operations, including seam decimation criteria and optimization placement constraints. Our mesh decimation allows the same texture to be used across all decimation levels—notably along seams.
- **Seam straightening** (Section 6): The seam edge collapse criteria may be violated often for a given mesh, preventing significant decimation along the seams. We describe a one-time preprocessing algorithm to modify a mesh's parameterization to "straighten seams," allowing many more seam edges to be collapsed.

The *combination* of these contributions enables (i) a generalization of geometry images to dynamic, deforming shapes (ii) that runs on the standard, hardware-supported graphics pipeline (iii) without auxiliary data structures, non-standard sampling, or taking full control of the parameterization.

We demonstrate our seam erasure on a variety of surface signals: color, normals, ambient occlusion maps, geometry, and skinning weights. We apply our mesh decimater to generate level-of-detail meshes, all of which can re-use the same seam-free surface signals. We further leverage this decoupling of surface signal from mesh resolution with straightforward, crack-free subdivision in the tessellation stage of modern GPU's. We show, for the first time, adaptively deforming surfaces with skinning *weight maps*. Weight maps allow artists to 3D paint skin weights at a high resolution decoupled from the mesh.

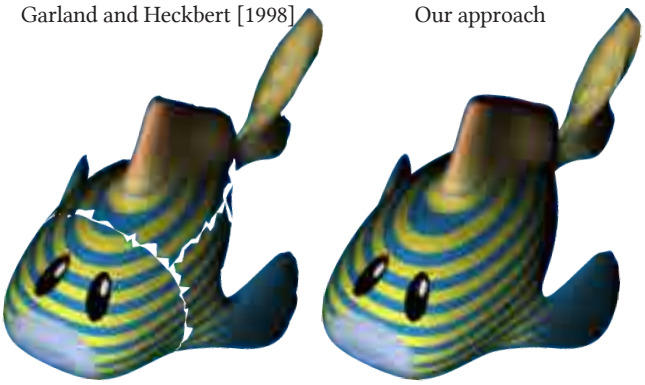


Fig. 2. When the parameterized signal encodes geometry, discontinuities manifest as cracks due to out of range sampling (the red areas in Figure 1, inset). Garland and Heckbert's approach exhibits cracks, because their decimation does not operate in the space of seam-free edge collapses. In contrast, our seam-aware decimation prevents such discontinuities.

2 RELATED WORK

Our contributions involve rendering, mesh decimation, and texture mapping, topics nearly as old as computer graphics.

Textures. Texture memory on the GPU has been exploited far beyond its original use for storing sampled color information. We briefly review methods storing *geometric* information in a texture map. The most common geometric information stored in texture memory is a normal map or displacement map. Displacement mapping is well-established technique in computer graphics [Blinn 1978; Cook 1984]. While displacement mapping is widely used, especially for subdivision and parametric surfaces in non-real-time settings [Guskov et al. 2000; Lee et al. 2000], it has been less widely used (apart from *normals*) in real-time rendering scenarios due to GPU limitations. Recently, however, geometry and tessellation shaders have enabled dynamic geometry generation on the GPU, leading to a surge of interest in the literature [Jang and Han 2012; Loop et al. 2009; Nießner et al. 2016; Nießner and Loop 2013; Schäfer et al. 2012; Szirmay-Kalos and Umenhoffer 2008; Tatarchuk et al. 2010]. One can also store the mesh's geometric immersion by writing the coordinate functions into the texture as in Geometry Images [Gu et al. 2002]. All of these works spend considerable effort to eliminate seam artifacts *by construction*, such as constraining the parameterization to one or more regular rectilinear patches and adding auxiliary data structures. Our seam erasure (Section 4) eliminates seam artifacts with an offline preprocess that projects a texture into the space of seam-free textures; no changes are needed at render-time. Moreover, creating a high-quality displacement map to turn a coarse mesh into an existing high-resolution one can be challenging without a bijection [Nießner et al. 2016]. Ray casting along the normal direction and closest point sampling have common failure modes. Cohen et al. [1998] show that decimation can be used to create a natural bijection via the parametric domain, enabling the generation of perfect vector-valued displacements *away from seams*. In Section 5, we present conditions for decimation algorithms to prevent seam discontinuities. Our algorithms allow for a generalization of Geometry Images supporting arbitrary (bijective) parameterizations, enabling more natural and lower distortion parameterizations and eliminating the need for special seam handling in the *output*.

Seams. Many attempts have been made to tackle the problem of seams, or texturing discontinuities that arise when parameterizing non-disk-like surfaces. Most previous approaches either modify the rendering pipeline (altering sampling [Toth 2013] or blending multiple textures [Piponi and Borshukov 2000]), take complete control of the parameterization (generating rectangular patches or seams which are 90° rotations and integer offsets) [Aigerman et al. 2015; Bright et al. 2017; Kovalsky et al. 2016; Myles and Zorin 2013; Nießner and Loop 2013; Purnomo et al. 2004; Ray et al. 2010; Sheffer and Hart 2002], or avoid parameterizations entirely [Loop et al. 2009; Yuksel et al. 2010]. Our work is orthogonal to the choice of the original parameterization and stands to make this family of work more useful. A seamless parameterization does not provide seamless texture values or collapsible seams. A surface signal is undefined in texels outside the parameterization. Practical considerations, such as designer-specified adaptive resolution or seam layout, manifold mesh or genus restrictions, simulation-quality element requirements, and performance limitations, often preclude the use of global parameterization techniques in many workflows. A seamless parameterization's integer offsets are also insufficient when multiple texture resolutions are used for the same model or the texture resolution is chosen later. The closest approach to ours was developed in industry and briefly described by Iwanicki [2013], who also optimize texture values; they penalize discontinuity at finite sample points along seams and L_2 deviation from the input texture. We share the motivation to operate on existing parameterizations and leave the rendering pipeline unchanged (pay no runtime cost). In our experiments, their solutions exhibit non-smoothness and artifacts near corners (see Figure 7). In contrast, we provide a closed form expression for the space of seam-free textures and impose its zero-energy null space as a linear constraint to completely eliminate seam artifacts while minimizing value change and non-smoothness. Our solution allows for a global, gradient domain term. This is important because artists often create textures with a global mismatch across seams. Our energy has as a special case the "constant value" seam corner handling of Nießner et al. [2013] or the equivalence classes of Ray et al. [2010]. Setting texels along the seam to be rotated or reflected copies of each other requires specially generated parameterizations; we show that this is more strict than necessary.

Decimation. Previous works on mesh decimation either fix all seam edges, introduce seam discontinuities by allowing adjacent straight seam edges to collapse, or do not preserve the uv-space boundary at all. The seminal Quadric Error Metrics approach of Garland and Heckbert [1997], often referred to as QSlim, serves as a widely used de facto benchmark but does not handle surface attributes other than 3D position. A follow-up work considers multi-dimensional surface attributes in the same quadric metric framework [Garland and Heckbert 1998]. Hoppe [1999] proposed several quality and efficiency improvements to this quadric metric and modifications to handle attribute discontinuities such as sharp normals. While these works put mesh decimation and multi-dimensional surface attributes on very solid theoretical footing, Garland and Heckbert do not address the issue of bijectivity or directly address discontinuities. Indeed, implementations of Garland and Heckbert in practice [Cignoni et al. 2008] do not preserve seams or bijectivity, important conditions for texture reuse. Hoppe fixes the geometry of attribute discontinuities such as seams completely. In contemporary work, Cohen et al. [1997; 1998] develop bijectivity conditions and a parameterization-preserving decimation algorithm; they assume rectangular parametric boundaries and either fix all parametric boundaries or allow any collinear edges to collapse. Allowing collinear seam edges to collapse while maintaining collinearity may still introduce seam artifacts (Section 5, Figure 8). Also contemporaneously, Lee et al. [1998] obtain a bijection between coarse and fine surfaces, though without UV parameterization. Sander et al. [2002; 2001] introduced techniques for making effective use of texture resolution and sharing the same texture among all levels of detail; they do not address seams. We are similarly motivated to store high resolution surface signals in textures in a manner decoupled from mesh resolution. Our contribution is decimation criteria and constraints (Section 5) for collapsing seam edges while preserving seamless texture mappings. Our criteria and constraints are agnostic to the decimation approach. In Section 6, we describe an approach to adjust a parameterization to increase the number of seam edges that can be decimated.

Complementing to these methods, previous works have considered optimal decimations of an input mesh given knowledge of its probable deformations. Mohr and Gleicher consider a static simplification of a mesh given frames of an animation sequence [2003]. Huang et al. extend this to a simplification with dynamically changing topology [2006]. These approaches could be added as edge costs to our approach.

Adaptive meshing. Dynamically deforming meshes found in computer animation immediately invite adaptivity. In the simplest form, instead of defining the space deformation with displacements of regularly sampled lattice points (cf. Eulerian simulation), the grid adapts spatially to the complexity of the deformation (e.g. via octree subdivision [Botsch et al. 2007]). Instead, our use of the tessellation shader better utilizes existing real-time graphics hardware.

We postulate that cracks along seams like those in Figure 2 are an important reason that the tessellation shader is used infrequently in video game graphics engines. The tessellation shader in the OpenGL pipeline activates after the vertex shader for each (triangle) primitive. The shader can control the over-tessellation (subdividision) of the triangle and move resulting vertices before rasterization and fragment shading. This has been exploited in the past for view-dependent subdivision of coarse meshes into smooth surfaces [Boubekeur and Alexa 2008] or approximations of subdivision limit surfaces [Boubekeur 2010]. For example, approximating a subdivision limit surface via the tessellation shader provides a simple way to create high-resolution deformations: move the control mesh on the CPU or with the vertex shader (e.g., with traditional skinning) and then subdivide the deformed control cage within the tessellation shader [Holländer and Boubekeur 2010; Nießner et al. 2012]. However, the deformation is limited by the resolution of the skinning weights defined at control cage vertices. Moreover, deforming a subdivision cage with skinning and *then subdividing* to produce the smooth (approximate) limit surface can deviate arbitrarily from applying skinning as a pointwise deformation to the rest pose limit surface

[Liu et al. 2014]. Liu et al. propose an optimization of the deforming control cage vertices so that their limit surface follows the pointwise skinning rest-pose limit surface. We do not assume a particular subdivision paradigm and treat the input mesh after displacement offsets have been applied or geometry image as the underlying surface to be deformed pointwise. When deforming, our tessellation shader subdivides incoming triangles based on the deviation of their linearly deforming surface from their ideal pointwise skinning (nonlinearly deforming) surface, allowing flat triangles in the rest pose to curve and bend in space (Figure 18).

3 SEAMS

Seam edges in a triangulated 3D surface mesh are edges whose incident "face flaps" become combinatorially disconnected along the edge in the u,v parameterization, typically to different geometric locations (Figure 3). From the point of view of the 3D surface mesh, seams are edges that have been "cut" and correspond to multiple edges on the parametric domain (two for manifolds). From the point of view of the 2D parametrized mesh in uv space, seams are edges glued together in the 3D surface mesh.

Bilinear reconstruction is the standard technique (implemented in GPU hardware) for reconstructing a signal at any parametric point uv in a texture. Viewing the texture samples or *texels* as lying at the intersections of grid lines, it is defined as the bilinear interpolation of the four corners $p_{00}, p_{10}, p_{01}, p_{11}$ of the grid cell in which uv lies (Figure 3, left):

$$p_{00} + t(p_{01} - p_{00}) + s(p_{10} - p_{00}) + st(p_{00} - p_{10} - p_{01} + p_{11}) \quad (1)$$

where $s, t \in [0, 1]$ are the coordinates of uv within the grid cell.

For interior points far from seams, the bilinear reconstruction is unambiguous and continuous. Along a seam edge, however, the bilinear reconstruction will sample different texels depending upon which incident faces are used for the parametrization. When the two reconstructions do not match, the seam is visible as a discontinuity (Figures 4 and 5). This discontinuity manifests as cracks when the signal encodes geometry (e.g., geometry images or displacement maps, see Figure 2).

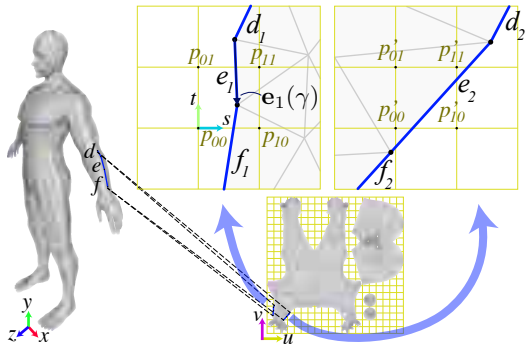


Fig. 3. A seam edge e on the 3D surface mesh corresponds to two half-edges e_1 and e_2 in the u, v parameterization. Similarly, for other edges d, f. We use boldface $\mathbf{e}_1(\gamma)$ to refer to the vector-valued function mapping a relative 1D position along on the edge to its half-edge positions along e_1 in u, v space. The half-edges e_1 and e_2 will pass through (possibly different) texels where signals stored at corners (p_{00}, p_{01}, etc.) are bilinearly interpolated.

Let us formalize seams and their associated issues. For every manifold edge on the surface mesh e, there are two functions $\mathbf{e}_1(\gamma)$ and $\mathbf{e}_2(\gamma)$ mapping a parameter $\gamma \in [0, 1]$ along the edge to uv coordinates by linearly interpolating texture coordinates stored at the half-edge endpoints (also known as *wedges*). See Figure 3. For non-seam edges, $\mathbf{e}_1 = \mathbf{e}_2$. For seam edges, these functions will in general disagree as they map seam half-edge e_1 and e_2 to different locations in the uv parameterization. Each half-edge e_i may pass through multiple texel-grid cells. In each grid cell, bilinear interpolation can be expressed as a function that is quadratic in γ and *linear* in the four relevant nodal values p_{kl}:

$$B(\mathbf{e}_i(\gamma), \mathbf{p}) = \gamma^2 \mathbf{a}_i(\gamma)^T \mathbf{p} + \gamma \mathbf{b}_i(\gamma)^T \mathbf{p} + \mathbf{c}_i(\gamma)^T \mathbf{p}, \quad (2)$$

where $\mathbf{p}$ is the (column) vector of the four samples and $\mathbf{a}_i(\gamma), \mathbf{b}_i(\gamma), \mathbf{c}_i(\gamma)$ are (column) vectors whose entries are constant functions of γ when restricted to this grid cell; these functions map the uv endpoints $e(0)$ and $e(1)$ to the appropriate cell s, t values in Equation 1. Equivalently, we choose to think of $\mathbf{p}$ in Equation 2 as the (column) vector of *all* samples in the texture and $\mathbf{a}_i(\gamma), \mathbf{b}_i(\gamma), \mathbf{c}_i(\gamma)$ as *sparse*, piecewise-constant (column) vectors with four non-zero elements (at the relevant indices in $\mathbf{p}$). In this view, B is piecewise quadratic; it is quadratic in each interval for which $\mathbf{e}_i(\gamma)$ lies within a single grid cell.

Seam artifacts occur when bilinear interpolation along the half edges of an edge (a seam edge) produce different values. We can measure the total discontinuity along a seam edge as

$$D(\mathbf{e}_1, \mathbf{e}_2) = \int_0^1 |B(\mathbf{e}_1(\gamma), \mathbf{p}) - B(\mathbf{e}_2(\gamma), \mathbf{p})|^2 \, \mathrm{d}\gamma. \quad (3)$$

We can express D in matrix form as

$$D(\mathbf{e}_1, \mathbf{e}_2) = \mathbf{p}^T \left(\int_0^1 \mathbf{m}(\gamma)^T \mathbf{m}(\gamma) \mathrm{d}\gamma \right) \mathbf{p} \quad (4)$$

where $\mathbf{m}(\gamma) = \gamma^2(\mathbf{a}_1 - \mathbf{a}_2) + \gamma(\mathbf{b}_1 - \mathbf{b}_2) + (\mathbf{c}_1 - \mathbf{c}_2)$.

To analytically evaluate the definite integral, we find the sorted parameters $\gamma_{i \ldots k} = \{0, \ldots, 1\}$ where $\mathbf{e}_1(\gamma)$ or $\mathbf{e}_2(\gamma)$ cross a grid cell boundary. We break up the integration to obtain

$$D(\mathbf{e}_1, \mathbf{e}_2) \quad (5)$$

$$= \mathbf{p}^T \left(\sum_{i=1}^{k-1} \int_{\gamma_i}^{\gamma_{i+1}} \mathbf{m}(\gamma)^T \mathbf{m}(\gamma) \mathrm{d}\gamma \right) \mathbf{p} \quad (6)$$

$$= \mathbf{p}^T \left(\sum_{i=1}^{k-1} M_i \right) \mathbf{p} \quad (7)$$

$$= \mathbf{p}^T M_{\mathbf{e}_1, \mathbf{e}_2} \mathbf{p}, \quad (8)$$

where each M_i is a sum of constant matrices scaled by monomials of γ, $\int_{\gamma_i}^{\gamma_{i+1}} \gamma^i \mathrm{d}\gamma$, for $i = 1, 2, 3, 4$. These integrals have trivial closed form solutions, and so we obtain an analytic, closed form expression for D, the seam discontinuity. Importantly, D is quadratic in the texels $\mathbf{p}$.

The following two sections answer the following two questions. For a given fixed mesh, what is the space of seam-free textures

(Section 4)? Assuming seam-free textures, what is the space of seam-free mesh decimations (Section 5)?

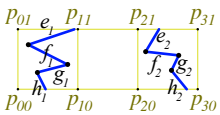
4 TEXTURE SEAM ERASURE

For a given mesh, we can express the space of seam-free textures as those for which integrated seam discontinuity in Equation 5 over all seam-edges is zero:

$$D_{total} = \sum_{\mathbf{e}_1,\mathbf{e}_2 \in \text{seams}} D(\mathbf{e}_1,\mathbf{e}_2) = \mathbf{p}^T \Big(\sum_{\mathbf{e}_1,\mathbf{e}_2 \in \text{seams}} M_{\mathbf{e}_1,\mathbf{e}_2} \Big) \mathbf{p} = \mathbf{p}^T M\, \mathbf{p}.$$

Our texture seam erasure eliminates seams by adjusting texel values $\mathbf{p}$. The space of seam-free textures is the null space of M. This is because $D_{total} = 0$ if and only if $M\,\mathbf{p} = 0$. [1] Computing M depends only on the parameterization of the mesh seams and the resolution of the texture.

This null space is guaranteed to be infinite; in particular, it contains all constant vectors. In our results, the curves we observe along seam edges are quite complex. Empirically, the null-space dimensionality (for pixels in bilinear cells the seam passes through) ranges from the hundreds to the thousands, or approximately 6–30% of the total dimensionality (number of seam pixels). For example, Hercules in a 512×512 texture has a 7130 pixel seam and nullity 586, where nullity is the number of seam pixels minus the rank of M.[2] The Lemon in a 512×512 texture has 4053 seam pixels and nullity 1183. The Teapot in a 256×256, 512×512, and 1024×1024 texture has 5274, 10641, and 21345 pixels along its seam and nullity 347, 959, and 2308, respectively. The Cow in a 512×512 texture has 8101 seam pixels and nullity 657. The null space contains solutions previously proposed in the literature (equivalence classes [Ray et al. 2010], 90° rotations [Aigerman et al. 2015; Myles and Zorin 2013], and translations [Nießner and Loop 2013]) as special cases. Bilinear interpolation defines a particular bivariate quadratic function in each grid cell. Each term M_i in Equation 7 only considers values along lines, so the constraint that ties seam halfedges to each other imposes a particular shared univariate quadratic function.



As the texture resolution decreases, more and more vertices of a seam may lie within a single bilinear cell. When a chain of four edges (separated by three vertices) lies in general position within a single cell (inset left), the null space locally reduces to a single dimension, constant functions. With chains of one, two, and three edges, the null space locally has five, three, and two dimensions, respectively. See Figures 6 and 17 for more complex examples with very low texture resolution.

In practice, because we impose $M\,\mathbf{p} = 0$ with a penalty method, we normalize M to avoid having to adjust weights for different length seams or texture sizes. To do so, we use

$$M_{total} = \frac{1}{L} \sum_{\mathbf{e}_1,\mathbf{e}_2 \in \text{seams}} l_e M_{\mathbf{e}_1,\mathbf{e}_2}. \tag{9}$$

[1] It is obvious that $D_{total} = 0$ if $M\,\mathbf{p} = 0$. In the other direction, since M is positive semi-definite, there exists an A such that $M = A^T A$. Hence, $D_{total} = \mathbf{p}^T M\,\mathbf{p} = \mathbf{p}^T A^T A \mathbf{p} = |A\mathbf{p}|^2 = 0$ implies that $M\,\mathbf{p} = A^T A \mathbf{p} = 0$.

[2] We compute the rank via NumPy's `matrix_rank()` function, which employs a widely used singular value threshold.

					D_{total}		
Example	Mode	Texture Size	Channels	Seam Edges	Before	After	Runtime
Animal (Diffuse)	local	640×640	4	835	0.009	$1 \cdot 10^{-10}$	33
Chimp (Diffuse)	local	640×640	3	1122	0.003	$4 \cdot 10^{-11}$	25
Cow (Displacement Map)	local	1024×1024	3	2441	3.744	$5 \cdot 10^{-8}$	65
Cow (Normal Map)	global	2048×2048	3	2441	0.078	$1 \cdot 10^{-10}$	227
Cow (Weight Map)	local	1024×1024	15	2441	0.072	$1 \cdot 10^{-11}$	71
Boy (Diffuse)	local	1024×1024	4	4802	0.056	$2 \cdot 10^{-09}$	234
Hercules (Ambient Occlusion)	local	256×256	4	1098	0.056	$6 \cdot 10^{-09}$	31
Lemon (Diffuse)	local	2048×2048	3	98	0.0003	$2 \cdot 10^{-07}$	431
Lemon (Normal Map)	global	2048×2048	3	98	0.031	$2 \cdot 10^{-07}$	408
Teapot (Ceramic)	local	992×992	3	954	0.056	$8 \cdot 10^{-10}$	56
Teapot (Ceramic)	global	992×992	3	954	0.056	$7 \cdot 10^{-11}$	56
Wolf (Weight Map)	local	1024×1024	20	653	0.068	$1 \cdot 10^{-11}$	58

Table 1. Performance information for our seam erasure. Runtime is measured in seconds. Please see the text for details.

where, for normalization, l_e is the 3D edge length of e and L is the sum of all such edge lengths. This does not affect the null space.

4.1 Choosing a solution

To choose a solution from the null space, we balance energy terms measuring the deviation from the input texture and smoothness across the seam. We briefly note that minimizing the deviation of interior texels (E_{change} below) subject to the null space constraint results in an under-constrained system.

Deviation. Given an input texture, we consider texels in the interior of the parameterization to be reliable. Texels outside the parameterization are, in general, completely unreliable. For example, a texture obtained by rasterizing surface data into the *uv* mesh will record no values outside. However, we also account for scenarios with additional information: surface data may be stored on mesh vertices (e.g. 3D painting or finite element solution), and therefore its linear interpolation along seam edges is known; artists may create the texture in 2D and deliberately set outside texels near the seam. Finally, note that not all *uv* mesh edges bordering outside texels (we call these *uv silhouette edges*) are seam edges; 3D surface boundary edges and *uv*-space foldovers are also *uv* silhouette edges. They can produce artifacts, since the signal along such edges will be bilinearly reconstructed using texels inside and outside the *uv* mesh. For example, in Figure 3, the circles in the parameterization are spheres mapped to themselves as discs. We consider all *uv* silhouette edges to be "seams" for our decimation (Section 5).

We express the reliability of interior texels (denoted as the set T_{in}) in the primary and gradient domains. We add a primary domain term,

$$E_{change}(\mathbf{p}) = \frac{1}{|T_{in}|} \sum_{i \in T_{in}} \|\mathbf{p}[i] - \mathbf{p}_0[i]\|^2,$$

where $\mathbf{p}_0$ are the initial texel values. We add a gradient domain term to allow for global effects resulting from enforcing seam continuity. For example, color or normal maps designed in 2D often have unintended non-local changes across seams. The gradient domain term we add is

$$E_{\nabla in}(\mathbf{p}) = \|\nabla_{in}\mathbf{p} - \nabla_{in}\mathbf{p}_0\|^2,$$

where ∇_{in} is the discretized 2D gradient operator restricted to interior texels T_{in}. We allow users to choose the balance between the primary and gradient domain terms as appropriate. In practice, we offer two parameter settings, given below.

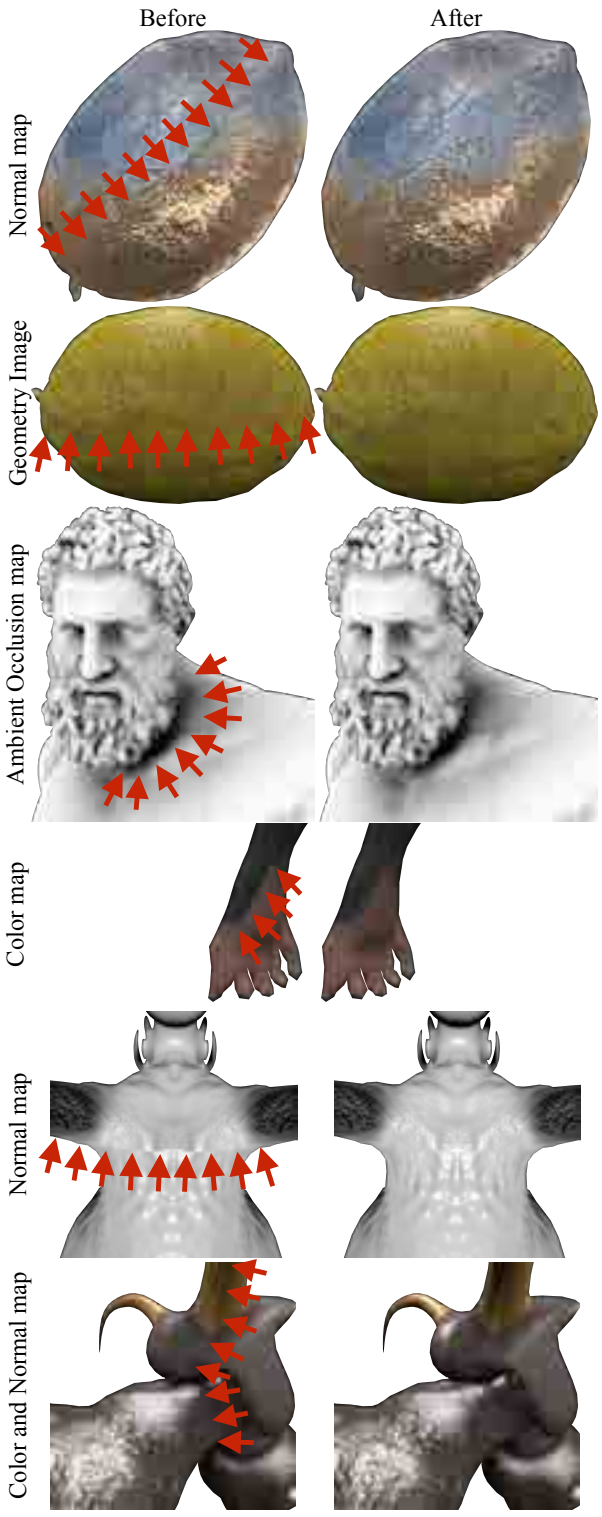


Fig. 4. Our texture seam erasure eliminates seam discontinuities from textures encoding any signal (colors, normals, geometry, ambient occlusion). Backfaces, visible through cracks in the input geometry image example, are rendered in red. Normal maps and the cow color map's seams were erased using the global settings (Section 4).

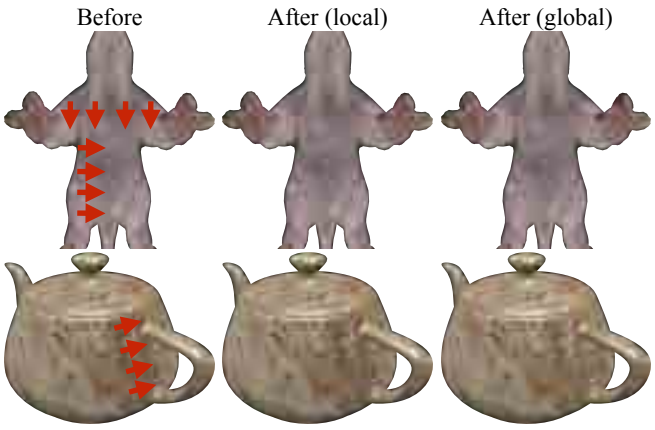


Fig. 5. Seam erasure can remove the discontinuity resulting from mismatched bilinear interpolation, but global discontinuities in the signal may remain. This typically occurs when textures are painted directly in the parametric domain. Seam erasure in the gradient domain has a more global effect.

When given additional information by the user, either linearly interpolated values along seam edges or reliable texels outside the seam, we add an appropriate term to the energy. For linearly interpolated values x_0 and x_1 along an edge e, we minimize the integrated squared deviation of the seam from the linear interpolated function,

$$E_{lerp}(\mathbf{e}_i, x_0, x_1) = \int_0^1 |B(\mathbf{e}_i(\gamma), \mathbf{p}) - (\gamma x_0 + (1-\gamma)x_1)|^2 \, \mathrm{d}\gamma.$$

For all edges, $E_{lerp}(\mathbf{p}) = \frac{1}{L} \sum_{\mathbf{e}_i, x_0, x_1 \in \text{silhouettes}} l_e E_{lerp}(\mathbf{e}_i, x_0, x_1)$. When texels outside the seam are considered reliable, we minimize the integrated squared deviation of the seam from its original function

$$E_{outside}(\mathbf{e}_i) = \int_0^1 |B(\mathbf{e}_i(\gamma), \mathbf{p}) - B(\mathbf{e}_i(\gamma), \mathbf{p}_0)|^2 \, \mathrm{d}\gamma.$$

Summing all edges, $E_{outside}(\mathbf{p}) = \frac{1}{L} \sum_{\mathbf{e}_i \in \text{silhouettes}} l_e E_{\nabla_{out}}(\mathbf{e}_i)$. We do not discretize these energy terms; we analytically integrate them. The derivation is similar to that of D and results in a quadratic energy in $\mathbf{p}$.

Smoothness. For smoothness, we introduce an energy term that measures the integrated C^1 discontinuity across seams. Namely, the directional derivative of B perpendicular to each seam edge (towards the interior) should have opposite sign:

$$E_{C^1}(\mathbf{e}_1, \mathbf{e}_2) = \int_0^1 \left| \nabla_{u,v} B(\mathbf{e}_1(\gamma), \mathbf{p}) \cdot \frac{\mathbf{e}_1^\perp}{\|\mathbf{e}_1\|} + \nabla_{u,v} B(\mathbf{e}_2(\gamma), \mathbf{p}) \cdot \frac{\mathbf{e}_2^\perp}{\|\mathbf{e}_2\|} \right|^2 \mathrm{d}\gamma.$$

Summing all edges, $E_{C^1}(\mathbf{p}) = \frac{1}{L} \sum_{\mathbf{e}_1, \mathbf{e}_2 \in \text{seams}} l_e E_{C^1}(\mathbf{e}_1, \mathbf{e}_2)$. As before, we analytically integrate E_{C^1} to obtain a quadratic energy in $\mathbf{p}$.

Finally, to account for any remaining degrees of freedom—in particular, non-seam uv silhouette edges trivially satisfy or are underconstrained by $D = 0$, E_{change}, $E_{spatial}$, and E_{C^1}—we add a

Fig. 6. Decreasing the resolution of a color texture while computing a seam-free texture. The original texture, whose seam runs underneath the handle, is 1024×1024. This example uses local smoothness parameters (see text for details). As the texture size decreases, the area of influence increases. The change eventually bleeds into the teapot's lid, a disconnected component. A seam never appears, though at 8×8 resolution, the texture becomes virtually constant.

Dirichlet energy involving only the outside texels of bilinear cells through which uv silhouette edges pass:

$$E_{\nabla out}(\mathbf{p}) = \|\nabla_{out}\mathbf{p}\|^2,$$

where ∇_{out} is the discretized 2D gradient operator restricted to grid edges involving the outside texels. (Texels that do not belong to bilinear cells containing the uv mesh are left untouched by our optimization.)

Optimization. All of our energy terms are quadratic energies in the texels $\mathbf{p}$. Our total energy is:

$$\begin{aligned} E(\mathbf{p}) = & \\ \text{(deviation)} \quad & w_{change}E_{change}(\mathbf{p}) + w_{\nabla in}E_{\nabla in}(\mathbf{p}) \\ \text{(smoothness)} \quad & + w_{C^1}E_{C^1}(\mathbf{p}) + w_{\nabla out}E_{\nabla out}(\mathbf{p}) \\ \text{(optional)} \quad & + w_{outside}E_{outside}(\mathbf{p}) + w_{lerp}E_{lerp}(\mathbf{p}) \end{aligned}$$

subject to

$$M\mathbf{p} = 0$$

We impose the null space constraint via the penalty method by adding $w_{seam}\mathbf{p}^T M\mathbf{p}$. This allows complex functions that have effectively zero energy. All examples use $w_{seam} = 10^{10}$, except the Lemon which used $w_{seam} = 10^6$ due to numerical issues. We obtain the global minimum of the resulting quadratic expression by solving the associated sparse linear system of equations. Multiple signal channels are optimized simultaneously as the system matrices are identical. We use the following weights in our experiments: $w_{change} = 10^4$; $w_{\nabla in} = w_{\nabla out} = 10^0$; $w_{C^1} = w_{lerp} = w_{outside} = 10^2$. If the user requests non-local (global) continuity across the seam, we decrease w_{change} to 10^2. This decreases the importance of the primary domain in favor of the gradient domain.

Experiments. In Figure 4, we erase seams from color, normal, [3] geometry, and ambient occlusion maps. The images are also available in the supplemental materials for inspection. (Seam erasure was used for all examples in the paper.) While seams are thin by nature, they are visually jarring artifacts on an otherwise smoothly varying surface. When textures are created in the parametric domain, global discontinuities are often unintentionally introduced. A comparison between our local and global parameter settings is shown in Figure 5.

[3]We use object-space rather than tangent-space normal maps in our experiments. Tangent space transformation matrices differ across half-edges and would need to be inserted into the definition of D, requiring numerical integration. However, there is no standard definition of tangent space transformation matrices; disagreement between the definition used by seam erasure and when rendering would reintroduce seam discontinuities.

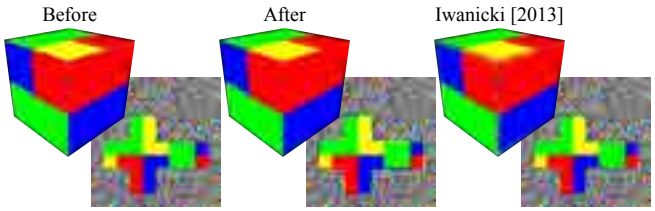


Fig. 7. Our algorithm erases seam mismatches without influence from mismatching or undefined texels outside the parameterization mesh. Minimizing bilinear interpolation mismatch and the change in interior texels results in an underconstrained system that cannot be solved. Minimizing the change in exterior texels as well [Iwanicki 2013] results in undesirable influence.

In Figures 6 and 17, we erase seams from progressively smaller texture maps (color and weight maps); a seam never appears, though the effect of seam erasure necessarily becomes more global and eventually a constant function as each texel influences a larger region. Skin weights textures (Section 8) as small as (16×16) are sufficient for many deformations, resulting in much less data storage than per-vertex weights.

Table 1 shows the running time and integrated seam deviation D_{total} before and after our optimization. Seam deviation is always decreased to approximately zero. Our implementation was written in Python/SciPy. We used a large, sparse, linear system and a direct solver to minimize our energy (SciPy's LU-decomposition-based `spsolve()`). Runtime numbers were obtained on a 2.6 GHz Intel Core i7-4720HQ CPU with 16GB of RAM. Optimization takes on the order of 30 seconds for a small example to 7 minutes for a 2k texture. This is a one-time preprocess, so it is not performance critical.

Figure 7 illustrates the effectiveness of our constrained optimization versus a simpler energy: minimizing the squared seam deviation at sample points and squared change in all texels [Iwanicki 2013]. Penalizing the change in texels outside the parameterization leads to unwanted influence from unreliable information. Not penalizing the change in outside texels leads to an underconstrained system. Iwanicki's approach erases seams when there are reasonable values outside the parameterization and only a minor touch-up is needed. In contrast, our approach can handle examples requiring global texture changes. Our optimization directly minimizes the analytic differences across seams, and chooses a solution from its numerical null space. Our cross-seam smoothness terms produce a properly determined system with pleasing results.

We experimented with a priori lexicographic multiobject optimization, in which less important terms (according to the weights w) are minimized subject to the constraint that more important terms achieve a best possible solution. We tried the approaches in de Lasa and Hertzmann [2009] and Kanoun et al. [2009] but found them to be numerically unstable. We were able to solve as a sequence of quadratically constrained quadratic programming problem using Mosek, but the quality of the solution was only marginally better than our weighted sum at a much higher computational cost.

5 DECIMATION

Our edge collapse conditions ensure that mesh decimation does not introduce new seams. Mesh decimation seeks to decrease the complexity of a mesh in the least obtrusive way possible. Decimation algorithms allow one to create simpler meshes for level-of-detail hierarchies. Approaches have been proposed for meshes with accompanying uv parameterizations [Cohen et al. 1998; Garland and Heckbert 1998; Hoppe 1999; Sander et al. 2002, 2001]. This allows decimated meshes to share the same surface signals stored efficiently in the parametric domain as images. For example, detailed surface colors, normals, or 3D positions can be stored as images. A bijective parameterization allows signal reuse. However, bijectivity can be destroyed by naive seam-oblivious decimation. We focus on edge-collapses, the fundamental operation in most popular triangle mesh decimation algorithms. The operation collapses an edge, destroying its triangle flaps and merging its vertex endpoints into a single new vertex. The edge-collapse rules proposed by existing approaches for decimating seams (and uv silhouettes in general) have either been too strict (leave seams untouched) or too permissive (preserve only the uv shape of seams or, worse, don't); see Figure 1. Leaving seams untouched prevents mesh complexity from being allocated where it will be more beneficial. Allowing seams to collapse freely introduces seam discontinuities, including sampling undefined texels. Preserving only the uv shape of seams can still introduce discontinuities due to mismatched sampling (Figure 8).

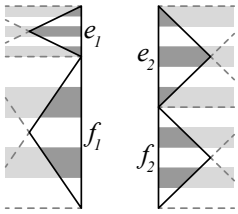


Fig. 8. Seam half-edges e_1, e_2 and f_1, f_2 are collinear, but merging $\overline{e_1 f_1}$ and $\overline{e_2 f_2}$ would cause the stripe texture to be misaligned across the seam.

In this section, we describe the space of seam-free edge collapses. We take as given that seam-free textures exist for which $D_{total} = 0$ (e.g. the output of Section 4). We describe criteria that must be satisfied to be able to collapse an edge without introducing a seam, and conditions that the new vertex's uv parametric coordinates must satisfy. Our conditions preserve the topology and shape of seams, preventing edge-collapses from "cutting corners," effectively omitting parts of the input signal (edge-collapses near convex boundaries) or spilling into unspecified regions not in the original surface's texture. Our conditions also prevent discontinuities due to mismatched sampling along the half-edges of a seam edge (Figure 8).

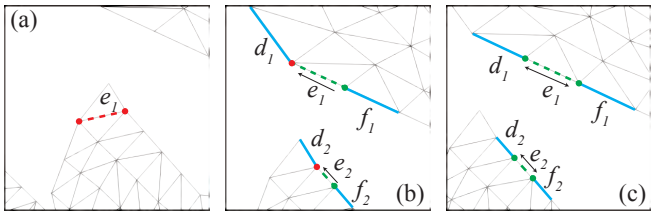


Fig. 9. Collapsible edge conditions. In all examples, e is the edge collapse under consideration. According to our criteria and conditions, green endpoints are free to move, while red endpoints must remain fixed. In (a), collapsing e would violate the Link Conditions. In (b), e and f are unifiable because they are collinear and satisfy the condition: $\frac{\|e_1\|}{\|f_1\|} = \frac{\|e_2\|}{\|f_2\|}$. However, e and d cannot be unified because they are not collinear. Therefore, when collapsing e, the only satisfying new vertex placement in uv is for the green endpoints to move to the location of the red ones; the xyz placements will be determined by constrained minimization of the quadric metric. In (c), e, f and e, d are unifiable, so both endpoints of e_1 and e_2 are free to move. The new vertex placement in uv and xyz will be determined by minimizing the quadric metric subject to collinearity constraints.

5.1 Criteria and Conditions for Collapsible Edges

Link Conditions: In any manifold-preserving mesh decimation algorithm, for an edge to be collapsible, its surface mesh must satisfy the link conditions [Dey et al. 1999] to guarantee that the resulting surface mesh is still manifold. In our setting, we additionally check that the parametric mesh satisfies the link conditions. This can occur if the edge to be collapsed is not a uv silhouette edge yet connects two uv silhouette vertices; such an edge is not collapsible.

One uv silhouette endpoint: If an edge has only one vertex on a uv silhouette, the collapsed edge must take the silhouette vertex's uv to preserve the shape of the silhouette.

Seam edges: For a seam edge to be collapsible, we must be able to unify both of its uv half-edges e_1, e_2 with the half-edges of at least one of its neighbors along the seam into a uniformly parameterized line. This is a stronger condition than uv collinearity, because it also requires that the ratio of uv edge lengths are the same: $\frac{\|e_1\|}{\|f_1\|} = \frac{\|e_2\|}{\|f_2\|}$, where f_1, f_2 are the half-edges of the adjacent seam edge. Collinearity preserves the shape of the seam. Matching uv edge lengths ensures that the collinear lines formed by $\overline{e_1 f_1}$ and $\overline{e_2 f_2}$ are similarly parameterized, so that merging them and reparameterizing them linearly results in the same bilinear interpolated signal reconstruction as before (Figure 8). Formally, the seam discontinuity D (Equation 2) after unifying the edges must remain identically zero. The edge length ratio condition arises from the change of variables of the integral.

There may be zero, one, or two adjacent unifiable seams edges. A seam edge with no such satisfactory adjacent seams cannot be collapsed (Figure 9).

If the seam edge e with uv half-edges e_1, e_2 is adjacent to two unifiable edges d_1, d_2 and f_1, f_2 (on either side), we impose uv collinearity and matching length ratios as linear equality and inequality constraints. Let $\overline{\mathbf{d}_1(0)\mathbf{e}_1(0)\mathbf{e}_1(1)\mathbf{f}_1(1)}$ be the sequence of uv endpoints for the half-edge sequence $\overline{d_1 e_1 f_1}$, and let $\overline{\mathbf{d}_2(0)\mathbf{e}_2(0)\mathbf{e}_2(1)\mathbf{f}_2(1)}$ be the analogous sequence for $\overline{d_2 e_2 f_2}$. We express collinearity for the

new vertex's uv positions $\mathbf{w}_1, \mathbf{w}_2$ on either side of the seam as:

$$\mathbf{w}_1 - \mathbf{d}_1(0) = \frac{(\mathbf{w}_1 - \mathbf{d}_1(0)) \cdot (\mathbf{f}_1(1) - \mathbf{d}_1(0))}{\|\mathbf{f}_1(1) - \mathbf{d}_1(0)\|^2}(\mathbf{f}_1(1) - \mathbf{d}_1(0))$$

$$0 \leq \frac{(\mathbf{w}_1 - \mathbf{d}_1(0)) \cdot (\mathbf{f}_1(1) - \mathbf{d}_1(0))}{\|\mathbf{f}_1(1) - \mathbf{d}_1(0)\|^2} \leq 1$$

and analogously for $\mathbf{w}_2$. We express the edge length ratio constraint as

$$\frac{(\mathbf{w}_1 - \mathbf{d}_1(0)) \cdot (\mathbf{f}_1(1) - \mathbf{d}_1(0))}{\|\mathbf{f}_1(1) - \mathbf{d}_1(0)\|^2} = \frac{(\mathbf{w}_2 - \mathbf{d}_2(0)) \cdot (\mathbf{f}_2(1) - \mathbf{d}_2(0))}{\|\mathbf{f}_2(1) - \mathbf{d}_2(0)\|^2}$$

If the seam edge is only adjacent to one unifiable edge—without loss of generality d_1, d_2—then the only solution to the constraints is for the unifiable edges $\overline{d_1 e_1}$ and $\overline{d_2 e_2}$ to merge by collapsing $\mathbf{w}_1$ and $\mathbf{w}_2$ to $\mathbf{e}_1(1)$ and $\mathbf{e}_2(1)$, respectively. This is a straightforward equality constraint.

For a non-seam uv silhouette edge, the collinearity constraints are necessary but not the length ratio. Any re-parameterization of the line is seam-free, so long as the shape is preserved.

5.2 Algorithm

We implement our criteria and conditions as an extension to Garland and Heckbert [1998]'s n-dimensional follow-up to their seminal QSlim method [Garland and Heckbert 1997]. (Our extension could be added to any edge-collapsing approach, such as Hoppe [1999].) These approaches extend the greedy edge-collapsing approach of QSlim to initialize each face's decimation metric with one that considers all attributes of the face.

The basic idea of Qslim in nD is to use a point-to-plane metric for a plane embedded in a higher-dimensional space (e.g. $x, y, z, u, v, \ldots$). In QSlim or any metric-based edge-collapse algorithm, the metric is stored first on faces, then averaged onto vertices, and finally onto edges. The cost of collapse and the nD value of the resulting vertex are determined by the nD point minimizing the metric. The algorithm repeatedly collapses the lowest cost edge until the mesh is sufficiently decimated. After each collapse, nearby edges' costs must be recomputed. The cost of an un-collapsible edge is ∞.

For a parameterized mesh, the surface mesh and texture mesh have corresponding faces, but the faces will not, in general, share the same edge or vertex connectivity. Let the input model be represented as a list of n_3 3D surface vertices V, a list of n_2 2D parametric (uv) vertices U, and two lists of m triangles F_V and F_U, which are ordered triplets of indices into V and U, respectively. This matches the way in which surface and texture meshes are stored in an OBJ-like format. We assume all edges have at most two incident faces, though extension to non-manifold inputs does not seem out of the question. We also assume that the surface mesh is closed; for open meshes, we add a new virtual vertex (with 3D position and 2D texture coordinate at infinity) and triangles stitching this virtual vertex to all boundaries edges.

Since the vertex sets V and U are *not* in correspondence, it is useful to define the set of *unique* vertices Z. Vertices in Z are in $\mathbb{Z}^2$ and refer indirectly to V and U. In the worst case, every corner of every triangle is a unique vertex in Z; then $|Z| = 3m$. In this way, we may re-interpret both F_V and F_U as a single face list F indexing Z; a vertex in Z specifies where in V or U to look for 3D surface coordinates or 2D texture coordinates. Similarly, half-edges reference vertices in Z. With these definitions in hand, a seam edge is one whose two half-edges reference different vertices in Z. We store nD metrics with vertices in Z rather than V. Each face contributes its metric to its three corners in Z. When an edge to be collapsed is a seam edge, each halfedge contains two Z vertices, all of which have nD metrics. We sum the metrics of each halfedge's two vertices separately and associate each metric with the two new Z "half-vertices" that result from the collapse. Note that we never explicitly create Z. We create Z vertices on-the-fly as pairs of vertex and texture coordinate indices. We look up the nD metrics via a hash table whose keys are Z vertices. This is similar to Hoppe [1999]'s *wedge* data structure to store the distinct attributes a vertex may have in each of its incident faces, but simpler to create and maintain for our purposes.

To find the 3D position and UV coordinates of the two Z half-vertices, we build a $2n$D metric (each of the two nD metrics is a block along the diagonal) involving two copies of every dimension, one for each half-vertex: $x_1, y_1, z_1, u_1, v_1, x_2, y_2, z_2, u_2, v_2$. We solve this metric subject to the constraints that $x_1 = x_2, y_1 = y_2, z_1 = z_2$, since the 3D positions must match, and the uv silhouette-related constraints defined above ($\mathbf{w}_1 = (u_1, v_1)$ and $\mathbf{w}_2 = (u_2, v_2)$. These are linear equality and inequality constraints imposed on the minimization of a quadric metric; we solve it using an implementation of the quadratic programming solver by Goldfarb and Idnani [1983].

6 SEAM STRAIGHTENING

In Section 5, we show that only seam edges along straight lines with matching edge-length ratios can be collapsed if we want to share texture-domain signals between the high-resolution and decimated meshes. Seam parameterizations of most models do not often meet these criteria and consequently decimation is hindered (Figure 12). We propose preprocessing a given model's uv parameterization to straighten seam edges and therefore increase the effectiveness of our seam-aware mesh decimation. This step can be interpreted as a one-time remapping of a model's parameterization and texture-domain signals so that here-on-out, different mesh resolutions can share data.

The input to this subroutine is a model composed of a surface mesh (only the connectivity F_V is needed) and a parameterization mesh (F_U and U) as described in Section 5, a list of signals (e.g., color textures, normal maps, geometry image, weight maps, etc.) defined on the parametric domain, and a tolerance parameter controlling the amount of straightening to perform. We assume that the input model's surface and parametrization meshes are edge-manifold, but they may contain boundaries.

The output is a displacement of the parametrization mesh's vertex coordinates and a corresponding warp of each of the input signals on the parametric domain (Figure 10).

While we cannot make a strong guarantee that the output parameterization will be free of foldovers (a difficult problem in general [Rabinovich et al. 2017]), we do take steps to prevent unsatisfiable constraints. We observe in practice that the boundary coarsening for reasonable tolerances does not cause cross-overs. Preventing

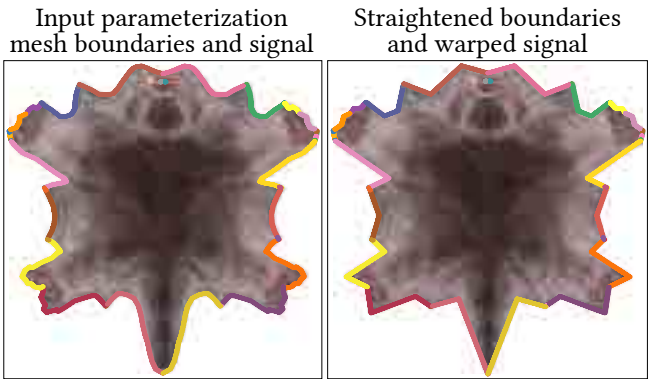


Fig. 10. We identify "straightenable" components of the parameterization meshes boundary edges (colored uniquely), coarsen these as piecewise-linear curves up to a given tolerance, and warp the interior.

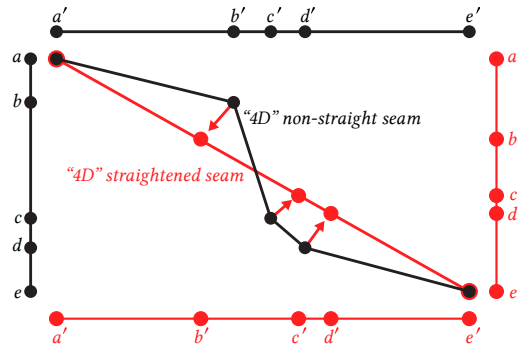


Fig. 11. Two halves of a seam with original vertices $\{a, b, c, d, e\}$ and $\{a', b', c', d', e'\}$ are illustrated as black chains with mismatching edge-length ratios (left and top). We straighten the seam, treating it as a 4D curve (illustrated here as a black 2D curve reduced to a red curve). The vertices are repositioned along the curve in 4D, and these define the parametric vertex positions along the original seams (right and bottom). Because their parametrizations agree, edge-length ratios now also agree.

foldovers during the internal warping is more challenging. We provide a simple heuristic, though more complex methods could be substituted, e.g., [Rabinovich et al. 2017].

Protected parametrization vertices. Protected parameterization vertices demarcate the boundary of *straightenable* edge-chain components and will not be moved.

A parameterization vertex will be protected if any of the following are true.

(1) Its corresponding surface mesh vertex is incident on any number of seam edges except zero or two. Vertices with zero incident seam edges are of no concern, and vertices with two incident seam edges have been "cut cleanly" and need not protection.
(2) Its corresponding surface mesh vertex is incident on at least one seam edge and a surface mesh boundary edge. We would like to straighten boundaries along with seams, but vertices joining seams and boundaries must not be moved.
(3) It is incident on an "ear" face of the parameterization mesh (a face with exactly two surface boundary or seam half-edges). Ears are problematic for the subsequent internal warping and protected as a preventative measure.
(4) It shares a corresponding surface mesh vertex with a protected parameterization vertex. We will straighten both sides of each seam simultaneously. To ensure compatibility, we *propagate* protection via the surface mesh vertex correspondence.

Edge component straightening. Next we run connected component analysis boundary and seam half-edges of the parameterization mesh, where two half-edges are *adjacent* if they share a non-protected vertex. Due to the protection process, each component will contain either all seam or all surface boundary half-edges. If component A_1 contains seam half-edges $d_1, e_1, f_1, \ldots$, then another component A_2 of equal size will contain all copies $d_2, e_2, f_2, \ldots$. We call A_2 the "copy" of component A_1. We then process components along with their copies in turn.

If the component contains boundary edges, then we straighten the chain or closed loop of edges by treating it as a 2D piecewise-linear curve. We coarsen this curve to the user provided tolerance using the iterative Ramer-Douglas-Peucker method [Douglas and Peucker 1973; Ramer 1972]. We reposition vertices removed during coarsening according to their position along the coarsened curve, parameterizing the segment between the previous and next surviving vertex according to normalized original edge-length. We ensure that closed boundary loops do not coarsen to less than three edges.

Otherwise the component contains seam half-edges, and we coarsen using Ramer-Douglas-Peucker in four dimensions; we concatenate the coordinates from each copy. We reparameterize the vertices of the non-straight 4D curve onto the straightened 4D curve according to arc-length. Using this parameterization to move each side of the seam, we guarantee that edge-length ratios match (see illustration in Figure 11).

Warping the interior. If we only move seam and boundary vertices on parameterization mesh, there will be high distortion and many foldovers of the attached faces. To alleviate this, we perform planar shape morphing. We experimented with a variety of ready-made options.

A simple method is to map interior vertices according to a discrete harmonic displacement field (see, e.g., [Joshi et al. 2007]). For small changes in the boundary this is sufficient. However, for large changes, especially where convex regions becoming concave or vice-versa, this simple linear mapping creates many inverted triangles. To accommodate large changes, we compose harmonic mappings along small linear steps (similar in spirit to [Schneider et al. 2013] and inspired by [Xu et al. 2011]).

Input parameterizations often contain exactly or nearly degenerate triangles that cause numerical instabilities when conducting discrete harmonic mappings. We observe that texture meshes are often much *messier* than surface meshes. Even more than surface meshes, they often are not intended for computation. In the presence of degeneracies, we construct a constrained Delaunay triangulation

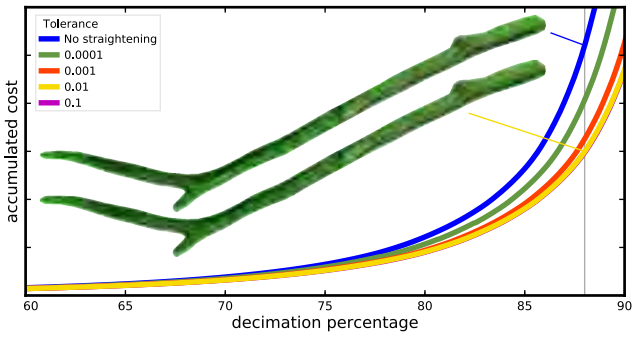


Fig. 12. The effect of straightening tolerance on decimation quality. The accumulated metric cost of edge collapses to decimate a model (stick) decreases as more straightening is performed. Each curve represents a different straightening tolerance. (The 0.1 curve is almost entirely covered by the 0.01 curve.) Straightening increases the number of seam edges that can be collapsed, allowing for a more effective use of the mesh resolution and therefore a lower total error.

	Before			After Straightening	After Decimation		
Example	# faces	# seam edges	# un-collapsible edges	# un-collapsible edges	# faces	# seam edges	# un-collapsible edges
Lemon	50368	196	0	-	168	58	58
Chimp	52352	1507	805	171	2594	901	894
Cow	23328	1929	1923	-	3494	1929	1929
Hercules	79691	626	626	290	3293	470	469
Animal	39040	419	369	17	1166	200	194
Wolf	10018	390	374	173	894	290	282
Boy	80000	2281	1462	-	3994	1878	1877

Table 2. Information about the effects of our straightening (Section 6) and decimation (Section 5) on various models. Straightening increases the number of collapsible seam edges. This allows for greater flexibility, and therefore higher quality, decimations (e.g. Figure 1).

of the entire parametric domain (i.e., unit square) treating input parametrization mesh vertices as point constraints and boundary edges as segment constraints. Then we conduct the composite harmonic mapping on this mesh, bringing along the interior vertices as part of the ambient mesh's triangulation. This part of our pipeline will immediately benefit from future improvements in bijective planar mapping algorithms.

Whether using an ambient mesh or not, ears incident on the morphing vertices could be *forced to flip* as their behavior is fully determined by the morphing constraints. Hence, we preemptively protect all ears. Finally we simply *render* new signals using the new parametrization coordinates, with the old coordinates as texture coordinates, into each input signal.

In our experiments, we always use tolerance = 0.01 when straightening. Beyond this value, few additional seam edges become collapsible (Figure 12).

7 EVALUATION

Our seam erasure, seam straightening, and seam decimation can be used to create seam-free signals shared by all decimation levels. Importantly, they can be used for crack-free tessellation on the GPU with the standard rendering pipeline. This enables the straightforward application of geometry images for dynamic level-of-detail, with no special data structures to avoid seam artifacts. Perfect vector-valued displacement maps (storing *xyz* offsets) can also be easily created and used. We also explore skin weight textures for the first time (Section 8).

Decimation and Straightening. Figure 13 and 20 show a variety of models decimated using our algorithm. These decimated models share the same parameterization as the original mesh and so all levels of detail can still be rendered, seam-free, with the original color and normal map (Figure 13) or by baking the original mesh's normal map (Figure 20). Furthermore, we can rasterize the original model's positions into a geometry image (Figure 13) or vector-valued displacement map (Figure 20), erase seams, and obtain obtain crack-free dynamic level-of-detail with the tessellation shader on the GPU. See Table 2 for information about the decimation of these models. It is trivial to create a vector-valued displacement map via the shared parameterization by subtracting the un-decimated and decimated mesh's geometry images. Vector-valued displacement maps avoid foldover artifacts that result from ray casting in the normal direction and artifacts that arise from closest point sampling [Nießner et al. 2016]. The shared parametric domain provides a natural correspondence.

Rendering Performance. Figure 14 shows the rendering performance of static and dynamic level of detail. Rendering performance was measured on a 2.30 GHz Intel Xeon E5-2630 CPU with 32 GB of RAM and a GeForce GTX 1080 GPU. More models' performance plots can be seen in the supplemental materials. Our decimation generates multiple meshes that can be rendered in the typical way, with 3D positions as vertex attributes. Because our decimations share a seam-free parametric domain, however, we can submit a decimated mesh's *uv* coordinates as the sole vertex attributes and load all information from textures (here, positions from a geometry image), without cracks. With a tessellation shader, this extends to choosing the level-of-detail dynamically on the GPU. In this plot, we choose a variety of uniform tessellation levels. All our examples render in real time on a 2012 MacBook Air 2GHz Intel Core i7 with 8GB RAM and an Intel HD Graphics 4000 GPU. Videos were recorded on a 2015 13" MacBook Pro with a 2.9 GHz Intel Core i5 processor with 16 GB of RAM and an Intel Iris Graphics 6100 GPU.

We store geometric surface attributes as floating point textures for greater fidelity. For example, if stored in 8-bit images, geometry images or displacements maps would lose fidelity when quantized and normalized to lie within $[0, 255]$; weight maps (Section 8) can be stored as 8-bit images without loss of fidelity, but must be normalized on-the-fly.

8 WEIGHT MAPS

Subspace methods describe the deformation of any point on a surface as a function of small number of degrees of freedom. For linear subspaces, this function is a simple weighted sum of subspace functions defined on the surface, i.e. *subspace signals*. Linear blend

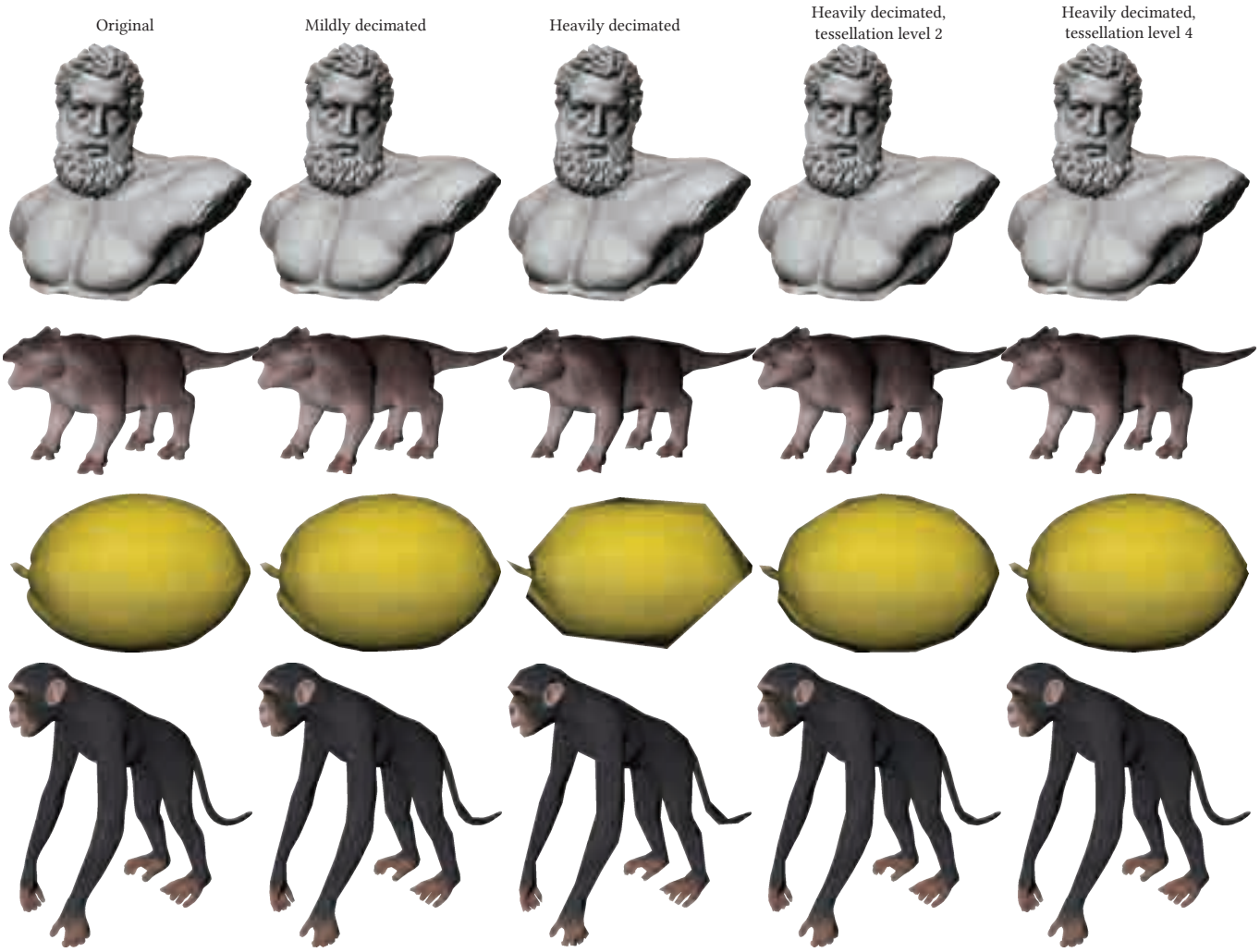


Fig. 13. Decimating a model while reusing its color and normal map without seam artifacts. Tessellating the model on the GPU using a geometry image for crack-free dynamic level of detail.

skinning—perhaps the most well known linear subspace method defines subspace signals as *bone weights*, and its subspace parameters are the bones' transformations. Modern applications of linear blend skinning encompass not just skeletons, but also point, region, and cage handles [Jacobson et al. 2014].

As the deformation is pointwise, we also store skinning weights in the parametric domain as textures, decoupling them from mesh resolution. State-of-the-art automatic weighting algorithms employ variational principles and the finite-element method to approximate smooth weight functions (e.g., [Baran and Popović 2007; Jacobson et al. 2011; Joshi et al. 2007]). In classic linear blend skinning, computing these functions on a more accurate discretization is mostly for naught if only values on the original mesh are retained. With weight maps, we compute more accurate automatic weights and efficiently store them in floating-point images. We erase the seams with E_{lerp} enabled, since the weight signal was initially computed on a high resolution mesh. When rendering, we tile all weight maps into one or two floating point textures.

Adaptive Subdivision. Because deforming models may require additional detail *locally*, we use a simply adaptive tessellation level scheme when rendering them. The GPU's tessellation control shader allows one to specify, for each triangle, separate tessellation levels for each edge and the interior. To prevent cracks, an edge should be given the same tessellation level from both incident triangles. We evaluate, at each edge midpoint, the length-weighted deviation between the exact deformation and the piecewise-linear approximation (Figure 15, red arrow):

$$e_{ij} = \|\mathbf{v}_i - \mathbf{v}_j\| \times \|\text{deform}(\mathit{lerp}(\mathbf{v}_i, \mathbf{v}_j, \frac{1}{2})) - \mathit{lerp}(\text{deform}(\mathbf{v}_i), \text{deform}(\mathbf{v}_j), \frac{1}{2})\|$$

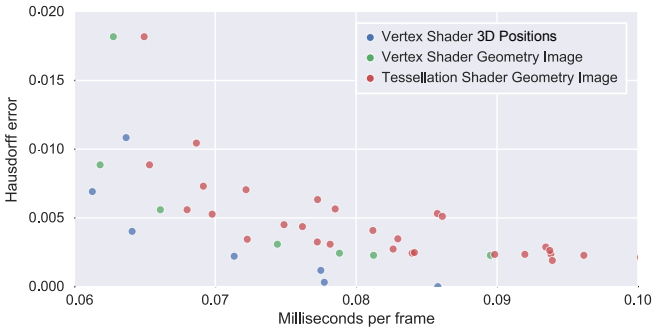


Fig. 14. Rendering performance. We plot rendering scenarios as points with coordinates given by their Hausdorff error and milliseconds per frame. Rendering scenarios, classified by color, are degree of decimation, whether using the geometry image, and uniform tessellation level (as applicable). Hausdorff error is measured as the distance to the undecimated mesh as a fraction of bounding box diagonal. (Note that even without decimation, using a geometry image introduces error due to rasterization; hence the geometry image scenarios are vertically offset.) See the supplemental materials for additional models' performance plots.

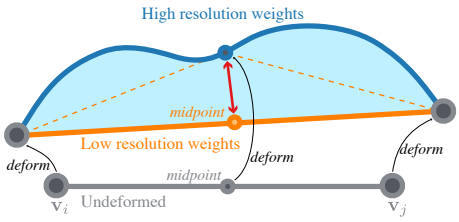


Fig. 15. An edge between $\mathbf{v}_i$ and $\mathbf{v}_j$ stays straight when per-vertex weights are used, but can take on any shape with weight maps. Our adaptive subdivision evaluates the distance indicated with a red arrow.

where deform($\mathbf{v}$) looks up the position from the geometry image at $\mathbf{v}$'s texture coordinate or adds the displacement vector and then, if using weight textures, applies the skinning deformation. This amounts to a 0-th order numerical approximation of the integrated length-weighted deviation along the edge (Figure 15, shaded region): $\|\mathbf{v}_i - \mathbf{v}_j\| \int_0^1 \|\text{deform}(lerp(\mathbf{v}_i, \mathbf{v}_j, t)) - lerp(\text{deform}(\mathbf{v}_i), \text{deform}(\mathbf{v}_j), t)\| dt$. We use $quality \cdot e_{ij}$ as the tessellation level for each edge and $quality \cdot \frac{e_{ij}+e_{jk}+e_{ki}}{3}$ as the tessellation level for each face, where *quality* is a user-determined parameter that we scale by the inverse distance of the object to the camera.

Experiments. Treating skinning weight signals as a potentially much higher or lower resolution signal than the base surface mesh has important ramifications.

Analogous to Z-BRUSH-style sculpting interfaces, riggers and animators can paint weights directly onto any point of the surface, ignoring its geometric representation or particular vertex placement. In Figure 16, we sketch a proof-of-concept demo where an animator 3D paints high-resolution weights. Animators and modelers have comfortable tool suites for painting high-resolution texture colors and displacement maps either in the texture domain or directly on the surface (e.g. ZBRUSH). Current weight painting tools (e.g. in MAYA) are limited by the resolution of the mesh. More expressive weights require subdivision of the mesh geometry (and interpolation of other attributes) just to create vertex degrees of freedom for weights.

Fig. 16. Just like painting texture colors, an animator can now 3D paint *high-resolution* skinning weights and see their effect immediately. In this example the weights for the bony skull of the animal are painted to make it move rigidly with the rest of the head. Please see the supplemental materials for a video of this example.

Fig. 17. High resolution weight textures can be shrunk to extremely small sizes while maintaining virtually all rendering fidelity. Even a weight texture as small as 16×16 captures the character's movement. At 4×4 resolution, the space of seam-free textures is insufficient and the weight maps become nearly constant.

In Figure 17, we see that weight maps as small as 16×16 are sufficient for many animations.

In Figure 20, we show linear blend skin deformations with adaptive tessellation. Decimation concentrates on areas with low Gaussian curvature, often producing long, skinny triangles. Despite this, weight maps and displacement maps restore the shape and deformability of the original, detailed model.

In Figure 18, we consider the canonical bending and twisting tests (e.g., [Kavan and Sorkine 2012]). Per-vertex skinning does not have enough degrees of freedom to create an interesting bend or twist. Skinning with weight textures reproduces the expected ground-truth pointwise skinning behavior (e.g. linear blend skinning's candy wrapper effect). Additional detail is added only as needed (zoom to see wireframe). To further show the agnosticity of our approach to skinning algorithms, in Figure 19 we reproduce a classic result from Sederberg et al. [1986] using Free-Form Deformation weights.

Figure 21 separates the effects of high resolution skinning weights from displacement mapping. A coarse, flat surface is augmented with a bumpy displacement map. With our pipeline, the surface bends as expected. With per-vertex skin weights, the mesh vertices are first deformed by skinning and then the tessellation shader adds applies the displacement map offsets after transforming them via linear blend skinning applied to the tangent space.

9 CONCLUSION

Seams, long an irritant of computer graphics, play an important role in the creation and reuse of surface signals. Without consideration of seams, signals cannot be stored or retrieved properly and levels-of-detail cannot make use of a shared parametric domain. They are a tax on artists, who must carefully place them to minimize the discontinuity artifacts. We have shown how to erase seams from

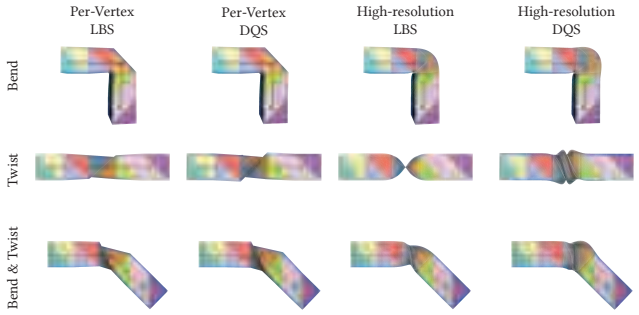


Fig. 18. Per-vertex skinning on a 24-vertex box suffers severe artifacts when bending or twisting under linear blend skinning (LBS) [Magnenat-Thalmann et al. 1988]) and Dual Quaternion Skinning (DQS) [Kavan et al. 2008]. Weight maps reproduce expected ground-truth pointwise skinning.

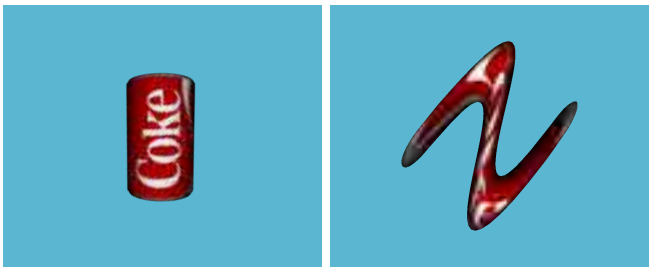


Fig. 19. Coca-Cola Classic. Our dynamic refinement approach can be used for Free-Form Deformation [Sederberg and Parry 1986].

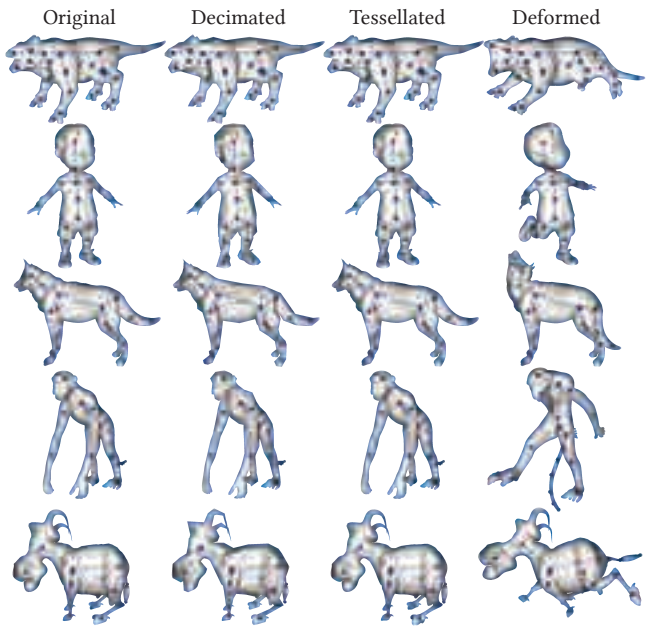


Fig. 20. Decimating a detailed input mesh into a coarse mesh and accompanying vector-valued displacement map, normal map, and skin weight map. These allow our the coarse model to reproduce the shape and deformability of the detailed mesh.

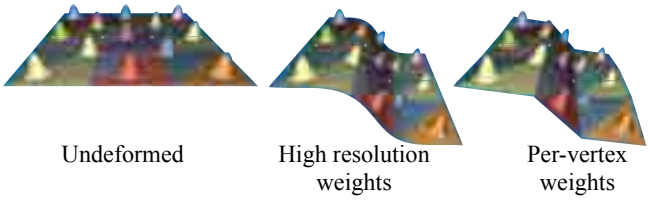


Fig. 21. A displacement map applied to a flat square mesh, original triangles indicated by color. Using high resolution weights, the expected smooth deformation is achieved. With per-vertex weights, the coarse base surface cannot smoothly deform.

any existing mesh and texture and how to correctly consider them during decimation. Our approach places no explicit rectilinear or symmetry constraints on the shape of seams or constraints on signals' values. As a result, we are able to decouple surface signals, geometric and otherwise, from mesh resolution. This frees designers from worrying about discretization, and implementors from worrying about seam artifacts. Our approach naturally supports mip maps (and anisotropic filtering via rip maps) by optimizing each downsampled image independently. Mipmapping maintains seamlessness. It continuously blends continuous functions. Our approach has broad applicability, requiring minimal changes to existing workflows and no changes to the standard rendering pipeline. We expect our approach to be widely adopted, as it imposes no runtime costs or parameterization requirements. We hope that eliminating cracks will lead to widespread use of GPU tessellation shaders. Finally, we believe that our approach may provide an expanded set of desiderata for parameterization.

Limitations and Future Work. Seam erasure is limited by the resolution of the texture. Small textures cause more "locking" where the texture becomes constant. Seam erasure will also produce undesirable results when the parametrization mesh seams cross each other or come within one texel of each other; distant seams then become constrained to the same value. Closely packed structures in a parameterization, like fingers, can pose a problem for geometry images. A given resolution may be fine for a color texture, since all fingers have similar colors, but not for a geometry image, which must distinguish the fingers. Seam erasure, to eliminate cracks, may merge them.

In the future we would like to explore additional adaptive subdivision criteria for the tessellation shader. For example, it may be profitable to encode the potential tessellation gain due to the geometry, displacement, or weight maps of an edge or face. We also believe that the creation of vector-valued displacement maps and geometry images would benefit from an optimization that minimizes the bilinear reconstruction error of the signal. To correctly transform normals while linear blend skinning, *in general*, weight gradients are needed [Tarini et al. 2014]. The inverse transpose is often used in practice. In the future we will explore computing weight gradients on the fly; their seam-free computation may require an additional subspace constraint in our texture seam optimization. Finally, we also plan to explore applications of our optimization to volumetric textures (trilinear interpolation).

ACKNOWLEDGMENTS

We are grateful to Guilin Liu and Jyh-Ming Lien for lending us hardware; to Tamy Boubekeur for a discussion on tessellation shaders; to Keenan Crane for the fish model (Figs. 1 and 2), to Maurizio Nitti for the Boy (Fig. 20) and to Turbosquid users Sumatra3d (lemon, Figs. 4 and 13), Deniz Ozemre (Hercules, Figs. 4 and 13), SpinQuad1976 (chimp, Figs. 4, 13, and 20), Pabong (cow, Figs. 4 and 20; human, Fig. 3), Tornado Studio (stick, Fig. 12), and mnphmnmn (wolf, Fig. 20).

This work was supported by the United States National Science Foundation (IIS-1451198 & IIS-1453018), a Google research award, NSERC Discovery Grants (RGPIN-2017-05235 & RGPAS-2017-507938), the Connaught Fund (NR-2016-17), and a gift from Adobe Systems Inc.

REFERENCES

Noam Aigerman, Roi Poranne, and Yaron Lipman. 2015. Seamless Surface Mappings. *ACM Trans. Graph.* 34, 4, Article 72 (July 2015), 13 pages.

Ilya Baran and Jovan Popović. 2007. Automatic Rigging and Animation of 3D Characters. *ACM Trans. Graph.* 26, 3 (2007), 72:1–72:8.

James F. Blinn. 1978. Simulation of Wrinkled Surfaces. *SIGGRAPH Comput. Graph.* 12, 3 (Aug. 1978), 286–292.

Mario Botsch, Mark Pauly, Martin Wicke, and Markus Gross. 2007. Adaptive Space Deformations Based on Rigid Cells. *Comput. Graph. Forum* 26, 3 (2007), 339–347.

Tamy Boubekeur. 2010. A view-dependent adaptivity metric for real time mesh tessellation. In *Proc. ICIP*. 3969–3972.

Tamy Boubekeur and Marc Alexa. 2008. Phong Tessellation. *ACM Trans. Graph.* (2008).

Alon Bright, Edward Chien, and Ofir Weber. 2017. Harmonic Global Parametrization with Rational Holonomy. *ACM Trans. Graph.* 36, 4, Article 89 (July 2017), 15 pages.

Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. 2008. MeshLab: an Open-Source Mesh Processing Tool. In *Eurographics Italian Chapter Conference*, Vittorio Scarano, Rosario De Chiara, and Ugo Erra (Eds.). The Eurographics Association.

Jonathan Cohen, Dinesh Manocha, and Marc Olano. 1997. Simplifying Polygonal Models Using Successive Mappings. In *IEEE Visualization (VIS)*.

Jonathan Cohen, Marc Olano, and Dinesh Manocha. 1998. Appearance-preserving Simplification. In *ACM SIGGRAPH*.

Robert L. Cook. 1984. Shade Trees. *ACM SIGGRAPH* 18, 3 (Jan. 1984), 223–231.

Martin de Lasa and Aaron Hertzmann. 2009. Prioritized Optimization for Task-Space Control. In *International Conference on Intelligent Robots and Systems (IROS)*.

Tamal K Dey, Herbert Edelsbrunner, Sumanta Guha, and Dmitry V Nekhayev. 1999. Topology preserving edge contraction. *Publ. Inst. Math. (Beograd) (N.S.)* 66, 80 (1999), 23–45.

David H Douglas and Thomas K Peucker. 1973. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization* 10, 2 (1973), 112–122.

Michael Garland and Paul S. Heckbert. 1997. Surface Simplification Using Quadric Error Metrics. In *ACM SIGGRAPH*.

Michael Garland and Paul S. Heckbert. 1998. Simplifying Surfaces with Color and Texture Using Quadric Error Metrics. In *IEEE Visualization (VIS)*.

Donald Goldfarb and Ashok Idnani. 1983. A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical programming* 27, 1 (1983), 1–33.

X. Gu, S. J. Gortler, and H. Hoppe. 2002. Geometry images. *ACM Trans. Graph.* 21, 3 (2002), 355–361.

Igor Guskov, Kiril Vidimče, Wim Sweldens, and Peter Schröder. 2000. Normal Meshes. In *ACM SIGGRAPH (SIGGRAPH)*.

Matthias Holländer and Tamy Boubekeur. 2010. Synthesizing Subdivision Meshes using Real Time Tessellation. In *IEEE Pacific Graphics*.

Hugues Hoppe. 1999. New Quadric Metric for Simplifying Meshes with Appearance Attributes. In *IEEE Visualization (VIS)*.

F C Huang, B Y Chen, and Y Y Chuang. 2006. Progressive deforming meshes based on deformation oriented decimation and dynamic connectivity updating. In *Proc. SCA*.

Michal Iwanicki. 2013. Lighting Technology of "The Last of Us". In *ACM SIGGRAPH 2013 Talks*. ACM, New York, NY, USA.

Alec Jacobson, Ilya Baran, Jovan Popović, and Olga Sorkine. 2011. Bounded Biharmonic Weights for Real-Time Deformation. *ACM Trans. Graph.* 30, 4 (2011), 78:1–78:8.

Alec Jacobson, Zhigang Deng, Ladislav Kavan, and JP Lewis. 2014. Skinning: Real-time Shape Deformation. In *ACM SIGGRAPH 2014 Courses*.

Hanyoung Jang and JungHyun Han. 2012. Feature-Preserving Displacement Mapping With Graphics Processing Unit (GPU) Tessellation. *Computer Graphics Forum* 31, 6 (2012), 1880–1894.

Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. 2007. Harmonic Coordinates for Character Articulation. *ACM Trans. Graph.* 26, 3 (2007).

Oussama Kanoun, Florent Lamiraux, Pierre-Brice Wieber, Fumio Kanehiro, Eiichi Yoshida, and Jean-Paul Laumond. 2009. Prioritizing linear equality and inequality systems: application to local motion planning for redundant robots. In *IEEE Robotics and Automation (ICRA)*.

Ladislav Kavan, Steven Collins, Jiri Zara, and Carol O'Sullivan. 2008. Geometric Skinning with Approximate Dual Quaternion Blending. *ACM Trans. Graph.* 27, 4 (2008), 105:1–105:23.

Ladislav Kavan and Olga Sorkine. 2012. Elasticity-inspired deformers for character articulation. *ACM Trans. Graph.* 31, 6 (2012).

Shahar Z. Kovalsky, Meirav Galun, and Yaron Lipman. 2016. Accelerated Quadratic Proxy for Geometric Optimization. *ACM Trans. Graph.* 35, 4, Article 134 (July 2016).

Aaron Lee, Henry Moreton, and Hugues Hoppe. 2000. Displaced Subdivision Surfaces. In *ACM SIGGRAPH*.

Aaron W. F. Lee, Wim Sweldens, Peter Schröder, Lawrence Cowsar, and David Dobkin. 1998. MAPS: Multiresolution Adaptive Parameterization of Surfaces. In *ACM SIGGRAPH*. 95–104.

Songrun Liu, Alec Jacobson, and Yotam Gingold. 2014. Skinning Cubic Bézier Splines and Catmull-Clark Subdivision Surfaces. *ACM Trans. Graph.* 33, 6 (2014).

Charles Loop, Scott Schaefer, Tianyun Ni, and Ignacio Castaño. 2009. Approximating Subdivision Surfaces with Gregory Patches for Hardware Tessellation. *ACM Trans. Graph.* 28, 5, Article 151 (Dec. 2009), 9 pages.

Nadia Magnenat-Thalmann, Richard Laperrière, and Daniel Thalmann. 1988. Joint-dependent local deformations for hand animation and object grasping. In *Graphics Interface*. 26–33.

Maya. 2017. Autodesk, `http://www.autodesk.com/maya`. (2017).

A Mohr and M Gleicher. 2003. *Deformation sensitive decimation*. Technical Report. Univ. of Wisc.

Ashish Myles and Denis Zorin. 2013. Controlled-distortion Constrained Global Parametrization. *ACM Trans. Graph.* 32, 4, Article 105 (July 2013), 14 pages.

M Nießner, B Keinert, M Fisher, M Stamminger, C Loop, and H Schäfer. 2016. Real-Time Rendering Techniques with Hardware Tessellation. *Computer Graphics Forum* 35, 1 (2016), 113–137.

Matthias Nießner and Charles Loop. 2013. Analytic Displacement Mapping Using Hardware Tessellation. *ACM Trans. Graph.* (2013).

Matthias Nießner, Charles Loop, Mark Meyer, and Tony Derose. 2012. Feature-adaptive GPU Rendering of Catmull-Clark Subdivision Surfaces. *ACM Trans. Graph.* (2012).

D. Piponi and G. Borshukov. 2000. Seamless texture mapping of subdivision surfaces by model pelting and texture blending. In *Computer Graphics (SIGGRAPH Conference Proceedings)*. 471–478.

Budirijanto Purnomo, Jonathan D Cohen, and Subodh Kumar. 2004. Seamless texture atlases. In *Symposium on Geometry Processing (SGP)*. 65–74.

Michael Rabinovich, Roi Poranne, Daniele Panozzo, and Olga Sorkine-Hornung. 2017. Scalable Locally Injective Mappings. *ACM Trans. Graph.* 36, 2 (April 2017).

Urs Ramer. 1972. An iterative procedure for the polygonal approximation of plane curves. *Computer graphics and image processing* 1, 3 (1972), 244–256.

Nicolas Ray, Vincent Nivoliers, Sylvain Lefebvre, and Bruno Lévy. 2010. Invisible seams. *Computer Graphics Forum* 29, 4 (2010), 1489–1496.

Pedro V. Sander, Steven J. Gortler, John Snyder, and Hugues Hoppe. 2002. Signal-specialized Parametrization. In *Eurographics Workshop on Rendering (EGRW)*. 87–98.

P. V. Sander, J. Snyder, S. J. Gortler, and H. Hoppe. 2001. Texture mapping progressive meshes. In *Computer Graphics (SIGGRAPH Conference Proceedings)*. 409–416.

Henry Schäfer, Magdalena Prus, Quirin Meyer, Jochen Süßmuth, and Marc Stamminger. 2012. Multiresolution attributes for tessellated meshes. In *Symposium on Interactive 3D Graphics and Games (I3D)*. ACM, 175–182.

T. Schneider, K. Hormann, and M. S. Floater. 2013. Bijective Composite Mean Value Mappings. In *Symposium on Geometry Processing (SGP)*.

Thomas W. Sederberg and Scott R. Parry. 1986. Free-Form Deformation of Solid Geometric Models. In *Proc. ACM SIGGRAPH*. 151–160.

Alla Sheffer and John C. Hart. 2002. Seamster: Inconspicuous Low-distortion Texture Seam Layout. In *IEEE Visualization (VIS)*.

László Szirmay-Kalos and Tamás Umenhoffer. 2008. Displacement Mapping on the GPU - State of the Art. *Computer Graphics Forum* 27, 6 (2008), 1567–1592.

Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. 2014. Accurate and Efficient Lighting for Skinned Models. *Comput. Graph. Forum* 33, 2 (May 2014), 421–428.

Natalya Tatarchuk, Joshua Barczak, and Bill Bilodeau. 2010. Programming for real-time tessellation on gpu. *AMD whitepaper* 5, 3 (2010).

Robert Toth. 2013. Avoiding Texture Seams by Discarding Filter Taps. *Journal of Computer Graphics Techniques (JCGT)* 2, 2 (6 December 2013), 91–104.

Yin Xu, Renjie Chen, Craig Gotsman, and Ligang Liu. 2011. Embedding a Triangular Graph Within a Given Boundary. *Comput. Aided Geom. Des.* 28, 6 (Aug. 2011), 349–356.

Cem Yuksel, John Keyser, and Donald H. House. 2010. Mesh Colors. *ACM Trans. Graph.* 29, 2, Article 15 (April 2010), 11 pages.