Indirection Mapping for Quasi-Conformal Relief Texturing

Morgan McGuire* Kyle Whitson
Williams College

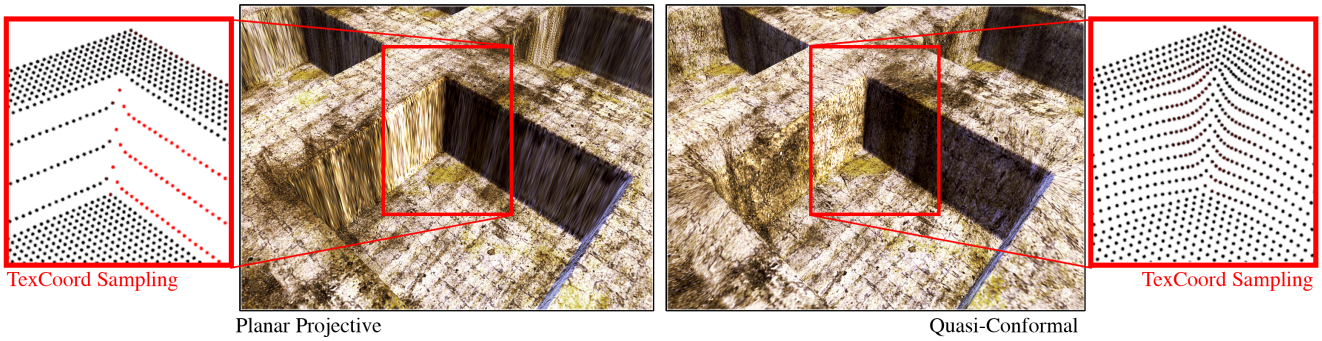

Figure 1: A single polygon rendered with parallax occlusion mapping (POM). *Left:* Directly applying POM distorts texture resolution, producing stretch artifacts. *Right:* We achieve approximately uniform resolution by applying a precomputed indirection in the pixel shader.

Abstract

Heightfield terrain and parallax occlusion mapping (POM) are popular rendering techniques in games. They can be thought of as per-vertex and per-pixel relief methods, which both create texture stretch artifacts at steep slopes.

To ameliorate stretching artifacts, we describe how to precompute an *indirection map* that transforms traditional texture coordinates into a quasi-conformal parameterization on the relief surface. The map arises from iteratively relaxing a spring network. Because it is independent of the resolution of the base geometry, indirection mapping can be used with POM, heightfields, and any other displacement effect. Noisy textures like grass and stucco can be used with indirection mapped (which is convenient when texturing terrain.) We pre-warp structured textures by the inverse of the indirection map to maintain their appearance.

Our process gives approximately uniform texture resolution on all surfaces. During rendering, the time and space overhead are one texture fetch and one texture map.

Keywords: parallax occlusion, relief, bump, heightfield, terrain

1 Introduction

A game development company approached us [Lomerson 2007] with a rendering challenge for a next-generation game engine. They wanted to use a popular new real-time displacement technique [Tatarchuk 2006] for rendering relief detail but were concerned about texture stretch artifacts that arise from it. Figure 1 shows a concrete tile from the ceiling of a Washington, D.C. subway station rendered as a single quadrilateral. The relief, i.e., the appearance of depth and parallax, is simulated by offsetting the texture coordinates according to a bump map during shading. The figure uses the actual material texture maps and bump map from the developer's game. The texture stretch visible in the left subfigure along the sides of the indent, is the kind of artifact that they were concerned about.

This paper describes a solution to texture stretch called *indirection mapping*, which was used to generate figure 1(right). Indirection mapping achieves approximately uniform texture resolution on even steep slopes. It operates by replacing the color texture read in a pixel shader with an indirect read through an *indirection map*. The indirection map changes the texture parameterization at each pixel in order to provide uniform sampling. In addition to POM, indirection mapping can be applied to parallax offset mapping [Kaneko et al. 2001], true displacement subdivision surfaces produced by a geometry shader, and heightfield terrain rendering (figure 9.)

The cost of indirection mapping is that one needs to precompute (and then load at runtime) an additional texture map of approximately the same memory footprint as the relief map. The runtime cost is negligible. We see no decrease in frame rate — both left and right images render at 320 fps with the shader applied to 1.3 M pixels — because the cost of a single additional texture fetch at runtime is unmeasurably small on modern hardware, compared to the overhead of a parallax shader or frame buffer swap.

1.1 Relief Surfaces

Surfaces are often represented for real-time rendering by low-frequency *base geometry* and a set of high-frequency displacements. To distinguish between this general tool and specific techniques, in this paper we follow architecture vocabulary and call any displacement function represented by samples on a regular 2D grid a *relief map*. Indirection mapping improves the appearance of any relief that is also colored by a texture map.

Relief detail can be applied at any scale. For large-scale relief such as terrain hills and valleys, the relief map is called a *heightfield*. It is rendered by displacing the vertices of subdivided base geometry. For small-scale relief such as grooves and rivets, the relief map is called a *bump map*. In this case, the geometry remains coarse and shading is adjusted to provide proper parallax occlusion and self-shadowing effects across the surface. Both scales are popular techniques for real-time applications like video games. Offline rendering employs various medium-scale displacement methods, but these have not yet become standard in real-time graphics.

*e-mail: morgan@cs.williams.edu

Relief may be applied to a non-planar base geometry, e.g., a patch of relief terrain applied to the curve of the Earth or a bump map applied to a character's round head. For any base geometry, a smoothly-varying local tangent space reference frame defines the two in-plane axes of the tangent $\hat{T}$ and bitangent $\hat{B}$, and the out-of-plane normal $\hat{N}$ direction along which the displacement acts. A well-known algorithm [Lengyel 2003] computes the $\hat{T}\hat{B}\hat{N}$ frame from (u,v) texture coordinates. That is, although the base surface is non-planar, in any small neighborhood it can be considered locally planar. Artists are skilled at maintaining approximately uniform resolution for the (u,v) values both manually and using optimization tools. This ensures approximately uniform application of the relief map, which is desirable.

The advantages of working on a regular grid are that relief maps can be authored with standard image tools and paradigms, filtered (e.g., for level of detail) in a straightforward manner, and that elevation and intersection queries are both easy to implement and fast to execute. The last two properties are critical for real-time applications, and are why heightfields and bump mapping remain popular in practice despite substantial scientific literature on more sophisticated tessellation and parameterization methods.

One drawback of representing relief as regular grid samples is that material maps (e.g., diffuse color) appear distorted when they follow the same (u,v) parameterization used for the relief map. In figure 1 left, the diffuse color and relief maps used identical (u,v) parameterizations that were uniform on the *base* surface. During rendering, the diffuse color parameterization was displaced to some other (u',v') at each pixel, based on the relief and the eye vector. This produced both desirable parallax and undesirable stretch. Such stretch artifacts are visible in many commercial products, for example, on the heightfield terrain of the *Battlefield 1942* game and the bump-mapped features like cobblestones in ATI's *Toy Shop* demo. This makes relief self-limiting: **steeper relief features create more desirable differentiation from the base surface, but also more undesirable stretch artifacts.** There are no artifacts when the relief magnitude is zero, but at that point there is also no relief!

The reason that indirection mapping can produce better results, like figure 1(right), is that it uses independent parameterizations (u,v) and (s,t) for relief and color maps. In figure 1, the (s,t) values were precomputed such that after parallax displacement the (s',t') distribution would be quasi-conformal. This means that the mapping from the 3D surface to texture space approximately preserves both angles and relative scale. (From here on, we are always working with post-parallax/post-projection coordinates, so we drop the u' notation.)

Indirection mapping uses the existing relief representations, thus leveraging their benefits; it renders at the same frame rate as previous methods; and can be applied to any relief technique, e.g., both large-scale terrain heightfields and small-scale bump maps. Two authoring scenarios are creating pre-warped material textures for specific relief and applying synthesized or tiling textures in a uniform manner over an a priori unknown relief. Our method is only appropriate for static relief data (although that data may be mapped onto a dynamic or skinned object.) In order of decreasing significance, the contributions of this paper are:

1. A fast multi-scale optimizer that produces a quasi-conformal parameterization M: $(u,v) \rightarrow (s,t)$ for relief maps.
2. Algorithms and advice for helping texture authors to leverage increased resolution on steep slopes.
3. Accurate surface distance metric to speed convergence.
4. The *indirection map* $I(u,v)$, a compact, geometry-independent encoding of M for GPUs.
5. Extension of parallax occlusion mapping to use $I(u,v)$, with demonstration of many positive results.

2 Related Work

Indirection mapping builds on many previous methods. Although many of the pieces have been around since 2001, no-one has previously used them to fix the stretch artifacts on parallax surfaces or heightfield terrain. Hence, these artifacts are common in games, geographic fly-bys, and other 3D applications as shown in figure 2.

2.1 Quasi-Conformality

An artist wants the texture map to "conform" to a displaced surface. Mathematical conformality [Marsden and Hoffman 1999] is consistent with this intuitive notion and provides a rigorous definition of the desirable properties of a texture parameterization. For our purposes, the essential property of a conformal texture parameterization is that it preserves local scale and angles, thus minimizing texture stretch artifacts. We also desire approximately uniform resolution globally, although we are willing to sacrifice that where needed to avoid local distortion. Conformality implies continuity, which is convenient since that allows interpolating between texels during rendering and also means that it is possible to efficiently use all of the texture map.

However, as noted by Lévy et al. [2002] it is impossible to achieve a truly conformal texture parameterization for general polygonal meshes. Thus texture parameterization schemes seek the minimal distortion *quasi-conformal* sampling. Both this paper and previous work find that minimum by optimization; the differences in approach are in how the deviation from the minimum is quantified and the optimizer algorithm itself.

2.2 Small-scale: Parallax Occlusion Mapping

Bump (a.k.a. normal) mapping adds the appearance of additional detail to a polygon mesh by specifying per-pixel surface normals. This decouples the physical geometry from the relief detail and is a standard effect in real-time rendering systems today and is easily implemented in a pixel shader. Oliveira et al.'s [2000] "relief mapping" used a method of pre-warping textures to extend traditional bump mapping with parallax effects. An efficient approximation of this technique known as "parallax mapping" [Kaneko et al. 2001] that works for features with *shallow* slopes is used in currently available applications. The primary drawback to parallax mapping is that it does not produce correct occlusions at *steep* slopes. Brawley and Tatarchuk's [2004] "parallax occlusion mapping" (POM) method introduced per-pixel heightfield ray tracing in a practical manner on a GPU to fix this. Their approach has since been augmented as summarized in table 1.

Techniques [Kaneko et al. 2001; Policarpo et al. 2005; Tatarchuk 2006] that use only the bump map at runtime have generally been the most interesting to the industry [Lomerson 2007]. We believe that our addition of a precomputed texture is justified because it improves rendering quality and because the texture is small. Previous precomputation techniques required much larger data, yet did not significantly improve quality over [Brawley and Tatarchuk 2004]. All previous methods exhibit texture stretch at steep slopes. Figure 2 shows a selection of artifacts from recent publications.

2.3 Large-scale: Terrain Parameterization

Heightfields have regular tessellation in 2D, but are irregular in 3D after relief displacement is applied. There is significant work on adjusting terrain vertices to achieve an approximately-regular 3D tessellation, e.g., [Duchaineau et al. 1997; Cignoni et al. 2003]. However, adjusting the geometry away from a regular 2D grid forgoes many of the features of the heightfield, such as constant-time elevation queries and linear-time line-of-sight checks. It is also not

[Kaneko et al. 2001]	Parallax offset for smooth slopes.
[Wang et al. 2003], [2004]	Ray-tracing and silhouettes with massive precomputed visibility.
[Brawley and Tatarchuk 2004]	Linear trace for steep slopes.
[Donnelly 2005]	Sphere tracing on 3D texture.
[Policarpo et al. 2005]	Added binary search for accuracy; two-sided heightfields.
[McGuire and McGuire 2005]	MIP-bias for sharp features; depth modification.
[Oliveira and Policarpo 2005]	Quadratic silhouette approx.
[Tatarchuk 2006]	Soft shadows; shader level of detail; piecewise-linear final step for fast, smooth surfaces; glancing angle correction.
[Policarpo and Oliveira 2006]	Multi-layer impostors to allow overlaps and discontinuity.
[Dachsbacher and Tatarchuk 2007]	Tetrahedra for silhouettes.
This paper	Parameterization correction for steep slopes.

Table 1: Summary of Selected Parallax Occlusion Methods.

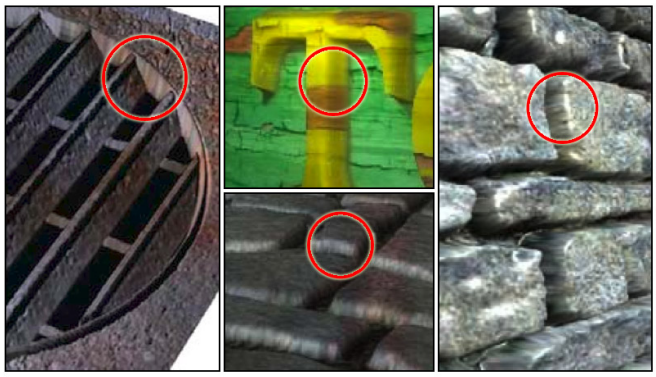

Figure 2: Texture stretch in previous work. *Left:* [Donnelly 2005] *Center:* [Tatarchuk 2006] *Right:* [Policarpo and Oliveira 2006].

advantageous on today's GPUs, where static vertex buffers provide optimal performance and one wants to apply samples displacements to a single, instanced vertex grid. Furthermore, an irregular tessellation is harder to use with filtering and level of detail algorithms and is less efficient to store because the triangle strip index array cannot be shared between terrain tiles and the horizontal positions of each vertex must be explicit instead of generated. We therefore seek a solution that retains the heightfield vertices and instead adjusts the texture coordinates.

From the earliest manual tools [Krishnamurthy and Levoy 1996] to more recent standalone optimizers [Lévy et al. 2002; Khodakovsky et al. 2003; Purnomo et al. 2004; Ray et al. 2006], most parameterization work focused on placing discontinuities in the parameterization to maximize conformality, decreasing the visibility of discontinuities, and efficiently packing the segments into a single texture map. Yet these discontinuous parameterizations are not well suited to relief surfaces—that would mean authoring textures that were cut up around each relief feature, hard to interpolate across smoothly, and that waste some space. We instead take advantage of our restricted geometry to find mappings that waste no texture space and are more intuitive for texture authoring.

A notable exception to previous parameterization methods that is closer to our work is signal-specialized parameterization (SSP) [Terzopoulos and Vasilescu 1991; Sloan et al. 1998; Sander et al. 2002], which adjusts texture coordinates and resolution according to an importance signal. Our parameterization can be viewed mathematically as SSP where the signal is the inverse of projected area. Mesh relaxation is a technique from the finite element community that has also been used in parameterization. Relaxation uses a mass-spring system to solve for uniform vertex distribution. Sander et. al [2001] relaxed pieces of the texture itself to eliminate wasted space in the texture map. We instead use relaxation in a manner similar to Wang et al. [2005] by relaxing the texture coordinates and holding the mesh geometry fixed. Compared to previous multi-scale optimizers [Wang et al. 2005; Lévy et al. 2002], the new surface distance metric presented in section 3.5 brings the coarse-scale approximation much closer to the fine-scale solution to dramatically speed convergence.

Compared to previous optimization work, we operate specifically on relief maps and do not introduce discontinuities. This allows us provide a more accurate stretch metric for the early passes of the optimizer, encode our parameterization efficiently for runtime, and use texture space efficiently. We separate the resolution of the geometry, the relief map, and the parameterization. This is desirable for all relief applications because it gives texture artists the freedom to assign resolution independently of modeling artists. Separating these is essential for use with POM, where by definition the bump map is much higher resolution than the geometry.

3 Preprocessing Algorithm

3.1 Overview

The preprocessing operation computes a quasi-conformal mapping M from an arbitrary relief map B by optimization and then encodes it efficiently in a differential *indirection map*, I. Note that any texture coordinate optimization scheme can be used with indirection mapping. We specifically introduce a new quasi-conformal one that is well-suited to particular characteristics of relief maps. These characteristics are that relief maps are usually continuous functions on a rectangle–or in the case of wrapping texture coordinates, a torus–and therefore can always be stretched to a rectangle without cuts. Our method also has the practical advantage that it is relatively easy to implement given a rasterizer.

Consider the mesh that would represent the relief-displaced surface if it contained real geometry. It has vertices that are points in tangent space and labels that are texture coordinates. Traditional tessellation algorithms would move the vertices to correct texture stretch, however we want to keep the vertices in place because they correspond to regular relief samples. The optimizer instead creates the dual of this mesh. In the dual, the vertices are in texture space and the labels are in tangent space. The optimizer then adjusts the dual mesh by iterative relaxation until it has approximately uniform tessellation. The inverse of the net transformation applied during relaxation is M, our desired mapping.

3.2 Definitions

Let relief-(e.g., bump-) map $R(u,v)$ be a piecewise-bilinear function defined on tangent space points such that $(u,v,z) = u\hat{T} + v\hat{B} + R(u,v)\hat{N}$ is on the displaced surface.

Let $M(u,v) = (s,t)$ be the mapping from tangent to texture space that is quasi-conformal for the displaced surface, and $M^{-1}(s,t) = (u,v)$ be its inverse. These are the functions that we seek through

optimization. The identity case where $M(u,v) = (u,v)$ is the traditional texture parameterization for a relief mapped surface.

Let $I(u,v)$ be an *indirection map* that encodes the differential parameterization at runtime,

$$I(u,v) = (\Delta u, \Delta v) = M(u,v) - (u,v), \tag{1}$$

We store the deltas instead of the absolute offset because they have smaller magnitude and correctly interpolate around the texture boundary for tiling textures.

Let $C(s,t)$ be a diffuse color or other material texture map. Without loss of generality, assume that tangent space coordinates (u,v,z) and texture coordinates (s,t) all range from 0 to 1.

Let $\vec{m}$ be a 2D array of $(g+1)^2$ vertices, where g is an integer power of two. Let $\vec{m}_i[x,y] = (u,v,R(u,v))$ be the position of the vertex stored at 0-based indices $[x,y]$ after relaxation iteration i. When optimization begins, $\vec{m}$ is initialized to a regular grid:

$$\vec{m}_0[x,y] = (x/g, y/g, R(x/g, y/g)). \tag{2}$$

Let $\delta(x,y,a,b)$ measure the signed difference between the actual and rest lengths of a simulated spring between $\vec{m}[x,y]$ and $\vec{m}[a,b]$. Let $\vec{F}_i[x,y]$ be the net force on $\vec{m}_i[x,y]$ due to all springs.

3.3 Multi-Scale

We begin optimization with the $\vec{m}$ grid at size $g = 16$. That mesh is relaxed until it approximately converges, that is, when the change in $\sum_{[x,y]} \vec{F}[x,y]$ is minimal over two iterations. The optimizer then subdivides each quadrilateral of the grid into four such that $g_{i+1} = 2g_i$ and continues relaxation.

The final sampling frequency of the texture mesh should be half the width of the narrowest feature in R. This guarantees that R will be sampled without aliasing and with at least two samples per inflection point for the piecewise-linear surface formed by the mesh that approximates R.

This process gives much faster convergence than creating the initial mesh at full resolution. We see convergence on the order of two minutes for meshes up to 1024×1024, wheras previous work reports convergence in about an hour [Wang et al. 2005] for comparable complexity. Although it is not necessary to run a preprocessing step extremely fast, minutes of preprocessing accommodate an artist's workflow better than hours.

3.4 Relaxation

Each relaxation step computes the position $m_i[x,y]$ at iteration $i+1$ of every grid vertex, given the positions of all vertices at iteration i. The force that moves a vertex arises from simulated springs connecting it to its eight grid neighbors. The horizontal and vertical springs have rest length $1/g$. They maintain the local scale. The diagonals have rest length $\frac{1}{g\sqrt{2}}$ and maintain local angles. Together, these lead to quasi-conformality by resisting changes to either.

Real springs experience force linearly proportional to the displacement from their rest lengths. Real springs also oscillate, and simulated springs can even gain energy if the time step is too large. We experimentally found that although linear force does lead to convergence, increasing the order of the force produced faster convergence and allowed stability at larger time steps. A $k = 6$th order force is a practical value for trading oscillation against convergence speed. Intuitively, this causes highly displaced springs to pull much harder, thus correcting the most distorted areas first.

We define spring force $\vec{F}$ to have magnitude proportional to spring displacement raised to k and direction opposite the displacement. The net force at the vertex with indices (x,y) is the sum of the forces over all of its neighbors at vertices $\{(a,b)\}$:

$$\vec{F}_i[x,y] = \sum_{neighbor\ (a,b)} \frac{\vec{D}}{||\vec{D}||} \delta(x,y,a,b) * |\delta(x,y,a,b)|^{k-1}, \tag{3}$$

where direction $\vec{D} = \vec{m}_i[x,y] - \vec{m}_i[a,b]$ is the 3D vector between the neighbors.

Finally, integrate the positions forward in time and snap them to the relief map surface,

$$\vec{m}_{i+1}[x,y] = \text{snap}(\vec{m}_i[x,y] + \Delta t \cdot \vec{F}_i[x,y]), \tag{4}$$

where $\text{snap}(\vec{q}) = (\vec{q}_u, \vec{q}_v, R\vec{q}))$.

We choose the step size Δt to be less than a fraction (we use 1/10) of the rest length of the smallest spring, divided by the largest force to be applied:

$$\Delta t = (10g \cdot \max_{x,y} \vec{F}_t[x,y])^{-1}. \tag{5}$$

This helps prevent oscillation by never allowing any individual spring to move more than 1/10 of its ideal length.

Two shortcuts can be made when implementing the optimizer that together cut more than half of the computation. First, it is only necessary to compute half of the spring forces explicitly because the others can be obtained by negating the corresponding force from a neighbor vertex. Second, vertices are snapped to the relief map after simulation, so only the in-plane components need to be simulated.

3.5 Surface Distance

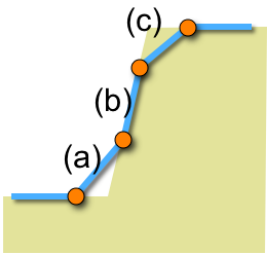

Figure 3: Cut corners

In equation 3, force is defined in terms of $\delta(x,y,a,b)$, the change from rest length of the spring between vertices with indices $[x,y]$ and $[a,b]$. The rest length of that spring is $\frac{1}{g^2} \cdot ||(x,y) - (a,b)||$. The current spring length is *not* the straight-line distance $\vec{m}[x,y] - \vec{m}[a,b]$ because the spring must travel along the surface of R. We want the "on the ground" (i.e., surface) distance, not the "through the air" distance. If we did not take distance travelled along the surface into account, then the convergence configuration of the mesh would not necessarily be close to the true shape of a mesh following the relief.

Figure 3 shows a side view of the problem that we seek to avoid with surface distance. When vertices in $\vec{m}$ (circles) slide across B (solid background), springs (lines) may either correctly lie along the geometric mesh as on edge (b), or shortcut concave (a) or convex (c) corners, giving an incorrect distance metric during optimization.

The surface distance is mathematically the arc length L of a curve from $\vec{\alpha} = m[x,y]$ to $\vec{\beta} = m[a,b]$. This is different than the geodesic distance, which is the *shortest* path across the relief; surface distance is the length of a planar shortest path after projection onto the relief. The path integral from $\vec{\alpha}$ to $\vec{\beta}$ is given by

$$L(\vec{\alpha}, \vec{\beta}) = \tag{6}$$
$$\int_0^1 || \left(\vec{\beta}_u - \vec{\alpha}_u, \vec{\beta}_v - \vec{\alpha}_v, \vec{\nabla} B(\vec{\alpha}_{uv} + \psi(\vec{\beta} - \vec{\alpha})) \cdot (\vec{\beta} - \vec{\alpha})_{uv} \right) || d\psi.$$

This can be approximated by numerical integration, that is, walking across the bump map in small steps and measuring the piecewise line segments. All of our results were computed with the integration step size at one-quarter of the 2D straight-line length of the spring, clamped to the range [2, 10].

3.6 Compute M and M^{-1}

After the final scale of relaxation has converged, the mesh encodes the inverse mapping:

$$M^{-1}(s,t) = (u,v) = \vec{m}[s \cdot g, t \cdot g]_{uv}, \tag{7}$$

where we assume that $\vec{m}$ is bilinearly interpolated between its integer indices.

We want the forward mapping, but symbolically inverting the $\vec{m}$ grid is hard. Fortunately, regularly sampling across a mesh in which vertices and labels have been swapped corresponds directly to a GPU operation: rasterizing a mesh with per-vertex labels.

We configure the GPU with an offscreen render target and orthographic projection matrix. We then render a distorted grid of quadrilaterals. The grid adjacency follows the $[x,y]$ indexing scheme of array $\vec{m}$. The vertex *position* corresponding to $[x,y]$ is $\vec{m}[x,y]_{uv}$ and vertex *color* at $[x,y]$ is $(x/g, y/g, 0)$. After rasterization, the render target now stores a discretized $M(u,v)$, which we convert it the indirection map $I(u,v)$ by equation 1. On a GPU, I can be efficiently packed into a two-channel, 8-bit luminance-alpha texture map. Like a normal map, the values must be scaled and biased to fit into the unsigned [0, 1] value range and use it effectively. Thus, the actual values in the indirection texture map are $I_{pack} = (I + \min_{u,v} I) * (\max_{u,v} I - \min_{u,v} I)$, which are unpacked by a single multiply-add instruction at runtime.

4 Real-time Rendering

Rendering with our parameterization is simple: replace each texture coordinate (u,v) with $(u,v) + I(u,v)$ in a pixel shader.

For example, to extend POM [Tatarchuk 2006] with an indirection map, leave the geometry and parallax offset unmodified. Replace the texture coordinate after parallax distortion but before fetching the material properties from their texture maps. Briefly, following figure 4: (a) Linearly march the tangent-space position along the eye vector until reaching a point below the bump map B, then use linear interpolation to step back close to the true intersection. This gives tangent-space point $(u,v,B(u,v))$ on the displaced surface. (b) Offset the texture coordinate to $(s,t) = (u,v) + I(u,v)$. (c) Shade using the original bump-mapped normal $N(u,v)$ and material properties $C(s,t)$. Note that we do not indirect the bump or normal maps; they define the heightfield and therefore need no additional resolution on steep slopes.

The dimensions of I need not match the dimensions of the material texture or relief map. Bilinear interpolation and MIP-mapping will automatically adjust the values to the appropriate resolution. For the color texture, OpenGL's (and DirectX's) MIP-map functions are defined to automatically compute the screen-space derivative of an indirect texture access and select the correct MIP-map level for the material textures.

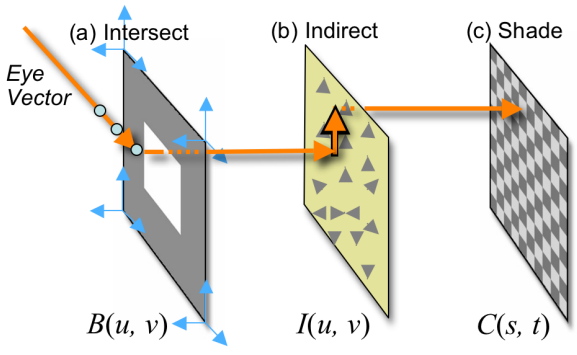

Figure 4: Parallax occlusion mapping on bump texture map B and material texture map C extended with indirection texture map I.

5 Texture Authoring

For any relief technique, there are two kinds of textures to manually author: relief maps (B) and material maps (C). We use traditional relief maps, that can be authored with any existing method. These include 3D tools like Z-brush, direct heightfield painting in Photoshop, depth rendering from a 3D modeling program, and terrain-specific modeling with tools like Terragen and WorldMachine.

Depending on the nature of the material texture maps, the authoring process may change. We consider three cases: unstructured, structured, and paletted-unstructured material texture maps.

5.1 Unstructured

Unstructured textures are ones like wood grain, grass, dust, and scratches that are not particularly sensitive to orientation or alignment relative to the relief features. For these, no additional work is necessary on the authoring side. Paint the textures as you normally would for a flat surface, e.g., in Photoshop, and the runtime engine will apply them correctly. Our parameterization will shift how these textures align with the features, but the point of an unstructured texture is that this shift is acceptable. Figure 1 uses an unstructured texture. Note that the cracks and scratches are aligned slightly differently in the left and right images.

5.2 Structured

Structured textures contain recognizable material features that must align with relief features, for example, eyes on a human face. If we were to change the underlying parameterization without altering the texture, the texture would appear to slip off the features.

The optimizer produces both the forward mapping M used at runtime and its inverse, M^{-1}. Pre-warping the texture by M^{-1} will exactly correct for the distortion applied when rendering. To do so, resample the texture so that the value at location (s,t) in the output texture is sampled from location $M^{-1}(s,t) = \vec{m}[s \cdot g, t \cdot g]_{uv}$ in the input texture.

The point of pre-warping is that it increases the amount of texture area devoted to the sides of features and slightly decreases the amount for flat surfaces. We could render an atlas from M^{-1}, but it is hard to paint a pre-warped texture. Instead we recommend a process similar to that used in creating the relief maps. Often a modeler builds a very high-polygon (e.g., 2M triangle) mesh and then reduces it to a low-polygon (e.g., 5k triangle) mesh and a relief map computed from their difference. We can do something similar with a texture. Paint textures at very high resolution, pre-warp them, and then resize down to the desired runtime resolution. When painting the high resolution version, paint as if planar projecting. That is, compress the detail for slopes as if they were viewed from directly above. For realistic terrain this process has conveniently been performed for us—high-resolution satellite or surveillance plane images are exactly the kinds of projected textures that the pre-warp algorithm takes as input.

5.3 Paletted-Unstructured

Terrains are often represented by a set of high-frequency, tiling unstructured textures and a low-frequency blend map that stretches across the entire terrain. We consider this a hybrid case. The blend map is like a palette—it specifies which materials cover what areas. For example, a typical blend map might have a base layer that specifies grass, but fades to dirt on cliff-sides and to rock and snow as the elevation rises. During rendering, a shader uses the blend map to combine contributions from each of the unstructured textures.

For such cases, leave the unstructured textures unmodified, but treat the blend map as if it were a structured texture. Paint (or compute) it at high resolution, and then pre-warp by M^{-1}. Thus

the blending weights will achieve approximately uniform resolution over the surface but stay aligned with relief features, and the unstructured textures will properly conform to that surface.

6 Results

Figure 6 shows the bump, normal, and indirection maps computed by our system for a relief map of statues downloaded from http://fabio.policarpo.nom.br/files/reliefmap3.zip that originally appeared as a bump map in [Policarpo and Oliveira 2006]. We visualize the indirection map with yellow as horizontal displacement and blue as vertical displacement.

Figure 5 shows the initial coarse spring network (a) and progressively finer spring networks during relaxation, until the final result (f) for the statue relief map. Springs are shown in red when they are stretched and black when they are near their rest length. Stretched, red springs correspond to areas that will exhibit texture stretch artifacts. Most of the springs are black because each subimage was captured after the optimizer converged at that level, so there is little stress in the springs. The final result (f) still has some stretch in the overconstrained region near the top, however it compares favorably to the texture distribution in the original mesh at the same resolution (g). Without our parameterization, the long vertical areas in (g) would all have low texture resolution. The total relaxation time for this mesh was 64s on one core of an AMD 4200 processor. Figure 8(a) shows a highly structured texture. Running the optimizer redistributes that texture over the relief, creating uniform resolution but creating misalignment with features (b). Our tool distorts the high-resolution texture by M^{-1} and then downsamples the result to create a warped texture (c), that gives both correct appearance and uniform resolution when applied (d) at runtime.

Figure 7(a-d) shows a subway station that we modeled entirely using indirection-mapped POM surfaces. The relief on the ceiling, train tracks, and tile floor are due to POM. Note texture detail on the sides of steep relief features, like the inset in the ceiling and the track grooves. The actual scene geometry is low resolution, as shown in the untextured rendering (e).

Figure 10 shows results for several difficult relief maps with near-vertical features. The left four columns show the data used at runtime and the images on the right are the rendered 3D views using POM. Rows 1 and 2 are cases with near-vertical edges on both boxes and cubes. The checkerboard texture demonstrates that the resulting parameterization is quasi-conformal because the rendered checkers appear approximately square and of the same scale everywhere. Row 3 shows a stone wall winding through a meadow. The color texture C is structured and has been pre-warped so that it covers the sides of the wall. If we had not used a pre-warped texture then the grass would have been pulled up onto the wall. Row 4 shows a complex, relief map with a structured texture.

For these results, optimizer times ranged from 10s to about two minutes. The longest time was for the lion wall downloaded from http://graphics.cs.brown.edu/games/SteepParallax/bump-maps.zip. That relief map has size 1024×1024, however, a $g = 256$ spring mesh (i.e., about 260,000 springs) accurately represents the features.

All 3D images in this paper were rendered in real time. We can shade every pixel on the screen at 1440×900 screen resolution with a GeForce 8800 GTX GPU at over 320 fps for indirected POM rendering with 100 steps in the ray marcher. The terrain renders at 500 fps with an indirection texture. The subway station scene renders at about 50 fps, with 20 light sources, shadows, and indirection relief mapping on every large surface.

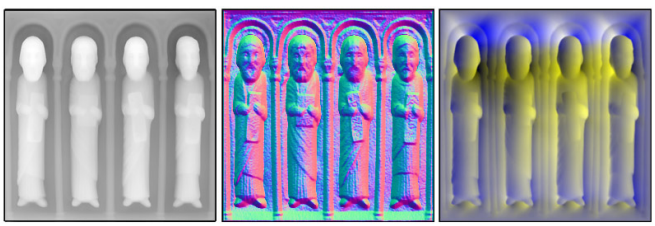

Figure 6: Corresponding bump, normal, and indirection maps.

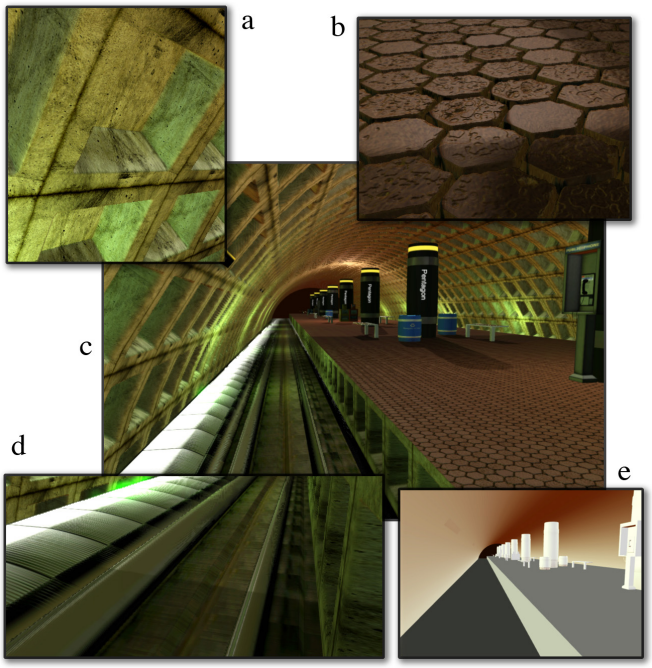

Figure 7: (a-d) Subway station with all surfaces rendered using POM with indirection mapping. (e) The underlying base geometry contains very few polygons.

Figure 8: (a) Original structured texture (b) indirection mapped without correction creates uniform sampling but distorts structure. (c) Pre-warping (d) achieves both uniform sampling and structure.

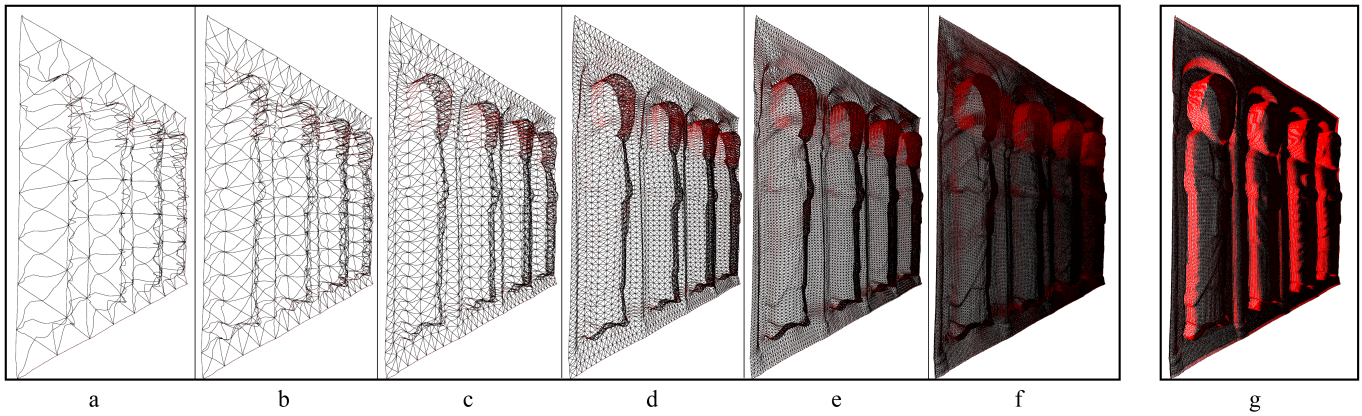

Figure 5: (a-f) The relaxation spring network after convergence, at levels from $g = 8$ up to $g = 256$. Red (gray if reading a monochrome printout) represents springs experiencing large forces. Note that the springs correctly follow the relief surface. (g) The unoptimized mesh at $g = 256$ shows distortions of the parameterization on steep slopes; these are the locations at which artifacts will appear.

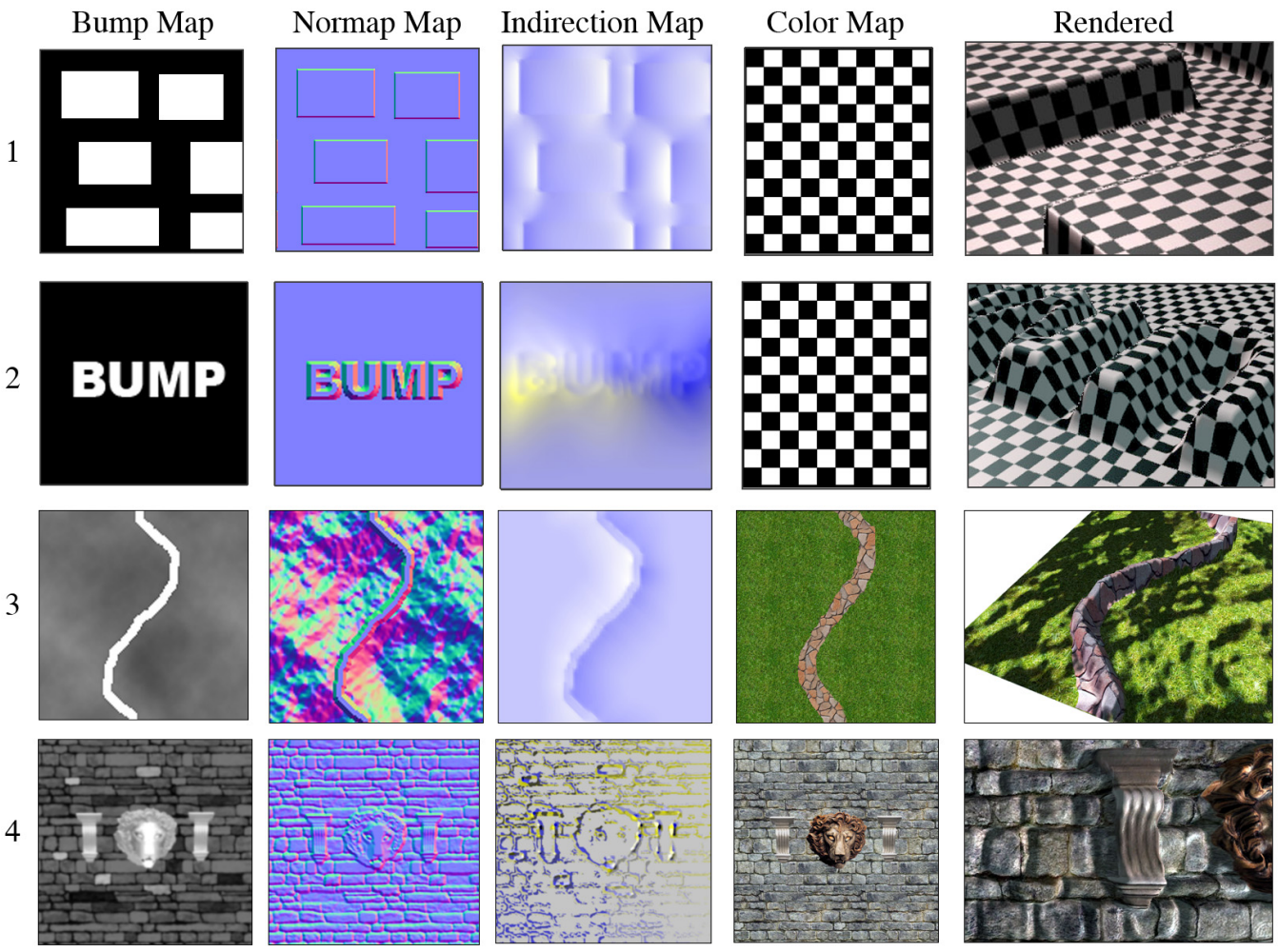

Figure 10: Surfaces rendered with POM with indirection mapping. From left to right: bump, normal, indirection, and color texture maps; and a rendered view. The 3D view shows a single quadrilateral–the appearance of depth is entirely from displacement in the pixel shader. Note the uniform texture appearance for curved, straight, and complex shapes on both flat and sloped features.

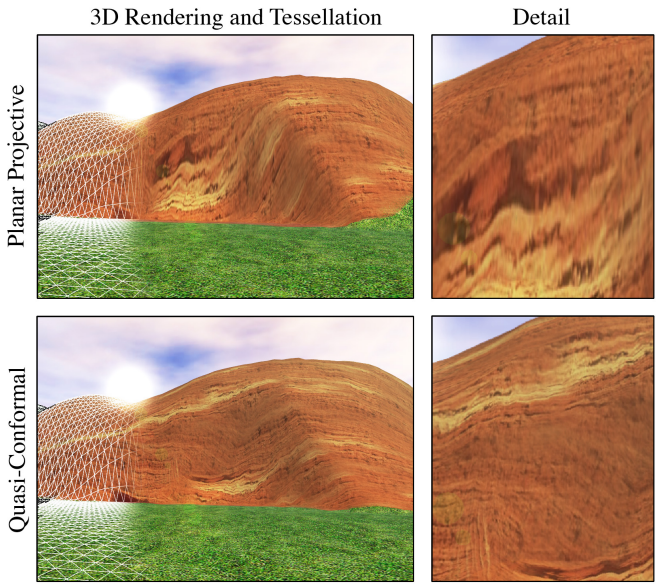

Figure 9: *Top:* Heightfield terrain... *Bottom:* improved by indirection mapping, with color from a paletted-unstructured texture.

7 Future Work

We used the GPU's rasterization capability to simplify the process of inverting a complex mapping and to pre-warp structured textures. It is natural to consider porting the relaxation step to the GPU as well. Relaxation is well-suited to a GPU implementation since it operates on a regular grid and only uses information from immediately adjacent nodes.

We assumed that distributing stress evenly over the spring network was desirable. However, there may be cases where artists are willing to tolerate differing levels of non-conformality for across a model. Consider the texture map for a clothed human figure. Areas with little detail, such as a solid-color shirt, better tolerate texture stretch than areas with high detail, like the person's face. One way to encode this notion would be to use the local frequency composition of the texture to create springs with different spring constants.

We believe that indirection mapping will remain applicable for a long time. Displacement mapping methods have been a staple of computer graphics nearly since its inception. We have focused our results on POM because it is rapidly gaining popularity in the games industry. Geometry shaders and other hardware tessellation mechanisms may someday obsolete POM. However, heightfields (see figure 9) and true geometry displacement mapping both benefit from our quasi-conformal relief parameterizations and are likely to remain essential techniques.

Acknowledgments

We thank NVIDIA for donating the rendering hardware, Mihai Stoiciu for explaining the complex analysis implications, Sara Carian for proofreading, and the anonymous reviewers for their presentation suggestions for this paper.

References

BRAWLEY, Z., AND TATARCHUK, N. 2004. Self-shadowing,perspective-correct bump mapping using reverse height map tracing. *ShaderX*^{3}, 135.

CIGNONI, P., GANOVELLI, F., GOBBETTI, E., MARTON, F., PONCHIO, F., AND SCOPIGNO, R., 2003. BDAM – batched dynamic adaptive meshes for high performance terrain visualization.

DACHSBACHER, C., AND TATARCHUK, N. 2007. Extended parallax occlusion mapping with accurate silhouette computation. *I3D 2007 Poster*.

DONNELLY, W. 2005. Per-pixel displacement mapping with distance functions. In *GPU Gems 2*, Addison-Wesley, M. Pharr, Ed., 123–136.

DUCHAINEAU, M., WOLINSKY, M., SIGETI, D. E., MILLER, M. C., ALDRICH, C., AND MINEEV-WEINSTEIN, M. B. 1997. Roaming terrain: real-time optimally adapting meshes. In *Proc. of Visualization '97*, IEEE Computer Society Press, Los Alamitos, CA, USA, 81–88.

KANEKO, T., TAKAHEI, T., INAMI, M., KAWAKAMI, N., YANAGIDA, Y., MAEDA, T., AND TACHI, S. 2001. Detailed shape representation with parallax mapping. In *Proceedings of ICAT 2001*, 205–208.

KHODAKOVSKY, A., LITKE, N., AND SCHRÖDER, P. 2003. Globally smooth parameterizations with low distortion. In *SIGGRAPH '03*, ACM Press, New York, NY, USA, 350–357.

KRISHNAMURTHY, V., AND LEVOY, M. 1996. Fitting smooth surfaces to dense polygon meshes. In *SIGGRAPH '96*, ACM Press, New York, NY, USA, 313–324.

LENGYEL, E. 2003. Mathematics for 3d game programming & computer graphics. Section 6.8.3.

LÉVY, B., PETITJEAN, S., RAY, N., AND MAILLOT, J. 2002. Least squares conformal maps for automatic texture atlas generation. *ACM Trans. Graph. 21*, 3, 362–371.

LOMERSON, A., 2007, June. Senior Artist, Activision (Vicarious Visions studio). Personal communication.

MARSDEN, J. E., AND HOFFMAN, M. J. 1999. *Basic Complex Analysis, 3rd Edition*. W. H. Freeman and Company.

MCGUIRE, M., AND MCGUIRE, M. 2005. Steep parallax mapping. *I3D 2005 Poster*.

OLIVEIRA, M., AND POLICARPO, F. 2005. An efficient representation for surface details. Tech. Rep. RP-351, UFRGS, January.

OLIVEIRA, M. M., BISHOP, G., AND MCALLISTER, D. 2000. Relief texture mapping. In *SIGGRAPH '00*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 359–368.

POLICARPO, F., AND OLIVEIRA, M. M. 2006. Relief mapping of non-height-field surface details. In *I3D '06*, ACM Press, New York, NY, USA, 55–62.

POLICARPO, F., OLIVEIRA, M. M., AND COMBA, J. L. D. 2005. Real-time relief mapping on arbitrary polygonal surfaces. In *I3D '05*, ACM Press, New York, NY, USA, 155–162.

PURNOMO, B., COHEN, J. D., AND KUMAR, S. 2004. Seamless texture atlases. In *Proc. of the 2004 Symposium on Geometry Processing*, ACM Press, New York, NY, USA, 65–74.

RAY, N., LI, W. C., LÉVY, B., SHEFFER, A., AND ALLIEZ, P. 2006. Periodic global parameterization. *ACM Trans. Graph. 25*, 4, 1460–1485.

SANDER, P. V., SNYDER, J., GORTLER, S. J., AND HOPPE, H. 2001. Texture mapping progressive meshes. In *SIGGRAPH '01*, ACM Press, New York, NY, USA, 409–416.

SANDER, P. V., GORTLER, S. J., SNYDER, J., AND HOPPE, H. 2002. Signal-specialized parametrization. In *EGRW '02*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 87–98.

SLOAN, P.-P., WEINSTEIN, D. M., AND BREDERSON, J. D. 1998. Importance driven texture coordinate optimization. *Computer Graphics Forum 17*, 3, 97–104.

TATARCHUK, N. 2006. Dynamic parallax occlusion mapping with approximate soft shadows. In *Proc. of I3D '06*, ACM Press, New York, NY, USA, 63–69.

TERZOPOULOS, D., AND VASILESCU, M. 1991. Sampling and reconstruction with adaptive meshes. In *CVPR*, 70–75.

WANG, L., WANG, X., TONG, X., LIN, S., HU, S., GUO, B., AND SHUM, H.-Y. 2003. View-dependent displacement mapping. In *SIGGRAPH '03*, ACM Press, New York, NY, USA, 334–339.

WANG, X., TONG, X., LIN, S., HU, S., GUO, B., AND SHUM, H.-Y. 2004. Generalized displacement maps.

WANG, L., GU, X., MUELLER, K., AND YAU, S.-T. 2005. Uniform texture synthesis and texture mapping using global parameterization. In *The Visual Computer*, vol. 21, 801–810.