Interactive On-Surface Signal Deformation

Tobias Ritschel[1] Thorsten Thormählen[1] Carsten Dachsbacher[2] Jan Kautz[3] Hans-Peter Seidel[1]

[1]MPI Informatik [2]VISUS / Universität Stuttgart [3]University College London

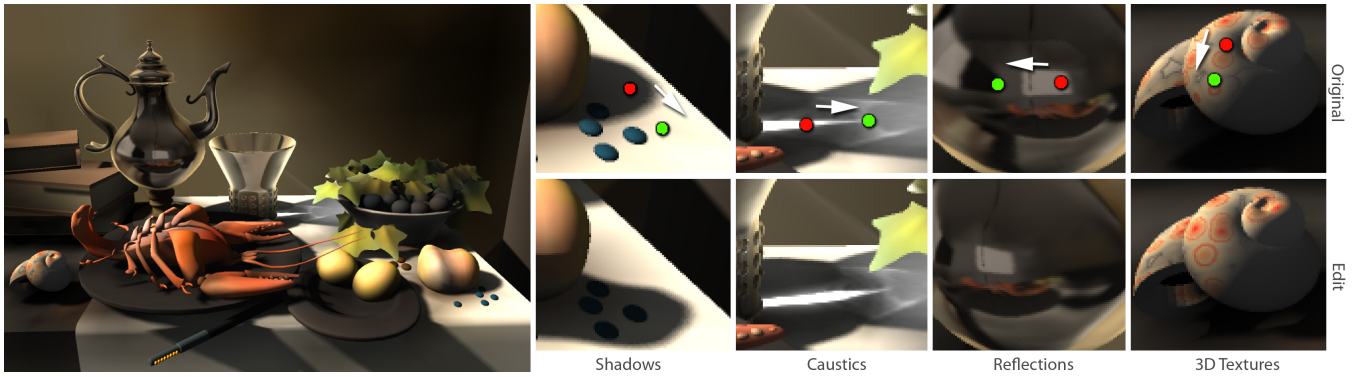

Figure 1: *Our framework allows artists to interactively edit (a) shadows, (b) caustics, (c) reflections, and (d) 3D-textures with on-surface deformations (7 fps and 11K deformation points). The user sets deformation constraints and the system interactively updates the rendering.*

Abstract

We present an interactive system for the artistic control of visual phenomena visible on surfaces. Our method allows the user to intuitively reposition shadows, caustics, and indirect illumination using a simple click-and-drag user interface working directly on surfaces. In contrast to previous approaches, the positions of the lights or objects in the scene remain unchanged, enabling localized edits of individual shading components. Our method facilitates the editing by computing a mapping from one surface location to another. Based on this mapping, we can not only edit shadows, caustics, and indirect illumination but also other surface properties, such as color or texture, in a unified way. This is achieved using an intuitive user-interface that allows the user to specify position constraints with drag-and-drop or sketching operations directly on the surface. Our approach requires no explicit surface parametrization and handles scenes with arbitrary topology. We demonstrate the applicability of the approach to interactive editing of shadows, reflections, refractions, textures, caustics, and diffuse indirect light. The effectiveness of the system to achieve an artistic goal is evaluated by a user study.

Keywords: intuitive editing, deformation, shadows, light design, texture, real-time rendering, graphics hardware

1 Introduction

With the advance of efficient rendering techniques and powerful graphics hardware, tools that enable the modification of computer-generated 3D scenes with immediate visual feedback are now available to artists [Pellacini et al. 2002; Ragan-Kelley et al. 2007; Obert et al. 2008]. Many of these techniques try to keep the resulting renderings physically correct; for instance, by repositioning light sources to cast an artist-defined shadow. However, artists are not necessarily interested in a physically correct rendering of the scene but rather want to achieve a certain artistic goal. Thus, they often revert to generating multiple renderings (2D layers) of the scene, which are later manipulated, enhanced, and composited in an off-line process [Birn 2006]. Of course, this approach is limited to simple 2D manipulations, and complex modifications, such as moving a shadow onto an adjacent object, are difficult or even impossible to support.

We propose a system to interactively perform such tweaks, including shadows, caustics, reflections, 3D textures, and indirect lighting, directly over scene surfaces in one framework. We argue that the most intuitive way of translating and deforming surface signals is to imagine them being painted on a piece of "elastic cloth" that can be dragged over the surface: the user can fix one or more points of the cloth that should not move, and drag others to move or deform the signal. Although this does not result in physically accurate modifications, it provides artists with the flexibility to modify only some shading components (e.g., shadows and not caustics) as well as the possibility to adjust them only locally. While this is a simple editing paradigm, it is challenging to achieve: the method must work in a meshless fashion (for instance, to support dragging a shadow from one object to another); discontinuous deformations need to be supported, as the user should be able to move a surface-signal across edges (such as from the floor onto a wall); and finally, the deformed surface must remain on the original surface, or artifacts such as self-shadowing may occur. We propose a novel surface deformation approach that overcomes these challenges and that offers many desirable properties: it can be computed in real-time, it is decoupled from the underlying geometric representation, and it affords an intuitive editing metaphor.

In summary, this paper makes the following contributions:

- An efficient, interactive, "what you see is what you get" deformation interface of on-surface signals.
- A unified method to deform many different shading components, like shadows, caustics, reflections, 3D textures, and indirect lighting.
- A meshless, multi-resolution GPU approach to compute on-surface deformations.

- A study of user task performance when editing on-surface signals with our system.

2 Related Work

Editing Light Interfaces for editing of light and shadows have been researched over the past two decades. Very early on, Poulin and Fournier [1992] suggested to infer light source positions from modified highlights and shadow volumes. Several methods have been proposed to use painting interfaces to infer and adjust light source intensities and positions [Schoeneman et al. 1993; Poulin et al. 1997] as well as to derive plausible environment maps [Okabe et al. 2007]. Direct interfaces, where the user directly modifies light sources, are probably the most widely used ones, as they are readily available in commercial editing tools, such as Autodesk Maya or Maxon Cinema 4D. Indirect interfaces exist as well, where the user is allowed to click-and-drag shadows, which in turn adjusts the light source [Pellacini et al. 2002]. Kerr and Pellacini [2009] have studied these different interfaces for lighting design and have found that painting interfaces are significantly less intuitive than direct or indirect editing interfaces.

We propose an indirect editing interface for the modification of various surface signals, including shadows and shading. In contrast to the methods above, our goal is not to relight the entire scene in order to match user input. We rather want to give the artist the flexibility to adjust the location of individual shading components, such as shadows or indirect illumination. This is more in the spirit of the work by Barzel [1997] (as used in Toy Story), who proposed a lighting model to support artist-driven modifications of shadows and lighting; for instance, this model allowed the user to move shadows independently of shading.

In non-photorealistic rendering, an approach was proposed by DeCoro et al. [2007] to edit and stylize shadows; e.g., by inflating or deflating it. Similarly, a method for editing stylized shading has been demonstrated [Todo et al. 2007].

While editing of direct illumination and shadowing has received most attention, there is also some work on editing other shading components. Tabellion and Lamorlette [2004] enable an artist to change the falloff and hue of indirect illumination through shader manipulation. The method of Obert et al. [2008] provides means to modify the indirect light transport in a general manner and they demonstrate changing the hue, saturation, and falloff. Our goal is different: we want to enable an artist to modify where indirect illumination appears, which we essentially achieve by mapping it to different surface locations through an intuitive user interface. Other modifications are orthogonal to our approach. Recently, Ritschel et al. [2009] presented an approach for the modification of reflections. A user can place constraints on the reflection and appropriate modified reflection directions are inferred. Our approach is similar in spirit. A user can impose several constraints and the shading component is transformed accordingly. In fact, we demonstrate that our approach can also be used for editing reflections.

Deformation While intuitive deformation of surfaces in 3D space has received much interest [Zorin et al. 1997; Kobbelt et al. 1998; Botsch and Kobbelt 2004; Müller et al. 2005; Sumner et al. 2007], there is little or no work considering the deformation of shapes *on* a 3D surface. However, some of the surface deformation methods are closely related to our work. Sumner et al. [2007] proposed embedded deformations to generate smooth and intuitive deformation fields. To this end, local deformation "nodes" on the surface are derived that then follow user-constraints. Consistency across the shape is ensured by regularization. Actual vertices are deformed by a linear combination of the deformed nodes. Our approach proceeds in a similar fashion but without an explicit graph structure and a different objective function that ensures that the deformation remains *on* the surface. Meshless deformations were introduced by Müller et al. [2005]. Similar to our work, deformations are directly computed over a point cloud without requiring connectivity information. However, we support deformations that remain on the objects' surfaces.

Deformation over a surface can also be understood as a reparametrization of a manifold by another manifold as described in the work by Schreiner et al. [2004] targeting off-line computation. Our work differs, since we do not need to match arbitrary surfaces but are interested in mapping a surface or collection of surfaces onto themselves under given user constraints. This allows for a more efficient solution enabling interactive editing.

Similarly, (constrained) mesh parametrization [Lévy and Mallet 1998; Zwicker et al. 2002; Schmidt et al. 2006; Hormann et al. 2007; Tzur and Tal 2009] could be used to derive an appropriate mapping. However, our goal is to allow users to make edits *across* different objects (e. g., a shadow dragged from the floor onto the adjacent wall). We therefore opt to use a meshless approach, to which mesh parametrization methods do not apply. A discrete approximation of the exponential map [Schmidt et al. 2006] can be used to compute local parametrizations for texture mapping. Our goal is different: we do not want to texture map an object or scene from scratch but rather deform existing surface signals.

Our approach also bears some resemblance to deformable shape matching algorithms [Li et al. 2009], as these try to maintain the intrinsic metric structure of a surface when matching another surfaces [Bronstein et al. 2006].

3 Editing Metaphor and User Interaction

The goal of our work is to enable an artist to easily and intuitively deform shading components, such as caustics or texture, over the surface. Common editing operations would include, for instance, dragging parts of a shadow from one location to another; or modifying the shape of a caustic to be more visually pleasing. To this end, we propose to use the metaphor of a virtual piece of cloth that allows for intuitive deformation of shading components covering the scene surfaces. We argue that users have a certain expectation or knowledge how a piece of cloth would behave when deformed over a surface. Users manipulate this virtual piece of cloth through constraints and regions of influence. We support simple point constraints as well as sketch-based constraints, see Figure 2 for some examples. More specifically, we support the following types of user interaction.

Constraints The user can generate a constraint by clicking on a 3D surface at location $\mathbf{x}_c$ and dragging it to some new location $\mathbf{y}_c$, which can be on the same as well as on a different surface. I.e., we want the surface signal to move from $\mathbf{x}_c$ to $\mathbf{y}_c$: $\mathbf{x}_c \rightarrow \mathbf{y}_c$. On the first click both constraint points $\mathbf{x}_c$ and $\mathbf{y}_c$ are generated at the surface position under the mouse pointer, i.e., $\mathbf{x}_c = \mathbf{y}_c$ in the beginning. Both constraint points are visualized by handles: a red one for the original surface position $\mathbf{x}_c$ and a green one for the deformed surface position $\mathbf{y}_c$ (see Fig. 2). The user can now drag the green handle over the surface away from the original position, defining a new position for $\mathbf{y}_c$. Our implementation ensures that $\mathbf{y}_c$ is always located on a surface, and is not floating in free space. While the user drags the constraint, the surface signal is following the mouse movements exactly. Thus, a single constraint is sufficient to specify a simple translation over the surface. Using multiple constraints allows the user to specify more complicated deformations, e.g., operations like rotation and scaling for two constraints, or arbitrary

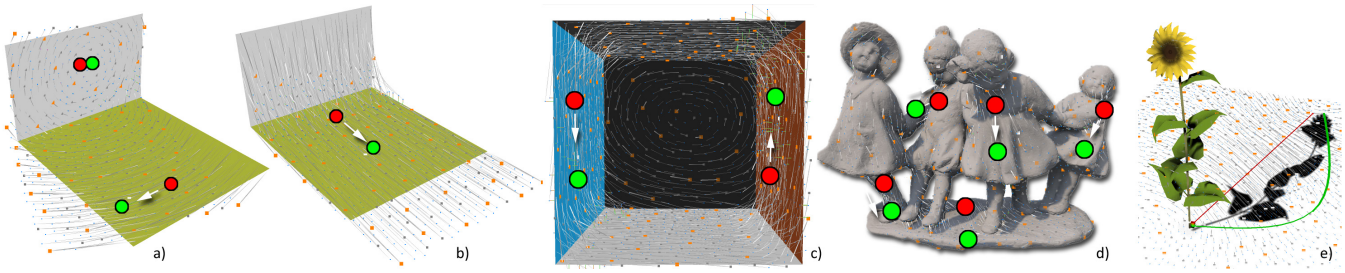

Figure 2: *Visualization of the on-surface deformation under given user constraints. Thin lines indicate how the surface signal deforms. (a) An example using two constraints. Note, how the translation of a constraint handle results in a rotational deformation field. (b) Moving of a single constraint results in a translation. (c) Two opposing constraints in a closed environment lead to a rotational deformation. (d) An edit on a complex geometry. Note, how the deformation field is smooth nonetheless. (e) A sketch-based edit.*

deformations for more constraints. As mention before, we intend the resulting deformation to behave roughly like a piece of cloth in order to make it intuitive for the user.

Regions The user can limit the influence of the deformation by placing a region of influence (defined by a center point and an Euclidian or geodesic radius). We then automatically generate positional constraints around the boundary of this region to ensure continuity. The deformation is only computed within this boundary, speeding up the computation.

Sketching We also provide a sketching interface to impose constraints. The user first sketches a curve (B-Spline) over the undeformed surface signal and a second curve, which specifies where the signal under the first curve should move to, see Figure 2. The two curves are turned into a set of constraints by sampling both B-Splines and imposing $\mathbf{x}_{c_n} \rightarrow \mathbf{y}_{c_n}$ for each sampled location n.

Shading Component The user can select which of the shading components to edit: shadows, reflections, refractions, texture, caustics, and indirect illumination. Note, that we support decoupled edits; i. e., different edits can be applied to different components.

4 On-surface Deformation

We will now introduce a more rigorous definition of our editing approach. We assume that the color or shading of a surface, $\mathcal{S} \subset \mathbb{R}^3$, is given by a function $f(\mathbf{x})$ mapping an arbitrary point, $\mathbf{x} \in \mathbb{R}^3$, to a color value. When rendering an object, this function is evaluated for points on the surface, $\mathbf{x} \in \mathcal{S}$. In the simplest case, this function represents a 3D texture and returns the color obtained from a volume texture at $\mathbf{x}$. However, it can also encode global information about the scene and, for instance, represent a shadow test yielding $f(\mathbf{x}) = 0$ if the point $\mathbf{x}$ is in shadow, and 1 otherwise. Deforming the surface signal means that we want to observe the result $f(\mathbf{x})$ at another location $\mathbf{y}$. Given the deformation mapping $g(\mathbf{x}) = \mathbf{y}$, which says that $\mathbf{x}$ should move to $\mathbf{y}$ ($\forall \mathbf{x} \in \mathcal{S}$), the result at $\mathbf{y}$ after deformation is then $f(g^{-1}(\mathbf{y}))$. Our main challenge is to efficiently derive the deformation function $g(\mathbf{x})$. Several properties of $g(\mathbf{x})$ are crucial for on-surface deformation and guide its optimization:

- It has to be editable in an interactive and intuitive fashion, and follow the artist's controls similar to a piece of cloth.
- The deformation has to be smooth (to prevent objectionable artifacts) and invertible.
- The deformation $g(\mathbf{x}) = \mathbf{y}$ must map points on the surface $\mathbf{x} \in \mathcal{S}$ to other points on the surface $\mathbf{y} \in \mathcal{S}$, i.e., $g(\mathbf{x}) : \mathcal{S} \rightarrow \mathcal{S}$.

In particular, the last property is of utmost importance for on-surface deformation and the fundamental difference to previous approaches performing geometric deformations. Consider the example of $f(\mathbf{x})$ being the shadow test: if $g(\mathbf{x}) : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ is an arbitrary smooth mapping, e.g., bending, that does not map points on the surface to the surface itself, then it would cause unintended self-shadowing (see Fig. 3). If the user deforms the shadow on the surface, or drags it over the surface, we have to ensure that we replace shadow tests at one surface location by a shadow test from another location on the surface – and not with shadow tests at arbitrary points in space.

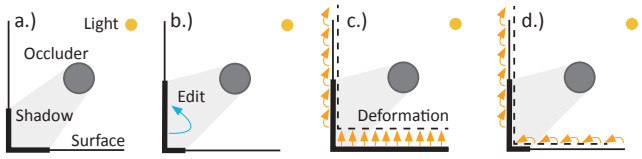

Figure 3: *An object casts a shadow on a wall (a). A user drags the shadow up the wall (b). Common surface deformation will result in self shadows on the ground (c). On-surface deformation will make the shadow slide over the surface (d).*

4.1 Meshless Deformation Representation

The deformation of the shading function $f(\mathbf{x})$ over $\mathcal{S}$ is captured and edited using the aforementioned virtual cloth metaphor. We will explain the basic algorithm first, which is later extended to a hierarchical method for improved stability and performance.

First, we compute a uniform distribution of *deformation points* on the surfaces of the scene. This initial set of points, denoted as $G = \{\mathbf{g}_i | \mathbf{g}_i \in \mathcal{S} \wedge 0 \leq i < N\}$, is created using the procedure described by Sumner et al. [2007], which ensures that no point has a neighboring point within a radius r, and thus exhibits good blue noise properties. These points can later be used to define constraints, i. e., the user can fix a point to reside at a certain location, or drag it to move or stretch the shading function. In order to solve for the constraints, we use a mass-spring system, similar to cloth simulations [Ko and Breen 2003]; however, with the additional constraint that the points reside on the surface (cf. Sec. 4.2). The initial locations of the points, $\mathbf{g}_i \in G$, define the undeformed state. After the system has been solved for user-imposed constraints, the new locations of the points are denoted as $\tilde{\mathbf{g}}_i \in \tilde{G}$. In contrast to most other approaches, where the cloth springs are related to a specific topology of the masses (points), we link every point to its $N = 18$ nearest neighbors by generating springs with a rest length equal to the Euclidean distance in the undeformed state. The number of springs is motivated by the blue noise properties [Balzer et al. 2009]: every point has on average 6 direct neighbors with a distance of approximately r, and 12 other points with an approximate distance of $2r$ (cp. Fig. 4). The springs with the rest length of ap-

proximately $2r$ allow us to prevent the virtual cloth from (strong) shearing and foldovers.

4.2 Deformation Estimation

The desired mapping function g, represented by the deformation points $\tilde{\mathbf{g}}_i$, has to satisfy three objectives: Firstly, the mapping g has to meet the user constraints c_n (hard constraint):

$$g(\mathbf{x}_{c_n}) = \mathbf{y}_{c_n} \ . \tag{1}$$

Secondly, deformation points in the direct neighborhood $\mathcal{N}$, defined by the N_{dist} nearest neighbors, should maintain their original distance. This requirement prevents (strong) shearing and foldovers and can be translated into an objective function as

$$E_{\text{dist}}(g) = \sum_i \sum_{k \in \mathcal{N}(i)} ||\mathbf{g}_k - \mathbf{g}_i||^2 - ||\tilde{\mathbf{g}}_k - \tilde{\mathbf{g}}_i||^2 \ . \tag{2}$$

Thirdly, all deformation points should be located on the surface. By defining a function $S(\tilde{\mathbf{g}})$ that maps a point in space to the respective closest point on the surface, we can write

$$E_{\text{surf}}(g) = \sum_i ||\tilde{\mathbf{g}}_i - S(\tilde{\mathbf{g}}_i)||^2 \ . \tag{3}$$

Thus, a combined objective function is given by:

$$E(g) = w_{\text{dist}} E_{\text{dist}} + w_{\text{surf}} E_{\text{surf}} \quad \text{subject to} \quad g(\mathbf{x}_{c_n}) = \mathbf{y}_{c_n} \tag{4}$$

We use the weights $w_{\text{dist}} = 0.3$ and $w_{\text{surf}} = 1.0$, and $N_{\text{dist}} = 18$ for all our examples.

This combined objective function E could be optimized with a non-linear least squares solver (as in Sumner at al. [2007]). However, we opt to minimize the objective function E using an implicit solver, as it is very amenable to a GPU implementation. We relax the system in a simple loop over all deformation points that performs three steps for each deformation point, one for each objective (similar to relaxation in cloth simulation [Ko and Breen 2003]), which yields the deformed points $\tilde{G}$. All three steps translate to simple and efficient GPU operations (cf. Section 4.5), which can be executed in parallel for all deformation points.

4.3 Inverse Mapping

The mapping from the initial point distribution G to the deformed distribution $\tilde{G}$ encodes the deformation. However, the deformation function $g(\mathbf{x})$ is only explicitly defined for distinct points $g(\mathbf{g}_i) = \tilde{\mathbf{g}}_i$, but in fact we need the inverse deformation $\mathbf{x} = g^{-1}(\mathbf{y})$ in order to evaluate the shading function at arbitrary points $\mathbf{y}$ on the surface. This inverse deformation can be approximated by a weighted average:

$$\mathbf{x} = g^{-1}(\mathbf{y}) \approx \sum_i w_{\text{gauss}}(|\mathbf{y} - \tilde{\mathbf{g}}_i|) \left(\mathbf{g}_i + \tilde{\mathbf{R}}_i^{-1}(\mathbf{y} - \tilde{\mathbf{g}}_i)\right) , \tag{5}$$

where the weight function w_{gauss} generates a Gaussian fall-off that sums to 1. The rotation matrices $\tilde{\mathbf{R}}_i$ describe the rotation that each deformation point $\tilde{\mathbf{g}}_i$ underwent in relation to its undeformed state (see Müller et al. [2005]). It can be computed for each $\tilde{\mathbf{g}}_i$ from the neighboring N_{dist} deformation points.

This approximation of the inverse deformation function can be evaluated efficiently for all the visible pixels in the scene with a splatting-based approach that is accelerated on the GPU. At each deformed point $\tilde{\mathbf{g}}_i$, a splat is drawn. The first term from Eq. (5), the scalar Gaussian fall-off, is encoded in the alpha-channel and the second (positional) term is encoded in the RGB channels of the splat (evaluated in a fragment shader). At each visible location $\mathbf{y}$, the point splats of the nearby deformed points will overlap $\mathbf{y}$. We blend them together based on their respective alpha-values, which effectively evaluates $\mathbf{x}$ according to Eq. (5) at every location $\mathbf{y}$. The result can simply be read from the framebuffer. This method is very efficient, and the complexity is only linear with the number of deformation points.

4.4 Hierarchical Solution

In contrast to surface deformation methods that create a spatially smooth deformation, our on-surface deformations are only piecewise spatially smooth. For instance, when a shadow is dragged over an edge, such as in Fig. 3, the deformation $g(\mathbf{x})$ is highly discontinuous at the edge (although the shading result might look smooth on the surface). To capture such discontinuities, a high number of deformation points is required. However, a system with many deformation points converges too slowly for interactive applications. To this end, we use a hierarchical relaxation approach (as used in parametrization [Schreiner et al. 2004]).

Analogous to the non-hierarchical approach described before, we first compute a set of undeformed points with a minimal distance of r_M, again denoted as G. From this point set we create the deformation hierarchy: we start by choosing a subset of points from G with a minimum distance of r_0 yielding the coarsest level $H_0 = \{\mathbf{h}_{0,i} | \mathbf{h}_{0,i} \in \mathcal{S} \wedge 0 \leq i < N_0\}$. The selection works as follows: we randomly select a point $\mathbf{g} \in G$, add it to H_0, and flag all points in G within a distance of r_0 from $\mathbf{g}$ (including $\mathbf{g}$ itself) as visited. This is repeated until all points are visited in G. To obtain the respective next finer hierarchy level, H_j, $j > 0$, we half the minimal distance from the previous level, i.e. $r_j = r_0/2^j$. The hierarchical meshless deformation requires that coarse point sets are included in finer ones, i.e. $H_0 \subset H_1 \subset ... \subset H_M$ (see Fig. 4), and consequently we first select all points $\mathbf{g} \in H_{j-1}$ from G, and thereafter continue with the random selection as described above. Note that the minimal distance r_M defines the editing granularity where small values allow more detailed on-surface deformations. The distance r_0 implicitly determines the number of hierarchy levels $l = \lfloor \log_2(r_0/r_M) \rfloor$. In our examples we chose approximately 5% of the scene extent for r_0, and r_M is chosen such that a sufficient resolution at the finest level is obtained. This leads to 5 or 6 levels in practice.

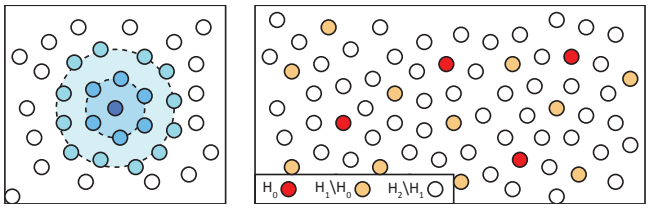

Figure 4: *Blue noise properties:* Left: *every deformation point has on average 6 direct neighbors (and 18 in total within twice the distance),* Right: *multiple levels (red to white) of a deformation hierarchy with blue noise properties in and between levels.*

When solving for the mapping function, we start by computing the solution for the coarser (and smaller) sets of deformation points first, and then continue with the respective next finer one. Consequently, we need to transfer the deformation from coarser to finer point sets after each intermediate solution. To do this, we first express every undeformed point $\mathbf{h}_{j,i} \in H_j \backslash H_{j-1}$ relative to the coarser undeformed points in H_{j-1} using the difference vectors $\mathbf{d}_{j,i} = \mathbf{h}_{j,i} - \mathbf{h}_{j-1,i}$.

After optimization of the coarse set of points H_{j-1} (yielding $\tilde{H}_{j-1}$), we transfer the solution as an initial solution to the next finer level $\tilde{H}_j$. To this end, we estimate the initial new positions of points in

$\tilde{H}_j$ as a weighted average of their relative locations:

$$\tilde{\mathbf{h}}_{j,i} = \sum_{k=0}^{N_{j-1}} w_{j,k}(\tilde{\mathbf{h}}_{j-1,k} + \tilde{\mathbf{R}}_{j-1,k} \cdot \mathbf{d}_{j,k}), \quad (6)$$

where the weights $w_{j,k}$ are the inverse magnitudes of the difference vectors (normalized to 1). Note that we apply the rotation $\tilde{\mathbf{R}}_{j-1,k}$, which the point $\tilde{\mathbf{h}}_{j-1,k}$ underwent during deformation, to the difference vector $\mathbf{d}_{j,k}$. This is required, if rotational movement was present on the coarser level.

The solution for $\tilde{H}_{j-1}$ serves as a starting point for $\tilde{H}_j$, but also as a constraint: in subsequent iterations, the deformation points computed in $\tilde{H}_{j-1}$ are treated as fixed and do not move anymore.

In order to allow the user to set constraints at the finest level, we perform a V-cycle, similar to multigrid methods [Trottenberg et al. 2000]. We start at the finest level H_M, perform a few relaxation steps, and then proceed to the next coarser level. We continue iterating until we have reached the coarsest level H_0. We then fully relax the coarsest level, and go back up to the finest level, as initially described above.

4.5 GPU Implementation

Deformation Estimation The simplicity of our hierarchical deformation estimation approach allows for an efficient parallel implementation on a graphics processor. The hierarchical deformation estimation part of the pipeline was implemented with CUDA because of its flexible memory management. The deformation points, connectivity information, and other constants, like $w_{j,k}$ and $\mathbf{d}_{j,k}$ used in Eq. (6), are precomputed on the CPU and are uploaded to GPU memory. We then relax all three parts of Eq. (4) in parallel for all deformation points, starting with enforcing the user constraints.

The three relaxation steps are repeated a number of times for each deformation point from all levels (we used 10 V-cycles in all our experiments). Afterwards, we calculate the inverse mapping $\mathbf{x} = g^{-1}(\mathbf{y})$ for each pixel using a vertex and a fragment shader, as outlined in Section 4.3.

Surface Projection To relax E_{surf}, we need to find the closest on-surface point for any location in the scene. To find this point efficiently, we use a discrete offset volume, which maps every location of a voxel center to the closest on-surface point. We build this offset volume in a preprocessing step and upload it to the GPU, where it is fetched using linear filtering.

Only voxels that are within close range to a surface are required by our algorithm, as the deformation points stay close to the surface during optimization. Thus, we have to calculate the closest on-surface point only for voxels that lie in regions close to a surface. This reduces the computation time of our pre-processing algorithm. It takes less than 10 seconds on a current CPU to generate a 256^3 volume for a typical scene. The required voxels are stored in a hierarchical data structure on the GPU. Therefore, even large volumes of up to 1024^3 typically require less than 20 MB of GPU memory.

In meshes with open boundaries, we achieve the most intuitive handling by extending the open edges into space. If the closest on-surface point for a particular voxel center is located on an open triangle edge, we replace this on-surface point with the closest point in space that is coplanar to the triangle vertices. This allows deformation points to float off the surface.

Per-Pixel Surface Projection As a last step in our pipeline, each location $\mathbf{x}$ is projected onto the closest surface using the discrete offset volume in a fragment shader. This step is always performed before evaluating the shading $f(\mathbf{x})$ to ensure that $\mathbf{x}$ matches the surface with per-pixel accuracy.

4.6 Animated Scenes

In general, our approach does support animated scenes, but there are a number of different cases to consider. Animating an object that casts a shadow, creates a caustic, or reflects indirect illumination requires no explicit consideration. It only means that the shading at locations $\mathbf{x}$ changes over time. However, our implementation does currently not support edits of signals on animated surfaces. Camera animations do not cause any problems. We simply keep the solution for the deformation field that was initially computed and only re-evaluate the inverse lookup at each screen pixel. We also allow key-framed signal deformations. For instance, the user can animate how a 3D texture deforms over an object by setting key-framed target positions for constraints.

5 Results

Our approach allows us to interactively edit a number of different visual phenomena on surfaces. In this section we briefly outline each application separately; a combination of all four applications is seen in Fig. 1. Additional results can be found in the supplemental video.

Shadows The editing of shadows is the primary application of our technique (although other on-surface signals can be deformed equally well). Fig. 8 shows shadow editing for highly complex geometry and models with high genus, for which editing the shadows manually, e.g. with image processing, would be a tedious task. With our system, all shown modifications can be specified using few mouse clicks, and the results can be observed at interactive frame rates.

Reflections and Refractions Fig. 1 shows that our system can even edit reflections to a certain degree. In this case, we apply the rotational component of Eq. (5) to the normals to ensure a consistent reflection vector. Note, however, that the modifications are not as general as the ones proposed by Ritschel et al. [2009], as we do not support direct editing of surface normals (and hence reflection directions).

Solid Textures 3D textures are a powerful tool to add surface color or detail to objects that are difficult to parametrize. However, control of their placement by changing a global or spatially varying transformation will almost always result in a change of visible surface texture, as the slice through the 3D texture changes. Using our technique, the pattern can be deformed on-surface without changing the pattern as shown in Fig. 5 and the accompanying video.

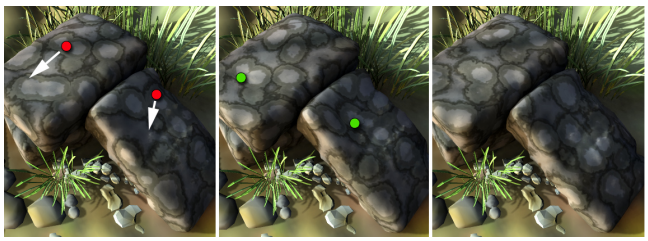

Figure 5: *Deforming a 3D texture on a rock using on-surface deformations preserves the structure of the texture, whereas conventional space deformation results in unpredictable changes of the visible texture (7.8 fps, 10 k deformation points).*

Caustics and Diffuse Bounces Our method also generalizes to indirect illumination, which allows us to interactively modify and deform it. Note, that our approach is independent of the method producing the indirect illumination, as is the case for all other mod-

ifiable shading components. Fig. 6 shows a scene rendered with Instant Radiosity for indirect illumination. We use our method to drag the indirect shadow of the fork lift from beneath to the floor in front of it. The modification only affects indirect illumination, all other lighting effects remain unchanged. Analogously, we can reposition and deform a caustic of a metal ring, see Fig. 7. Note that the smooth deformation due to our approach prevents objectionable artifacts in the high frequency details of the caustic.

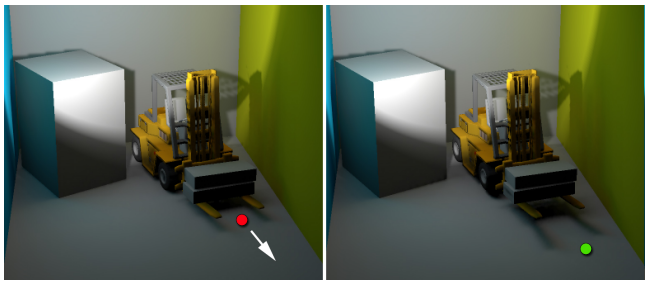

Figure 6: *This example demonstrates editing of indirect light: the shadow from underneath the fork lift is pulled towards the front (5.4 fps, 4 k deformation points).*

Sketching Shadows, Reflections, Caustics The user constraints for on-surface deformation can also be set by using the sketching interface, where our system automatically creates a deformation from two user-specified curves, such that one curve deforms to another one. Sketching can be used to intuitively edit on-surface signals in many scenarios, such as the caustics in Fig. 7.

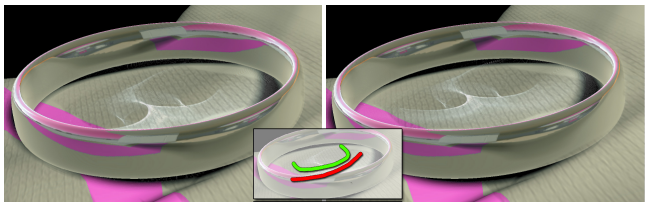

Figure 7: *Starting from a caustic (left), a user sketches a source and target curves on the shading signal (middle), and the system computes the new shape (right) (10.3 fps, 1 k deformation points).*

Complex Topology Our method efficiently handles geometrically complex scenes. The preprocessing, including the computation of the offset field, required for the scene in Fig. 8, left, which consists of 1.4 million triangles, takes only 23 seconds (10 s for the offset volume, and 13 s to generate 19 k deformation points). As our method works in a meshless fashion, it naturally handles scenes with low, medium and high genus (Fig. 8).

5.1 Performance

Performance numbers are indicated underneath each example edit. As can be seen, our method allows for interactive edits (measured on a GeForce 8800 GTX). Our GPU implementation of the solver is about 10 times faster than a pure CPU implementation and was key to achieve interactivity.

5.2 User Study

In order to evaluate the usability of our system for a particular editing task, a user study with 16 participants was conducted. All 16 participants had not used our system before. Most of them considered themselves skilled computer users in general (average of 8.2 ± 1.2 where 0 is worst and 10 is best, and $\pm$ denotes the standard deviation) and about half of them also had some experience with professional 3D modeling and animation packages (average of 4.4 ± 3). First, the participants received a short tutorial on our system, which took them on average $4{:}32 \pm 1{:}17$ (min:sec) to complete. Afterwards, they were asked to complete four tasks on their own for four different shading components: shadows, caustics, reflections, and 3D textures. The tasks were given by a textual description as well as before/after images (see the supplemental material). All of the participants found the tasks easy to solve with our system and completed them in a very short time (on average $1{:}06 \pm 0{:}33$, $1{:}27 \pm 0{:}57$, $1{:}06 \pm 1{:}10$, and $1{:}02 \pm 0{:}36$ (min:sec), for the shadow, caustic, reflection, and 3D texture editing task, respectively). Furthermore, the participants were very satisfied with the achieved results (average of 8.6 ± 1.8 over all tasks, where 0 is worst and 10 is best) and found the system extremely useful to achieve the requested task (average of 9.3 ± 1.3 over all tasks). An exact description on how the user study was conducted and more details on the results can be found in the supplemental material to this paper.

6 Discussion and Limitations

Our editing metaphor is based on a virtual piece of cloth that is deformed over the scene surface. We reason that users have a certain expectation or knowledge of how a piece of cloth would behave when deformed over a surface and our objective function is chosen accordingly. As a result, our deformations are not as-rigid-as-possible (ARAP) [Alexa et al. 2000], i. e., our solution may contain some shearing, as it does not explicitly prefer rotations. We have experimented with an ARAP objective function as well; however, we found that the resulting deformations tend not to be that different from our results, but more expensive to compute. Furthermore, a user can easily enforce curved deformations without shearing by using the sketching interface to map a straight line onto a curve.

The deformation function $g(\mathbf{x})$ should be invertible in order to find undeformed locations $\mathbf{x}$ for any surface point $\mathbf{y}$. However, the computed deformation function may not be strictly invertible. For instance, under very strong deformations, foldovers may occur. Adding more constraints can prevent this.

Our method does currently not support signal deformation on animated surfaces. It might be possible to use a temporally consistent parameterization over the animated surfaces that allows applying a deformation over time. However, this extension as well as an intuitive user interface to perform such edits remains future work.

Our system does not prevent the user from performing objectionable edits that might be perceived as physically incorrect (as can be observed in the *sphere* example shown in the supplemental video at around 1:00 min:sec). Especially smooth surfaces (e. g., a plane) or simple signals (e. g., a sphere's shadow) can lead to objectionable edits.

We have chosen on-surface deformations as our editing paradigm. It has proven to be intuitive (cf. Section 5.2), interactive, and generally applicable. It also prevents artifacts that would occur with space deformations (see Fig. 3c). The different shading components can be decoupled, enabling a workflow that artists are used to (i. e., layer-based editing).

An interesting extension of our system would be to allow the deformation of a shading effect to relate only to a *certain* object. For instance, if two objects cast a shadow onto the same location, an artist might want to edit only the shadow of one object. To enable such a feature, our system could be easily extended to hold a deformation field per object.

Another possible extension would be to support locally varying weights w_{dist} in Eq. (4) that depend on the local value of the shading

Figure 8: Left: *Statue with 1.4M faces (genus 8, 8.1 fps, 19 k deformation points).* Right: *A scene with several 100 k faces with a high genus broken topology (hundreds of disconnected components). It can be edited at 7.3 fps (21K deformation points), and would be impossible to do in image space.*

function. E.g., for a shadow editing task it could be useful to keep the shadow area more rigid and to allow the illuminated area to deform more.

Our editing metaphor is focused on on-surface signal deformations, i.e., we are interested in edits such as deforming cast shadows or caustics. Other edits, such as changing the hue of indirect illumination, are orthogonal to our method.

7 Conclusion

We have presented an interactive system for the artistic deformation of shading components such as shadows, caustics, and indirect illumination, directly over surfaces. Our main editing metaphor is that of a virtual piece of cloth modified by constraints. The system has an intuitive user interfaces to set these constraints. The constraints are solved on the GPU at interactive rates, which allows for an interactive workflow. We have chosen a meshless approach in order to support the deformation of surface signals across separate objects and across objects with difficult topologies. A number of challenging scenes were presented, where we have modified shadows, caustics, reflections, 3D texture, and indirect illumination. We further demonstrated the effectiveness of our system with a user study of task performance.

In future work, we would like to add higher-level constraints to allow for more general deformation fields. While we have focused on deformations of surface signals, it would be worthwhile to explore other modifications, such as intensity changes. Our technique could possibly be extended to higher-dimensional (4D) editing of light transport. Furthermore, we would like to investigate how to best handle animations that exhibit topology changes.

References

ALEXA, M., COHEN-OR, D., AND LEVIN, D. 2000. As-rigid-as-possible shape interpolation. In *Proc. SIGGRAPH*, 157–164.

BALZER, M., SCHLÖMER, T., AND DEUSSEN, O. 2009. Capacity-constrained point distributions: A variant of Lloyd's method. *ACM Trans Graph. (Proc. SIGGRAPH) 28*, 3, 86:1–8.

BARZEL, R. 1997. Lighting controls for computer cinematography. *Journal of Graphics Tools 2*, 1, 1–20.

BIRN, J. 2006. *Digital Lighting and Rendering*, 2nd ed. New Riders Press.

BOTSCH, M., AND KOBBELT, L. 2004. An intuitive framework for real-time freeform modeling. *ACM Trans. Graph. (Proc. SIGGRAPH) 23*, 3, 630–634.

BRONSTEIN, A. M., BRONSTEIN, M. M., BRUCKSTEIN, A. M., AND KIMMEL, R. 2006. Matching two-dimensional articulated shapes using generalized multidimensional scaling. In *AMDO*, 48–57.

DECORO, C., COLE, F., FINKELSTEIN, A., AND RUSINKIEWICZ, S. 2007. Stylized shadows. In *Proc. NPAR*.

HORMANN, K., LÉVY, B., AND SHEFFER, A., 2007. Mesh parameterization: Theory and practice. ACM SIGGRAPH 2007 Courses.

KERR, W. B., AND PELLACINI, F. 2009. Toward evaluating lighting design interface paradigms for novice users. *ACM Trans. Graph. (Proc. SIGGRAPH) 28*, 3, 1–9.

KO, H.-S., AND BREEN, D., 2003. Clothing simulation and animation. ACM SIGGRAPH 2003 Courses.

KOBBELT, L., CAMPAGNA, S., VORSATZ, J., AND SEIDEL, H.-P. 1998. Interactive multi-resolution modeling on arbitrary meshes. In *SIGGRAPH '98*, 105–114.

LI, H., ADAMS, B., GUIBAS, L. J., AND PAULY, M. 2009. Robust single-view geometry and motion reconstruction. *ACM Trans. Graph. (Proc. SIGGRAPH Asia) 28*, 5.

LÉVY, B., AND MALLET, J.-L. 1998. Non-distorted texture mapping for sheared triangulated meshes. In *Proc. SIGGRAPH*, 343–352.

MÜLLER, M., HEIDELBERGER, B., TESCHNER, M., AND GROSS, M. 2005. Meshless deformations based on shape matching. *ACM Trans. Graph. (Proc. SIGGRAPH) 24*, 3, 471–478.

OBERT, J., KRIVÁNEK, J., PELLACINI, F., SÝKORA, D., AND PATTANAIK, S. N. 2008. iCheat: A representation for artistic control of indirect cinematic lighting. *Computer Graphics Forum (Proc. EGSR) 27*, 4, 1217–1223.

OKABE, M., MATSUSHITA, Y., SHEN, L., AND IGARASHI, T. 2007. Illumination brush: Interactive design of all-frequency lighting. In *Proc. Pacific Graphics 2007*, 171–180.

PELLACINI, F., TOLE, P., AND GREENBERG, D. P. 2002. A user interface for interactive cinematic shadow design. In *ACM Trans. Graph. (Proc. SIGGRAPH)*, 563–566.

POULIN, P., AND FOURNIER, A. 1992. Lights from highlights and shadows. In *Proc. I3D*, 31–38.

POULIN, P., RATIB, K., AND JACQUES, M. 1997. Sketching shadows and highlights to position lights. In *Proc. CGI*, 56–63.

RAGAN-KELLEY, J., KILPATRICK, C., SMITH, B. W., EPPS, D., GREEN, P., HERY, C., AND DURAND, F. 2007. The

lightspeed automatic interactive lighting preview system. *ACM Trans. Graph. (Proc. SIGGRAPH) 26*, 3.

RITSCHEL, T., OKABE, M., THORMÄHLEN, T., AND SEIDEL, H.-P. 2009. Interactive reflection editing. *ACM Trans. Graph. (Proc. SIGGRAPH Asia) 28*, 5.

SCHMIDT, R., GRIMM, C., AND WYVILL, B. 2006. Interactive decal compositing with discrete exponential maps. *ACM Trans. Graph. (Proc. SIGGRAPH) 25*, 3, 605–613.

SCHOENEMAN, C., DORSEY, J., SMITS, B., ARVO, J., AND GREENBERG, D. 1993. Painting with light. In *SIGGRAPH '93*, 143–146.

SCHREINER, J., ASIRVATHAM, A., PRAUN, E., AND HOPPE, H. 2004. Inter-surface mapping. *ACM Trans. Graph. (Proc. SIGGRAPH) 23*, 3, 870–877.

SUMNER, R. W., SCHMID, J., AND PAULY, M. 2007. Embedded deformation for shape manipulation. *ACM Trans. Graph. (Proc. SIGGRAPH) 26*, 3, 80.

TABELLION, E., AND LAMORLETTE, A. 2004. An approximate global illumination system for computer generated films. *ACM Trans. Graph. (Proc. SIGGRAPH) 23*, 3, 469–476.

TODO, H., ANJYO, K.-I., BAXTER, W., AND IGARASHI, T. 2007. Locally controllable stylized shading. *ACM Trans. Graph. (Proc. SIGGRAPH) 26*, 3, 17.

TROTTENBERG, U., OOSTERLEE, C., AND SCHULLER, A. 2000. *Multigrid*. Academic Press.

TZUR, Y., AND TAL, A. 2009. Flexistickers: Photogrammetric texture mapping using casual images. *ACM Trans. Graph. (Proc. SIGGRAPH) 28*, 3, 1–10.

ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. 1997. Interactive multiresolution mesh editing. In *SIGGRAPH '97*, 259–268.

ZWICKER, M., PAULY, M., KNOLL, O., AND GROSS, M. 2002. Pointshop 3D: An interactive system for point-based surface editing. In *Proc. SIGGRAPH*, 322–329.