EUROGRAPHICS 2013 / I. Navazo, P. Poulin
(Guest Editors)

Volume 32 (2013), Number 2

Freeform Shadow Boundary Editing

O. Mattausch[1,2] and T. Igarashi[2] and M. Wimmer[3]

[1]University of Zurich, Switzerland
[2]The University of Tokyo, Japan
[3]Vienna University of Technology, Austria

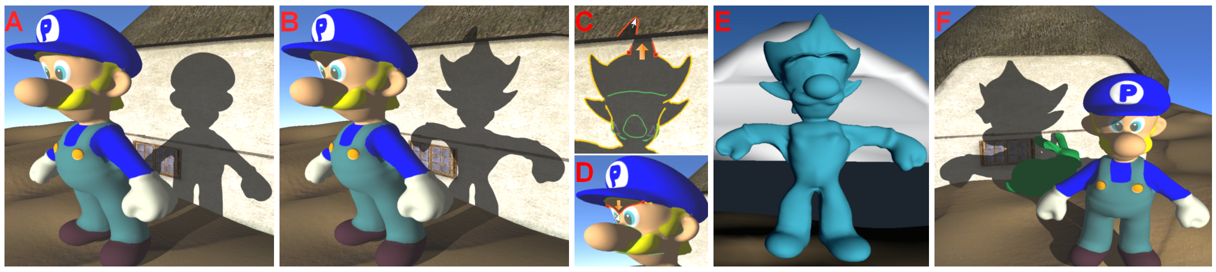


Figure 1: *(A) Paolo model with physical hard shadows. (B) Paolo model with edited shadows, giving him a demonic look and changing the mood of the scene. This result was created with a few simple operations on the shadow boundaries given by the silhouettes (C and D, drag direction indicated by arrows). Using a mesh created from the new shadow boundaries (E), a shadow volume algorithm creates plausible shadows when changing the scene configuration (F).*

Abstract

We present an algorithm for artistically modifying physically based shadows. With our tool, an artist can directly edit the shadow boundaries in the scene in an intuitive fashion similar to freeform curve editing. Our algorithm then makes these shadow edits consistent with respect to varying light directions and scene configurations, by creating a shadow mesh from the new silhouettes. The shadow mesh helps a modified shadow volume algorithm cast shadows that conform to the artistic shadow boundary edits, while providing plausible interaction with dynamic environments, including animation of both characters and light sources. Our algorithm provides significantly more fine-grained local and direct control than previous artistic light editing methods, which makes it simple to adjust the shadows in a scene to reach a particular effect, or to create interesting shadow shapes and shadow animations. All cases are handled with a single intuitive interface, be it soft shadows, or (self-)shadows on arbitrary receivers.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

1. Introduction

Various methods exist to create correct physically based shadows, which are convenient to use as they require no further artist input. However, for artistic purpose it is often more important to achieve a certain effect than to maintain physical correctness. In cinematography, for instance, shadows are considered a valuable means to express a certain mood or atmosphere, or to emphasize certain features of a character [Bar97]. For example, a large shadow cast on another character may convey dominance, or a character may be given an air of mystery by having one half of his face covered by shadow. However, obtaining a desired result just by properly setting up and tweaking the scene lighting might be very difficult, as illumination has a global influence – a small correction to a shadow might require relighting of a large portion of the scene. Conversely, it may be equally tedious to create all shadow effects in the scene by hand, and almost impossible to keep these artificial shadows consistent over an animation sequence. Inconsistencies or sudden changes in the shadows might instantly shatter the illusion and take someone out of a movie or game experience. Ideally, we want to choose and edit individual shadows, while everything else stays the same.

In this paper, we therefore propose a different approach, where we 1) start from the physical (hard or soft) shadow, 2) artistically edit the physical shadows for the current scene configuration, and 3) make the edited shadows consistent for varying light source directions and scene configurations. A typical editing example using our tool can be seen in Figure 1. Starting from the physical shadow (Image A), a user can create results like the demonic shadow shape shown in Image B. The shadows are edited by simply pulling on the *shadow boundaries* represented as *freeform curves* (visualized as colored lines in Images C and D). Editing shadows on distant shadow receivers (C) and self shadows (D) are handled uniformly in our framework. This algorithm has the following main properties:

Direct and intuitive control over the shadow edits. The artist can drag the shadow boundaries of the physically based shadows *directly* in the scene, and the shadow closely follows the user inputs.

Detailed and local control over the edits which exceeds that of previous algorithms. To our knowledge, this is the only method that allows changes to the *topology* of the shadow (as shown, e.g., in Figure 13 where the shadow of the nose is pulled out).

Animation of both characters and light sources.

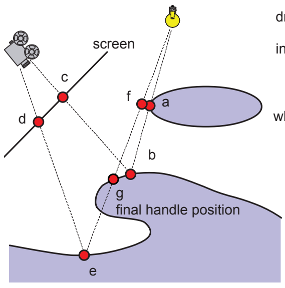
Plausible interaction of the generated shadows with a dynamic environment, e.g., when another object comes between shadow receiver and shadow caster.

A unified framework for editing self shadows, contact shadows, and shadows cast on non-planar distant receivers, as well as soft shadows.

To our knowledge, this is the only shadow editing method which can provide all of the listed properties simultaneously, using a single intuitive and local editing interface. This interface is suitable for a casual user to quickly create interesting shadow edits and animations, as well as for professional artists to create a desired effect in an animation movie.

2. Related Work

Several indirect methods for lighting design have been proposed to avoid the often tedious work of directly tweaking the light sources to reach a particular effect. In early work, Poulin et al. [PF92] described an algorithm where modifying the highlights and shadows in the scene lead to indirect modification of the scene lights. The algorithm of Pellacini et al. [PTG02] allows a user to directly move shadows in the scene, whereupon it calculates the corresponding light position for the new shadow location. However, they do not go beyond simple translations of the shadow.



```
dragging handle (for shadow boundary editing)

initialization:
   1) compute physical light, obtaining
   shadow silhouette (a) and shadow handle (b)

while (!release mouse):
   2) drag handle in screen space from (c) to (d)
   3) project dragged handle (d) to ground (e)
   4) compute modified silhouette (f) from
      (e) and (a)
   5) cast ray from (f) to (e) to find final handle
      position (g)

   If no surface is between silhouette (f) and (e),
   (g) follows screen space position (d) exactly.
```

Figure 2: *The process for finding a new handle position during shadow boundary editing.*

DeCoro et al. [DCFR07] proposed a number of operators for non-photorealistic shadow styles, including *abstraction*, which creates a stylized, simplified version of the original shadow, and *inflation*, but they do not support direct editing of the shadows as such. Todo et al. [TABI07] proposed a method for locally tweaking the diffuse and specular term for stylized shading. Its main use is in cartoon-style keyframe animation, where they smoothly interpolate their shading between the frames. Since they do not deal with shadows, our method could be used in combination with theirs. Mitra et al. [MP09] proposed an algorithm for creating shadow art, where depending on the light direction the created object casts multiple target shadow shapes. There is some similarity to our algorithm, as both methods create a shadow shape which in turn casts the desired shadow.

2.1. Editing the shadow shape

In contrast to the above methods, there are a couple of papers which discuss direct edits to the *shape* of the shadow.

The method of Obert et al. [OPP10] cleverly decouples visibility from lighting, allowing direct edits of the shadows. Instead of a unified interface as in our method, it offers a couple of editing tools. In terms of local and detailed edits, the lattice-editing metaphor supports only coarse-grained shadow warpings but no topology changes like our method. Dynamic scene changes are very limited since visibility is preprocessed. An artist can either edit in image-space (fixed view position) or *indirectly* in a *separate* uv-space window (more complex edits are fixed to a surface parametrization). Also, no object interaction is supported.

The on-surface signal editing method of Ritschel et al. [RTD*10] does not require parametrization. It uses the metaphor of cloth dragged over a surface for local warping and can uniformly handle a variety of illumination effects like caustics and shadows. This method is closest to ours in terms of direct and local control. However, editing fine details depends heavily on the sampling resolution and requires

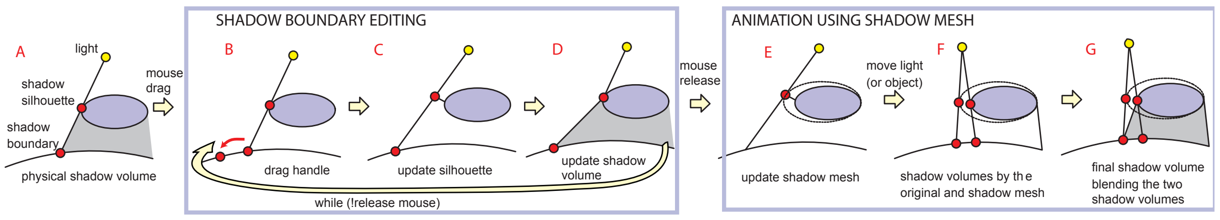


Figure 3: *Shadow volume modification during boundary editing (B,C,D), and after shadow mesh creation for animation (E, F, G). The rendered shadows immediately reflect new user edits.*

many control points near discontinuities, and does not permit topology changes. An inherent drawback of this approach is that edits are fixed to the surface, and hence does not support object interactions nor translations.

BendyLight [KPD10] enables editing by bending the shadow casting light rays and was later extended to global illumination [NJS*11]. It is related to our algorithm in the sense that it really deforms the underlying geometry. Also, it is the only other shadow editing method which allows both character animation as well as object interactions. However, the editing metaphor is neither direct nor local, since the artist has to bend a light beam to *indirectly* move a shadow. While it is an option to use per-object shadow dragging, all presented edits are of global nature.

All of these methods are useful in their own way. Nevertheless, none of them supports as direct and detailed shadow control as our method. They are limited to *global*, continuous transformations, basically equivalent to stretching and squishing parts of the surface where the shadow falls on. A first step towards shadow boundary editing is constituted by the method of Nakajima et al. [NSM07]. For rendering they use also shadow volumes, where they change the extrusion direction of the polygons. Unfortunately, the method has many shortcomings, as it can handle neither non-planar receivers, self shadowing, nor dynamic changes.

3. Algorithm Overview

Our algorithm displays the shadow boundaries and allows the user to modify them in the scene directly. In order to make this work, the main idea of our algorithm is to map these changes back to the the *silhouettes* of the shadow casting object, and use the modified silhouettes to cast the shadows. Since silhouettes become invalid when dynamically modifying the scene, we create a *shadow mesh* that conforms to the edited silhouettes, and can be used to create new silhouettes for different light or object positions. When rendering the shadows, we have to avoid self-shadow artifacts due to mismatches between shadow mesh and original mesh, and we have to make sure the shadow modifications have local effect by combining modified and original shadows.

We use the *shadow-volume* algorithm [Cro77] to render shadows, which creates shadow volumes by extending the silhouette of an object with respect to the light source to infinity and then determines for each receiver point whether it is inside our outside this volume. Whenever we mention silhouettes in this paper, we always mean the silhouettes as seen from the *position of the current light source*.

The following lists the main steps of the algorithm, described in more detailed in the subsequent sections. Figure 2 shows the details of user input and projections and Figure 3 shows the overall workflow.

Silhouette and handle generation (Figure 3 A) First, we calculate the silhouettes of the mesh from the current light direction (depicted as point (a) in Figure 2). To create handles on the shadow boundaries for editing (denoted as *shadow handles*), we project the silhouette vertices onto the receiver surfaces using ray casting (point (b) in Figure 2).

Shadow boundary editing (Figure 3 B-D) The user edits the shadow boundaries by dragging a shadow handle in screen space (see Figure 2). The new handle position is used to deform all vertices in a shadow boundary segment similar to 2D freeform curves. Meanwhile the system interactively adjusts the silhouettes, recomputes new handle positions, and uses these to compute new (modified) shadow volumes to cast the desired shadows in each frame.

Animation using shadow meshes (Figure 3 E) At this point however, the shadow is only valid for the current set of silhouettes. Even a slight change of light direction or object transformation could lead to sudden changes in the shadow. Hence, after mouse release, the original mesh is warped to create a *shadow mesh* based on the deformed silhouette vertices. Such a shadow mesh is depicted in Image E of Figure 1. The shadow mesh is then able to produce the edited shadow shapes while providing smooth shadow transitions when the silhouettes change.

Rendering the edited shadows (Figure 3 F,G) After each modification of the scene, we recompute the silhouettes based on the shadow mesh and use them to cast the desired shadow. Notice however that a naive approach can cause

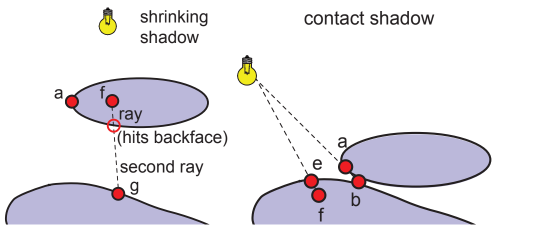


Figure 4: *(Left) In case of shrinking a shadow boundary (the silhouette vertex (a) moves to (f)), the shadow ray from (f) will terminate at a backface inside the geometry. A second ray is then cast to find the correct handle position (g). (Right) For contact shadows (f) might intersect the receiving surface. In this case we set (f) to (e).*

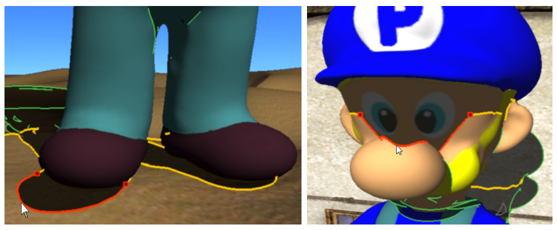


Figure 5: *Our editing approach provides intuitive behavior of the shadow with respect to close contact shadows and highly non-planar surfaces.*

self-shadowing artifacts (i.e., the shadow mesh casts shadows on the original geometry). An important part of our algorithm is therefore to set up the shadow volume polygons properly for rendering in order to minimize self-shadowing artifacts due to the shadow mesh. Our solution is to interpolate between the original shadows at the silhouettes and the edited shadows at the destination surface (Figure 3, G).

4. Silhouette and Handle Generation

The following two steps are required before the first shadow editing operation (using the original mesh), and the also after dynamic changes: silhouettes are recomputed based on the shadow mesh after each dynamic scene modification, while shadow handles are recomputed during dragging of the shadow boundary.

4.1. Silhouette Generation

The silhouettes are used both for calculating the shadow boundaries for editing, and for rendering the original shadows and those of the shadow mesh. However, we found that the silhouettes normally used for shadow volume rendering, i.e., the edges between front-facing and back-facing polygons, are too cluttered to be useful for editing, which was also pointed out by Nealen et al. [NSACO05]. Instead we approximate the silhouettes of the underlying smooth mesh as defined by the vertex normals [HZ00]. This method interpolates silhouette vertices s between vertices on opposite sides with respect to the light (placing s where $n(s) \cdot l = 0$), and outputs smooth silhouettes in connected loops.

4.2. Shadow Handle Creation

To be able to edit the shadows boundaries directly on the receiver surfaces, we need handles on the boundaries. Therefore we project each silhouette vertex (point (a) in Figure 2) onto the receiver geometry as seen from the light source. For this purpose we cast a shadow ray from the current silhouette vertex position (in direction $a - l$) and compute the first intersection with a receiver surface (point (b)). There are situations where the user drags a shadow handle inward to cause a shadow to shrink. In this case the shadow ray cast from the modified silhouette will potentially terminate inside the shadow caster geometry and does not provide a useful shadow handle. Therefore we always test if the intersected polygon is backfacing, and cast a *second* ray starting from the intersection point if this test is positive. This case is illustrated in Figure 4. We render the shadow boundaries using line segments between individual shadow handles. In order to cull handles which should be hidden from the current view (so that the user cannot accidentally select them), we utilize hardware occlusion queries [BMH98] to test their visibility.

If the number of silhouettes is large (e.g, in case of a noisy mesh), the shadow boundaries can consist of many small intersecting lines which are hard to edit, while we are mostly interested in the long silhouette loops that define the main features of a model. To avoid such situations, we introduce a user-defined parameter which removes all silhouette loops which are smaller than a certain threshold (e.g., loops under 20 silhouette vertices). More specifically, they are displayed in a neutral gray color, cannot be selected and do not influence the editing as they are not taken into account as constraints for the mesh deformation (see for example Figure 14).

5. Shadow Boundary Editing

The editing process starts when the user selects an object with the mouse, causing the system to display its shadow boundaries. The user can drag the shadow handles directly on the receiver surface. In the following we will first explain how to move a 3D shadow handle responding to the user's 2D dragging operation on the screen, and then show how we move all vertices in a selected part of the boundary using freeform deformations.

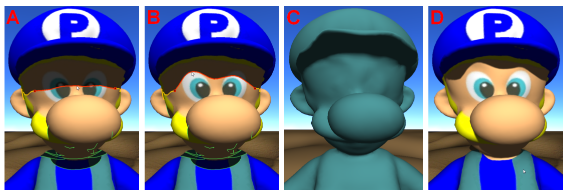


Figure 6: *The user simply selects a shadow boundary, sets two fixed end points (A), and edits the (freeform) curve in between (B). The created shadow mesh (C) provides consistency with respect to another light position (D).*

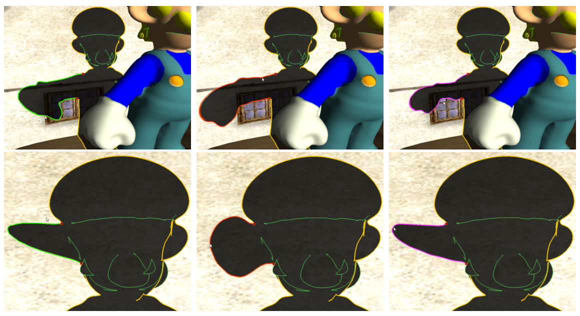


Figure 7: *Different deformation styles on the shadow boundaries: affine (left), similar (middle), rigid (right).*

5.1. Dragging a Shadow Handle

The following operations are explained using the points marked in Figure 2. The user edits the shadow boundaries by dragging a shadow handle in screen space (from screen position (c) to (d)). In order to compute the shadow boundary positions from the user input, we first project the screen space position (d) on the receiver surface (point (e)) using ray casting. We then obtain the modified silhouette position (f), which is a point on the light ray passing through (e) and with the same depth as (a). An exception applies when (e) is closer to the light than (a), which happens in the case of close contact shadows like those cast by the feet in Figure 5. In this case, we set the depth of (f) to that of (e) (right image of Figure 4). The final position (g) of the new shadow handle is then calculated by casting a shadow ray from new silhouette vertex (f) to get the intersection with the surface where the edited shadow will finally project to. This method gives a very intuitive behavior of how the shadow changes in response to the user input, also in case of highly non-planar surfaces such as in the right image of Figure 5.

The handle position (g) usually follows the mouse pointer (d). The only exception is given by the case where another surface blocks the visibility from (f) to (e) as shown in Figure 2, in which case the handle is placed on top of that surface. Note that this is the desired behavior, because the edited shadow will also be cast onto this surface.

5.2. Deforming the Whole Boundary

Dragging each individual shadow handle of the shadow boundary can be tedious. Therefore we use the shadow handle dragging-operation to control a freeform curve deformation process on the whole boundary. Figure 6 shows a sequence of the user interface and the full editing process, demonstrated on the self shadows cast by the cap on the face. In order to deform shadow boundaries of the original shadow, we compute and display the handles on the shadow boundary (Image A) as described in Section 4.2. The user selects two points on the boundary (the two fixed end points of a segment). Selecting another point on the curve uniquely defines the segment of our boundary loop to be edited. The user deforms the shadow boundary like an ordinary 2D freeform curve by dragging it with the mouse (Image B). The three specified points (the fixed end points and the moving point) are used as control vertices for the deformation algorithm.

For computing the shadow boundary deformation, we chose the moving least-squares image deformation method by Schaefer et al. [SMW06], because it is quite fast and versatile, and we are able to choose from several warping styles. The method allows affine-, similarity-, and rigid deformations. We offer these styles in our system to provide variable means for shadow editing (see Figure 7).

A modified shadow volume algorithm (explained in detail in Section 7) uses the newly computed shadow handles to immediately update the shadow volumes with respect to the user input and render the desired shadows (Image B of Figure 3). Hence the user receives important real-time feedback on these actions at any time.

Choosing a deformation space One question is in which 2D space the deformation should take place. Directional and spotlights are associated with an intrinsic (orthogonal or perspective) frustum. For point lights, a frustum to enclose the shadow caster can be set up automatically. If the light is allowed to enter the bounding volume of a shadow caster, a more robust solution is to automatically calculate a shadow frustum for each silhouette loop separately. No matter what frustum is chosen, we simply use the (x,y)-coordinates of the silhouette vertices with respect to this frustum, keeping the z-coordinate constant during deformation. This also means that only the shadow handle actually dragged by the user is treated as in Section 5.1, whereas the other silhouette vertices are deformed using the freeform deformation, and the final handle position is computed by casting a shadow ray as in step 5 of Figure 2.

Our particular choice of deformation space has the advantage that only *small deformations* of the caster geometry are needed to achieve a desired shadow edit. As an alternative, we also tried to use the deformation plane with respect to the

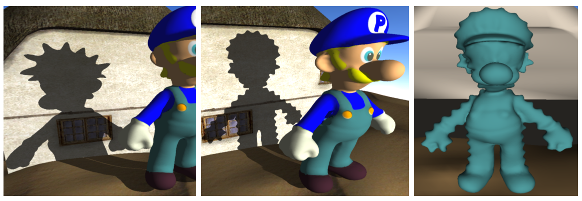


Figure 8: *Sinusoidal noise function to create effects like wild hair (left) or something more extreme (middle). Right: The shadow mesh corresponding to the middle image.*

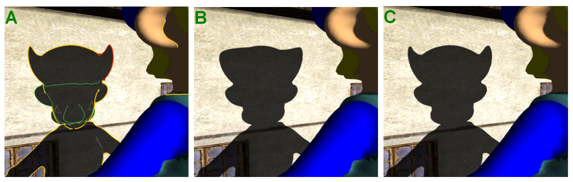


Figure 9: *When computing the shadow mesh from the edited boundaries (A) using standard isotropic weights, details are lost (B). By transforming the vertices into light space and using non-uniform weights in light direction before deformation, the details are captured in the shadow (C).*

average orientation of each silhouette loop (to minimize distortion). However, this caused unacceptably large deformations of the geometry. Consider the case of a silhouette oriented nearly parallel to the light direction, where a huge deformation would be needed for a tiny change in the shadow. Note also that our choice of 2D coordinates does not restrict the allowed orientations of the receiver geometry, as it only affects the deformation of the caster geometry.

5.3. Alternative Operations on the Shadow Boundaries

While the styles defined by the moving least squares deformation algorithm are our main means for editing, our paradigm of working on shadow boundaries allows more operations that yield interesting and useful shadowing effects. For example, we use a sinusoidal noise function which adds an offset $h\sin(fu)$ to each vertex v of the shadow boundary, in the local normal direction of the boundary. u is a parametrization of the shadow boundary curve, h the amplitude, and f the frequency of the noise function. We calculate the normal direction by first using central differences to compute a tangent vector $\vec{t} = v_{i-1} - v_{i+1}$ and from that the normal as $n = (-t_y, t_x)/|\vec{t}|$. u is advanced as $u_{i+1} = u_i + |v_{i+1} - v_i|$. Of course, various other kinds of operators would be possible. Figure 8 shows an application of the noise functionality on the Paolo model.

6. Animation Using Shadow Meshes

In order to allow dynamic changes in the scene after the user has edited the shadow boundary, we deform the original mesh with respect to the deformed silhouettes to create a *shadow mesh* (the Image E in Figure 1). This allows recalculating new silhouettes (for the new light or object position) that still incorporate the intent of the modifications to the original silhouettes.

6.1. Creating the Shadow Mesh

For this, we could extend the moving least squares algorithm that we use for shadow boundary editing to 3D. However, there are several issues in using moving least squares for 3D mesh editing (e.g., having to use the Geodesic distance metric). Instead the system solves for the deformed shape-preserving vertex Laplacians [Sor06]. Note that the deformation procedure is very similar to silhouette-based editing [NSACO05], with the difference that the silhouette offsets are not directly controlled by the user but calculated based on the desired shadow shape. We use cotangent weights [Sor06] to handle non-uniform tessellations.

We slightly modify the original algorithm to take advantage of the fact that we almost deal with a 2D problem. Also, we want to control the shape of cast shadow on a surface rather than the 3D shape of the object. In particular, we want to ensure that the influence of the deformations reaches sufficiently far in the third direction (the light direction). Otherwise some deformations might not cast the desired shadow, as other geometry blocks the light. Therefore it is important to treat the third direction differently: We transform the geometry and the silhouette vertices into post-perspective light space and apply a non-uniform scale $(1.0, 1.0, \frac{1}{w_z})$ before solving the Laplacian optimization problem (afterwards we go back by applying the inverse transform). The factor w_z controls the influence of the deformation along the light direction. In our experience, setting w_z to a value of 2.5 ensures that the shadows cast by the shadow mesh are very similar to the one specified by the user. An example where using non-uniform weights improves the accuracy of the shadow editing is depicted in Figure 9.

6.2. Animating the Shadow Mesh

In order for our approach to be used in interactive applications, for example to attach an edited shadow to a game character, it has to be compatible with character animation. Luckily, due the versatility of the shadow mesh concept, character animation techniques like skeletal animation can be supported in a quite straightforward fashion. In particular, we use the fact that there is a one-to-one correspondence of the shadow mesh vertices and the vertices of the original mesh. Hence we simply copy the blend weights of the original vertices and use the same skeleton to deform both the original geometry as well as the shadow mesh. Note that this sim-

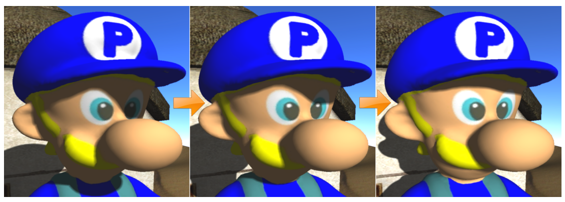


Figure 10: *For large deviation from the light direction when the shadows were edited, interpolation between the deformed shadow mesh and the original geometry can be used to achieve smoothly blended shadows.*

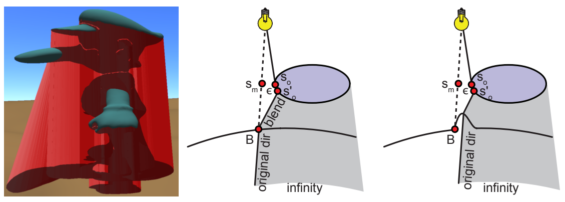


Figure 11: *(Left) Shadow mesh and its modified shadow volumes. (Middle) Illustration of the modified shadow volumes. (Right) Case where B is not the first intersection from s'_o.*

ple approach only works if the original pose and the pose of the shadow mesh stay roughly the same as a result of the edits. See Figure 16 for an example. Another possibility is to animate the shadow mesh *independently* of the original geometry while the geometry itself stays unchanged. Interesting artistic effects can be created with our system by simply blending between two states of the shadow mesh with further applications in keyframe animation.

Dealing with larger deformations As the shadow mesh is deformed to cast the desired shadows for the *current* scene configuration, we cannot guarantee that the shadows are still plausible if the scene configuration is very different from the initial one where the shadow was modified. One way to solve this problem is to define multiple keyframes for different configurations. An even simpler way to avoid this problem is to quantify the degree of change in the scene configuration. For example, we can linearly blend between the original geometry and the deformed shadow mesh depending on the angle between the initial light direction $\vec{l}_{initial}$ where the editing took place and the current one ($\vec{l}_{current}$), where the light vector is given in local object space to take relative changes of the object to the light into account. An example is shown in Figure 10. The falloff factor is computed as $(\vec{l}_{initial} \cdot \vec{l}_{current})^{\alpha}$, where α is a user-defined factor which gives no falloff if it is set to 0.

7. Rendering the Edited Shadows

While shadow maps [Wil78] can be considered the most popular algorithm for generating real-time shadows, we chose to use the competing shadow volume algorithm [Cro77] for rendering the edited shadows. This is because it has the required property that the volumes can be constructed from the silhouettes. More specifically, the faces of the volumes are extruded from the silhouette edges towards infinity (away from the light position), and for hard shadows the shadows can be efficiently rendered using the stencil buffer [EK02].

7.1. Modified Shadow Volumes

In every frame, no matter whether the user is currently dragging a shadow boundary or moving the light or an object, we need to render shadows that respect the user's modifications, but don't produce artifacts. In particular, we need to take the following constraints into account: (1) At the shadow caster itself, we need to avoid self shadowing artifacts, which arise easily when the shadow boundary is enlarged. For example, in Figure 3 E, the shadow mesh occludes the the original mesh, thus casts undesirable shadow on it. Thus, the modified shadow volume should be close to the original shadow volume near the shadow caster. (2) At the shadow receiver, the shadow volume should meet the modified shadow boundary. (3) Behind the shadow receiver, the shadow volume should continue without causing artifacts. This means that it should be extruded along the original light direction.

Figure 11 visualizes the process. As input to the rendering, for a single shadow volume we have the modified silhouette s_m created either from the shadow mesh or during a dragging operation, and the corresponding boundary handle curve B where the shadow should land. We also need a silhouette loop s_o close to the original mesh. While we could compute s_o from the current light position using the original mesh, this would produce a silhouette with a different number of vertices not matching s_m. Instead, we calculate s_o by mapping back the modified silhouette loop s_m to the original surface. This is done by calculating the silhouette vertex positions s_o on the original mesh using the same vertex indices as for calculating s_m.

In order to respect the three constraints mentioned above, we create a modified shadow volume in three parts to blend between the original shadow volumes and the modified ones: (1) To make sure we initially avoid any self-shadow artifact, we extrude s_o a small distance ε along the original light direction, leading to a second silhouette loop s'_o. (2) We connect s'_o with B, thus interpolating between the original shadow volume and the modified one. (3) We extrude B along the original light direction to infinity. The distance ε is calculated as a small percentage of the distance between a vertex on s_o and the corresponding vertex on B.

While this produces plausible shadows matching the three constraints in the majority of cases, there is a realistic chance that the modified shadow volume meets the receiver surface *before* B, violating constraint (2). For example, consider a situation where a dune intersects the line-of-sight between B on the receiving terrain and the silhouette vertex s'_o (Figure 11, right). In this case, the *first* surface intersection is with the surface of the dune, and can be detected by slightly modifying step (2). In particular, an additional ray is cast from each vertex s'_o to the corresponding vertex on B, and a possible new intersection is then substituted in B.

7.2. Soft Shadows

In this section we explain how to extend the method to the editing of soft shadows. Soft shadows could theoretically be created by multisampling (creating a multitude of slightly offset shadow volumes and blending), but this brute-force approach would be infeasible with respect to interactive editing. Instead we chose a variation of Penumbra Wegdes [AAM03], because that algorithm is sufficiently fast to allow interactive editing. In the original Penumbra Wedge algorithm, for each silhouette edge, a so-called penumbra wedge is computed, which conservatively bounds the influence region of this edge with respect to the penumbra computation. When a receiver fragment lies within a penumbra wedge, the soft shadow term is computed as follows: the silhouette edge generating that wedge is projected onto the light source (with the fragment position as center of projection), and then extended away from the center of the light source. The covered area on the light source gives the percentage of occlusion of the corresponding penumbra volume (Figure 12, right). The hard shadow volume is used to separate the volume of those shading points where this percentage has to be *added* to the shadow term from those shading points where it has to be *subtracted*.

Luckily the bulk of the algorithm does not change when used in combination with our modified shadow volumes. We just have to account for the fact that the modified shadow volumes blend between the original geometry and the shadow mesh. As depicted in Figure 12, the key for correct softness is to notice that the projection of a silhouette edge should always intersect the light source center if the point lies on a hard shadow volume boundary b_s extruded from this edge. We use the distance d to the light source to compute an interpolated silhouette vertex position s_i which fulfills this requirement, and is then used as input to the Penumbra Wedge algorithm. In particular, this method linearly interpolates the silhouette vertex between original and modified silhouette when the fragment lies between caster and receiver position:

$$s_i = \begin{cases} s_0, & \text{if } d(p) < d(s'_0) \\ s_m, & \text{if } d(p) > d(B) \\ s_0 + (s_m - s_0)\frac{d(x)-d(S'_0)}{d(B)-d(s'_0))}, & \text{else.} \end{cases}$$

p is the current point to be shaded, s'_0 is the starting point of the blended shadow volume, and B the end point. Hence for receiver point p'' in Figure 12, which is farther away than

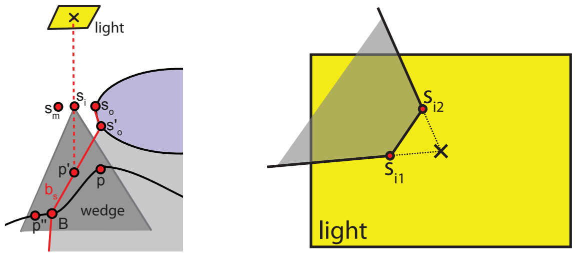


Figure 12: *Soft shadow volumes: (Left) Penumbra wedges are centered around an interpolated silhouette edge s_i. (Right) The penumbra volumes induced by an edge are projected onto the light source to compute the percentage of occlusion for receiver points p, p', and p'' inside the wedge.*

Model	tris	orig	hard	soft	smesh
Paolo	30,5K	4.5ms	20ms	32ms	0.2s
Bunny	70K	9ms	32ms	38ms	1.01s
Dragon	100K	19ms	101ms	123ms	1.28s

Table 1: *Typical timings for different models: The baseline using stencil shadow volume rendering (orig), editing and rendering hard shadows (hard), editing and rendering soft shadows (soft), and shadow mesh creation (smesh).*

intersection point B, just the modified silhouette s_m is used as input to the Penumbra Wedge algorithm. Receiver point p uses the same interpolated silhouette s_i as p' instead, since they are at the same distance from the light source. In particular, p' lies at the hard shadow boundary and the projection of its silhouette edge onto the light source correctly intersects the light source center. Note that the shadow handles still coincide with hard shadow boundaries, and hence the editing framework stays the same.

8. Results

All results were generated with an Intel Core i7-3770K CPU (using a single core) and a NVIDIA GTX 580 GPU, using a

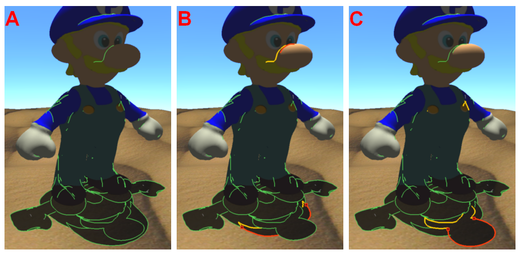


Figure 13: *Editing sequence where Paolo's nose is emphasized, so that it is lit and at the same time visible in the shadow (note the changing shadow topology).*

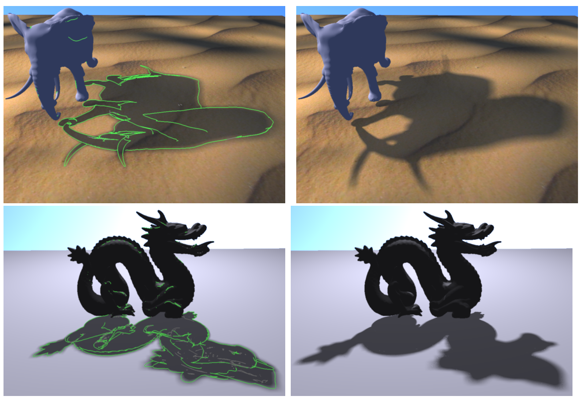


Figure 14: *The same interface is used for soft shadow editing: like pronouncing tusks, trunk, and ear of the elephant and the mouth of the dragon (note the topology changes).*

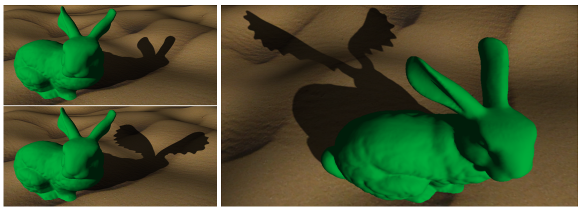


Figure 15: *(Left) Result created with only 4 clicks. (Right) Same edit under changed light configuration.*

deferred shading rendering pipeline with a render target resolution of 1024×768. For ray casting the shadow handles we employ the flexible PBRT framework [PH10]. A non-planar terrain is used in most of our results in order to underline the fact that a planar receiver surface is not required for our method (as opposed to Nakajima et al. [NSM07]).

Performance In Table 1 we show algorithm timings for 3 different models, the Paolo model, the Bunny model, and the Dragon model. The considerable difference to the timings with the original shadow volumes stem mostly from the fact that we use (unoptimized) ray casting to determine the next surface, and partly from the fact that our modified shadow volumes are more complex and require processing of two silhouettes. As the number of rays cast is typically quite low (it corresponds to the number of silhouette vertices), the ray casting overhead should be significantly reduced in an optimized implementation. The performance drop in the Dragon scene can be explained by the more complex silhouettes. Shadow mesh creation takes longer but is only invoked after a complete dragging operation (mouse release) and is therefore not bothering a user.

Editing examples Figure 13 shows an editing sequence where we emphasize Paolo's nose by making it lit and its silhouettes being reflected in the shadow boundaries. Note that this combined effect is almost impossible to achieve by just changing the light direction. We believe that a similar degree of control over the shadow topology cannot be reached by approaches which not operate on the shadow boundaries. Results like the Bunny ears in Figure 15 can be created easily with only 4 actions in our system (dragging the ear and using the noise function to create the feathers). It can further be seen that the shadow remains plausible when lit from a different direction. Examples for soft shadow editing are shown in Figure 14, where interactive framerates are still possible for editing fairly complex models like the Dragon.

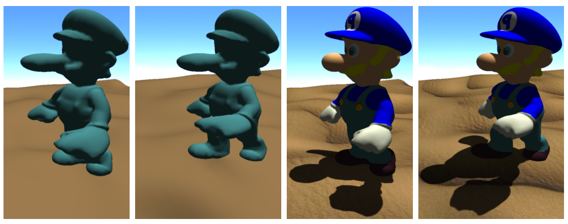


Figure 16: *By animating the shadow mesh with the same skeleton and using the same blend weights as the original geometry, the edited shadow can be animated.*

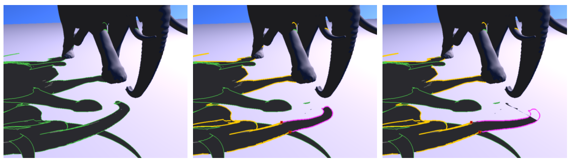


Figure 17: *Large edits in cases where the caster is close to the receiver are a problem since the geometry gets very distorted. While the distortion in the middle image is still fine, it becomes too much and artifacts follow (right image).*

Animation example Once we finish editing a shadow and finalize it by creating the shadow mesh, it can be used as the shadow of an animated character (see Figure 16). We also enlarged the fists to demonstrate that the edited parts of the mesh can be moved. With surface deformation methods [OPP10, RTD*10], such edits would be bound to the receiving surface and not move with the shadow casting object.

9. Discussion and Limitations

As an inherent problem when dealing with edits that are not physical, no guarantee can be given that the system behaves as desired in all cases. A failure case can be seen in Figure 17, where the desired edit cannot be completed by our method because it is too different from the physical shadow. The level of control one can exert on the shadow boundary currently depends on the tessellation level of the shadow caster. Methods have been proposed which can adaptively refine coarse shadow boundaries [MSWI12]. However, to make those changes permanent would also require remeshing of the shadow mesh.

While we did not conduct a formal user study, we presented our system in the R&D department of a major Japanese Anime Studio. The feedback was very positive, such as that the method could be useful in an Anime-style production pipeline, where shadows are frequently used for artistic purposes.

Dependence on shadow volumes Since we exploit some properties os the shadow-volume algorithm which are not shared by shadow mapping, it would be hard to use shadow mapping. As an alternative, we plan to adapt a method which solves the known shadow-volume issues with alpha mapping and robustness [SOA11]. For production rendering, it would be straightforward to replace shadow volumes with shadow ray tracing. On the other hand, we believe that for potential application scenarios (for example, editing the shadows of a couple of objects in a medium-sized scene), shadow volumes should work fine and not run into performance issues. To avoid explicit ray casting, the ray queries could be replaced by capturing and querying a depth map from the light source after some modifications of the volume construction.

10. Conclusions

We proposed a novel method for artistic shadow editing based on modifying the physical shadow boundaries on the receiver surface. The main strength of this algorithm is the intuitive interface for the manipulation of shadow boundaries, which handles hard and soft shadows on any kind of receiver geometry, is easily understandable and allows both fine grained shadow adjustments as well as large shadow deformations. Through the introduction of shadow meshes, the modified shadows can be plausibly used in interactive applications and with animated characters and light sources.

Acknowledgments

This work has been supported by funding from the Japan Society for the Promotion of Science (JSPS) and by the EU FP7 People Programme (Marie Curie Actions) under REA Grant Agreement no. 290227.

References

[AAM03] Assarsson U., Akenine-Möller T.: A geometry-based soft shadow volume algorithm using graphics hardware. *ACM Trans. Graph. 22*, 3 (2003), 511–520. 8

[Bar97] Barzel R.: Lighting controls for computer cinematography. *J. Graph. Tools 2* (October 1997), 1–20. 1

[BMH98] Bartz D., Meissner M., Hüttner T.: Extending graphics hardware for occlusion queries in opengl. In *Proceedings Workshop on Graphics Hardware* (1998), pp. 97–ff. 4

[Cro77] Crow F. C.: Shadow algorithms for computer graphics. *Computer Graphics (Proceedings SIGGRAPH) 11* (July 1977), 242–248. 3, 7

[DCFR07] DeCoro C., Cole F., Finkelstein A., Rusinkiewicz S.: Stylized shadows. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)* (Aug. 2007). 2

[EK02] Everitt C., Kilgard M.: *Practical and Robust Stenciled Shadow Volumes for Hardware-Accelerated Rendering*. Tech. rep., NVIDIA Corporation, mar 2002. 7

[HZ00] Hertzmann A., Zorin D.: Illustrating smooth surfaces. In *Proceedings SIGGRAPH* (2000), pp. 517–526. 4

[KPD10] Kerr W. B., Pellacini F., Denning J. D.: Bendylights: Artistic control of direct illumination by curving light rays. *Computer Graphics Forum 29*, 4 (2010), 1451–1459. 3

[MP09] Mitra N. J., Pauly M.: Shadow art. *ACM Trans. Graph. 28* (2009), 156:1–156:7. 2

[MSWI12] Mattausch O., Scherzer D., Wimmer M., Igarashi T.: Tessellation-independent smooth shadow boundaries. *Computer Graphics Forum 31*, 4 (2012), 1465–1470. 10

[NJS*11] Nowrouzezahrai D., Johnson J., Selle A., Lacewell D., Kaschalk M., Jarosz W.: A programmable system for artistic volumetric lighting. *ACM Trans. Graph. 30*, 4 (July 2011), 29:1–29:8. 3

[NSACO05] Nealen A., Sorkine O., Alexa M., Cohen-Or D.: A sketch-based interface for detail-preserving mesh editing. *ACM Trans. Graph. 24* (2005), 1142–1147. 4, 6

[NSM07] Nakajima H., Sugisaki E., Morishima S.: Tweakable shadows for cartoon animation. In *Proceedings WSCG* (2007). 3, 9

[OPP10] Obert J., Pellacini F., Pattanaik S. N.: Visibility editing for all-frequency shadow design. *Comput. Graph. Forum 29*, 4 (2010), 1441–1449. 2, 9

[PF92] Poulin P., Fournier A.: Lights from highlights and shadows. In *Proceedings symposium on interactive 3D graphics* (1992), pp. 31–38. 2

[PH10] Pharr M., Humphreys G.: *Physically Based Rendering, Second Edition: From Theory To Implementation*, 2nd ed. Morgan Kaufmann Publishers Inc., 2010. 9

[PTG02] Pellacini F., Tole P., Greenberg D. P.: A user interface for interactive cinematic shadow design. *ACM Trans. Graph. 21* (2002), 563–566. 2

[RTD*10] Ritschel T., Thormählen T., Dachsbacher C., Kautz J., Seidel H.-P.: Interactive on-surface signal deformation. *ACM Trans. Graph. 29*, 4 (2010), 36:1–36:8. 2, 9

[SMW06] Schaefer S., McPhail T., Warren J.: Image deformation using moving least squares. *ACM Trans. Graph. 25* (July 2006), 533–540. 5

[SOA11] Sintorn E., Olsson O., Assarsson U.: An efficient alias-free shadow algorithm for opaque and transparent objects using per-triangle shadow volumes. *ACM Trans. Graph. 30*, 6 (2011), 153:1–153:10. 10

[Sor06] Sorkine O.: Differential representations for mesh processing. *Computer Graphics Forum 25*, 4 (2006), 789–807. 6

[TABI07] Todo H., Anjyo K.-i., Baxter W., Igarashi T.: Locally controllable stylized shading. *ACM Trans. Graph. 26* (July 2007). 2

[Wil78] Williams L.: Casting curved shadows on curved surfaces. *Computer Graphics (Proceedings SIGGRAPH) 12*, 3 (1978), 270–274. 7