EUROGRAPHICS 2011 / M. Chen and O. Deussen
(Guest Editors)

Volume 30 (2011), Number 2

GeoBrush: Interactive Mesh Geometry Cloning

Kenshi Takayama[1,2], Ryan Schmidt[3], Karan Singh[3], Takeo Igarashi[1], Tamy Boubekeur[4] and Olga Sorkine[2]

[1]The University of Tokyo [2]New York University [3]University of Toronto [4]Telecom ParisTech – CNRS LTCI

Abstract

We propose a method for interactive cloning of 3D surface geometry using a paintbrush interface, similar to the continuous cloning brush popular in image editing. Existing interactive mesh composition tools focus on atomic copy-and-paste of pre-selected feature areas, and are either limited to copying surface displacements, or require the solution of variational optimization problems, which is too expensive for an interactive brush interface. In contrast, our GeoBrush method supports real-time continuous copying of arbitrary *high-resolution surface features between irregular meshes, including topological handles. We achieve this by first establishing a correspondence between the source and target geometries using a novel generalized discrete exponential map parameterization. Next we roughly align the source geometry with the target shape using Green Coordinates with automatically-constructed cages. Finally, we compute an offset membrane to smoothly blend the pasted patch with C^1 continuity before stitching it into the target. The offset membrane is a solution of a bi-harmonic PDE, which is computed on the GPU in real time by exploiting the regular parametric domain. We demonstrate the effectiveness of GeoBrush with various editing scenarios, including detail enrichment and completion of scanned surfaces.*

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems

1. Introduction

3D surface meshes with high-resolution geometric details are frequently used in computer graphics and modeling applications, such as product prototyping and digital content creation. They are typically created by scanning real-world objects or by digital sculpting with sophisticated software tools such as ZBrush [Pix10] or MudBox [Aut10]. Since 3D modeling is a time-consuming process, it is highly desirable to be able to reuse the results of previous efforts and combine features from existing models when making new ones.

In 2D image editing, one very popular tool for content reuse is the *cloning brush*, which allows incremental interactive copy of image parts onto a new location on the canvas (see e.g. [Ado10]). In addition to composition, it is often used to fill holes in an image after an object was removed. In this work, our goal is to design a similar interactive cloning operation for high-resolution meshes in 3D. Our proposed GeoBrush tool enables continuous copy-paste of detailed geometry, allowing combining features from different models, filling missing geometry parts (e.g. due to scanning artifacts) and cloning of arbitrary geometric details.

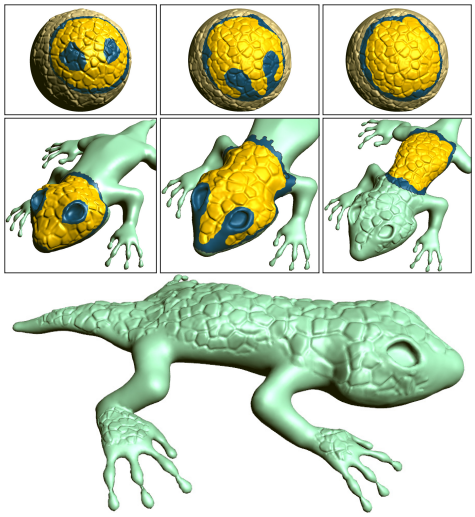

Figure 1: *Cloning geometric details using GeoBrush.*

Geometry cloning significantly differs from image cloning and is much more challenging in many respects.

Firstly, while images are essentially height functions over a regular parametric domain, general 2-manifolds in 3D are not. Surfaces in 3D do not possess canonical "up" and "right" directions, and the correspondence between the source and target surfaces for cloning purposes is difficult to define: one needs to establish intrinsic coordinate systems that match in terms of orientation and scale. A one-to-one correspondence may even be impossible, e.g., if the surface patches have different topology. Secondly, in the surface case it is not clear what exactly one wants to clone: the features need to be "peeled" from the source and glued onto the target surface, taking into account the difference in the overall curved shapes of both surfaces and deforming the features accordingly. Finally, the smoothness of the result is a concern: while in the image case one may get away with C^0 or even discontinuity at the boundary of the pasted region, for surfaces one typically needs at least C^1 continuity.

Previous works partially address some aspects of the geometry cloning problem, however none provide a complete solution which could handle general input and work in real time. Most existing tools either allow copy-paste of height-field geometric details expressed as displacements over an explicitly defined smooth base surface [BMBZ02, Aut10, Pix10], or transplanting of entire large-scale features such as arms and legs [SLCO*04, YZX*04, SBSCO06, FAT07], which is realized as a one-time operation rather than a continuous cloning brush. Note that defining a base surface may be unintuitive for the user and restricts the type of geometry one can copy to embossed "stamps". Copying displacement vectors independently of each other easily leads to distortions and self-intersections if the copied features are large and the target surface is curved; the smoothness at the pasted boundary may also suffer [BS08]. For this reason the more recent approaches above employ a variational formulation which requires solving a global optimization problem (typically a discretized polyharmonic PDE over the cloned patch). While providing better results, such optimization is costly and prohibits the use of a paint brush interface in real time even for moderately-sized meshes, since a new system of equations over an irregular domain is set up and solved for each cloned patch.

The contribution of our work is a general tool for cloning geometry with a paint brush interface that can handle irregular meshes of high resolution and arbitrary surface details in real time. In particular, GeoBrush enables copy-paste of features that significantly differ from simple normal displacements, including highly-protruding geometry and non-disk topology. We avoid the explicit separation of the source surface into smooth base and details and rather deform the entire source patch to follow the overall shape of the target surface, which allows working with a very broad range of shapes of both source and target surfaces. Our formulation ensures a smooth C^1 connection of the cloned geometry to the rest of the surface, and we preserve the source patch connectivity while stitching it into the target mesh in real time,

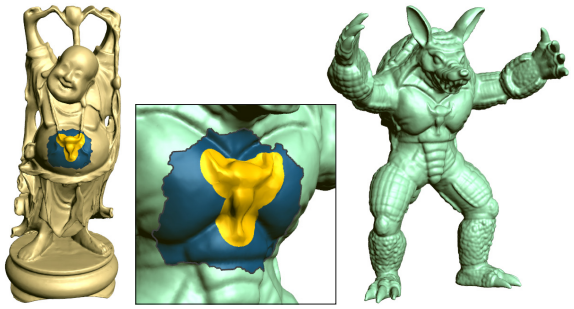

Figure 2: *GeoBrush enables to clone various areas (gold) from a selected canvas region onto a target surface.*

as the user moves the brush. Moreover, the entire cloned region may be interactively transformed and dragged to various locations on the target surface.

Our user interface allows the user to interactively modify the cloning area, including or excluding features on the surface by painting and erasing while watching continuously updated cloned geometry on the screen. This interaction supports creative exploration much more fluently than the traditional lasso-and-paste approach. Furthermore, our painting algorithm automatically corrects the brushed region to "flood" partially painted features, such that the selected area is always valid for cloning. We demonstrate the effectiveness of our approach through various artistic cloning scenarios, and also show that our tool can be useful for completing scanned surfaces.

2. Previous work

Many previous works have considered the problem of reusing existing geometry to synthesize new models. Some have been largely automated; for example, the Shuffler system [KJS07] enables point-and-click swapping of similar parts using precomputed compatible segmentations. Closer to our work, many systems focus on reproducing the "cut-and-paste" interface metaphor on 3D surfaces.

An arbitrary part can be inserted into a target surface by first cutting a suitable hole in the target, aligning the part with the hole, defining a boundary correspondence, and then finally blending the two surfaces. Kanai et al. [KSMK99] described techniques for manually authoring each of these steps, while the Modeling By Example system [FKS*04] automated them via "intelligent scissor" cutting, ICP alignment and automatic generation of smooth fillet surfaces. Variational techniques based on Poisson [YZX*04,HFAT07] and bi-Laplacian [SLCO*04,FAT07] systems have also been proposed to generate smoother transitions and handle differences in scale. To increase user control, SnapPaste [SBSCO06] automatically "snaps" a part into place as a user interactively drags it near a suitable target hole. This snapping is driven by a Soft-ICP alignment parameterized by the cursor speed, after which a smooth blend surface is geometrically derived. Note that in each of these tools, the artist

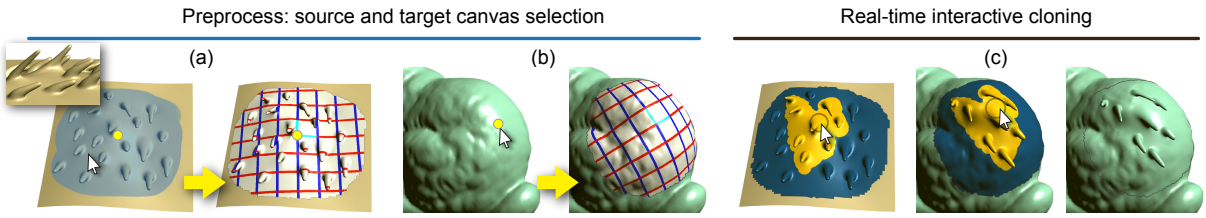

Figure 3: *The workflow in GeoBrush. (a) The user clicks on the origin point for the source canvas (yellow point) and selects the desired canvas region by painting. After mouse release, the system parameterizes the canvas and displays the coordinate lines (the user may adjust their orientation). (b) The user clicks on the origin location for the target canvas. The system suggests the initial parameterization, and the user may adjust the scaling and orientation of the coordinate lines. (c) The user paints on either the source or target surface canvas, and the geometry is cloned and stitched onto the target in real time. The corresponding source and target brush locations are displayed as circles.*

must define an initial global orientation for the part, and also a target hole. The meshmixer system [SS10] introduces a "drag-and-drop" metaphor which automates these two steps using a parameterization of the part boundary, which is projected onto the target surface via local parameterization. As a result, the artist can interactively drag the part across the target surface with real-time visual feedback.

Surface details that can be encoded as tangent-frame displacements can be copied using compatible local parameterizations, which in turn free the user from having to specify global orientation and a suitable target boundary. Biermann et al. [BMBZ02] took advantage of subdivision topology to extract displacement vectors from a source region and resample them at the target. This interface was extended to arbitrary meshes by Fu et al. [FTZ04]. Similar strategies have been applied to transfer details represented in differential form. For example, Sorkine et al. [SLCO*04] copied differential vectors between suitably-parameterized source and target regions, then solved a bi-Laplacian problem on an irregular mesh domain to reconstruct the new target surface. Zatzarinni et al. [ZTS09] propose to copy relief height functions, which are normal displacements computed via an optimization, without the need for an explicit base surface.

Commercial 3D sculpting tools [Pix10, Aut10, Lux10] also support the extraction of vector-displacements from existing models. These displacement "stamps" can then be applied to any other surface. Note, however, that as in all the tools described thus far, the source geometry is fixed. Schmidt and Singh [SS10] presented a simple clone-brush-style application of these techniques to arbitrary meshes, in which the artist first indicates canvas regions on the source and target meshes. The source is then smoothed and vector displacements are extracted; compatible parameterizations are defined between the smoothed source and target. After this preprocessing, the artist uses a brushing interface to define the area and magnitude of detail transfer. Like other displacement-based techniques, this approach is limited to source regions with disk topology and assumes that the tangent frames at the smoothed source and target have a relatively similar distribution, otherwise the features may become severely distorted or even self-intersecting. Our GeoBrush system is inspired by this technique, but does not share these significant limitations.

Viewing this body of work as a whole, we see that existing works are either limited to transfer of local displacements between disk-like patches using artist-friendly brushing interfaces, or composition of discrete parts using complex multi-step interactions. We propose an alternative, where the copied shape is not limited to disk topology or displacement features, but can still be interactively cloned via an intuitive brushing interface. An additional advantage of GeoBrush is maintaining a valid manifold mesh at all times, even during real-time paint-brushing, such that the user may stop any time and no "baking" postprocess is necessary.

3. User interface

We use a successfully established image cloning workflow [Ado10] as a guideline for GeoBrush. In image cloning, the screen coordinates define a natural mapping between the source and target images. The user only has to indicate a single *stamp* point on the source that is then mapped to the point where the user begins to paint on the target image. The source can be restamped at any time to change the (translational) correspondence between the source and target for subsequent cloning. Advanced options allow the user to set a relative rotation and scale between source and target, but these are seldom used in practice. The opposite is true for 3D surface cloning: Rarely is the view-projected planar parameterization for arbitrary 3D surfaces foldover-free and with low distortion, making a direct view-based correspondence between source and target surfaces ill-defined. Further unlike images, it is very common to desire 3D surface details to be cloned with different scale and orientation.

We thus introduce a step into the surface cloning workflow (see Fig. 3), where the user explicitly defines *canvases* on both source and target surfaces. The canvases roughly enclose the region to be cloned and provide a local parametric correspondence between source and target. Users interac-

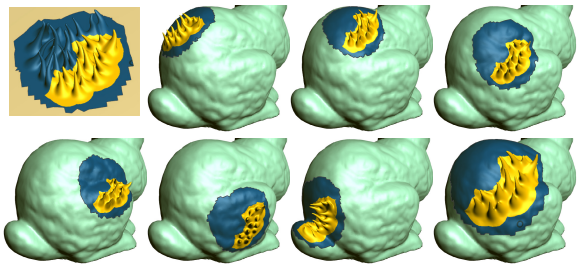

Figure 4: *The user is free to drag the target canvas region across the surface, rotate and scale it; the painted ROI geometry is preserved.*

tively control the relative position, orientation and scale of the canvases. Having established this parametric correspondence between canvases, the user can freely clone detail by painting in real-time within either source or target canvas. As with restamping in image cloning, the canvases can be redefined at any time.

3.1. Interactive canvas placement

As the role of the canvases is to roughly demarcate the region of operation of the clone brush, it is quickly specified by the user on the source surface by painting with a large brush. The first point the user clicks on defines the stamp point, i.e. the origin of the source canvas (see Fig. 3a, left). The canvas boundary should be a single loop, but the topology of the inner surface itself need not be a disk and can have a higher genus. The system then parameterizes the source canvas and displays (u,v) coordinate lines as texture on the surface (Fig. 3a), with the origin and the u,v axes' directions marked (Fig. 3a, right). The v-direction is initially aligned with the screen-space up-vector, and the user can interactively rotate it if desired.

Next, the user clicks on a single point on the target surface as the corresponding origin of the target canvas; the system grows the target canvas and displays its 2D parameterization as a texture (Fig. 3b). Note that the u- and v-coordinate lines in the source and target canvases (the red and blue lines in the texture) correspond to each other, as do the cyan tangent frame vectors painted at the origin points (they signify corresponding "right" and "up" directions on the canvases). The user can freely define a 2D rotation and scale of the target canvas relative to the source; initially the system aligns the v-coordinate direction with the screen-space up-vector.

3.2. Real-time painting

The user now applies the clone brush by selecting the desired brush size and painting either on the source or target canvas. The painting operation defines a region of interest (ROI) on the source canvas that is continuously copied to the target. Boolean operations on the ROI are also possible (using modifier keys), subtracting or adding to the already painted parts. The system performs the cloning operation instantly and displays the result in real time (Fig. 3c). GeoBrush enables automatic ROI extension to include whole surface features (Fig. 9d) that the user brushed only partially. Additionally, the user can interactively re-adjust the placement, orientation and scale of the target canvas at any time by transforming and dragging it on the target surface, while retaining the painted and cloned ROI (Fig. 4).

4. Algorithm

Our guiding principles for designing GeoBrush are interactive speed and treatment of nontrivial details and topology. We take advantage of the recent advances in discrete differential geometry to be able to generate smooth transitions between the cloned patches and the target surface; however, we wish to avoid the computational complexity involved in solving the related variational problems (i.e., solving a PDE on an irregular mesh domain, which requires expensive matrix factorizations). Our key idea is to exploit the regular 2D domain provided by local surface parameterization and solve the variational problem on the GPU. This approach already proved to be very effective for image editing [MP08].

4.1. Overview

Figures 5 and 6 show the overview of the preprocessing and the real-time stages of our algorithm. Let $M_s = (V_s, T_s)$ and $M_t = (V_t, T_t)$ denote the source and target triangle meshes, respectively. To prepare for the cloning brush operation, we establish correspondence between *canvas* areas on the source and the target meshes via common 2D parametric domain (Fig. 5a). Given the user-defined source canvas $P_s \subset M_s$ we compute a mapping $E_s : P_s \rightarrow \mathbb{R}^2$. Similarly, the target canvas $P_t \subset M_t$ is parameterized onto the same 2D domain by $E_t : P_t \rightarrow \mathbb{R}^2$, such that $E_s(P_s) \subset E_t(P_t)$, and the user-specified origin points of P_s and P_t map to the same point in $\mathbb{R}^2$. The details of the parameterization definition are described in Section 4.2. Note that this is not a classical one-to-one parameterization since P_s and P_t may have different topologies and generally E_* are many-to-one mappings; however, for our purposes it suffices to require that the boundaries ∂P_s and ∂P_t are both single loops and to pose mild conditions on the pasted regions (see Section 4.5).

The second stage of the preprocess is the deformation of P_s to approximately non-rigidly align it with the overall shape of P_t (Fig. 5b). We do this by applying the Green Coordinates (GC) space deformation $G : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ and obtain a "draft geometry" $G(P_s)$. The details of the automatic cage construction necessary to define G are given in Section 4.3.

Once the above preprocessing is done, the user can start using the cloning brush and painting on the target canvas. Each time the brush is applied, it implies a new region of interest (ROI) on the source canvas and a corresponding ROI $Q \subset \mathbb{R}^2$ in the parametric domain (Fig. 6a). Note that the ROI may have multiple boundary loops; the only requirement is that these boundary loops have a unique mapping and do not self-intersect the ROI in the parametric domain (see Section 4.5).

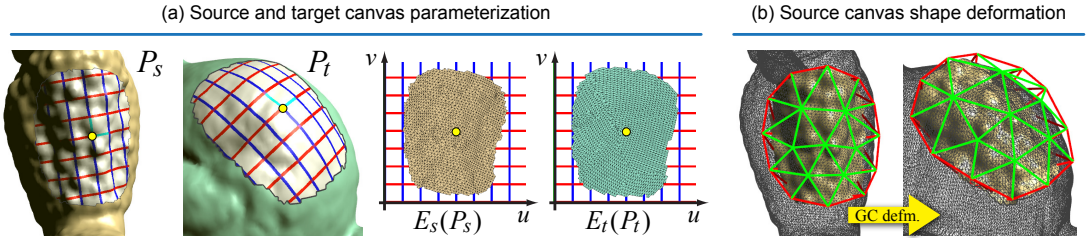

Figure 5: *Overview of the GeoBrush preprocess. (a) Canvas selection and parameterization. (b) Construction of compatible cages and deformation of the source canvas to roughly align it with the target (via Green Coordinates [LLCO08]).*

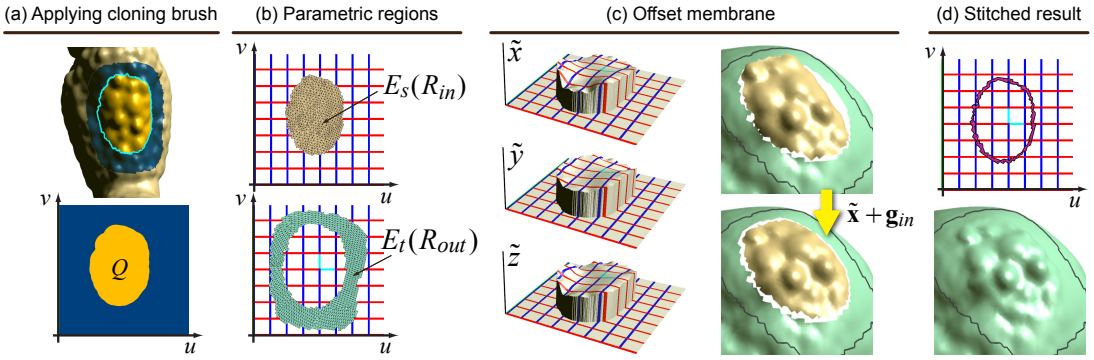

Figure 6: *Real-time cloning. (a) The user paints the ROI inside the source (or target) canvas; it corresponds to the parametric region Q. (b) Flattened submesh R_{in} (allowed to contain inner foldovers) and flattened part of the target canvas* outside *of the cloned area (R_{out}). (c) The components of the offset membrane (left) which smoothly deforms $\mathbf{g}_{in}$ to match with the target surface (right). (d) We triangulate the gap between $E_s(R_{in})$ and $E_t(R_{out})$ (top) to yield the cloned result (bottom).*

With every brush movement, we collect the contiguous subset mesh R_{in} of the source canvas to be cloned onto the target (its vertices are $\{\mathbf{v} \in P_s | \; E_s(\mathbf{v}) \in Q\}$, Fig. 6b). We would like to paste the warped shape $\mathbf{g}_{in} = G(R_{in})$ onto the target surface, but naturally, there is some mismatch in the shape and placement of $\mathbf{g}_{in}$ w.r.t. P_t (Fig. 6c). We offset $\mathbf{g}_{in}$ so that it smoothly attaches to the target (Fig. 6c-d) by computing a C^1 offset membrane, as described in Section 4.4.

Apart from painting with the cloning brush, the user may also change the location of the target canvas origin, as well as the orientation and scale parameters, while retaining the cloned ROI (the ROI will naturally translate, rotate and scale along the surface together with the canvas, see Fig. 4). Dragging and transforming the target canvas is fast thanks to the efficient parameterization procedure (Section 4.2) and quick target GC cage generation (Section 4.3): we can reuse the most expensive part, which is Green Coordinates computation for the source (1-5 sec.), and only need to parameterize the target canvas, construct its cage and apply GC, which is about 40 times faster than computing the GC themselves.

4.2. Canvas selection and parameterization

Given the 2D painted area of the source canvas in screen space (Section 3.1), we compute the 3D canvas on the mesh (P_s) as follows. First, we pick a point on the source mesh under the starting point of the canvas painting (the origin). We then iteratively expand the region around the origin on the surface by flood fill, stopping when we hit the boundary of the painted canvas on the screen.

The clone brush operation requires a correspondence between source and target surfaces; however, this is only a rough correspondence since the shapes may be very different in terms of details and topology. If we were to represent the surfaces as a smooth base plus displacements, a one-to-one correspondence between the base surfaces would be sufficient; however, we wish to avoid this explicit decomposition and allow cloning more general geometry, e.g. extreme extrusions and handles. Therefore we employ a projection-like parameterization where many-to-one mappings are possible; this suffices for our purposes as long as the boundary of the domain has a one-to-one parameterization.

To compute a parameterization, we generalize the discrete exponential map (DEM) algorithm [SGW06]. Firstly, we provide DEM with highly smoothed normals instead of the original surface normals, such that in a sense, DEM would act as a projection onto an imaginary smooth surface. For boundary vertices $\mathbf{w} \in \partial P_s$ we use the original surface normals $\mathbf{n}(\mathbf{w})$, and we interpolate these normals at the interior

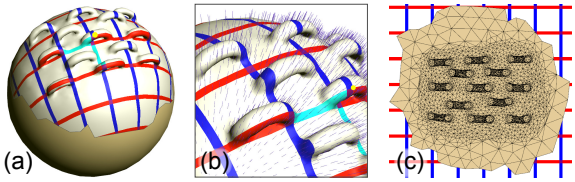

Figure 7: *(a) Generalized Discrete Exponential Map. (b) To flatten complex geometric features we first generate a smooth normal field and then compute the DEM parameterization. (c) This effectively projects the complex geometry onto the imaginary surface described by the normal field, but the parameterization still respects the larger-scale curvature of the surface, as is visible in (a).*

vertices, i.e. for each interior vertex $\mathbf{v} \in P_s \setminus \partial P_s$ the normal is defined as

$$\mathbf{n}(\mathbf{v}) = \frac{\tilde{\mathbf{n}}(\mathbf{v})}{\|\tilde{\mathbf{n}}(\mathbf{v})\|}; \quad \tilde{\mathbf{n}}(\mathbf{v}) = \sum_{\mathbf{w} \in \partial P_s} \frac{\mathbf{n}(\mathbf{w})}{(\|\mathbf{w}-\mathbf{v}\|^2 + \varepsilon)}. \quad (1)$$

This interpolation is very easy to compute and sufficiently smooth for our purpose (Figure 7).

We also modify the local parameterization procedure of a vertex $\mathbf{u}$ adjacent to an already parameterized vertex $\mathbf{v}$. In the original DEM, local vectors $(\mathbf{u}-\mathbf{v})$ are projected onto the tangent plane defined by $\mathbf{n}(\mathbf{v})$, and then re-scaled to preserve (approximate) geodesic distances, effectively rotating $(\mathbf{u}-\mathbf{v})$ about $(\mathbf{u}-\mathbf{v}) \times \mathbf{n}(\mathbf{v})$. If we were to project the vertices onto the hypothetical surface described by our smoothed normal field, components orthogonal to the normals would be discarded. Hence, we can approximate this cancellation simply by skipping the re-scaling step. Similarly, we use a k-nearest neighborhood for DEM propagation instead of the mesh connectivity, as this more accurately represents adjacency in the (hypothetical) projected mesh and further improves the results.

Note that for the target canvas P_t we can typically use the original DEM parameterization, since usually P_t does not contain significant geometric features. We stop the growth of P_t's parameterization when the 2D parametric space $E_t(P_t)$ contains the entire image $E_s(P_s)$. Note also that P_t and its parameterization are computed simultaneously, whereas E_s is defined after obtaining P_s from the user's selection.

4.3. Automatic generation of cage meshes for source canvas deformation

We compute cages C_s and C_t in order to perform space deformation of P_s and align it with P_t. A cage is a low-resolution closed mesh whose interior contains the object to be warped (preferably with some distance from the cage surface, to avoid problems with the deformation smoothness). C_s and C_t should have the same connectivity and be in full correspondence; C_s should enclose P_s while C_t should approximate the overall shape of P_t.

We assume shell-like connectivity for the cages. We thus first create a 2D mesh C by coarsely triangulating the 2D region $E_s(P_s)$. The connectivity of the cages is formed by duplicating C into "top" and "bottom" sheets (green and blue cage parts in Fig. 8) and generating a triangle strip connecting the two copies along the boundary (red edges Fig. 8). Therefore, for every vertex $\mathbf{u} \in C$ there are two corresponding vertices of the cage C_s, denoted by $\mathbf{v}_{\text{top}}(\mathbf{u})$ and $\mathbf{v}_{\text{btm}}(\mathbf{u})$.

We compute the geometry of the cage such that it approximately encloses the canvas. For every $\mathbf{u} \in C$ we assign 3D positions to $\mathbf{v}_{\text{top}}(\mathbf{u})$ and $\mathbf{v}_{\text{btm}}(\mathbf{u})$ by essentially forming a local shell around the corresponding region of the surface P_s. Let $\ell(t) = \mathbf{p}_0 + t\mathbf{n}(\mathbf{u})$ be a 3D line, where $\mathbf{n}(\mathbf{u})$ is the normal used for DEM and $\mathbf{p}_0 \in P_s$ is some point such that $E_s(\mathbf{p}_0) = \mathbf{u}$ (there is at least one). Let $\mathcal{V}(\mathbf{u})$ be the Voronoi region of $\mathbf{u}$ in the mesh C; we project all 3D points $\mathbf{p} \in E_s^{-1}(\mathcal{V}(\mathbf{u}))$ onto the line ℓ and assign the extremal projected positions to $\mathbf{v}_{\text{top}}(\mathbf{u})$ and $\mathbf{v}_{\text{btm}}(\mathbf{u})$. This simple displacement technique may lead to self-intersections and badly-shaped cage triangles (Fig. 8d); therefore we apply Laplacian mesh optimization [NISA06] with the above initial position assignments as soft constraints. While theoretically, the optimization is not guaranteed to eliminate all self-intersections, we have not encountered any problems in practice.

The positions of C_t's vertices are computed by deforming the C_s mesh using Laplacian editing [SLCO*04]:

$$\min_{\mathbf{w}_t \in C_t} \|\Delta \mathbf{w}_t - \delta\|^2. \quad (2)$$

We first compute the mesh Laplacian vectors Δ_{C_s} and then transform them according to the local frame to obtain δ. Let $\mathbf{v}_s \in C_s$ and $\mathbf{w}_t \in C_t$ be corresponding vertices on the source and target cages, and $\mathbf{u} \in C$ the corresponding point in the parametric domain; we attach a local frame to $\mathbf{v}_s$ by taking the DEM normal $\mathbf{n}(\mathbf{u})$ and the two tangent vectors (the derivatives w.r.t. the u,v coordinates) obtained from the E_s mapping; similarly we compute the local frame at $\mathbf{w}_t$ from E_t; we then use $\delta = \lambda T \Delta_{C_s}(\mathbf{v}_s)$ as the Laplacian coordinate, where λ is the scaling factor of P_t (specified by the user, see Section 3) and T is the transformation between the local frames of $\mathbf{v}_s$ and $\mathbf{w}_t$. The minimization in Eq. (2) requires positional constraints; we constrain the boundary midpoints $(\mathbf{w}_{\text{btm}} + \mathbf{w}_{\text{top}})/2 = E_t^{-1}(\mathbf{u}_{\mathbf{w}})$, where $\mathbf{u}_{\mathbf{w}} \in C$ is a boundary vertex in the parametric domain and $\mathbf{w}_{\text{btm}}, \mathbf{w}_{\text{top}}$ are its corresponding vertices in C_t.

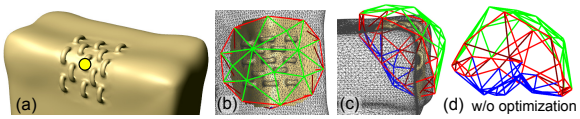

Figure 8: *(a) Cage generation for a challenging case where the source canvas is curved. Our method produces a reasonable shape (b-c), whereas the displacement-only approach leads to self-intersections (d).*

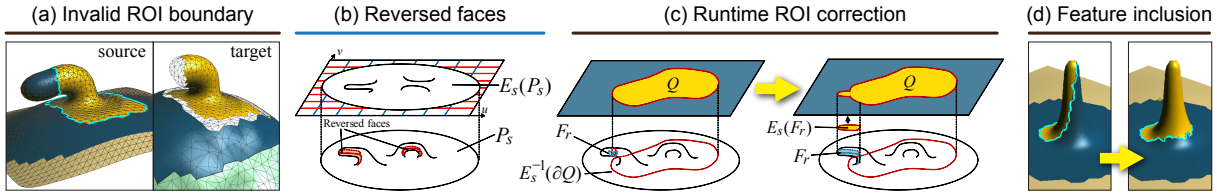

Figure 9: *ROI correction. (a) The painted ROI boundary is invalid (it crosses foldover regions in the parametric domain). GeoBrush automatically detects and corrects such ROI on the fly. (b) We flag the faces that got reversed in the parametric domain during the preprocess. (c) During painting, we detect whether the ROI boundary ∂Q crosses the set of reversed ROI faces F_r and grow F_r until ∂Q becomes valid. (d) The face flagging can be extended such that protruding features are automatically included in the ROI.*

4.4. Computing the offset membrane and stitching

As explained in Section 4.1, each time the user applies the cloning brush, it defines a new ROI $Q \subset \mathbb{R}^2$ in the parametric domain. We collect the ROI source mesh R_{in} (its vertices are $\{\mathbf{v} \in P_s |\ E_s(\mathbf{v}) \in Q\}$), and we also compute the submesh R_{out} of the target canvas that maps completely outside of Q and thus remains unchanged after cloning (its vertices are $\{\mathbf{v} \in P_t |\ E_t(\mathbf{v}) \notin Q\}$). See Fig. 6 for notation explanation. Recall that we have a draft geometry $\mathbf{g}_{in} = G(R_{in})$ from the GC deformation of the source canvas, but it does not match the target surface precisely. We wish to deform $\mathbf{g}_{in}$ such that it can be attached to the boundary ∂R_{out} of the target surface with tangent continuity. For this, we can solve the following variational problem, akin to transplanting in [SLCO*04]:

$$\min \int_\Omega \|\Delta \mathbf{x} - \delta_{in}\|^2, \quad \text{s.t. } \mathbf{x}|_{\partial\Omega} = R_{out}, \quad \left.\frac{\partial \mathbf{x}}{\partial \mathbf{n}}\right|_{\partial\Omega} = \frac{\partial R_{out}}{\partial \mathbf{n}},$$

where $\mathbf{x}$ is the stitched geometry, $\delta_{in} = \Delta \mathbf{g}_{in}$ are the Laplacians of the draft geometry and Ω is the domain mesh of the pasted patch, i.e. $\mathbf{g}_{in}$. This equation means that the local geometric details of $\mathbf{x}$ should be as close as possible to those of the draft geometry, and $\mathbf{x}$ should coincide with the target surface along the boundary, both in positions and derivatives. Minimizing the above amounts to solving a bi-Laplacian PDE:

$$\Delta^2 \mathbf{x} = \Delta \delta_{in}, \quad \text{s.t. } \mathbf{x}|_{\partial\Omega} = R_{out}, \quad \left.\frac{\partial \mathbf{x}}{\partial \mathbf{n}}\right|_{\partial\Omega} = \frac{\partial R_{out}}{\partial \mathbf{n}}. \tag{3}$$

To avoid expensive numerical solution of the PDE on the irregular Ω domain, we use the membrane trick, i.e. we compute a smooth offset membrane function $\tilde{\mathbf{x}} : Q \subset \mathbb{R}^2 \rightarrow \mathbb{R}^3, \tilde{\mathbf{x}}(u,v) = (\tilde{x}, \tilde{y}, \tilde{z})$ by solving the bi-harmonic equation in the parametric domain instead:

$$\begin{aligned} \Delta^2 \tilde{\mathbf{x}} &= 0, \quad \text{s.t.} \\ \tilde{\mathbf{x}}|_{\partial Q} &= R_{out} - \mathbf{g}_{in}, \quad \left.\frac{\partial \tilde{\mathbf{x}}}{\partial \mathbf{n}}\right|_{\partial Q} = \frac{\partial R_{out}}{\partial \mathbf{n}} - \frac{\partial \mathbf{g}_{in}}{\partial \mathbf{n}}. \end{aligned} \tag{4}$$

The membrane has the same topology as Q and offsets all points on $\mathbf{g}_{in}$ whose parametric mapping is (u,v) to $\mathbf{g}_{in} + \tilde{\mathbf{x}}(u,v)$ (see Fig. 6c). Solving the above equation can be efficiently done on the GPU. The final geometry to be stitched is then $\mathbf{x} = \tilde{\mathbf{x}} + \mathbf{g}_{in}$, or in other words, each vertex $\mathbf{v} \in R_{in}$ is mapped to $\tilde{\mathbf{x}}(E_s(\mathbf{v})) + G(\mathbf{v})$. We triangulate the gap between ∂R_{in} and ∂R_{out} in the parameter space to stitch the mesh connectivity.

Instead of solving the bi-harmonic PDE in Eq. (4), we could have used a fast approximation based on mean value coordinates [FHL*09], but this technique is applicable only when the parametric region Q is a topological disk, and does not allow constraining tangents at the boundary. We thus choose to solve the actual PDE on the GPU, similarly to [MP08]. Since we work in the parametric domain, we convert the Neumann boundary conditions into constraints on the parametric derivatives:

$$\left.\frac{\partial \tilde{\mathbf{x}}}{\partial u}\right|_{\partial Q} = \frac{\partial R_{out}}{\partial u} - \frac{\partial \mathbf{g}_{in}}{\partial u}, \quad \left.\frac{\partial \tilde{\mathbf{x}}}{\partial v}\right|_{\partial Q} = \frac{\partial R_{out}}{\partial v} - \frac{\partial \mathbf{g}_{in}}{\partial v}. \tag{5}$$

In order to easily solve the bi-harmonic problem using an iterative GPU solver, we factor it into two second-order problems: we solve a Laplace equation for the derivatives $D_u = \partial\tilde{\mathbf{x}}/\partial u$ and $D_v = \partial\tilde{\mathbf{x}}/\partial v$ using the constraints above, and then we solve for $\tilde{\mathbf{x}}$ using the Poisson equation $\Delta\tilde{\mathbf{x}} = \mathrm{div}(D_u, D_v)$ with the positional boundary constraints. We employ simple iterations as in [MP08] and interleave the iterations of the derivative and the position equations for better robustness, recomputing the derivatives from the positions each time. The GPU implementation is described in the appendix.

Fig. 10 shows the effect of using derivative constraints for the membrane computation. We artificially set $G(\cdot)$ to constant zero in Fig. 10a-b, such that the membrane essentially forms a hole-filling surface; the advantage of having tangent continuity is evident (Fig. 10b). For the actual cloning scenario, we clearly observe that the derivative constraints work effectively, especially when the target geometry is highly curved (Fig. 10c-d).

4.5. Automatic ROI correction

The cloning operation becomes undefined when the boundary of the ROI, when mapped to the parametric domain, crosses a part where E_s maps multiple points of P_s onto to a single point in $\mathbb{R}^2$, because the topology of ∂R_{in} then differs from that of ∂R_{out} (Fig. 9a). Similarly, the ROI bound-

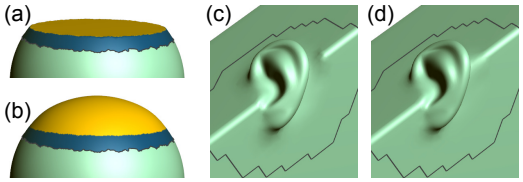

Figure 10: *Test cases with $G(\cdot) = 0$ to compute hole-filling surfaces without (a) and with (b) derivative constraints. Results of cloning the* Mannequin*'s ear onto the edge of a cube (90° bend) without (c) and with (d) derivative constraints.*

ary should not cross any foldovers of the target canvas (although this is rare since usually the target does not contain significant features and can be parameterized without self-intersections). Therefore, we check whether Q is in an invalid configuration every time the user paints, and automatically correct Q if necessary, as follows. In the preprocess, after computing the parameterization E_s, we assign a *reversed* flag to each face of P_s that is reversed in the parameter space (i.e., the dot product of the face normal and the normal used for DEM is negative, see Fig. 9b). When the user paints, we detect a set of reversed faces $F_r \subset P_s$ that are crossed by ∂Q in the parameter space, and grow F_r further by adding other reversed faces in P_s that are adjacent to F_r. Finally, we add a region slightly larger than $E_s(F_r)$ to Q (Fig. 9c).

The above procedure can be extended to automatically include sharply protruding features into the painted ROI, to make the cloning process even easier on the user. We change the criterion for reversed face flagging: instead of checking whether the dot product between the face normal and the DEM normal is negative, we check whether it is below a small positive threshold (Fig. 9d). We also propagate the *reversed* flag to k-ring neighbors ($k = 3$ in our code).

5. Results and discussion

We implemented our system in C++ and GLSL and tested it on various machines, including 2.6 GHz CPU with 3.0 GB of RAM and an NVIDIA Quadro FX 570M GPU. From our experience, the preprocessing step (source and target canvas parameterization, cage computation and Green Coordinates precomputation) typically takes 1-5 seconds, with the GC precomputation accounting for most of the time spent.

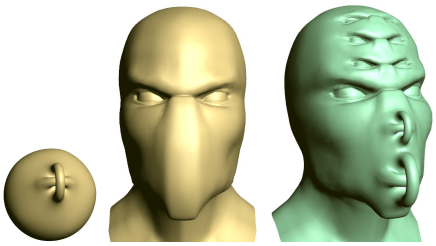

Figure 11: *Scary monsters created by cloning nontrivial topology and concave features.*

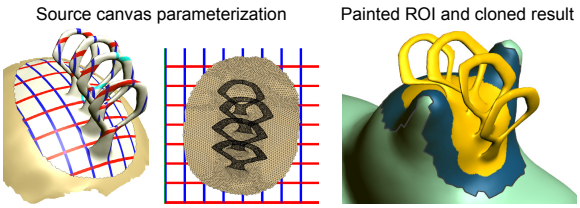

Figure 12: *Cloning handle-like features.*

The cloning brush operations run in real time. Dragging of the target canvas across the target mesh runs at interactive speed as well since no GC precomputation is necessary, as discussed in Section 4.1. Please refer to the accompanying video for live captures of GeoBrush in action.

Figures 1, 2, 11-16 demonstrate various results achieved with our system. GeoBrush is quite versatile and enables cloning intricate geometry in an intuitive manner. The painting metaphor allows to create complex ROI shapes, including non simply-connected domains, as can be seen in Fig. 1 (this example took a user with limited 3D modeling experience under 5 minutes to create). The automatic feature inclusion (or exclusion) makes it easy to control the ROI, relieving the user from highly-accurate painting (see Fig. 15).

GeoBrush easily handles complex non-height-field features such as the bent spike in Fig. 14. In contrast, many existing works such as [SS10, Lux10] attempt to find a smooth base surface via fairing, which tends to degenerate for such features (see Fig. 14c). Although additional smoothing will collapse the spike to the plane, in the clone brush tool of Schmidt et al. [SS10] the degenerate geometry cannot be reconstructed on the target surface (Fig. 14d).

We conducted a stress test for GeoBrush in Fig. 13, where we cloned intricate features onto concave and curved surfaces. Previous pasting approaches which are based on displacement copying would lead to self-intersections in the cloned geometry, but our method avoids artifacts and preserves the features robustly, while bending them to adapt to the shape of the new base domain. The surface quality of our results is consistent with that of variational cut-and-paste methods (e.g. [SLCO*04]), while enjoying real-time speed and continuous brush interface.

GeoBrush can be useful for interactive scanned surface completion and repair. Automatic surface reconstruction methods typically fill in holes where scanned data is missing by a smooth surface [KBH06]. Our tool allows to restore geometric details in the missing regions by cloning them from another part of the surface or from another model. Fig. 16a-d shows a synthetic example where we artificially created a smooth "hole" in the *Armadillo* model and completed it in GeoBrush; Fig. 16e-g demonstrates a real-life example, where the head model in Fig. 16e was obtained from stereo acquisition using Poisson surface reconstruction [KBH06] (due to significant missing data, Poisson reconstruction tends to create some blobs on the top of the

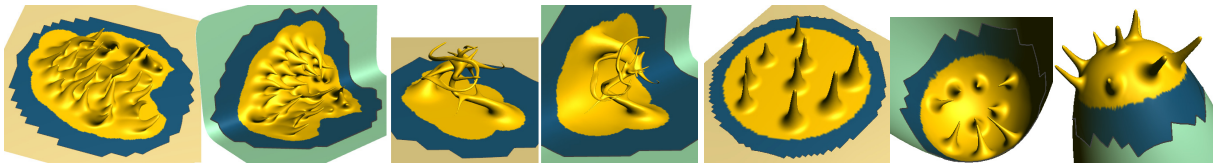

Figure 13: *source; target. A stress test for our cloning method. Highly-protruding complex features are challenging to clone, especially onto concave or high-curvature regions.*

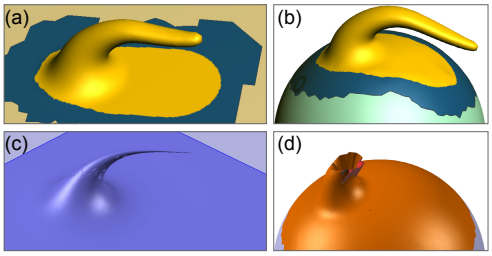

Figure 14: *(a) Cloning a bent spike. (b) GeoBrush successfully handles this protruding feature, while the cloning tool of Schmidt et al. [SS10] (c-d) fails, because the smoothing necessary to define a base surface causes degeneracy.*

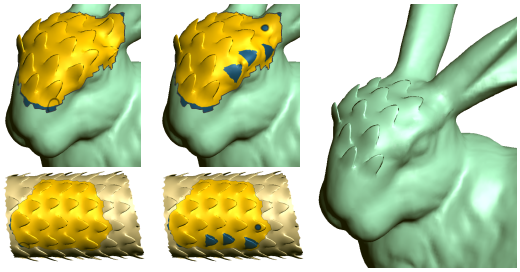

Figure 15: *Our automatic feature (dis)selection (Section 4.5) removes individual features with a single click.*

head). We completed this model by cloning parts from different meshes (Fig. 16f-g). Differently from the automatic context-based surface completion of [SACO04], GeoBrush allows full and interactive control over the source of the completed geometry while being very simple to use.

Limitations. In our current setup, the user is somewhat limited in the canvas and ROI selection due to the topological restrictions: the canvas must have a single boundary, and boundaries of the canvas and ROI must not fold over in the source and target parameterization. In particular, the canvas cannot have a cylinder or sphere topology, although this would be useful in practice. In future work, we would like to alleviate this restriction; at the least, inner holes (i.e., inner boundary loops) in the canvas can be achieved with minimal effort since the DEM parameterization can handle such cases. Moreover, while changing the target canvas is fast, re-stamping or changing the location of the source canvas is currently slow due to the need to compute GC. We would like to look into faster alternatives for canvas warping to enable interactive re-stamping.

6. Conclusion

We presented a technique for interactive geometry cloning using a painting interface, extending the popular image cloning metaphor to surfaces in 3D. We handled the added complexity of dealing with curved manifolds, involved surface features and non trivial topology by using a generalized notion of DEM parameterization, detail-preserving space warping with automatic cage construction, and smooth membrane computation. Compared with the image cloning interface, we had to introduce an explicit canvas definition step which can be seen as extended source stamping, resulting in an interface that retains the fluidity of image cloning but with the additional functionality needed to clone 3D surface detail. Our novel technique of automatically adapting the clone brush to completely include/exclude surface features, as proposed in Section 4.5, would be equally applicable and useful in image cloning.

In addition to the future work mentioned earlier, an important issue to explore is cloning brush re-application, i.e. adding detail to the already cloned geometry. Currently our working assumption is that the initial target canvas geometry should be completely replaced by the source; however, being able to perform additive cloning while retaining existing features would be a useful functionality.

Acknowledgments

We are grateful to Scott Schaefer and Eric Landreneau for the beautiful models with scale-like features [LS10] and to Alexander Hornung for valuable comments. This work was supported in part by an NSF award IIS-0905502, by MITACS, by 3DLife EU N.o.E. and by a gift from Adobe Systems.

References

[Ado10] ADOBE SYSTEMS INC.: Photoshop CS5, July 2010. http://www.adobe.com/photoshop/.

[Aut10] AUTODESK INC.: Mudbox 2011, July 2010. http://www.autodesk.com/mudbox.

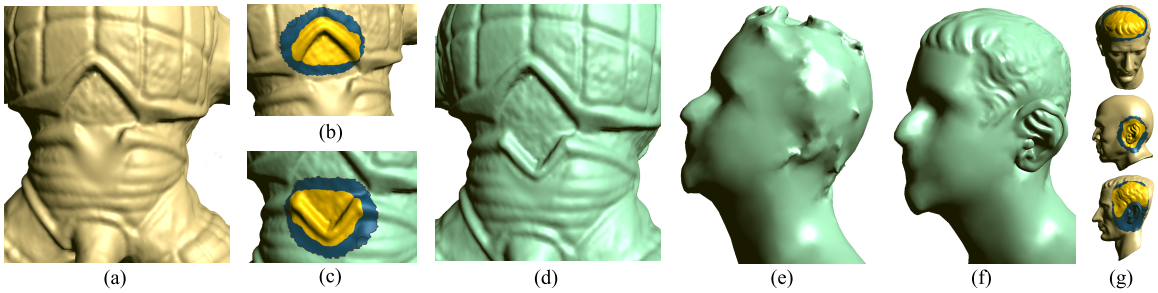

Figure 16: *(a-d) Missing or damaged features of a surface can often be repaired by cloning a similar region. (e) In other cases, no similar regions are available, so we synthesize the missing details (f) using other models (g).*

[BMBZ02] BIERMANN H., MARTIN I., BERNARDINI F., ZORIN D.: Cut-and-paste editing of multiresolution surfaces. In *Proc. ACM SIGGRAPH* (2002), pp. 312–321.

[BS08] BOTSCH M., SORKINE O.: On linear variational surface deformation methods. *IEEE TVCG 14*, 1 (2008), 213–230.

[FAT07] FU H., AU O. K.-C., TAI C.-L.: Effective derivation of similarity transformations for implicit Laplacian mesh editing. *Computer Graphics Forum 21*, 1 (2007), 34–45.

[FHL*09] FARBMAN Z., HOFFER G., LIPMAN Y., COHEN-OR D., LISCHINSKI D.: Coordinates for instant image cloning. *ACM Trans. Graph. 28*, 3 (2009), 67:1–67:9.

[FKS*04] FUNKHOUSER T., KAZHDAN M., SHILANE P., MIN P., KIEFER W., TAL A., RUSINKIEWICZ S., DOBKIN D.: Modeling by example. *ACM Trans. Graph. 23*, 3 (2004), 652–663.

[FTZ04] FU H., TAI C.-L., ZHANG H.: Topology-free cut-and-paste editing over meshes. In *Proc. Geom. Model. and Proc.* (2004), pp. 173–182.

[HFAT07] HUANG X., FU H., AU O. K.-C., TAI C.-L.: Optimal boundaries for Poisson mesh merging. In *Proc. SPM* (2007), pp. 35–40.

[KBH06] KAZHDAN M., BOLITHO M., HOPPE H.: Poisson surface reconstruction. In *Proc. Symposium on Geometry Processing* (2006), pp. 61–70.

[KJS07] KRAEVOY V., JULIUS D., SHEFFER A.: Model composition from interchangeable components. In *Proc. Pacific Graph.* (2007), pp. 129–138.

[KSMK99] KANAI T., SUZUKI H., MITANI J., KIMURA F.: Interactive mesh fusion based on local 3D metamorphosis. In *Proc. Graphics Interface* (1999), pp. 148–156.

[LLCO08] LIPMAN Y., LEVIN D., COHEN-OR D.: Green coordinates. *ACM Trans. Graph. 27*, 3 (2008).

[LS10] LANDRENEAU E., SCHAEFER S.: Scales and scale-like structures. *Computer Graphics Forum 29*, 5 (2010), 1653–1660.

[Lux10] LUXOLOGY INC.: Modo 401, July 2010. http://www.luxology.com/modo/.

[MP08] MCCANN J., POLLARD N. S.: Real-time gradient-domain painting. *ACM Trans. Graph. 27*, 3 (2008).

[NISA06] NEALEN A., IGARASHI T., SORKINE O., ALEXA M.: Laplacian mesh optimization. In *ACM GRAPHITE* (2006), pp. 381–389.

[Pix10] PIXOLOGIC, INC.: ZBrush 3.5R3, July 2010. http://www.pixologic.com/zbrush/.

[SACO04] SHARF A., ALEXA M., COHEN-OR D.: Context-based surface completion. *ACM Trans. Graph. 23*, 3 (2004), 878–887.

[SBSCO06] SHARF A., BLUMENKRANTS M., SHAMIR A., COHEN-OR D.: SnapPaste: an interactive technique for easy mesh composition. *Vis. Comput. 22*, 9 (2006), 835–844.

[SGW06] SCHMIDT R., GRIMM C., WYVILL B.: Interactive decal compositing with discrete exponential maps. *ACM Trans. Graph. 25*, 3 (2006), 605–613.

[SLCO*04] SORKINE O., LIPMAN Y., COHEN-OR D., ALEXA M., RÖSSL C., SEIDEL H.-P.: Laplacian surface editing. In *Proc. Symposium on Geometry Processing* (2004), pp. 179–188.

[SS10] SCHMIDT R., SINGH K.: meshmixer: an interface for rapid mesh composition. In *ACM SIGGRAPH Talks* (2010).

[YZX*04] YU Y., ZHOU K., XU D., SHI X., BAO H., GUO B., SHUM H.-Y.: Mesh editing with Poisson-based gradient field manipulation. *ACM Trans. Graph. 23*, 3 (2004), 644–651.

[ZTS09] ZATZARINNI R., TAL A., SHAMIR A.: Relief analysis and extraction. *ACM Trans. Graph. 28*, 5 (2009).

Appendix

Here we give the details of the GPU implementation for the offset membrane optimization described in Section 4.4. Denote by f the coordinate value at the cell (i.e., $f = \tilde{x}$ or $\tilde{y}$ or $\tilde{z}$) and let $g = (g_u, g_v)$ with $g_u = \partial f/\partial u$ and $g_v = \partial f/\partial v$. The values of f and g at the boundary remain constant since they are constrained. We first compute new f by diffusing values from surrounding cells while considering g, then compute g' as finite differences between the previous f values, and finally compute new g by diffusing the gradients from surrounding cells. We use damping for both f and g for stable convergence. The GPU program runs the following computation in a multigrid fashion, for each cell in parallel in each step t of the iterations. It takes f^t, g^t as input and returns f^{t+1}, g^{t+1} as output.

$$f^{t+1}[i,j] = \tfrac{1}{2} f^t[i,j] + \tfrac{1}{8}\left(f^t[i-1,j] + g^t_u[i-1,j] + f^t[i+1,j] - g^t_u[i,j] + f^t[i,j-1] + g^t_v[i,j-1] + f^t[i,j+1] - g^t_v[i,j]\right).$$

$$g'[i,j] = \begin{cases} g^t[i,j] & \text{if constrained} \\ (f^t[i+1,j] - f^t[i,j], f^t[i,j+1] - f^t[i,j]) & \text{else} \end{cases}$$

$$g^{t+1}[i,j] = \tfrac{1}{2} g'[i,j] + \tfrac{1}{8}\left(g'[i-1,j] + g'[i+1,j] + g'[i,j-1] + g'[i,j+1]\right).$$