First Person Sketch-based Terrain Editing

Flora Ponjou Tasse, Arnaud Emilien, Marie-Paule Cani, Stefanie Hahmann, Adrien Bernhardt

First Person Sketch-based Terrain Editing

Flora Ponjou Tasse[1]* Arnaud Emilien[2,3]† Marie-Paule Cani[2]‡ Stefanie Hahmann[2]§ Adrien Bernhardt[2]¶

[1] University of Cambridge
[2] Laboratoire Jean Kuntzmann (Grenoble University, CNRS) and Inria
[3] LIGUM, Dept. I.R.O., Montreal University

Abstract

We present a new method for first person sketch-based editing of terrain models. As in usual artistic pictures, the input sketch depicts complex silhouettes with cusps and T-junctions, which typically correspond to non-planar curves in 3D. After analysing depth constraints in the sketch based on perceptual cues, our method best matches the sketched silhouettes with silhouettes or ridges of the input terrain. A specific deformation algorithm is then applied to the terrain, enabling it to exactly match the sketch from the given perspective view, while insuring that none of the user-defined silhouettes is hidden by another part of the terrain. As our results show, this method enables users to easily personalize an existing terrain, while preserving its plausibility and style.

Keywords: First person editing, terrain, sketch-based modelling, silhouettes

Index Terms: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—

1 Introduction

Terrain is a key element in any outdoor environment. Applications of virtual terrain modelling are very common in movies, video games, advertisement and simulation frameworks such as flight simulators. Two of the most popular terrain modelling methods are procedural [9, 20, 17, 21] and physics-based techniques [21, 25, 4, 22, 24, 16]. The former are easy to implement and fast to compute, while the latter produce terrains with erosion effects and geologically sound features. However, the lack of controllability in these methods is a limitation for artists.

Sketch-based or example-based terrains have been very popular recently in addressing these issues [5, 28, 29, 10, 12, 27, 11]. However, many of these methods assume that the user sketch is drawn from a top view, which makes shape control from a viewpoint of interest very difficult. Others only handle a restricted category of mountains, with flat silhouettes. Lastly, terrains fully generated from sketches typically lack details. Dos Passos et al. [6] recently presented a promising approach where example-based terrain modelling and a first person point-of-view sketch are combined. However their method does not support local terrain editing and cannot handle typical terrain silhouettes with T-junctions. Moreover, terrain patches are often repeated which may spoil the plausibility of the results from other viewpoints.

*e-mail: flora.ponjou-tasse@cl.cam.ac.uk
†e-mail:arnaud.emillien@inria.fr
‡e-mail:marie-paule.cani@inria.fr
§e-mail:stefanie.hahmann@inria.fr
¶e-mail:adrien.bernhardt@inria.fr

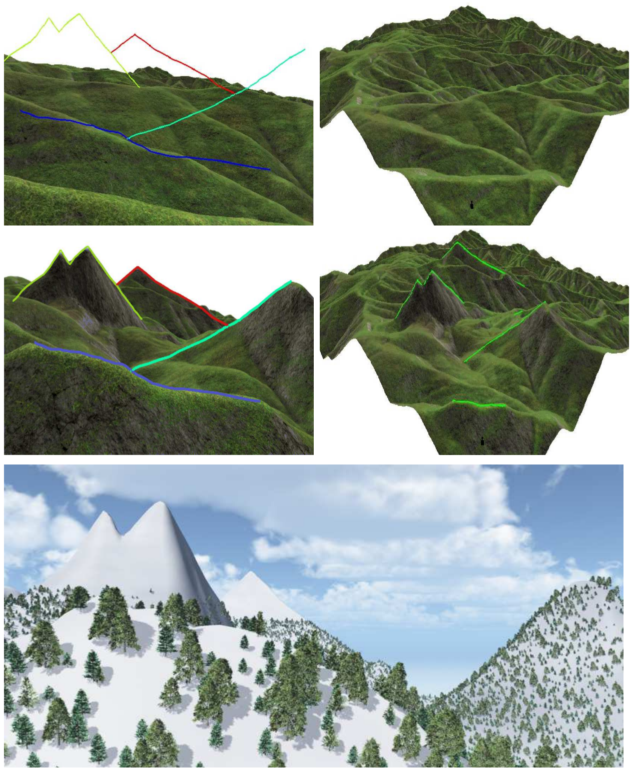


Figure 1: A typical artist sketch (top left), is used to edit an existing terrain (right). Results are shown on the second row from the same two viewpoints. Note the complex silhouettes with T-junctions, matched to features of the input terrain. The bottom image shows a rendering of the resulting terrain, from a closer viewpoint.

In this work, we address the problem of intuitive shape control of a terrain from a first person viewpoint, while generating a detailed output, plausible from anywhere. To achieve the intuitive shape control goal, we stick to the sketch-based approach, but allow the user to input complex silhouettes with cusps and T-junctions, as those typically used to represent terrains (see Figure 1). To get plausible, detailed results from anywhere, we focus on editing an existing terrain rather than starting from scratch. This approach captures the coherent small details from the existing terrain, while avoiding the patch blending and repetition problems that are typical of example-based methods. The use of an existing terrain also enables matches of sketched silhouettes with plausible, non planar curves on the terrain.

In practice, the user edits the input terrain by over-sketching it from a first person viewpoint. The user strokes, forming a graph of curves with T-junctions, represent the desired silhouettes for the terrain. The input terrain is then deformed such that its silhouettes exactly match the strokes in the current perspective view. This means that each stroke segment is to be some silhouette of the output terrain, and that no other part of the deformed terrain should hide them. Previous sketch-based modelling methods have successfully use feature curves to deform surfaces [26, 30]. Our work explores the use of terrain features for sketch-based terrain editing.

First, we order the sketched strokes by inferring their relative depth from the height of their end-points and from the T-junctions detected in the sketch. Next, features of the input terrain such as silhouette edges and ridges are assigned to each stroke and extended if necessary, to cover the length of the stroke. This assignment is the solution of a minimization problem expressing the similarity between a terrain feature and a stroke in the drawing plane, and the amount of deformation caused by their matching. The selected features then become constraints for an iterative diffusion-based terrain deformation method. Our main contributions are:

- An algorithm for ordering strokes in a complex, perspective sketch with respect to their distance from the camera.
- A method for matching terrain features with user-specified silhouettes, drawn from a given first-person viewpoint.
- A deformation method for matching silhouette constraints while preventing them from being hidden by other parts of the terrain.

Related work is discussed in Section 2. Then we present an overview of our solution in Section 3. This is followed by a description of stroke ordering in Section 4, generation of feature constraints in Section 5 and terrain deformation in Section 6. We discuss results in Section 7 before concluding.

2 Related work

Most terrain modelling systems use one or a combination of the following: procedural terrain generation, physics-based simulation, sketch-based or example-based methods. See [23] for a detailed survey.

Procedural terrain modelling methods are based on the fact that terrains are self-similar, i.e. statistically invariant under magnification [18]. These methods are the popular choice for landscape modelling due to their easy implementation and efficient computation. They mainly consist of pseudo-randomly editing height values on a flat terrain by using either adaptive subdivision [9, 20, 17] or noise [20, 21]. Adaptive subdivision progressively increases the level of detail of the terrain by iteratively interpolating between neighbouring points and displacing the new intermediate points by increasingly smaller random values. Noise synthesis techniques are often preferred because they offer better control. Superposing scaled-down copies of a band-limited, stochastic noise function generates noise-based terrains. For more information on fractal terrain generation methods, see Ebert et al. [7]. Fractal-based approaches can generate a wide range of large terrains with unlimited level of details. However, they are limited by the lack of user control or non-intuitive parameter manipulation, and the absence of erosion effects such as drainage patterns. To address the last issue, fractal terrains can be improved using physics-based erosion simulation [21, 25, 4, 22, 24, 16]. Alternatively, river network generation can be incorporated in the procedural method [15, 11]. In particular, Genevaux et al. [11] create procedural terrains from a hydrographically and geomorphologically consistent river drainage network, generated from a top-view sketch. However, this method only captures terrains resulting from hydraulic erosion, and there is no mechanism for controlling their silhouettes from a first person viewpoint.

Physically-based techniques generate artificial terrains by simulating erosion effects over some input 3D model. Musgrave et al. [21] present the first methods for thermal and hydraulic erosion based on geomorphology rules. Roudier et al. [25] introduce a hydraulic erosion simulation that uses different materials at various locations resulting in different interactions with water. Chiba et al. [4] generate a vector field of water flow that then controls how sediment moves during erosion. This process produces hierarchical ridge structures and thus enhances realism. Nagashima [22] combines thermal and fluvial erosion by using a river network pre-generated with a 2D fractal function. Neidhold et al. [24] present a physically correct simulation based on fluids dynamics and interactive methods that enable the input of global parameters such as rainfall or local water sources. Kristof et al. [16] propose fast hydraulic erosion based on Smooth Particle Hydrodynamics. The main drawback of all these methods is that they only allow indirect user-control through trial and error, requiring a good understanding of the underlying physics, time and efforts to get the expected results.

Sketching interfaces and more generally feature-based editing have been increasingly popular for terrain modelling. These methods can be combined with some input terrain data to generate terrains with plausible details.

Cohen et al. [5] and Watanabe et al. [28] present the first terrain modelling interfaces that take as input a 2D silhouette stroke directly drawn on a 3D terrain model. They only handle a single silhouette stroke, interpreted as a flat feature curve. McCrae and Singh [19] use stroke-based input to create paths which deform terrains. However user strokes are interpreted as path layouts and not as terrain silhouettes. Multi-grid diffusion methods enable generation of terrains that simultaneously match several feature curves, either drawn from a top view [12] or from an arbitrary viewpoint [2]. The main limitation is that generated terrains typically lack realistic details.

In contrast, Zhou et al. [29] use features (actually, sketch maps painted from above) to drive patch-based terrain synthesis from real terrain data. Closer to our concerns, Gain et al. [10] deform an existing terrain from a set of sketched silhouettes and boundary curves. The algorithm deforms the terrain based on the relative distance to the feature-curves in their region of influence, and on wavelet noise to add details to the silhouettes. In this work we rather use a diffusion-based deformation method to propagate feature constraints, avoiding the need for boundary curves. Lastly, Tasse et al. [27] present a distributed texture-based terrain generation method that re-uses the same sketching interface. Unfortunately, all these methods interpret each sketched silhouette as a planar feature curve, which reduces the realism of the result.

Dos Passos et al. [6] propose a different approach to address this issue. Given a set of sketched strokes drawn from a first person point-of-view, copies of an example terrain are combined such that the silhouettes of the resulting terrain match the strokes. This gives a realistic, varying depth to silhouettes. To achieve this, the algorithm assumes each stroke represents a terrain silhouette. A stroke is matched with a portion of a silhouette, selected from a set of silhouettes viewed from several standing viewpoints around the example terrain. Terrain slices representing portions of matched silhouette are cut from the example terrain and then combined through a weighted sum to produce a smooth terrain. A drawback of this method is that it does not handle complex sketches with T-junctions, which are common in landscape drawings. Moreover, the matching process may select the same silhouette portions for different strokes, thus producing unrealistic repeating patterns in the final result. Finally, the weighted sum function used for merging may fail to remove the boundary seams produced by combining different terrain slices. In this work, we address these issues by presenting a sketch-based method that handles T-junctions in complex sketches and deforms an input terrain to match the sketch rather than copy-pasting parts of it.

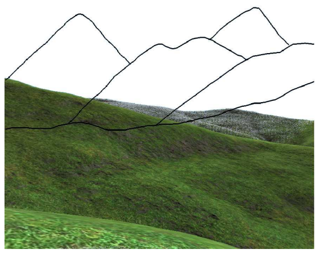
3 Overview

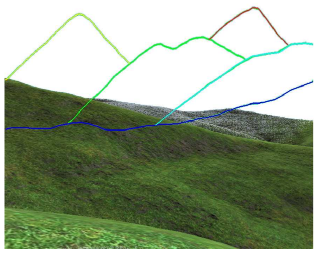
Let us describe our processing pipeline. As in many terrain modelling and rendering methods, our terrains are represented by a *height field*, namely a greyscale image storing elevation values. This representation cannot emulate features such as overhangs and caves, but it is the most prevalent format in terrain generation because of its simplicity and efficient use of storage space. For rendering purposes and silhouette detection, a 3D triangular mesh is constructed from the height field by connecting adjacent terrain points $(x, y, altitude(x, y))$. Users are able to navigate on a 3D rendering of the existing terrain, possibly flat, with a first-person camera always at a standing viewpoint. A sketch is created by drawing one or multiple strokes from the same camera position. The drawn strokes represent silhouettes that should be visible from that position. Our main goal is to deform the terrain such that these user constraints are respected. The following requirements should be satisfied:

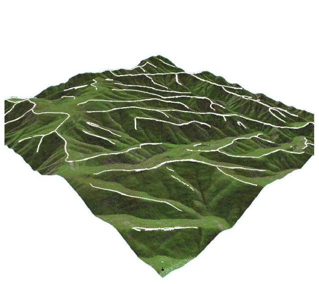
- Every sketched stroke should be a terrain silhouette, in the current perspective view from the first-person camera viewpoint.
- Each of these terrain silhouettes should be visible, i.e. not hidden by any other part of the terrain.
- The deformed terrain should not have artifacts nor contain unrealistic deformations, from any other viewpoint.

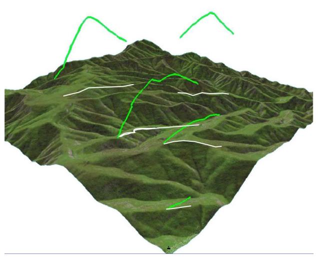
Our solution consists of five main steps, illustrated in Figure 2:

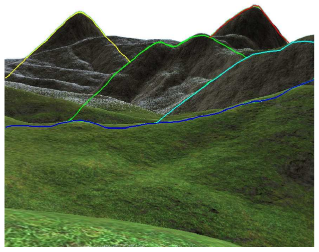
1. We order strokes according to their depth, from front to back with respect to the camera position. This order is used when we generate constraints for terrain deformation, so that a curve constraint is not occluded by another, when viewed from the first-person viewpoint.
2. Terrain features such as silhouettes and ridges are detected. Deforming existing terrain features to match the desired silhouettes results in a more realistic terrain since no extra features are added and thus, the nature of the existing terrain is best preserved.
3. For each stroke, we select a terrain feature that will be deformed to fit the stroke, when seen from the camera position. These deformed features represent the positional constraints that we use in the diffusion-based terrain deformation. A key idea of our framework is the expression of this feature selection step as an energy minimization problem, in which we penalize features with large altitude differences compared to their corresponding stroke as well as features that would result in too large deformations.
4. We use a multi-grid Poisson solver for diffusion-based terrain deformation. It solves for altitude differences instead of absolute terrain positions, thus preserving the small-scale features of the input terrain.
5. After terrain deformation, other parts of the terrain may hide the user-specified silhouettes. To address this issue, we run the following iterative process: we detect terrain silhouettes that do not fit any user stroke and yet hide one of the sketched silhouettes. Extra deformation constraints are constructed to enforce lowering these protruding silhouettes until the user-sketched silhouettes are no longer occluded. The terrain is deformed with a combination of previous constraints and the newly constructed constraints. We repeat this process until there is no longer protruding silhouette.

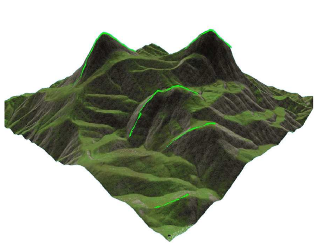

(a) User 2D sketch, in a 3D interface

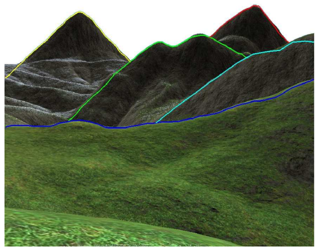

(b) Stroke ordering

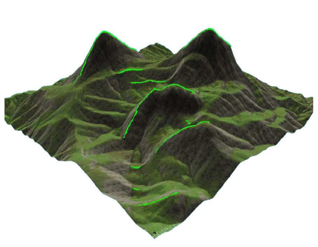

(c) Terrain feature detection (top view)


(d) Matching strokes to features (top view)


(e) Deforming terrain with matched features


(f) Terrain deformation result (from top view)


(g) Lowering protruding silhouettes


(h) Resulting terrain (top view)

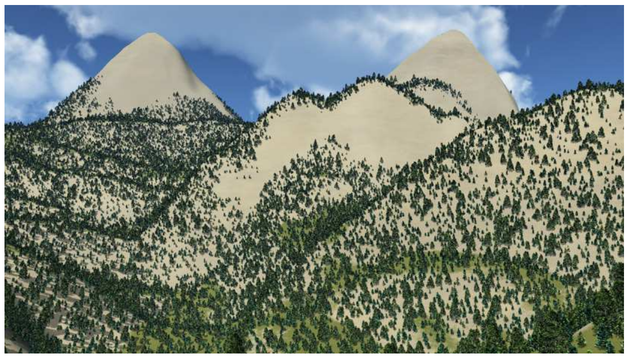

(i) Closer view of the generated terrain

Figure 2: Overview of our terrain editing framework. In (b), stroke colour indicates stroke ordering: blue indicates that a stroke is closer to the camera position and red indicates that it is the furthest. (c) illustrates detected terrain features in white and (d) shows the subset of terrain features that have been assigned to user strokes. In (f) the terrain features are deformed so that they match the strokes from the user viewpoint. The final result in (h) is obtained after removing some residual artifacts.

4 Analysing complex terrain sketches

In this section, we explain how depth ordering of silhouette strokes is extracted from the user sketch.

The different silhouette strokes in the input sketch first need to be ordered, in terms of relative depth from the camera viewpoint. This will enable us to ensure, when they are matched with features, that they will not be hidden by other parts of the terrain. Our approach to do so is based on two observations:

- If, in the viewing plane, a silhouette lies above another, it obviously corresponds to a mountain A farther away from the viewpoint than the other mountain B. Otherwise A would hide B. Using height coverage for ordering them in depth is however not sufficient, since some strokes may overlap in height, as for the green and blue strokes in Figure 3.
- Furthermore, the terrain being a height field, the projection of each stroke onto the horizon (x-axis of the viewing plane) is injective (no more than one height value per point).

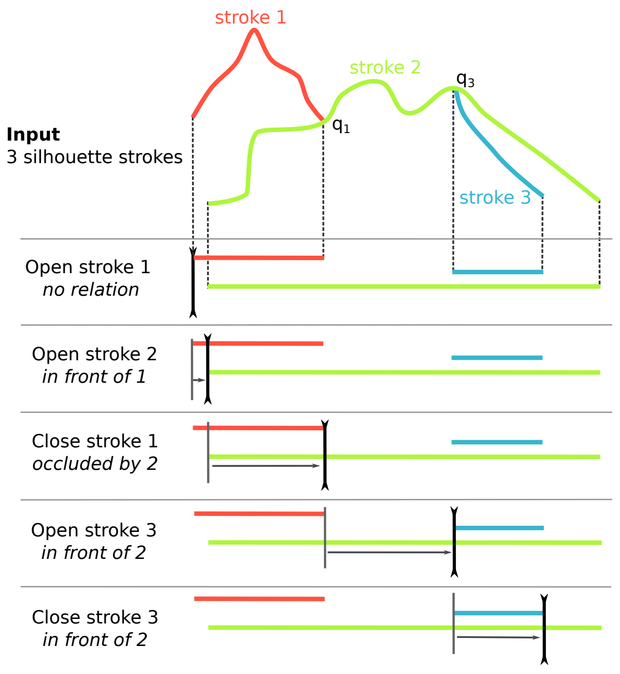


Figure 3: An input sketch (top) and the different steps of the sweeping algorithm used for scanning the sketch, labelling T-junctions and ordering strokes (bottom). As a result, stroke 3 is detected to be in front of stroke 2, which is itself in front of stroke 1.

These two observations allow us to solve the relative stroke ordering problem thanks to a new sweeping algorithm (see Figure 3): We consider the projections of all the strokes onto the horizontal x axis (depicted in the bottom part of the Figure) and sweep from left to right, examining the extremities (starting and endpoints in sweeping direction) and junction points of the silhouette strokes. While doing so, we label the strokes' extremities and the junction points in the following way: an extremity q_s of stroke s is a T-junction if its closest distance to another stroke r is smaller than a threshold. An endpoint q_s is labelled (*occluded-by*, r) if the oriented angle, measured counterclockwise, between the tangent[1] of s at q_s and the tangent of r at q_s, $\angle(t_s, t_r) < 180°$. This indicates that s is occluded by, and thus behind, r. Otherwise, s is in front of r and we label q_s as (*in-front-of*, r).

If a stroke s has no T-junctions, then it is behind a stroke r either if the projection of s completely contains the projection of r or if the smallest height value of s's endpoints is larger than the smallest height value of r's endpoints.

While scanning the sketch from left to right, we insert each stroke in a sorting structure, at a relative depth position determined by the cues above. This results in a relative ordering of the user strokes.

[1]Strokes are always oriented clockwise. Hence, stroke tangents are independent of the direction in which the stroke was sketched. When labelling a starting point q_s as T-junction, we flip its tangent.

5 Positioning strokes in world space

The key idea of our approach is to create a 3D terrain that matches the user drawing, by deforming an existing one. More precisely, we deform the features of the existing terrain, like its ridgelines, to match the user silhouette strokes. Because a terrain has many features, we first have to compute to which one of them it is the most appropriate to apply a deformation. In this section, we detail how we compute the set of terrain features (Section 5.1), how we affect one of them to each of the user strokes (Section 5.2) and we present a feature completion algorithm that infers the hidden parts of the silhouettes, enabling a more realistic terrain deformation result (Section 5.3).

5.1 Feature detection: silhouettes and ridgelines

Silhouette detection on the existing terrain is based on a common and naive algorithm for computing the exact silhouettes of a 3D mesh. Silhouette edges are detected by finding all visible edges shared by a front face and a back face in the current perspective view. Neighbouring silhouette edges are then linked to form long silhouette curves.

Ridge detection is based on the profile-recognition and polygon-breaking algorithm (PPA) by Chang et al. [3]. The PPA algorithm marks each terrain point that is likely to be on a ridge line, based on the point height profile. Segments, forming a cyclic graph, connect adjacent candidate points. Polygon-breaking repeatedly deletes the lowest segment in a cycle until the graph is acyclic. Finally, the branches on the produced tree structure are reduced and smoothed. The result is a graph where nodes are end points or branch points connected by curvilinear ridgelines. An improvement of the PPA algorithm connects all the terrain points into a graph using a height-based or curvature-based weighting and computes the minimum spanning tree of that graph [1]. Because we are mainly concerned with performance and detection of large-scale ridges, we simply connect candidate terrain points as in the original PPA algorithm and replace the polygon-breaking with a minimum spanning forest algorithm.

5.2 Stroke - Feature matching

In this section, we discuss a method for determining, for each stroke, the terrain features which can be used to construct deformation curve constraints. Viewed from the first person camera, these curve constraints should match the user-sketched strokes. To achieve this, we first construct a features priority list for each stroke and then select features for each priority list such that the sum of their associated cost is minimized.

5.2.1 Features priority list per stroke

For a stroke s, we project all terrain features on the sketching plane (i.e. we use the 2D projection of the feature from the first-person viewpoint) and select feature curves that satisfy the following condition: the x interval they cover matches the one of the stroke s. We deform the selected feature curves, and if necessary extend their endpoints, such that viewed from the camera position, they cover the length of s. This deformation is simply achieved by displacing the feature curve points according to their projection on the 2D stroke in the sketching plane, and their distance to the camera position. Let f be a terrain feature and f_p its projection on the stroke plane. If a portion of f_p is below or above s, f is truncated to that portion. Moreover, for each point $q \in f$, its altitude is modified as follows:

$$q.z = q.z + k * ||q_p - q_p^s|| * \frac{||q - eye||}{||q_p - eye||}$$

where eye is the camera position, $k = -1$ if f_p is below s and $k = 1$ otherwise, q_p the projection of q on the stroke plane, and q_p^s the intersection of s and the vertical line passing at q_p.

We used this deformed version of the feature to associate the following cost $E(f, s)$ to each feature f with respect to stroke s:

$$E = E_{dissimilarity} + E_{deformation} + E_{sampling} + E_{extension} \quad (1)$$

$$\begin{aligned}
E_{dissimilarity}(f) &= \frac{w_1}{curvelength(f_p)} * \int_{f_p} h_{f_p} dt \\
E_{deformation}(f) &= \frac{w_2}{curvelength(f)} * \int_{f} h_f dt \\
E_{sampling}(f) &= w_3 * \frac{longestedgelength(f)}{\max_{g \in prioritylist(s)} longestedgelength(g)} \\
E_{extension}(f) &= w_4 * \frac{extendedcurvelength(f)}{curvelength(f)}
\end{aligned}$$

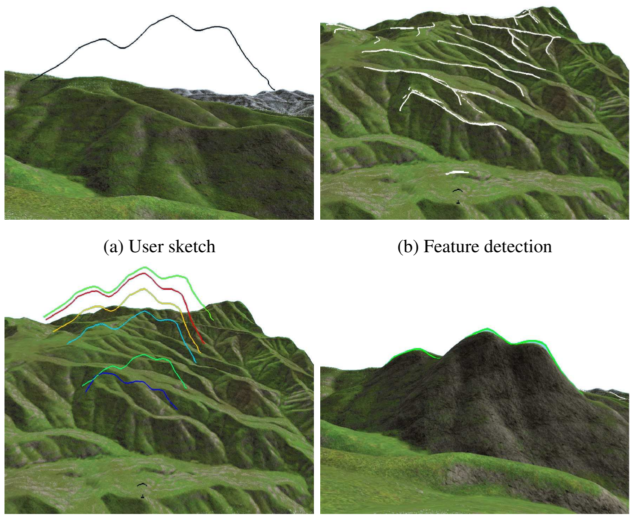
where, w_i are weights, f_p is the projection of f on the stroke plane, h_f is the altitude difference between f and f's projection on the terrain, and h_{f_p} is the altitude difference between f_p and the stroke s. $E_{dissimilarity}$ represents the dissimilarity between f and s, $E_{deformation}$ expresses the amount of deformation along f, $E_{sampling}$ penalizes features with long edges and $E_{extension}$ penalizes features that were extended to fully cover s when viewed from the camera position. All the results shown here were generated with $w_1 = w_2 = w_3 = w_4 = 1.0$.

All features are sorted in a priority list according to their cost. Figure 4 illustrates this process for a single stroke (in this simple case, the feature of minimal cost is selected).

5.2.2 Energy minimization

The goal here is the selection of a feature curve f from the priority list of each stroke s_i, to construct deformation constraints for terrain deformation. In addition to the feature order within the different priority lists, we need to take into account the depth ordering for silhouette strokes computed in Section 4.

Therefore, this selection process can be seen as a minimization problem. Let $S = \{s_i : i = 1, ..., n\}$ be the stroke list (ordered by depth) and f^i denote a feature in the priority list $L(s_i) = \{f_k^i : k = 1, ..., m_i\}$ for a stroke s_i. We are looking for $\{f^i : i \in 1...n\}$ such that $f^i < f^j$ if $i < j$ and $\sum E(f^i)$ is minimized. Here, $f^i < f^j$ means that f^i should not be occluded by f^j, so that all deformation curve constraints are visible from the first person viewpoint. We process the ordered stroke list from front to back, and after each stroke, we remove from the priority list of the next strokes, features that will be occluded if selected. We chose to process strokes from front to back for two main reasons. Firstly, strokes that are closest to the eye are processed first and due to $E_{deformation}$, the algorithm attempts to select constraints that will minimize the terrain deformation. Thus, features closer to the eye are more likely to be selected. Secondly, if all the features of interest for a given stroke s_i were already selected, and therefore its priority list was empty, an arbitrary curve on the terrain would be used instead. If this ever occurs, we prefer it to be for background silhouettes.


(a) User sketch (b) Feature detection

(c) Detect possible candidate matches (d) Terrain deformation using best match

Figure 4: Computing possible features to match with a user stroke. Images (a) and (d) show the terrain from the first person viewpoint used for editing, while image (b) and (c) use a higher viewpoint to better show features on the input terrain. Feature colour indicates cost: blue for the lowest cost and red for the highest.

In practice, feature selections that cause any stroke to have an empty priority list are penalized with a very high cost. Thus, a configuration that guarantees a non-empty priority list for each stroke is always selected, if it exists. If no such configuration exists and s_i has an empty priority list, we automatically compute a 3D embedding of the 2D stroke s_i and use the resulting curve as a deformation constraint. To easily compute this 3D embedding, we take the two strokes lying just in front and just behind s_i. Then we place s_i halfway between the terrain features assigned to these two strokes. If there is no stroke restricted to lie behind s_i, we place it behind the furthest stroke from the viewpoint. If there is no stroke restricted to lie in front of s_i, we place it in front of the closest stroke to the viewpoint. With this approach, each stroke is represented by a deformation constraint even if it was not matched to a terrain feature during the energy minimization.

The energy minimization problem we have described so far is NP-hard. We use a branch-and-bound algorithm to efficiently discard all partial solutions that have a cost higher than the current best cost, without having to explore the whole solution tree. The branch-and-bound algorithm consists of two steps: a *branching* step and a *bounding* step. The branching step consists of exploring possible choices for s_{i+1} once we have made a feature selection for s_i. In other words, we split the node (s_i, f^i) into multiple nodes (s_{i+1}, f_k^{i+1}), where f_k^{i+1} are features in the priority list of s_{i+1}. The bounding step allows the algorithm to stop exploring a partial solution if the total cost of features in the solution is higher than the cost of the best solution found so far.

It is possible for a feature to be the first choice in the priority lists for two or more strokes. To handle this, when exploring a possible solution, a feature curve assigned to a stroke is no longer considered for subsequent strokes. Our branch and bound algorithm will explore other solutions with the feature curve assigned to different strokes as long these solutions are guaranteed to have a smaller cost than the current best solution.

5.2.3 Stroke in world space

The previous minimization gives us, for each stroke s, an associated terrain feature f. However, the stroke s has its points in screen space, whereas the points of f are in the world space. Our goal is to place the stroke in the world space, in order to deduce terrain constraints, i.e. find the distance of their projection from the camera.

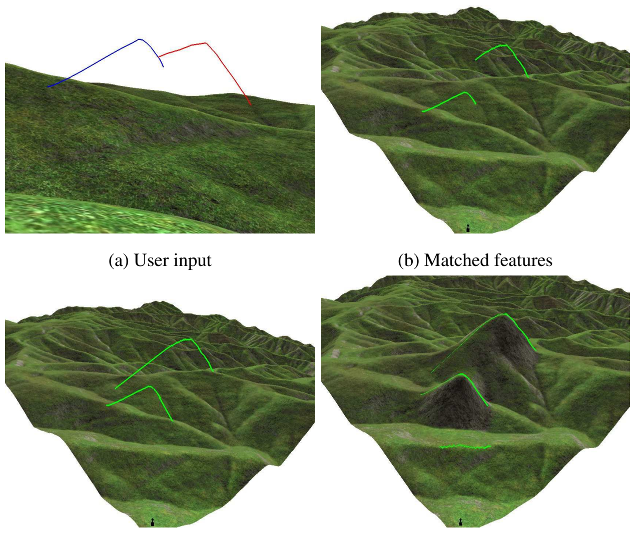
For each point of the stroke $q_s = (x_s, y_s)$, we check if there exists a feature point q_f whose projection on screen $q_p = P(q_f) = (x_p, y_p)$ has the same x-coordinate as q_s, i.e. $x_s = x_p$. If this point exists, we project the stroke point on the world space, using the distance of q_f from the camera as a depth value.

The possible undetermined points depth, at the stroke borders, are set in world space to follow the stroke tangent, in the world space.

5.3 Completing selected 3D features

Using user-specified endpoints of an occluded stroke during the generation of deformation constraints would create silhouettes that appear to start exactly at these endpoints. This can look quite unnatural when viewed from a different position than the first person camera position used for sketching: indeed, the endpoint of the occluded stroke (a junction) is typically above the terrain and thus, a sharp deformation will be created at that point.

We address this problem by simply extending 3D features assigned to strokes at both endpoints along their tangents, until they reach the surface of the terrain. An example of this feature completion is presented in Figure 5. More sophisticated contour completion methods such as the one presented in SmoothSketch [14] could alternatively be used, but this simple method was sufficient in our case, and is proposed as an optional step in the editing process.


(a) User input (b) Matched features

(c) Extend the matched features (d) Resulting terrain

Figure 5: Completing selected features: after matching 2D strokes to terrain features, we extend these features until they reach the surface of the terrain, to ensure a smooth transition from specified silhouettes to the terrain.

6 Terrain deformation

In the previous section, we analysed terrain features and used them to position the strokes in the world space. We present in this section how we use them as constraints to deform the existing terrain.

6.1 Diffusion-based equation solver

Our deformation algorithm relies on iterative diffusion of displacement constraints, which are computed from the 3D strokes positioned in the world space.

The diffusion method, first introduced by the authors in [8], consists in computing the difference of the curve height and the terrain height $\mathscr{H}$, and to diffuse these differences (instead of absolute height values) using a multi-grid Poisson solver similar to the one in [12].

More precisely, for each point $p = (x, y, z)$ of the stroke in the world space, we compute $\delta = z - \mathscr{H}(x, y)$, and set it as a displacement constraint. The constraints are rasterised on a grid, whose resolution is equal to the terrain resolution. After having set the constraints of all strokes, we perform the diffusion, which gives the displacement map $\mathscr{M}$.

The displacement is finally applied on the terrain height field $\mathscr{H}$, whose feature line silhouettes are now matching the user strokes, when seen from the first-person viewpoint used for sketching. The deformation only consists of adding the two heights, $\mathscr{H}'(x, y) = \mathscr{H}(x, y) + \mathscr{M}(x, y)$, where $\mathscr{H}'$ is the resulting terrain. Because height differences are propagated, instead of absolute heights, the terrain preserves fine-scale details during the deformation.

6.2 Lowering protruding silhouettes

After deformation, the user-defined silhouettes may be hidden by other parts of the terrain. To address this issue, we detect the unwanted protruding silhouettes and constrain them to a lower position so that the user-defined silhouettes become visible.

6.2.1 Detecting most protruding silhouette edges

First, all visible silhouettes are detected, with the algorithm discussed in Section 5.1. These silhouettes are projected onto the sketching plane. Let s be a silhouette of the deformed landscape, inherited from the example terrain. The mountain of silhouette s hides a user-specified silhouette g if s is closer to the camera than g and the projection s_p of s in the sketching plane has a higher altitude than g_p, the projection of g. In this case, s is an unwanted protruding silhouette. Determining how much s should be lowered is done as follows: Let h be the maximum height difference between s and a silhouette g hidden by s. h is the minimum altitude by which s should be lowered to ensure the silhouettes it hides become visible. Our solution is simply to uniformly lower s by an offset h. This method is applied to all unwanted protruding silhouettes and we use the set of lowered silhouettes to form new deformation constraints.

6.2.2 Updating deformation constraints

The new deformation constraints from the lowered protruding silhouettes are added to the set of constraints associated to the sketched silhouettes, and the terrain is deformed once again using the method of section 6.1. This operation maintains the user-specified silhouettes while lowering areas around the unwanted protruding silhouettes, so that user specifications are satisfied.

The process of detecting protruding silhouettes and using this information to further constrain the terrain is repeated until protruding silhouettes are no longer detected. In practice, a single iteration is usually sufficient to make all user-specified silhouette strokes visible.

7 Results

Validation examples : The examples below and the joined video illustrate the results of our method in a variety of cases. In particular, Figure 6 shows editing of a terrain with a complex sketch containing 5 T-junctions. Our method is also able to handle complex mountains where ridges are not as well-defined as they are on smooth landscapes. An example of this is presented in Figure 7. Our proposed approach differs from other sketch-based methods in that non-planar silhouettes can be generated from planar user-sketched strokes. This is illustrated in Figure 8. Moreover, the method is robust enough to support terrains with few or no features, as shown in the example given in Figure 9. This complex sketch-based editing framework comes with interactive rates, as illustrated in the attached video, which makes it a very attractive alternative to other terrain generation/editing techniques discussed in Section 2.

User tests: We performed an informal user test with two experienced computer artists. The system was briefly introduced to the users, who had no prior knowledge about it. They were then asked to draw sketches to deform existing terrains. Both of them reported that our system was very easy to take in hand and use, and where able to quickly create new sceneries. Their feedback indicates that the approach is original, and seems a promising way to create a scene that matches their artistic intend. However, they had some difficulties for predicting the result of their sketch. This could have been improved by manually tuning the feature matching weights, which was not among their skills. These first users also asked for the ability to move within the scene and edit the terrain from multiple viewpoints. Note that this is not a technical issue: we simply did not implement it since this editing mode was already provided by previous methods. Lastly, the users insisted on the natural aspect of the resulting terrain, and noted that it matched their sketches in the expected way.

Limitations: Although our system succeeds in matching a complex user-sketch through a natural deformation of the terrain, based on its existing features, the lack of predictability of the stroke-feature solver may be a problem. A full user study would be useful to understand the user intent when sketching over the existing terrain, and identify the ranges of weight values allowing us to better match this intent. We could also improve our matching method using extra error functions.

Another limitation comes from our deformation solver. The diffusion-based deformation method sometimes creates small declivities around the extremity of a constraint curve, when the slope of the curve is high and the extremity is located on the terrain: in this case, the terrain locally inflates, except at this end-point where the deformation is zero, which causes the problem. Using an inverse distance to deform a terrain [13] does not work either, because of our use of curves as constraints. Future work still needs to be done on terrain deformation, especially for curve-based deformations.

8 Conclusion

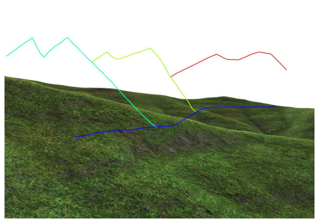
We presented the first sketch-based modelling method enabling the deformation of a terrain from a single viewpoint. The user sketches a few silhouette strokes forming a graph with T-junctions, similar to the silhouette representations used in artistic terrain sketching. A key feature of our method is that sketched silhouettes are matched with existing terrain features: this enables our technique to both match silhouette strokes with a non-planar curve, and produce a deformation that does not spoil plausibility, since the structure of ridges and valleys typically remains unchanged. This work could easily be extended to a multiple-view editing interface, where the user can move over the terrain and iteratively edit it from different points of view, while keeping his other sketches as constraints.

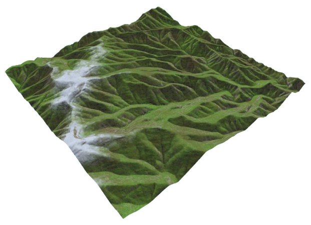
Acknowledgements

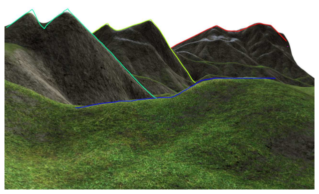
This work was conducted during an internship of Flora Ponjou Tasse at Inria Rhône-Alpes in Grenoble. It was partly supported by the ERC advanced grant EXPRESSIVE.

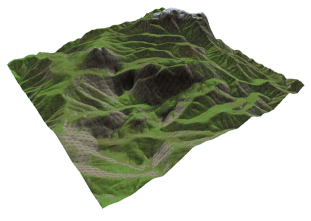
References

[1] S. Bangay, D. de Bruyn, and K. Glass. Minimum spanning trees for valley and ridge characterization in digital elevation maps. In *Proceedings of the 7th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa*, AFRIGRAPH '10, pages 73–82, New York, NY, USA, 2010. ACM.


(a) User input


(b) Existing terrain


(c) Deformed terrain


(d) Viewed from a different point

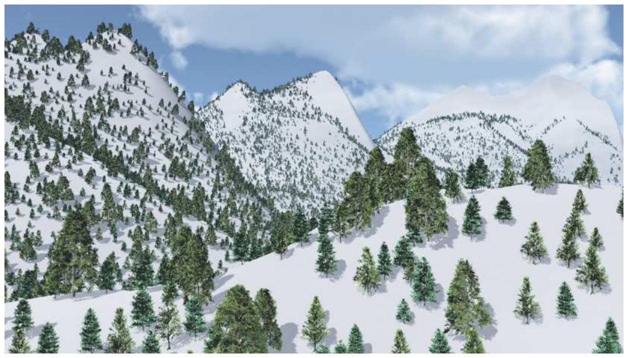

(e)

Figure 6: Terrain editing with a complex user sketch.

[2] A. Bernhardt, A. Maximo, L. Velho, H. Hnaidi, and M.-P. Cani. Real-time Terrain Modeling using CPU-GPU Coupled Computation. In *XXIV SIBGRAPI*, Maceio, Brazil, Aug. 2011.

[3] Y.-C. Chang and G. Sinha. A visual basic program for ridge axis picking on dem data using the profile-recognition and polygon-breaking algorithm. *Computers and Geosciences*, 33(2):229–237, 2007.

[4] N. Chiba, K. Muraoka, and K. Fujita. An erosion model based on velocity fields for the visual simulation of mountain scenery. *Journal of Visualization and Computer Animation*, 9(4):185–194, 1998.

[5] J. M. Cohen, J. F. Hughes, and R. C. Zeleznik. Harold: A world made of drawings. In *Proceedings of the 1st International Symposium on Non-photorealistic Animation and Rendering*, NPAR '00, pages 83–90, New York, NY, USA, 2000. ACM.

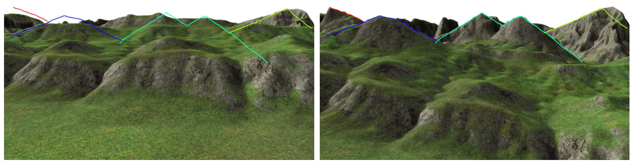
[6] V. Dos Passos and T. Igarashi. Landsketch: A first person point-of-view example-based terrain modeling approach. In *Proceedings - Sketch-Based Interfaces and Modeling, SBIM 2013 - Part of Expressive 2013*, pages 61–68, 2013.

[7] D. S. Ebert, F. K. Musgrave, D. Peachey, K. Perlin, and S. Worley. *Texturing and Modeling: A Procedural Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2002.

[8] A. Emilien, P. Poulin, M.-P. Cani, and U. Vimont. Interactive procedural modeling of coherent waterfall scenes. *Computer Graphics Forum*, 2014. to appear.

[9] A. Fournier, D. S. Fussell, and L. C. Carpenter. Computer rendering of stochastic models. *Commun. ACM*, 25(6):371–384, 1982.

[10] J. Gain, P. Marais, and W. Straer. Terrain sketching. In *Proceedings of I3D 2009: The 2009 ACM SIGGRAPH Symposium on Interactive*


(a) User input (b) Result

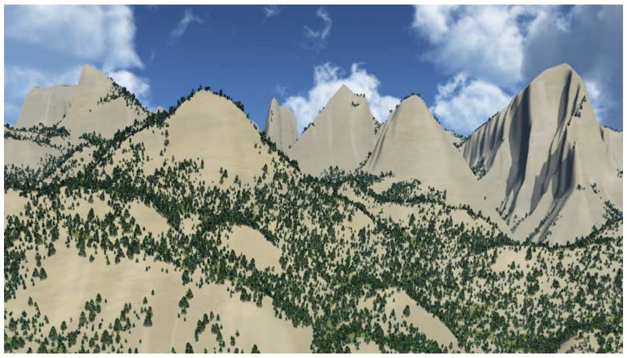
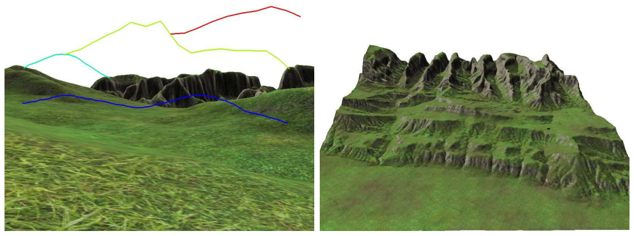

(c)

Figure 7: Editing a complex rocky mountain with a complex sketch.


(a) User input (b) Existing terrain

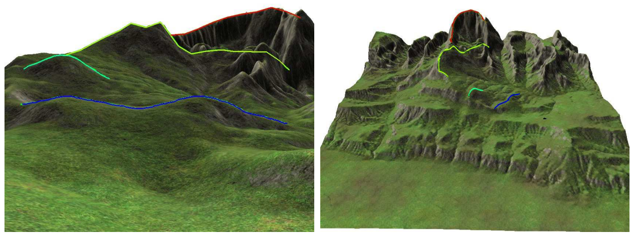

(c) Deformed terrain (d) Result viewed from a different point

Figure 8: Terrain editing produces non-planar silhouettes in the output, from 2D planar strokes.

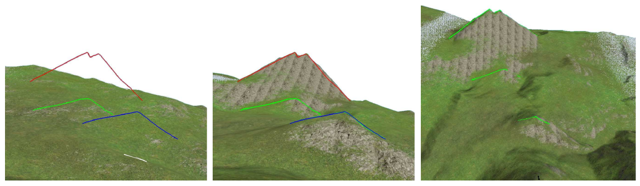

(a) User input (b) Result (c) View from another point

Figure 9: Adding deformation constraints automatically: the stroke furthest away from the user did not have an assigned feature to it and so one was automatically generated, and positioned on a plane orthogonal to the view direction, such that stroke ordering is respected.

3D Graphics and Games, pages 31–38, 2009.

[11] J.-D. Genevaux, E. Galin, E. Guerin, A. Peytavie, and B. Benes. Terrain generation using procedural models based on hydrology. *ACM Transactions on Graphics*, 32(4), 2013.

[12] H. Hnaidi, E. Guerin, S. Akkouche, A. Peytavie, and E. Galin. Feature based terrain generation using diffusion equation. *Computer Graphics Forum*, 29(7):2179–2186, 2010.

[13] H. Jenny, B. Jenny, W. E. Cartwright, and L. Hurni. Interactive local terrain deformation inspired by hand-painted panoramas. *Cartographic Journal, The*, 48(1):11–20, 2011.

[14] O. Karpenko and J. Hughes. Smoothsketch: 3d free-form shapes from complex sketches. In *ACM SIGGRAPH 2006 Papers, SIGGRAPH '06*, pages 589–598, 2006.

[15] A. D. Kelley, M. C. Malin, and G. M. Nielson. Terrain simulation using a model of stream erosion. *Computer Graphics (ACM)*, 22(4):263–268, 1988.

[16] P. Kristof, B. Benes, J. Krivanek, and O. Stava. Hydraulic erosion using smoothed particle hydrodynamics. *Computer Graphics Forum*, 28(2):219–228, 2009. Cited By (since 1996):27.

[17] J. P. Lewis. Generalized stochastic subdivision. *ACM Trans. Graph.*, 6(3):167–190, July 1987.

[18] B. B. Mandelbrot. *The fractal geometry of nature*. W. H. Freeman, New York, 1983.

[19] J. McCrae and K. Singh. Sketch-based path design. In *Proceedings of Graphics Interface 2009*, GI '09, pages 95–102, Toronto, Ont., Canada, Canada, 2009. Canadian Information Processing Society.

[20] G. S. P. Miller. The definition and rendering of terrain maps. *SIGGRAPH Comput. Graph.*, 20(4):39–48, Aug. 1986.

[21] F. K. Musgrave, C. E. Kolb, and R. S. Mace. The synthesis and rendering of eroded fractal terrains. In *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '89, pages 41–50, New York, NY, USA, 1989. ACM.

[22] K. Nagashima. Computer generation of eroded valley and mountain terrains. *Visual Computer*, 13(9-10):456–464, 1997.

[23] M. Natali, E. M. Lidal, I. Viola, and D. Patel. Modeling terrains and subsurface geology. In *Proceedings of EuroGraphics 2013 State of the Art Reports (STARs)*, pages 155–173. Eurographics, Eurographics 2013 - State of the Art Reports, May 2013.

[24] B. Neidhold, M. Wacker, and O. Deussen. Interactive physically based fluid and erosion simulation. *Natural Phenomena*, pages 25–32, 2005.

[25] P. Roudier, B. Peroche, and M. Perrin. Landscapes synthesis achieved through erosion and deposition process simulation. *Computer Graphics Forum*, 12(3):375–383, 1993.

[26] K. Singh and E. Fiume. Wires: A geometric deformation technique. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 405–414, New York, NY, USA, 1998. ACM.

[27] F. Tasse, J. Gain, and P. Marais. Enhanced texture-based terrain synthesis on graphics hardware. *Computer Graphics Forum*, 31(6):1959–1972, 2012.

[28] N. Watanabe and T. Igarashi. A sketching interface for terrain modeling. In *ACM SIGGRAPH 2004 Posters*, SIGGRAPH '04, pages 73–, New York, NY, USA, 2004. ACM.

[29] H. Zhou, J. Sun, G. b. b. Turk, and J. b. b. Rehg. Terrain synthesis from digital elevation models. *IEEE Transactions on Visualization and Computer Graphics*, 13(4):834–848, 2007.

[30] J. Zimmermann, A. Nealen, and M. Alexa. Silsketch: Automated sketch-based editing of surface meshes. In *Proceedings of the 4th Eurographics Workshop on Sketch-based Interfaces and Modeling*, SBIM '07, pages 23–30, New York, NY, USA, 2007. ACM.