Feature-based Terrain Editing From Complex Sketches

Flora Ponjou Tasse[a,*], Arnaud Emilien[b,c], Marie-Paule Cani[b], Stefanie Hahmann[b], Neil Dodgson[a]

[a] *University of Cambridge*
[b] *Laboratoire Jean Kuntzmann (Univ. Grenoble-Alpes, CNRS) and Inria*
[c] *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 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. We extend this sketch-based terrain editing framework to handle a collection of multi-view sketches. 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

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 [1, 2, 3, 4] and physics-based techniques [4, 5, 6, 7, 8, 9]. 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 popular recently in addressing these issues [10, 11, 12, 13, 14, 15, 16]. However, many of these methods [12, 14, 16] assume that the user sketch is drawn from a top view, which makes shape control from a viewpoint of interest difficult. Others [10, 11, 13, 15] only handle a restricted category of mountains, with flat silhouettes. Lastly, terrains fully generated from sketches typically lack details. Dos Passos et al. [17] 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.

In this work, we address the problem of intuitive shape control of a terrain from a first person viewpoint, while generating detailed output that is plausible from any viewpoint. To achieve the intuitive shape control goal, we stick to the sketch-based approach, but allow the user to input complex silhouettes, as those are typically used to represent terrains (see Figure 1). Our interpretation of the term "complex" is similar to the one used in SmoothSketch [18], where a complex sketch is a set of 2D strokes with hidden contours and cusps. To get plausible, detailed results from any viewpoint, 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 [19, 20]. Our work explores the use of terrain features for sketch-based terrain editing.

Paper contributions. This paper is an extended version of earlier work [21] in which we first introduced a framework for deforming terrain features to fit user strokes. First, sketched strokes are ordered 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 it-

*e-mail:flora.ponjou-tasse@cl.cam.ac.uk

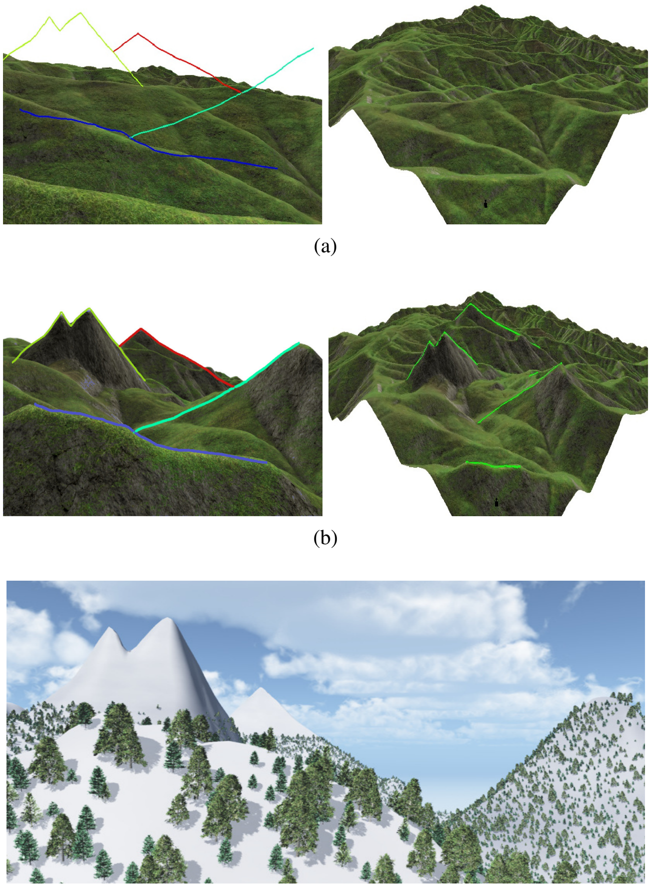


Figure 1: (a) An artist sketch (left), is used to edit an existing terrain (right). (b) Results shown from two viewpoints. Note the complex silhouettes with T-junctions, matched to features of the input terrain. (c) shows a rendering of the resulting terrain, from a closer viewpoint.

erative diffusion-based terrain deformation method. The main contributions of that earlier paper [21] 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.

This paper provides an in-depth discussion of the branch-and-bound search scheme used to address the energy minimization problem. Additionally, we propose an improved framework that supports terrain editing from multi-view sketches drawn from different viewpoints. In the context of film making, this additional tool can facilitate control of the exact shape of terrain silhouettes for two or three views, which will be used for key scenes. Although iteratively editing the terrain from multiple viewpoints could achieve realistic landscapes, there is no guarantee that silhouettes generated during one iteration will not be significantly modified by subsequent iterations. The stroke-to-feature matching algorithm is modified to handle all sketches at once, with additional constraints that ensure that no assigned feature is occluded by another. Finally, we claimed in the original paper that specifically deforming terrain features produces more realistic results. To illustrate this, we compare the use of feature-based curve constraints in terrain deformation against using 3D planar curve constraints obtained from projecting strokes on the drawing plane. We show how the two types of constraints affect terrain deformation and realism on 3 different test cases.

We begin by summarising related work (Section 2). We then give an overview of our whole system (Section 3), before describing, in detail, stroke ordering (Section 4), feature constraints (Section 5), terrain deformation (Section 6), and the modifications needed to handle multi-view sketches from various viewpoints (Section 7).

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. Natali et al. [22] provide a detailed survey.

Procedural terrain modelling methods are based on the fact that terrains are self-similar, i.e. statistically invariant under magnification. Fractals have the same concept of self-similarity [23] and thus, fractal-based methods have been widely used in terrain generation. 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 [1, 2, 3] or noise [2, 4]. 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. [24]. 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 [4, 5, 6, 7, 8, 9]. Alternatively, river network generation can be incorporated in the procedural method [25, 16]. In particular, Genevaux et al. [16] 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. [4] present the first methods for thermal and hydraulic erosion based on geomorphology rules. Roudier et al. [5] introduce a hydraulic erosion simulation that uses different materials at various locations resulting in different interactions with water. Chiba et al. [6] 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 [7] combines thermal and fluvial erosion by using a river network pre-generated with a 2D fractal function. Neidhold et al. [8] 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. [9] 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. [10] and Watanabe et al. [11] 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 [26] 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 [14] or from an arbitrary viewpoint [27]. The main limitation is that generated terrains typically lack realistic details.

In contrast, Zhou et al. [12] 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. [13] 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. [15] 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. [17] 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*, implemented as 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 the artist wishes to 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:

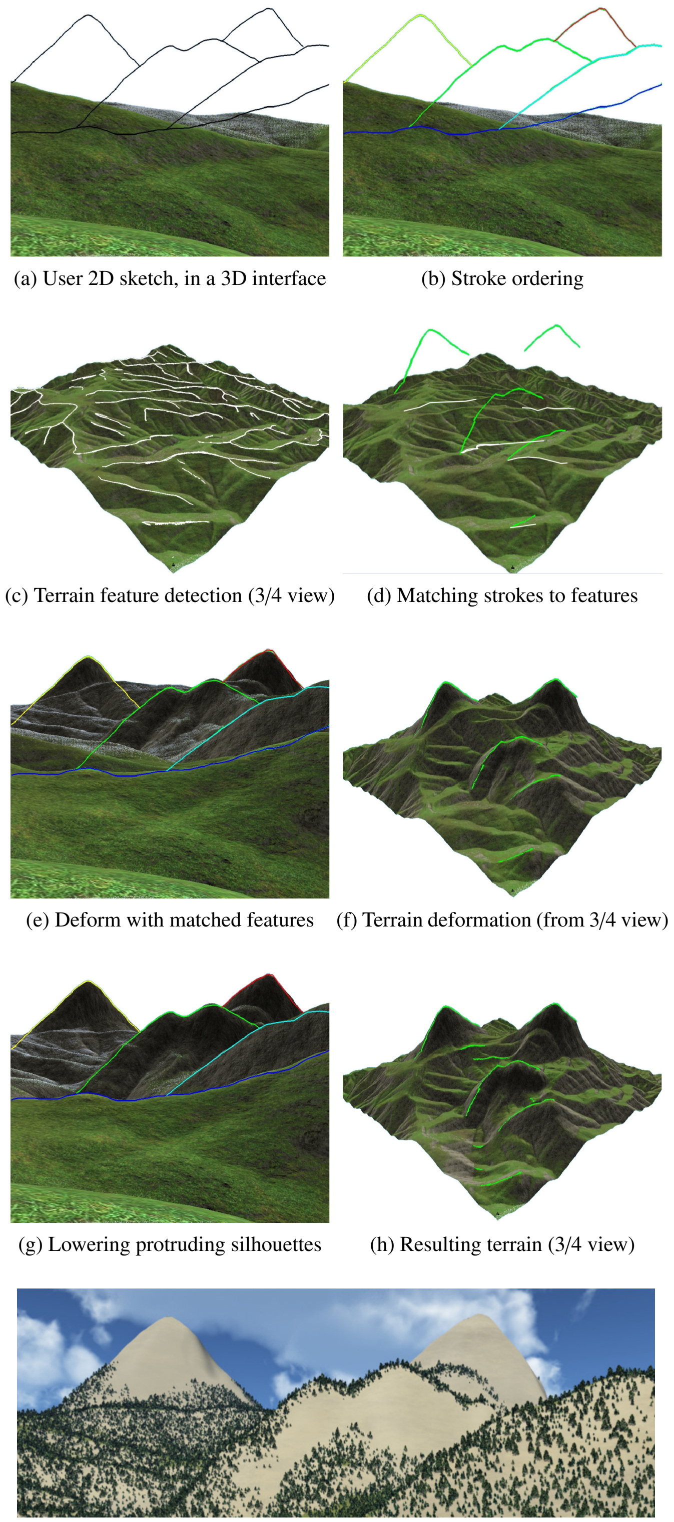
- Every sketched stroke should be a terrain silhouette, in the current perspective view from the first-person camera viewpoint.
- Each of these terrain silhouettes should be visible, i.e. not hidden by any other part of the terrain.
- The deformed terrain should not have artifacts nor contain unrealistic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

Our solution consists of five steps, illustrated in Figure 2.

Stroke ordering: 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.

Feature detection: 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.

Stroke-feature matching: 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



(a) User 2D sketch, in a 3D interface (b) Stroke ordering

(c) Terrain feature detection (3/4 view) (d) Matching strokes to features

(e) Deform with matched features (f) Terrain deformation (from 3/4 view)

(g) Lowering protruding silhouettes (h) Resulting terrain (3/4 view)

Figure 2: Overview of our terrain editing framework. (a) Unlabeled user sketch. 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 features in white and (d) shows the subset of features that are assigned to user strokes. In (e,f) the terrain features are deformed so that they match the strokes from the user viewpoint. The final result in (g,h) is obtained after removing some residual artifacts.

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.

Terrain deformation: 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.

Lowering protruding silhouettes: 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.

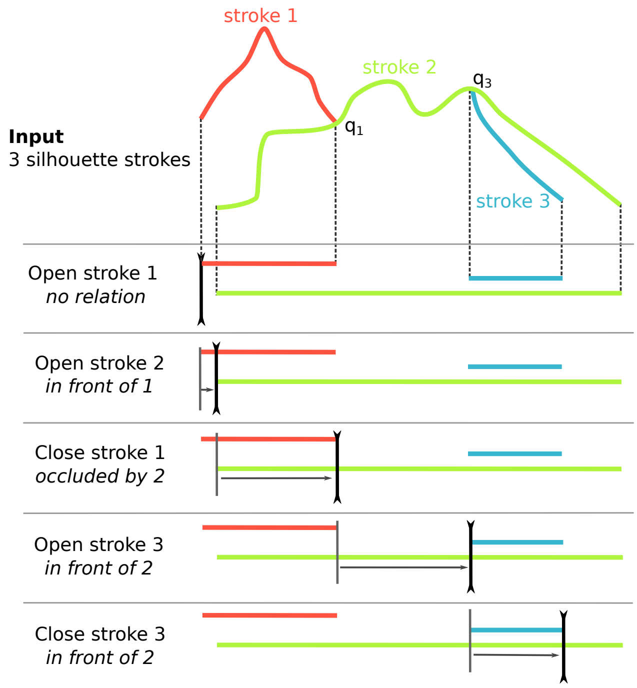


Figure 3: An input sketch (top) and the different steps of the sweeping algorithm used for scanning the sketch, labelling T-junctions and ordering strokes (bottom). As a result, stroke 3 is detected to be in front of stroke 2, which is itself in front of stroke 1. Note that the stroke colouring at the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

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 is necessary since the input strokes are not labeled and thus there no information of the order in which they should be processed. 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).

These two observations allow us to solve the relative stroke ordering problem using our 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. Information about the junction point of two strokes is used to unambiguously decide which stroke is occluded and thus, further from to the camera. 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).

In the absence of T-junctions, stroke ordering is determined using the height values at extremeties. First, we check if once both strokes are projected on the horizontal axis, the interval $[r_{right}, r_{left}]$ is a subset of $[s_{right}, s_{left}]$. If this is the case, we say that the projection of s completely contains the projection of r and s is behind r. Otherwise, the stroke with the lowest height is considered closer to the camera and thus, s is behind r 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.

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 ridge lines, 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 allocate 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. [28]. 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 [29]. 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 feature 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. Feature 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. We sweep s from

[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.

one extremity to another with a vertical line and sections of f whose projection on f_p never intersect this line are removed. Moreover, for each point $q \in f$, its altitude is modified as follows:

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

where p_c 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:

$$E = E_{\text{dis}} + E_{\text{def}} + E_{\text{sam}} + E_{\text{ext}} \tag{1}$$

$$\begin{aligned}
E_{\text{dis}}(f) &= \frac{w_1}{\text{CurveLength}(f_p)} \int_{f_p} h_{f_p} dt \\
E_{\text{def}}(f) &= \frac{w_2}{\text{CurveLength}(f)} \int_{f} h_f dt \\
E_{\text{sam}}(f) &= \frac{w_3 \times \text{LongestEdgeLength}(f)}{\max_{g \in \text{list}(s)} \text{LongestEdgeLength}(g)} \\
E_{\text{ext}}(f) &= \frac{w_4 \times \text{ExtendedCurveLength}(f)}{\text{CurveLength}(f)}
\end{aligned}$$

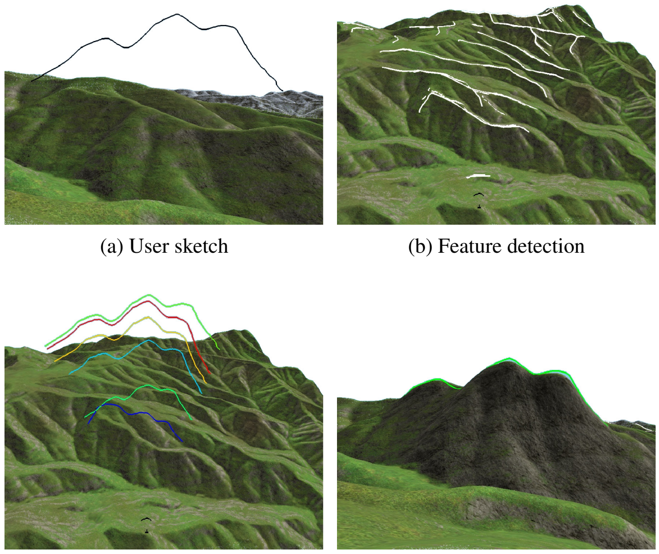
where w_i are weights, f_p is the projection of f on the stroke plane, h_f is the altitude difference between f and f's projection on the terrain, and h_{f_p} is the altitude difference between f_p and the stroke s. The cost E_{dis} represents the dissimilarity between f and s, E_{def} expresses the amount of deformation along f, E_{sam} penalizes features with long edges and E_{ext} 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. We want to find a set of stroke-feature matches such that the total cost of the assignments is minimized and the assigned features respect the pre-computed stroke ordering. 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



(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.

from front to back for two main reasons. Firstly, strokes that are closest to the eye are processed first and due to E_{def}, 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.

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 at least one valid feature match 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 a NP-hard combinatorial optimization problem. Branch-and-bound approaches are often used to overcome such computationally expensive exhaustive searches [30], since they are de-

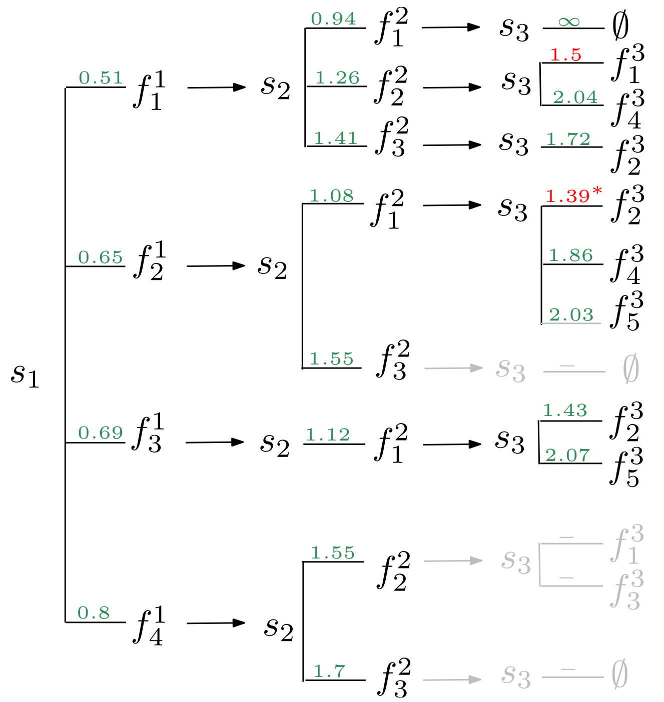


Figure 5: Energy minimization. We use a branch and bound search scheme to find the best stroke-feature matching that minimizes the total cost. Each stroke (in this example, s_1, s_2, s_3) has a priority list of potential candidate features, ordered from the most to the least preferable. Here s_1 has four candidates, s_2 has three and s_4 has five. Note how assigning one feature to a stroke often invalidates some features for subsequent strokes. Moreover, if a stroke no longer has a valid feature it can be assigned to, the corresponding branch has an infinite cost. Once a solution is found, branches that are guaranteed to have a cost higher than the current optimal solution are not explored (indicated in gray). The asterisk (∗) indicates the current best solution.

signed to discard non-optimal solutions early on. Here, we use the branch-and-bound scheme 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 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. Figure 5 illustrates the search for an optimal solution, given a sketch with 3 strokes.

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.

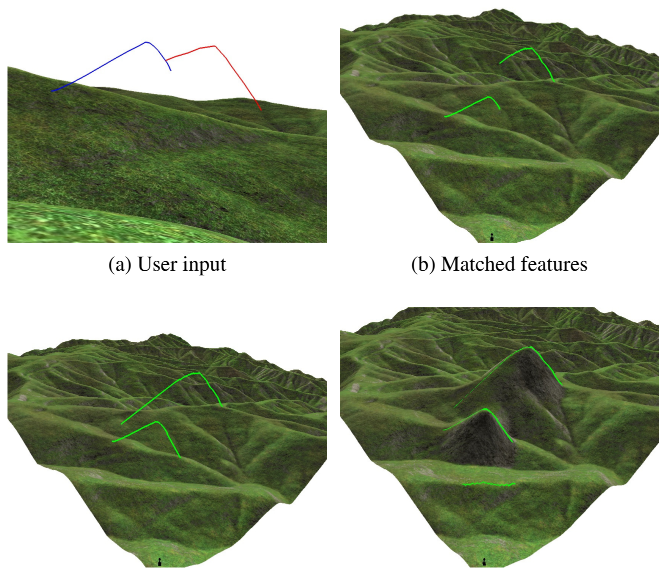
For each point of the stroke $q_s = (x_s, y_s)$, we check if there exists a feature point q_f whose projection on screen $q_p = P(q_f) = (x_p, y_p)$ has the same x-coordinate as q_s, i.e. $x_s = x_p$. If this point exists, we project the stroke point on the world space, using the distance of q_f from 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 extending 3D features assigned to strokes at both endpoints along their tangents, until they reach the surface of the terrain. This is provided as an optional step in the editing process. An example of feature completion is presented in Figure 6. This simple approach only produces realistic terrain silhouettes for strokes with a low-frequency structure. More sophisticated contour completion methods such as the one presented in SmoothSketch [18] could alternatively be used to support elaborate strokes.



(a) User input (b) Matched features

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

Figure 6: 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 in work by Emilien et al. [31], consists in computing the difference of the curve height and the terrain height $\mathcal{H}$, and to diffuse these differences (instead of absolute height values) using a multi-grid Poisson solver similar that used by Hnaidi et al. [14].

More precisely, for each point $p = (x, y, z)$ of the stroke in the world space, we compute $\delta = z - \mathcal{H}(x, y)$, 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 $\mathcal{M}$.

The displacement is finally applied on the terrain height field $\mathcal{H}$, 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, $\mathcal{H}'(x, y) = \mathcal{H}(x, y) + \mathcal{M}(x, y)$, where $\mathcal{H}'$ 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. It therefore follows that 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.

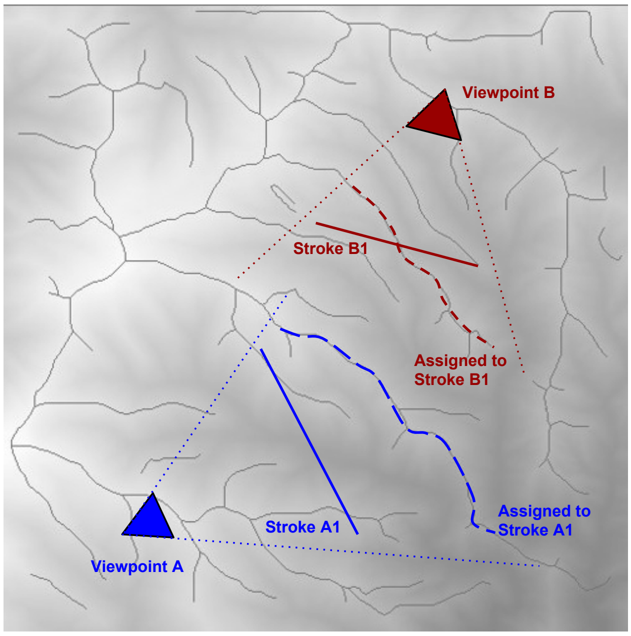
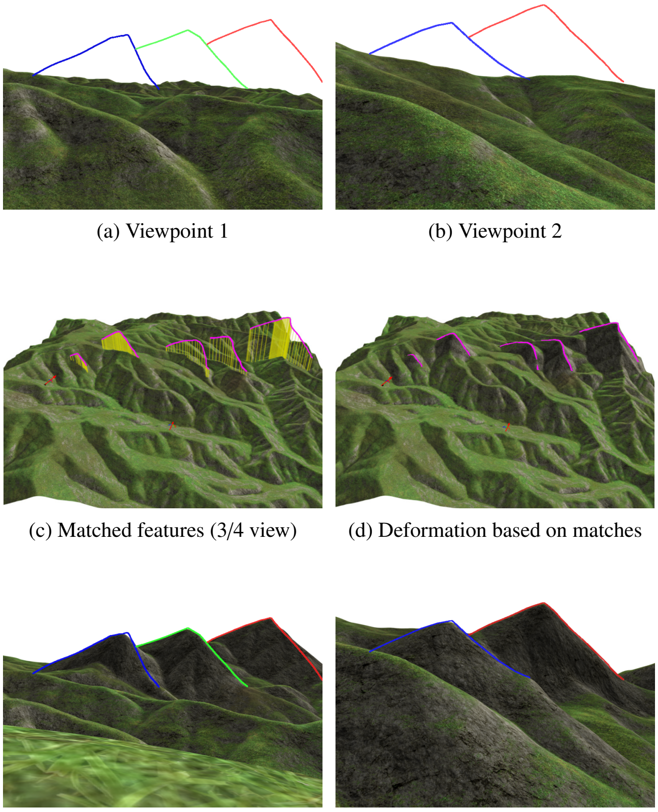


Figure 7: Multi-view from two overlapping viewpoints. Let sketch A consists of stroke A1 and sketch B consists of stroke B1. A and B are intersecting sketches since stroke B1 is visible from A and stroke A1 is visible from B. If the indicated terrain features (shown in dashed lines) are assigned to each stroke and deformed to fit the user-specified heights, then either the silhouette created by B1 will be protruding viewed from A, or the silhouette created by A1 will be protruding viewed from B.This situation can only be avoided if the section of stroke A1 visible from B has the same height values as B1.

7. Handling multi-view sketches

With respect to our earlier work [21], we improve our framework to support multi-view sketches from different viewpoints. We assume that the sketches provided by the artist do not cross each other. Two sketches cross or intersect if parts of both sketches are visible from the two sketching viewpoints. It would be difficult to generate terrain silhouettes that match one sketch and yet, are not detected as protruding from the other sketching viewpoint. Figure 7 shows an example of two intersecting sketches. The problem of having silhouettes generated from one sketch viewed as protruding silhouettes from a different viewpoint cannot usually be solved, unless the intersecting sections have the same height values or the assigned features for



(a) Viewpoint 1 (b) Viewpoint 2

(c) Matched features (3/4 view) (d) Deformation based on matches

(e) Final output (viewpoint 1) (f) Final output (viewpoint 2)

Figure 8: Sketch-based terrain editing from two different viewpoints (shown in (a) and (b)). (c) shows the terrain features assigned to each stroke, based on a modified stroke-feature matching algorithm that handles all sketches at once, while ensuring that curve constraints for different sketches do not occlude each other. The yellow lines indicate the height displacements of assigned terrain features. (d) The height displacements are used as constraints to a terrain deformation. (e,f) shows the deformed terrain from the two viewpoints. (g,h) shows the resulting terrain after lowering protruding silhouettes visible from each viewpoint.

one sketch are so far from the other sketch viewpoint that they are not visible. Thus for overlapping viewpoints that are far enough from each other, terrain features closer to the camera can be assigned such that no conflict occurs. Instead of including this additional complexity to our method, we decided not to support intersecting sketches. We argue that in the case where viewpoints are far from each other, iterative drawing can be used, since the algorithm always try to assign features that are closer to the corresponding sketch viewpoint. Iterative drawing could be used for non-intersectingl multi-view sketches as well, but especially in cases where multiple viewpoints are close to each other, taking into account all the sketches when deciding the assignment of features to strokes is important.The approach discussed here provides a guarantee that each generated silhouette will fit the corresponding user strokes, with no other silhouette protruding when seen from the corresponding viewpoint.

To handle non-intersecting multi-view sketches, we first process each sketch separately by computing its stroke ordering and a list of potential terrain features for each stroke. Note that for each sketch, we generate this list from terrain ridges and silhouettes edges detected from the sketch camera position. Once we have a priority list of candidate features for each stroke in each sketch, we run an energy minimization process that takes into consideration all the sketches at once.

The energy minimization problem (Section 5.2.2) changes as follows: for each input sketch I_t, we want to assign a terrain feature to each of its strokes s_i^t such that the total cost of all the assignments is minimized, with the additional constraint that no terrain feature assigned to a given stroke s_i^t should fall between the camera and an already assigned feature in a different sketch $I_{t'}$. This additional constraint ensures that all assigned terrain features remain visible from their respective sketch viewpoint. To handle all sketches at once in the branch-and-bound algorithm, we first explore solutions for the first drawn sketch and then proceed to the next one. Similarly to the process in Section 5.2.2, the list of candidates for every stroke is updated according to constraints within each sketch and across sketches. This modified branch and bound scheme effectively generates stroke-feature matches for all sketches.

Once we have assigned a terrain feature to each stroke, all the combined matched features are used to deform the terrain (Section 6). To handle residual artifacts from the deformation, we lower protruding silhouettes one sketch at a time, for all sketches. Because the influence of terrain deformation is localized, lowering protruding silhouettes for one sketch have a limited effect on terrain silhouettes for another sketch. Figure 8 shows a terrain editing from two sketches, each drawn from a different camera position and orientation. Note how for both sketches, user strokes correspond to terrain silhouettes, while the whole terrain remains plausible from different viewpoints. This would not have been the case if the two sketches had been processed sequentially, since deformations due to the second sketch would have likely modify silhouettes generated for the first sketch.

8. Results

Validation examples . The examples below and the associated video illustrate the results of our method in a variety of cases. In particular, Figure 9 shows editing of a terrain with a complex sketch containing 4 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 shown in Figure 10. 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 11. Moreover, the method is robust enough to support terrains with few or no features, as shown in the example given in Figure 12. Indeed if the terrain contains no features, we compute a 3D embedding of stroke closest to the camera by projecting the stroke on the drawing plane determined by the camera direction and a 3D point where the stroke touches the terrain. The rest of the user strokes can then be placed in 3D with respect to the embedding of the first stroke, using the same technique we apply to strokes with no matching features in Section 5.2.2.

Fig.	Features	Matching	Deformation	Silhouettes
1	0.14	0.24	0.09	4.9
2	0.14	1.5	0.11	2.6
9	0.15	0.21	0.10	2.1
10	0.12	0.13	0.10	9.4
11	0.12	0.04	0.09	3.4

Table 1: Computation times (in seconds) for examples illustrated in this paper. We show computation times of the following steps: feature extraction, stroke-feature matching, terrain deformation, lowering protuding silhouettes.

Our complex sketch-based editing framework can be implemented at interactive rates, as illustrated in the attached video, which makes it an attractive alternative to other terrain generation/editing techniques discussed in Section 2.

Performance. The terrain editing system is implemented in C++, and the computations are measured on an Intel® Xeon® E5-1650 CPU, running at 3.20 GHz with 16 GB of memory. We present the computation times of resuts illustrated in this paper in the Table 1. The feature extraction and terrain deformation computation times only depend on the terrain resolution, which is 512×512 in the examples. Feature matching performance depends on the number of strokes and the number of extracted features. In our examples, the average number of extracted features was around 1000 and mostly consisted of short terrain silhouette features. The most expensive algorithm is the lowering protruding silhouettes, due to the expensive sihouette detection. Our naive implementation of silhouette detection could be optimised to significantly impact the overall performance of our algorithm. The stroke ordering algorithm has a negligible computation time. The average manual editing time was less than a minute.

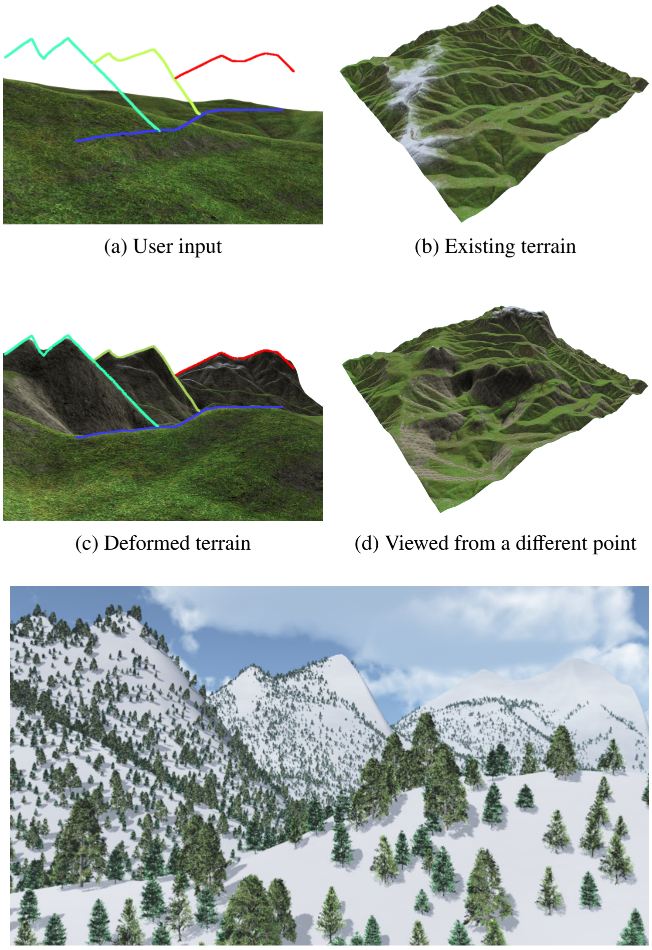
Comparing feature-based constraints against planar curve constraints. Typical sketch-based terrain deformation techniques [13, 27, 15] use planar curve constraints computed from user strokes. Such planar curves can be obtained by computing the drawing plane from the user sketch and projecting strokes on this plane to obtained their 3D position in world coordinates. The normal to the drawing plane is the camera view direction and one point on this plane is obtained by computing the world coordinates of a stroke point touching the terrain. We argue that using such planar curve constraints for terrain editing produce inferior results, compared to the use of feature-based constraints. To illustrate this, we compared the two different deformation schemes, our method and the standard method, on three different input. Each input consists of a real landscape and a one-stroke sketch drawn from a first person perspective view (see Figure 13). Our method uses the matched terrain features obtained from Section 5 as deformation constraints. The standard method simply uses curve constraints obtained by projecting user strokes on the drawing plane. Figure 13 shows the 3D constraints used in the terrain deformation and the final terrain produced by each method. Note that the final terrain is generated by first deforming the input terrain with feature-based constraints or planar curve constraints, and then lowering protruding silhouettes. In the case of planar curve constraints, this last step generates non planar silhouettes, which is already an improvement since the main pitfall of the standard method is that it produces unrealistic mountains with planar silhouettes. Even after this improvement, note how landscapes produced by the standard method have more prominent silhouettes in front of the user-specified silhouettes and thus may not reflect the user intent. This happens when a planar curve constraint is behind a terrain feature and thus the deformation raises the terrain feature making it a prominent silhouette. In contrast our proposed method is feature-aware and by generating deformation constraints based on terrain features, reduces the risk of prominent silhouettes appearing in front of user-specified silhouettes. In addition, the silhouettes we generate are non-planar, since they are matched with the depth of the associated terrain features (Figure 13(h, i)). This makes the resulting terrains look much more natural when seen from above.

User tests. We performed an informal user test on our single viewpoint system with two experienced computer artists. The system was briefly introduced to the users, who had no prior knowledge of it. They were asked to draw sketches to deform existing terrains. Both of them reported that our system was very easy to learn and use, and were able to quickly create new sceneries. Their feedback indicated that the approach is original, and seems a promising way to create a scene that matches their artistic intend. These first users also asked for the ability to move within the scene and edit the terrain from multiple viewpoints. This led to the work described in Section 7. Lastly, the users emphasised the importance of the realistic 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. It is often not clear during the drawing stage which terrain feature will be assigned to a stroke. The artist may draw a stroke with the intention of turning a large-scale feature into a terrain silhouette, but the algorithm chooses to deform a different terrain feature instead. To address this, we could also improve our matching method using extra error functions, that take into account the placement of user strokes relative to the projection of terrain features on the drawing plane.

The editing framework is also limited in the type of strokes drawned and the type of terrain. For instance landscapes with high frequency details and a complex style such as the Grand Canyon are particularly difficult to edit since depending on the nature of the strokes and which features are assigned to strokes, the deformed region can differ significantly from the other. In general, elaborate strokes that are unlikely to be terrain silhouette, except from a specific viewpoint, often cause several iterations of terrain deformation in the neighbourhood of the assigned features, that either do not suceed in removing all protruding silhouettes or look unrealistic when viewed from a different viewpoint.

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 [32] 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.



(a) User input (b) Existing terrain

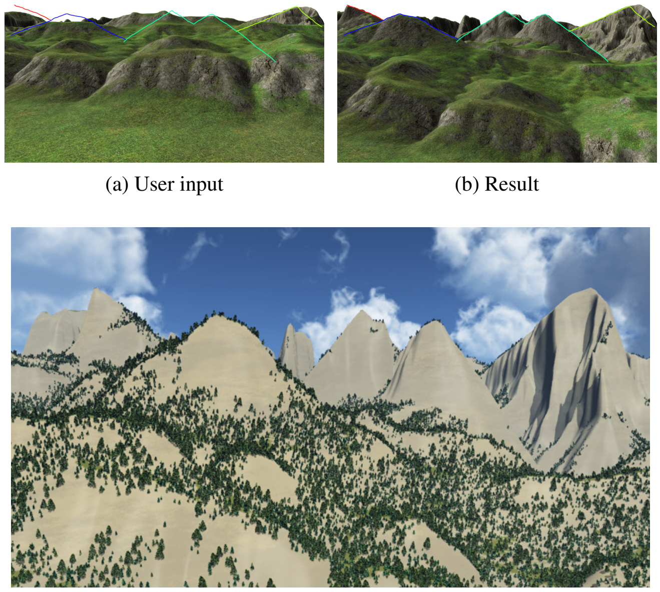
(c) Deformed terrain (d) Viewed from a different point

(e)

Figure 9: Terrain editing from a complex user sketch.

9. Conclusion

We presented a sketch-based modelling method enabling the deformation of a terrain from a single viewpoint, and then extended it to handle multiple viewpoints simultaneously. 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.



(a) User input (b) Result

(c)

Figure 10: Editing a complex rocky mountain from a complex sketch.

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] Fournier A, Fussell DS, Carpenter LC. Computer rendering of stochastic models. Communications ACM 1982;25(6):371–84.

[2] Miller GSP. The definition and rendering of terrain maps. SIGGRAPH Comput Graph 1986;20(4):39–48.

[3] Lewis JP. Generalized stochastic subdivision. ACM Trans Graph 1987;6(3):167–90.

[4] Musgrave FK, Kolb CE, Mace RS. The synthesis and rendering of eroded fractal terrains. In: Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH '89; New York, NY, USA: ACM; 1989, p. 41–50.

[5] Roudier P, Peroche B, Perrin M. Landscapes synthesis achieved through erosion and deposition process simulation. Computer Graphics Forum 1993;12(3):375–83.

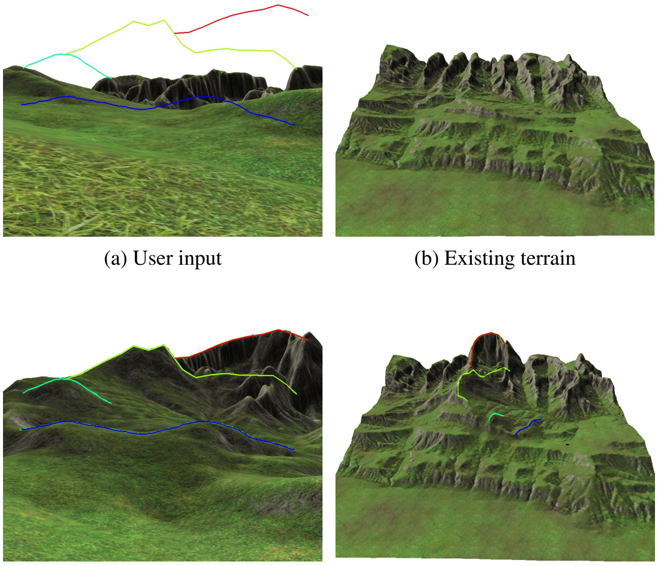
[6] Chiba N, Muraoka K, Fujita K. An erosion model based on velocity fields for the visual simulation of mountain scenery. Journal of Visualization and Computer Animation 1998;9(4):185–94.

[7] Nagashima K. Computer generation of eroded valley and mountain terrains. Visual Computer 1997;13(9-10):456–64.

[8] Neidhold B, Wacker M, Deussen O. Interactive physically based fluid and erosion simulation. Natural Phenomena 2005;:25–32.

[9] Kristof P, Benes B, Krivanek J, Stava O. Hydraulic erosion using smoothed particle hydrodynamics. Computer Graphics Forum 2009;28(2):219–28.

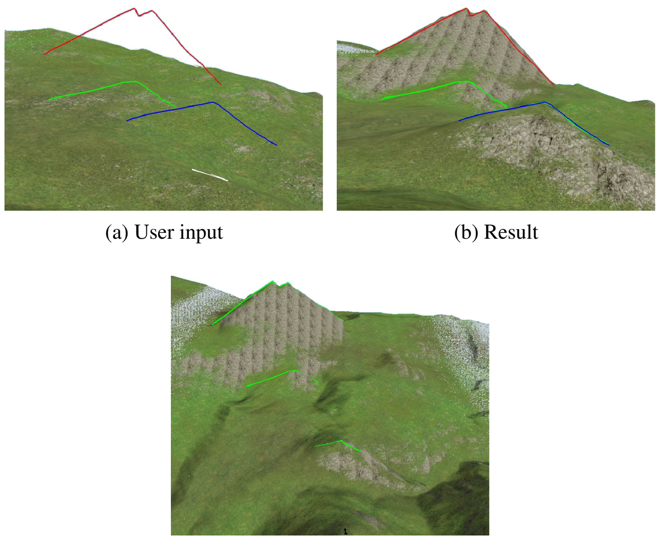
[10] Cohen JM, Hughes JF, Zeleznik RC. Harold: A world made of drawings. In: Proceedings of the 1st International Symposium on Non-photorealistic Animation and Rendering. NPAR '00; New York, NY, USA: ACM; 2000, p. 83–90.



(a) User input (b) Existing terrain

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

Figure 11: Terrain editing produces non-planar silhouettes in the output, from 2D planar strokes.



(a) User input (b) Result

(c) View from another point

Figure 12: 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.

[11] Watanabe N, Igarashi T. A sketching interface for terrain modeling. In: ACM SIGGRAPH 2004 Posters. SIGGRAPH '04; New York, NY, USA: ACM; 2004, p. 73–.

[12] Zhou H, Sun J, Turk Gbb, Rehg Jbb. Terrain synthesis from digital elevation models. IEEE Transactions on Visualization and Computer Graphics 2007;13(4):834–48.

[13] Gain J, Marais P, Straer W. Terrain sketching. In: Proceedings of I3D 2009: The 2009 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games. 2009, p. 31–8.

[14] Hnaidi H, Guerin E, Akkouche S, Peytavie A, Galin E. Feature based terrain generation using diffusion equation. Computer Graphics Forum 2010;29(7):2179–86.

[15] Tasse F, Gain J, Marais P. Enhanced texture-based terrain synthesis on graphics hardware. Computer Graphics Forum 2012;31(6):1959–72.

[16] Genevaux JD, Galin E, Guerin E, Peytavie A, Benes B. Terrain generation using procedural models based on hydrology. ACM Transactions on Graphics 2013;32(4).

[17] Dos Passos V, Igarashi T. 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. 2013, p. 61–8.

[18] Karpenko O, Hughes J. Smoothsketch: 3d free-form shapes from complex sketches. In: ACM SIGGRAPH 2006 Papers, SIGGRAPH '06. 2006, p. 589–98.

[19] Singh K, Fiume E. Wires: A geometric deformation technique. In: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH '98; New York, NY, USA: ACM; 1998, p. 405–14.

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

[21] Tasse FP, Emilien A, Cani MP, Hahmann S, Bernhardt A. First person sketch-based terrain editing. In: Proceedings of the 2014 Graphics Interface Conference. GI '14; Canadian Information Processing Society; 2014, p. 217–24.

[22] Natali M, Lidal EM, Viola I, Patel D. Modeling terrains and subsurface geology. In: Proceedings of EuroGraphics 2013 State of the Art Reports (STARs). Eurographics; Eurographics 2013 - State of the Art Reports; 2013, p. 155–73.

[23] Mandelbrot BB. The fractal geometry of nature. New York: W. H. Freeman; 1983.

[24] Ebert DS, Musgrave FK, Peachey D, Perlin K, Worley S. Texturing and Modeling: A Procedural Approach. 3rd ed.; San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 2002.

[25] Kelley AD, Malin MC, Nielson GM. Terrain simulation using a model of stream erosion. Computer Graphics (ACM) 1988;22(4):263–8.

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

[27] Bernhardt A, Maximo A, Velho L, Hnaidi H, Cani MP (IMPA, Rio de Janeiro, Brésil). Real-time Terrain Modeling using CPU-GPU Coupled Computation. In: XXIV SIBGRAPI. Maceio, Brazil; 2011,.

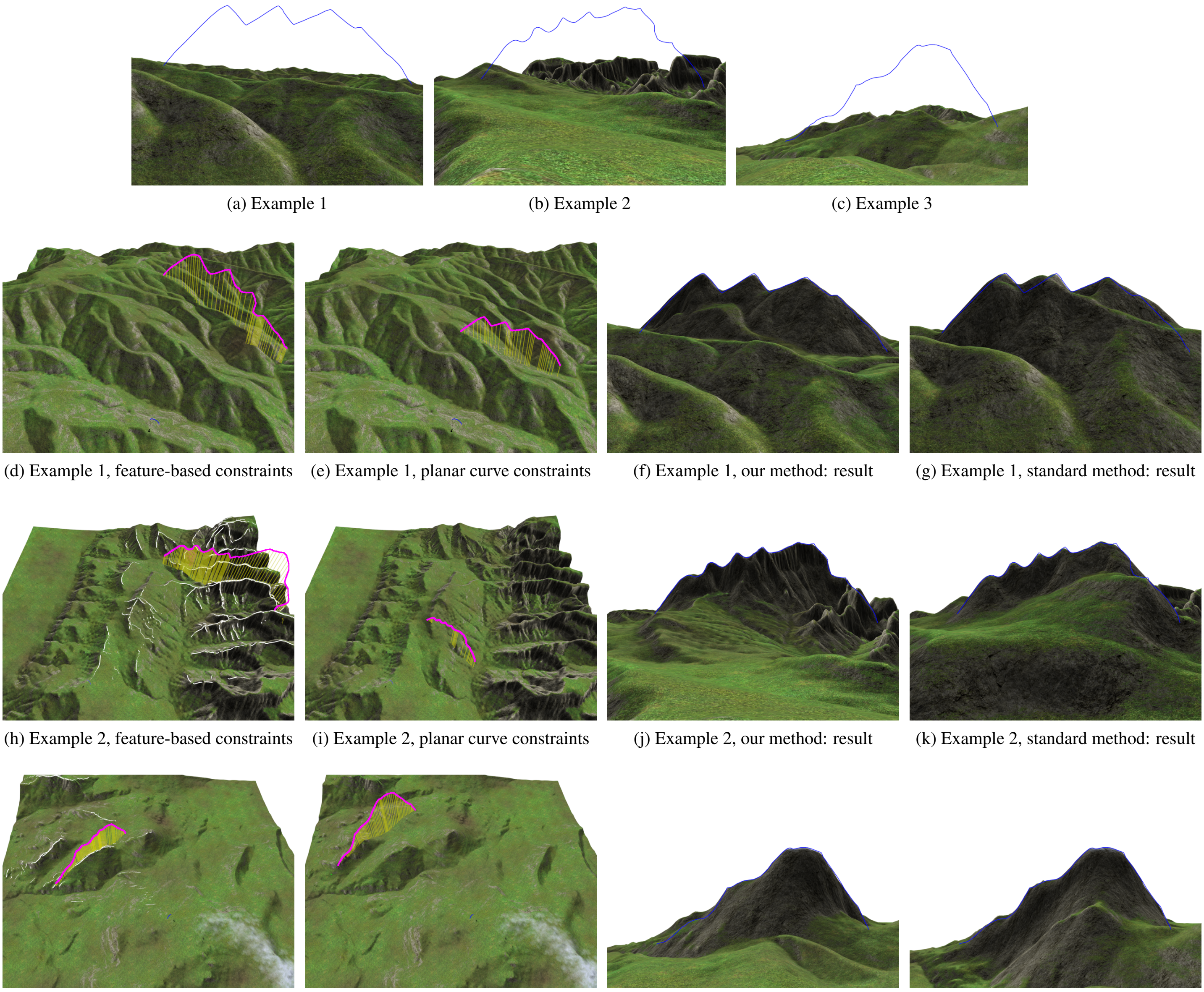
[28] Chang YC, Sinha G. A visual basic program for ridge axis picking on dem data using the profile-recognition and polygon-breaking algorithm. Computers and Geosciences 2007;33(2):229–37.

[29] Bangay S, de Bruyn D, Glass K. 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; New York, NY, USA: ACM; 2010, p. 73–82.

[30] Clausen J. Branch and bound algorithms-principles and examples. Parallel Computing in Optimization 1997;:239–67.

[31] Emilien A, Poulin P, Cani MP, Vimont U. Interactive procedural modeling of coherent waterfall scenes. Computer Graphics Forum 2014;To appear.

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



(a) Example 1 (b) Example 2 (c) Example 3

(d) Example 1, feature-based constraints (e) Example 1, planar curve constraints (f) Example 1, our method: result (g) Example 1, standard method: result

(h) Example 2, feature-based constraints (i) Example 2, planar curve constraints (j) Example 2, our method: result (k) Example 2, standard method: result

(l) Example 3, feature-based constraints (m) Example 3, planar curve constraints (n) Example 3, our method: result (o) Example 3, standard method: result

Figure 13: Comparing terrain deformation with feature-based constraints (our method) against editing from planar curve constraints (standard method). The final output produced by our deformation scheme has less prominent terrain silhouettes appearing between the camera position and the user-specified silhouettes, and thus is closer to the user intent.