DOI: 10.1111/j.1467-8659.2012.03076.x

COMPUTER GRAPHICS *forum*
Volume 00 (2012), number 0 pp. 1–14

Enhanced Texture-Based Terrain Synthesis on Graphics Hardware

F. P. Tasse, J. Gain and P. Marais

Department of Computer Science, University of Cape Town, South Africa
{ftasse, jgain, patrick}@cs.uct.ac.za

Abstract
Curvilinear features extracted from a 2D user-sketched feature map have been used successfully to constraint a patch-based texture synthesis of real landscapes. This map-based user interface does not give fine control over the height profile of the generated terrain. We propose a new texture-based terrain synthesis framework controllable by a terrain sketching interface. We enhance the realism of the generated landscapes by using a novel patch merging method that reduces boundary artefacts caused by overlapping terrain patches. A more constrained synthesis process is used to produce landscapes that better match user requirements. The high computational cost of texture synthesis is reduced with a parallel implementation on graphics hardware. Our GPU-accelerated solution provides a significant speedup depending on the size of the example terrain. We show experimentally that our framework is more successful in generating realistic landscapes than current example-based terrain synthesis methods. We conclude that texture-based terrain synthesis combined with sketching provides an excellent solution to the user control and realism challenges of virtual landscape generation.

Keywords: terrain generation, patch-based texture synthesis, terrain sketching, seam removal, patch merging, GPU

ACM CCS: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Colour, shading, shadowing and texture

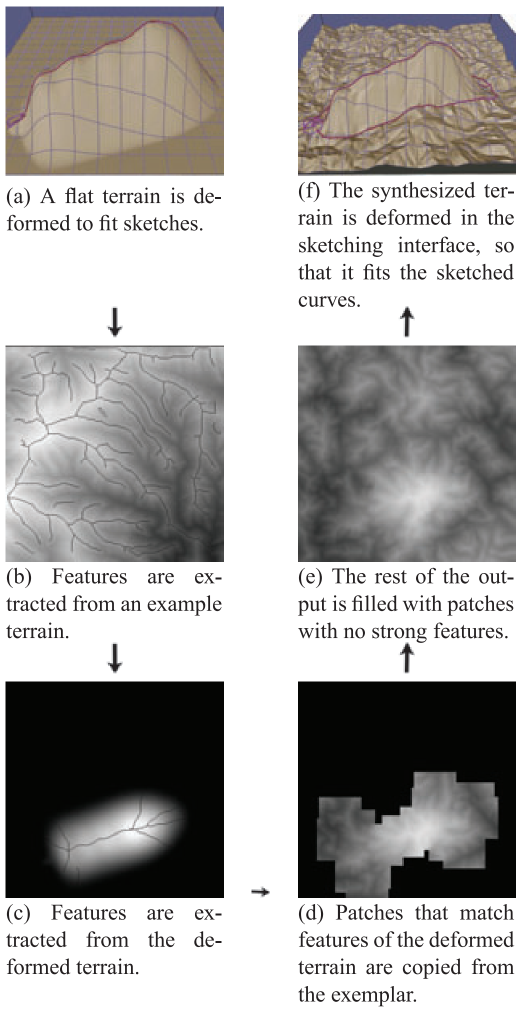
1. Introduction

Landscapes form an integral part of many virtual environments. Virtual terrains are used in video games, films, advertisements and many other software systems involving outdoor environments such as flight simulators. Believable terrains play an important role in ensuring immersion in a virtual environment. The U.S. Geological Survey provides freely downloadable models of real landscapes that can be added to 3D environments if needed [US11]. The usage of real terrains has the advantage of realism with little effort, but limits the range of terrains that can be modelled. Thus, artists often create the terrain manually using 3D modelling tools or paint a 2D image such that luminosity values represent height values. As the need for larger, more believable terrains increase, manually modelling becomes more complex and tedious.

Procedural terrain generation concerns the algorithmic generation of landscapes, as an alternative to manual creation. Fractal-based methods for terrain generation randomly displace the height values of a flat terrain [Man82, Lew87, GMS09]. However, these methods cannot simulate the erosion effects present in real landscapes, such as drainage patterns. To achieve these effects, landscapes need to undergo physically based erosion simulations [MKM89, NWD05, PGGM09]. Such simulations have a high computational cost and require that the user has a working knowledge of various erosion models. A new trend in procedural terrain generation is based on texture synthesis methods able to create landscapes from information extracted from an example terrain [BSS06, Dac06, ZSTR07]. Texturesynthesis generates a texture that can be regarded as having undergone the same stochastic process as an example texture. Terrain is commonly represented by a greyscale image, referred to as a heightmap, with elevation values stored as pixel values. This image 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. Heightmaps have large-scale curvilinear features such as ridges and valleys that are not stochastic, and thus, cannot easily be reproduced by current texture synthesis methods. Zhou *et al.* propose a texture-based terrain synthesis of an example heightmap, constrained by curvilinear features [ZSTR07]. Given a user sketch map and a real landscape, their algorithm creates a new terrain by copying square blocks of pixels (*patches*) from the real terrain such that curvilinear features specified in a user-sketched map appear in the output. A realistic terrain is produced that has the same small-scale characteristics as the real terrain. However, the Poisson seam removal, a patch merging technique used to remove boundary artefacts created by overlapped patches, creates artefacts clearly noticeable under 3D lighting. Furthermore, users can only specify the general location of valleys and ridges. The 2D sketch map does not allow intuitive control over the height profile of these features.

Our proposed system, as illustrated in Figure 1, builds on this technique. For usability, we use a terrain sketching interface [GMS09] that enables users to fully control the generation of new terrain by deforming a default terrain such that it fits $2\frac{1}{2}D$ drawn silhouette and boundary curves. The sketching system applies a multi-resolution surface deformation and wavelet noise to deform terrains and add detail to the surface. Despite the intuitive control users have over the generation, the terrains produced by this interface share the poor realism of fractal-based terrains. *We propose combining the sketching interface and patch-based terrain generation in a hybrid system that gives wider control to the user and produces realistic terrains.*

Patch-based synthesis has a high computational cost as it consists of several iterations, where each iteration compares patches from the example terrain to select the best patch to place in the output. As the size of the example increases, so does the computation time. Most modern computers now have programmable GPUs that can be used to accelerate parallelizable algorithms. Unfortunately, texture synthesis problems are generally order-dependent and thus do not map well to graphics hardware. However, each iteration of a patch-based synthesis performs a patch selection that involves computing distances between one patch and several candidate patches independently. This process is highly parallelizable and we thus accelerate the terrain generation by implementing this parallel work on graphics hardware.


(a) A flat terrain is deformed to fit sketches.

(b) Features are extracted from an example terrain.

(c) Features are extracted from the deformed terrain.

(d) Patches that match features of the deformed terrain are copied from the exemplar.

(e) The rest of the output is filled with patches with no strong features.

(f) The synthesized terrain is deformed in the sketching interface, so that it fits the sketched curves.

Figure 1: *Patch-based synthesis framework.*

This paper presents the following contributions:

1. Improved patch-based terrain synthesis that enhances the matching of user constraints and landscape realism.
2. Novel patch merging technique, more suitable to 3D structures, that removes boundary seams caused by the overlapping multiple patches.
3. GPU acceleration of patch-based landscape generation on graphics hardware.
4. User study that investigates terrain realism and success of patch merging.

2. Related Work

2.1. Texture synthesis

Two common schemes have emerged for synthesizing textures from an example texture or *exemplar*: pixel-based and patch-based approaches. Pixel-based methods generate a new texture pixel-by-pixel from an exemplar, with each pixel determined by a search of neighbourhoods in the exemplar. To optimize neighbourhood searches, schemes such as tree vector quantization [WL00], kd-trees [ZG02] or approximate nearest neighbour searching [LLX*01] are often used to approximate the best neighbourhood.

Patch-based schemes create a new texture by copying patches from the exemplar, constrained by overlap. These methods are more efficient and produce better results than pixel-based algorithms. However, patch overlaps create boundary seams. Several patch merging techniques exist to address this issue including minimum error cuts in Image Quilting [EF01], optimal cuts along the overlap region with Graphcut [KSE*03], alpha-blending [LLX*01], Poisson image editing [PGB03] or a combination of these [LZPW04, ZSTR07]. Most of these techniques are designed for 2D images, and not for terrains that will be rendered in a 3D environment. Thus, even patch merging methods that are successful for 2D images [ZSTR07] introduce discontinuities in the gradient that are visible in 3D terrain navigation.

2.2. Texture-based terrain synthesis

Achieving realism and easy user control is extremely difficult for fractal-based and physically based terrain generation. An alternative to these techniques is texture-based methods that build on example-based texture synthesis techniques. Using a real heightfield as the exemplar significantly improves the realism of the resulting terrain. Dachsbacher [Dac06] applies pixel-based texture synthesis by non-parametric sampling [EL99] to terrain generation. Their technique suffers from terrain artefacts visible during rendering and the high computational cost of the texture synthesis method. Brosz *et al.* [BSS06] present a terrain synthesis that extracts small-scale characteristics from a target terrain that are then applied to a base terrain using Image Quilting [EF01]. This technique increases the level of detail of a base terrain but cannot create a new terrain from scratch. Zhou *et al.* [ZSTR07] propose a feature-guided patch-based synthesis from real terrains that produce realistic results. However, patch merging introduces artefacts and users cannot intuitively specify height values for desired features. The framework proposed in this paper addresses these issues.

3. Existing Texture-Based Terrain Synthesis

We briefly describe Zhou *et al.* [ZSTR07]'s patch-based algorithm upon which our work is based. The algorithm uses patches from a real heightmap to form a new terrain that matches the features specified in a 2D user-sketched feature map. This approach consists of three key steps.

First, ridges and valleys are extracted from the example terrain and feature map using the Profile Recognition and Polygon Breaking Algorithm (PPA) [CSH98]. To extract ridges, the PPA algorithm marks each terrain point that is likely to be on a ridge line, based on the point height profile. Adjacent candidate points are connected by segments, forming a cyclic graph. 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 extracted feature tree is a graph where nodes are end points or branch points connected by curvilinear path features. Valley extraction uses a similar approach.

Next, curvilinear features extracted from the real landscape and sketch map are used to constrain which patches are selected from the exemplar and their position on the output terrain. The order of patch placement is specified by a breadth-first traversal of the sketch map feature tree. For a target patch centred at a feature point of the sketch map, a best matching patch is selected from the exemplar, based on the feature branch degree, the amount of deformation needed to fit the target and how well the patch matches already synthesized regions.

Finally, empty areas in the output are filled with patches with no strong features, from the already-filled areas. Here, patch selection is only determined by how well the patch matches non-empty areas.

During patch placement, the graphcut algorithm finds the optimal cut that minimizes the seam severity in the overlap area. Visible optimal seams are then removed with a height re-adjustment performed by the Poisson seam removal algorithm. Gradient values in the overlap are computed and gradient values along the optimal seam are set artificially to zero. A new set of elevation values that fit the modified gradient field is obtained by solving a Poisson equation. Unfortunately, the discontinuities in the gradient field are often visible during 3D terrain rendering.

Zhou *et al.* [ZSTR07] produce realistic terrains with this patch-based texture synthesis technique. We improve on their algorithm by using a more intuitive user control that allows low-level manipulation, applying a more efficient way of breaking polygons during feature extraction, adding more criteria for better patch matching and using a patch merging approach more suitable for terrains.

4. Enhanced Patch-Based Terrain Synthesis

We use a terrain sketching interface [GMS09] as the base for a new hybrid scheme that combines terrain sketching with patch-based terrain synthesis. The sketching interface deforms a given terrain to fit $2\frac{1}{2}D$ silhouette and boundary-sketched curves using multi-resolution surface deformation and wavelet noise extracted from the silhouette curve. Integrating this interface into our landscape generation framework offers an interactive environment that allows users to intuitively control the output through constraint curves, 3D

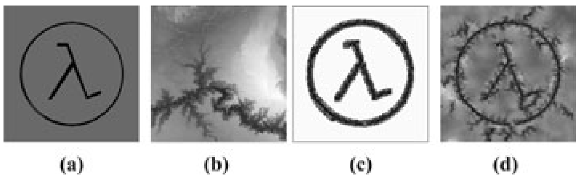


Figure 2: *Patch-based texture synthesis. (a) and (b) Valley lines are extracted from the target and the exemplar. (c) Patches are selected from the exemplar and placed in the output. (d) The rest of the output is completed. (The white segments represent valley lines).*

multi-resolution rendering and editing, while producing realistic terrains.

An overview of our hybrid scheme is illustrated in Figure 1. The user draws silhouette and boundary curves on a flat terrain in the terrain sketching interface, which deforms the flat terrain to fit the constraint curves. An example heightmap, or exemplar, and the deformed terrain are then passed into a patch-based texture synthesis that will generate a new terrain that has the large-scale features of the deformed terrain and yet exhibits the characteristics of the example heightfield. The new heightfield replaces the deformed terrain in the sketching interface. This new terrain no longer perfectly fits the sketched curves, and so it is deformed in a final phase to match height constraints. The user can edit the synthesized terrain by drawing another set of curves and the whole process restarts. The interface, combined with the texture-based synthesis, provides the capability to navigate, examine and edit the terrain as often as necessary.

Texture-based terrain synthesis consists of feature extraction, patch matching and merging, as shown in Figures 1 and 2 . The rest of this section covers these steps in more detail.

4.1. Feature extraction

Polygon breaking is a step of the PPA feature extraction algorithm [CSH98] that consists of repeatedly deleting segments in a cyclic graph until all cycles are removed. This step has a high computational cost, and thus, is not adequate for an interactive system. Furthermore, the performance drastically decreases as the size and the feature complexity of the terrain increases. Bangay *et al.* [BdBG10] propose a feature extraction scheme that connects all the terrain points into a graph using a height-based or curvature-based weighting and computes the *minimum spanning tree* (MST) of that graph. Results show that while the original PPA's cost is polynomial, this new version runs in quasi-linear time with respect to the number of edges in the graph. Because we are mainly concerned with the performance and the extraction of large-scale terrain features, we simply connect candidate terrain points as in the original PPA algorithm and replace the polygon breaking with a *minimum spanning forest* algorithm.

4.2. Patch matching

Once features have been extracted, patch matching selects square patches from the exemplar T_{exm} that match the user constraints. After experimenting with several patch sizes, we set the default patch size to 80×80 but the user may modify this value at runtime according to the desired level of detail and the exemplar resolution. We increase the flexibility and variability of the output by including mirrored patches about both x and y axes and rotated patches in increments of $\frac{\pi}{4}$. Patch matching is executed in two steps: searching for patches to place in the output T_{out} that match the feature tree Γ_{tar} extracted from the target T_{tar} and filling the remaining empty areas in the output T_{out} with patches that contain a minimal amount of detail.

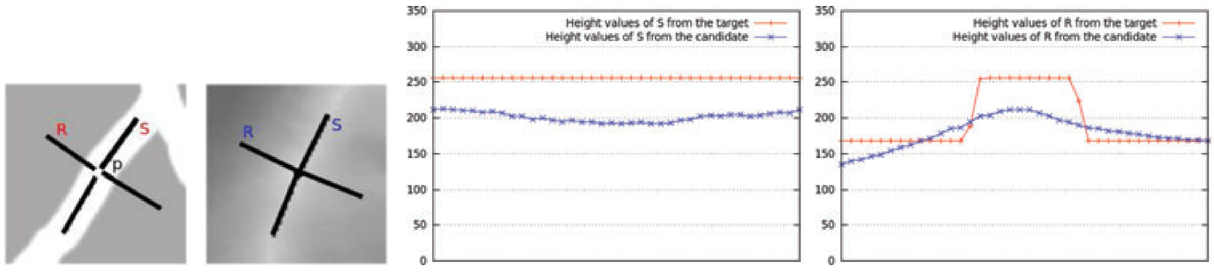
4.2.1. *Feature patch matching*

Feature patch matching selects patches from the exemplar that match features extracted from the target terrain. For each node or point in the tree, a set of control points $\{p_i\}$ is determined, which consist of the point itself and the intersections of its branches with a circle inscribed in a patch situated at the point. A breadth-first traversal of the feature tree Γ_{tar} is used to guide the order of patch placement.

Patch selection proceeds as follows: for each feature node $p \in \Gamma_{tar}$ with a patch Ψ^p_{tar} centred at p, its control points $\{p_i\}$ are computed and a cost is assigned to each patch from the exemplar centred at a feature node in Γ_{exm}. Candidates with a different branch degree than p are given a very high cost added to the overlap area cost c_o described below, so that they are unlikely to be selected. If q is a feature in Γ_{exm} with the same branch degree as p and Ψ^q_{exm} is the patch centred at q, then the cost of selecting q is a combination of the following criteria:

Feature dissimilarity c_f

This cost function determines feature similarity between Ψ^p_{tar} and Ψ^q_{exm} by comparing the height profile of p in Ψ^p_{tar} with its height profile in Ψ^q_{exm} using the normalized sum of squared differences (SSDs). The height profile of p consists of height values along the outgoing segments of p. Figure 3 shows a path feature p with two outgoing segments denoted by S. A graph illustrates the difference in height values along S. If p is a path feature, with two outgoing branches, height values along the segment orthogonal to the path at p are also compared and their normalized SSD is added to the feature dissimilarity cost. This differs from the feature dissimilarity in [ZSTR07] which only compares height profiles perpendicular to a path. Our modified dissimilarity cost improves



(a) Target Ψ_{tar}^{p} **(b)** Candidate Ψ_{exm}^{q} **(c)** **(d)**

Figure 3: *Feature dissimilarity between the target patch Ψ_{tar}^{p} and the candidate patch Ψ_{exm}^{q}.*

matching by penalizing candidates whose height values do not match the feature elevation values specified by the user.

Angle differences c_a

The angles of the outgoing paths of q and p are compared using the normalized SSD. Angle differences indicate how similar the structure of p and that of the candidate q are.

Noise variance c_n

We improve on previous texture-based landscape generation by considering the difference in noise variances. The Gaussian noise variances of the candidate Ψ_{exm}^{q} and Ψ_{tar}^{p} are computed at multiple levels of resolution and their normalized SSD is added to the candidate cost. A lower noise variance difference increases the chances that a patch with similar bumpiness at different resolutions will be selected.

Overlap area c_o

Placing Ψ_{exm} in the output T_{out} may overlap with previously placed patches. The difference in height values in the overlapping region should be minimized to reduce the severity of the seam created by the overlap. This difference is easily computed with the normalized SSD of already synthesized pixels in Ψ_{out}^{p} and their corresponding values in Ψ_{exm}^{q}.

The total cost c of selecting a specific candidate (q, Ψ_{exm}^{q}) as a match for the target (p, Ψ_{tar}^{p}) is computed by combining the above matching costs:

$$c(p,q) = \alpha_f c_f + \alpha_a c_a + \alpha_n c_n + \alpha_o c_o,$$

where α values normalize a particular criterion and determine its influence. For our test cases, the following values were used: $\alpha_f = 5$, $\alpha_a = 2$, $\alpha_n = 0.001$ and $\alpha_o = 1$. The list of candidates is sorted in increasing order according to their cost c. If there are no candidates with the same branch degree as p, c_o is the only criterion used for sorting. A set of k candidates with the lowest costs is selected and from that smaller set, the candidate with the lowest graphcut cost is chosen as the best match for (p, Ψ_{tar}^{p}). The default number of selected patches is set to $k = 5$. Once the best match is found, it is placed in the output at position p. Figure 2(c) shows a result of feature patch matching.

4.2.2. *Non-feature patch matching*

Some areas of the target terrain may not have any features, and hence the corresponding regions in the output are empty after feature patch matching. These empty areas are completed by non-feature patch matching. The unknown region of the output terrain is filled by iteratively copying patches with a minimal amount of detail from the exemplar T_{exm}, which match the already synthesized pixels. Criminisi *et al.* [CPT04] show that in exemplar-based filling, the quality of the output is highly affected by the order of the filling process and propose a filling algorithm that prioritizes patches along structures. We use a similar filling order during the non-feature patch matching to ensure that terrain features are preserved and correctly propagated.

The criteria used to find the best match differ from those used in feature patch matching. The cost associated with choosing a candidate patch Ψ_{exm} is the combination of only two of the criteria used in feature patch matching: the *noise variance difference* c_n and the *normalized SSDs on the overlap area* c_o. This differs from Zhou *et al.* [ZSTR07]'s algorithm that performs patch selection based on the overlap area cost. The noise variance difference encourages the selection of patches with minimal difference in bumpiness and c_o ensures that Ψ_{exm} matches the previously synthesized pixels in the patch Ψ_{out}^{p} centred at p. The total cost c for selecting Ψ_{exm} is computed from c_n and c_o as follows:

$$c(\Psi_{out}^{p}, \Psi_{exm}) = \beta_n c_n + \beta_o c_o,$$

with $\beta_n = 0.0001$ and $\beta_o = 10$.

The candidates are ranked in increasing order according to their cost, and the first k candidates are selected. Then, from the k candidates, the candidate with the lowest graphcut cost is selected as the best match and placed at position p. Figure 2(d) shows a final output terrain after non-feature patch matching is performed.

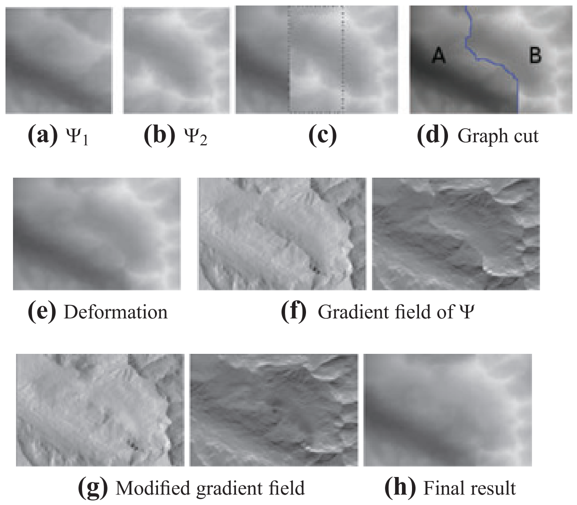


Figure 4: *Graphcut algorithm steps. (c) Ψ_1 and Ψ_2 overlap over a region ov (enclosed by the dotted lines). (d) Ψ_1 and Ψ_2 are merged along the optimal seam. (e) B is deformed to fit A along the seam. (f) The horizontal and vertical components of the gradient field after the graphcut. (g) Each component of the gradient field is deformed using Shepard Interpolation. Discontinuities at the seam are removed. (h) A Poisson solver is used to find a set of height values that fits the modified gradient field.*

4.3. Patch merging

Patch merging removes seams created by overlapping patches. Let Ψ_1 be an old patch and Ψ_2 a new patch that overlaps with Ψ_1 over a region ov. We merge Ψ_2 and Ψ_1 into a patch Ψ such that the seam across their overlap ov is invisible. This is achieved by a novel combination of three different techniques illustrated in Figure 4: Graphcut [KSE*03], Shepard Interpolation [She68] and a Poisson image editing [PGB03].

4.3.1. *Graphcut*

Our patch merging algorithm starts by performing a graphcut, which determines the optimal seam between Ψ_1 and Ψ_2. Graphcut effectively joins the two patches along an optimal seam that determines which pixels come from the old patch (sink A) and which pixels come from the new patch (source B) [KSE*03].

4.3.2. *Shepard interpolation*

The optimal seam may still be visible and we propose to remove it by warping the source B to match the sink A along the cut. Let x_i, $i = 0, 1, \ldots, N$, be the set of pixels along the seam. B', the deformation of B, is constrained by $B^{'}(x_i) = A(x_i),\ i = 0, 1, \ldots, N$. This constraint removes the seam by ensuring that B and A have the same values along the cut. We draw upon the deformation technique based on point features proposed by Milliron *et al.* [MJBF02] to compute $B^{'}$. Let $x \in B$, then the height value at x is displaced by an amount $\triangle(x)$ computed by summing displacements $A(x_i) - B(x_i)$, scaled by distance-based normalized we ights $\hat{w}_i(x)$. In other words, the deformation of B at a point x is

$$B'(x) = B(x) + \triangle(x),$$

$$\triangle(x) = \sum_{i=0}^{N} \hat{w}_i(x)(A(x_i) - B(x_i)),\ \ \hat{w}_i(x) = \frac{w_i(x)}{\sum_{j=0}^{N} w_j(x)},$$

where $w_i(x)$ is 1 at x_i and falls off radially to a distance $d_\emptyset$. We choose the weighting function w_i to be the simple inverse distance weighting function [She68]:

$$w_i(x) = \begin{cases} \left(\dfrac{d_\phi - d(x, x_i)}{d_\phi d(x, x_i)} \right)^\alpha, & \text{if} \quad d(x, x_i) < d_\phi, \\ 0, & \text{otherwise,} \end{cases}$$

where $d(x, x_i)$ denotes the distance between x and x_i, d_ϕ determines the area of influence, and α specifies the smoothness and the shape of the deformation. We set d_ϕ to be $\frac{1}{4}$ of the width after experimenting with different values of d_ϕ.

This type of deformation is often referred to as Shepard Interpolation. $B^{'}$ now has the same values as A on the seam, and hence, deforming B removes the discontinuity. However, Shepard Interpolation does not take into account gradient values and so the gradient field of the merged terrain may have discontinuities. Although a top-down view of the output will not show artefacts, 3D rendering immediately reveals that the terrain is not smooth. A possible approach to removing gradient differences is a bi-directional adjustment of Ψ_2 and Ψ_1 such that their height values agree along the seam. However, in our system, the old patch Ψ_1 is usually part of the already synthesized output, while Ψ_2 is a patch from the exemplar that needs to be placed in the output. A bi-directional adjustment may significantly modify the already placed pixels, and thus, the synthesized terrain features.

Instead of using Shepard Interpolation to obtain the merged patch Ψ, we remove the seam in its gradient field G_Ψ. G_Ψ is partitioned in two: the gradient field G_A of A and the gradient field G_B of portion B. Similarly to the above interpolation that deforms B to fit A along the seam, G_B is deformed to fit G_A along the optimal cut so that $G^{'}_B(x_i) = G_A(x_i),\ i = 0, 1, \ldots, N$, where $G^{'}_B$ is the deformation of G_B and x_i, $i = 0, 1, \ldots, N$, are pixels along the seam.

G_A and the deformed G_B now have the same values along the seam, and thus, the seam is removed from the modified field G_Ψ. An example of the Shepard Interpolation of a gradient field is presented in Figures 4(f) and (g). Patch merging

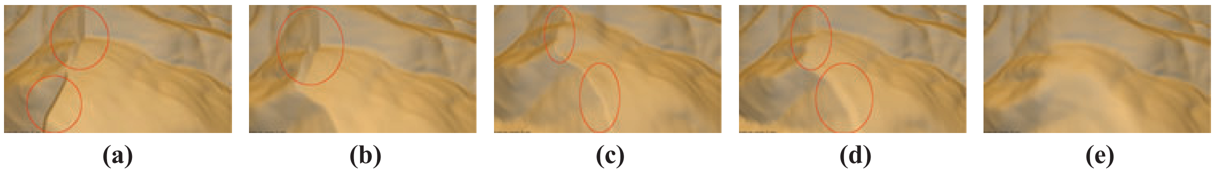


Figure 5: *Comparison of patch merging techniques. (a) No patch merging. (b) Graphcut algorithm. (c) Shepard Interpolation. (d) Graphcut+Poisson seam removal proposed by Zhou et al. [ZSTR07]. (e) Our method: Graphcut+Shepard Interpolation of the gradient+Poisson equation.*

ends with a new set of elevations Ψ' reconstructed from the modified gradient G_Ψ using Poisson image editing.

4.3.3. *Poisson image editing*

A Poisson solver is used to solve the linear system for the unknown values of Ψ'. The new set of height values Ψ' is pasted into the output heightfield and the result is a smoothly merged terrain, as shown in Figure 4(h). Figure 5 illustrates a result of our method compared with Shepard Interpolation and Poisson seam removal [ZSTR07].

5. GPU Acceleration

Texture-based terrain synthesis has a high computational cost, and thus is not suited for an interactive terrain sketching interface. We propose a GPU implementation that significantly accelerate the terrain generation time. Previous texture synthesis methods have been implemented on GPU [LH05, GN09, XLYD10], but to our knowledge, this is the first texture-based landscape generation on graphics hardware.

Implementing texture synthesis algorithms on a GPU is challenging because textures are generated in sequential order. The value of a new patch is used to determine the value of the subsequent patches, causing dependencies. Lefebvre and Hoppe propose a GPU implementation that builds on a order-independent pixel-based texture synthesis with no dependencies [LH05]. Other GPU algorithms accelerate texture synthesis by parallelizing the search for the best match at each iteration [GN09, XLYD10].

The search for the best matching patch consists of comparing the patch to be synthesized against all possible patches in the exemplar. These comparisons are based on the cost of selecting each candidate. Computing these costs is the main bottleneck of patch-based texture synthesis. Matching costs are independent from each other, and so can be calculated in parallel on graphics hardware. We use the NVIDIA CUDA C API for general-purpose GPU Programming to implement this parallelism.

Costs calculations in patch matching involve several memory transactions on the GPU memory. Maximizing memory throughput is a key factor in fully utilizing graphics hardware. This is usually achieved by reducing memory transactions to a minimum and using fast GPU memory such as shared memory [XLYD10]. However, the largest shared memory size is 48 KB on high-end GPU devices, which can only accommodate square patches of size less than 110×110, assuming that heights are represented using a single float. This means that a patch larger than 110×110 cannot be loaded to shared memory, and thus, using shared memory will place limitations on the range of allowable patch sizes. Other memory types include read-only texture memory and global memory. Global memory is off-chip memory, with the highest latency for non-coalesced access and better latency for coalesced memory requests. Garcia and Nielsen [GN09] keep the exemplar texture to texture memory and store the patch to be synthesized in global memory. Unfortunately, threads have to repeatedly perform rotation and mirroring operation to extract a patch from the exemplar. Furthermore, texture is slower than global memory when pixels are accessed in a coalesced fashion. We propose a new parallel implementation based on the use of coalesced memory access.

First, we design a faster CPU implementation that reuses candidate patches throughout the overall synthesis. In an analysis step, all candidates patches are extracted from the exemplar, rotated or mirrored, then converted to a linear array of pixels that is copied into a large array A. When calculating the cost of selecting a candidate patch, its pixel values are readily accessible from A. We refer to the single-threaded CPU solution described in Subsection 4.2 as CPU1 and the faster solution discussed here as CPU2.

A parallel implementation of CPU2 similarly copies all candidate patches into an array A in the global memory once, at the beginning of the synthesis process. No patch transformation or interpolation is performed during cost computation, which reduces the number of global memory reads and thus increases memory throughput. Other values used in patch matching such as control points and noise variances are copied to global memory. The number of concurrent threads spawned during cost computations is equal to the number of possible patches from the exemplar. Each thread determines the cost of selecting one candidate patch. Best performance is achieved by global memory transactions on the array A

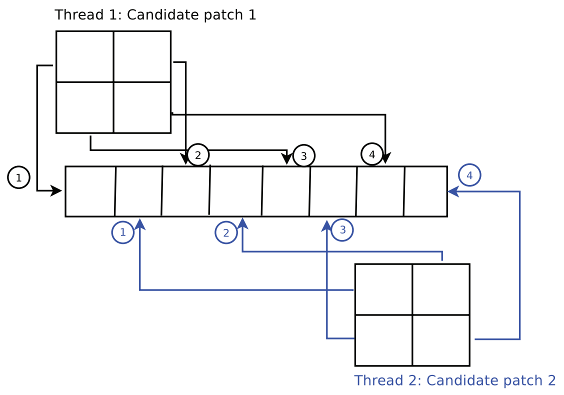


Figure 6: *Candidate patches are placed in a linear array such that threads can access their corresponding candidate in a sequentially aligned pattern. The encircled digits represent a patch pixel value and the arrows indicate where that pixel value is placed in the linear array.*

when thread i accesses the ith element in A [NVI10]. To achieve this, we do the following: for each position (x, y), pixel values at (x, y) of each candidate are placed in A successively. Figure 6 illustrates this process with two threads that each copy pixel information from a patch into the array A. This step is performed once, before the synthesis. During patch selection, threads access A directly for pixel values. Once every thread is terminated, a parallel insertion sort is used to find k candidates with the least costs. This process is repeated during each patch selection. This technique has a reduced number of CPU/GPU transfer since the target heightmap and patches are loaded onto the GPU once and subsequent CPU/GPU data transfers consist of copying the already synthesized pixels into the GPU memory at the beginning of each patch matching step. However, there are a large number of memory reads from GPU memory during cost computation. Even though access to global memory is coalesced, it is expensive and each concurrent thread still has to access all pixel values in its corresponding patch. We propose ways to improve our GPU acceleration technique in future work.

The limited amount of memory on GPUs places limitations on the maximum size of terrains. The parallel solution consumes a large amount of memory, because all candidate patches, including transformed patches, are transferred to the GPU memory. The maximum supported exemplar size on an NVIDIA GTX 280 with 1 GB of DDR3 memory is 2049×2049 when the patch size is 80×80. Table 1 presents the computation times of feature patch matching and non-feature patch matching for CPU1, CPU2 and the GPU solution as the exemplar size increases. The parallel solution performs up to 225 times faster than CPU1 and six times faster than CPU2. The timing results presented here were obtained on an Intel Core 2 Quad 2.33 GHz with 3 GB of

Table 1: *Patch matching performance (in seconds) as the exemplar size varies. Terrain size:* 1000×1000. *Patch size:* 80×80.

Feature patch matching: 271 selected patches			
Exemplar size	CPU1	CPU2	GPU
513×513	353.15	15.85	6.23
1025×1025	1491.29	56.13	10.23
1537×1537	2835.85	103.21	18.17
2049×2049	5906.7	145.43	26.19
Non-feature patch matching: 288 selected patches			
Exemplar size	CPU1	CPU2	GPU
513×513	239.13	9.93	6.33
1025×1025	1244.36	32.9	8.15
1537×1537	3098.5	78.23	17.75
2049×2049	5790.61	141.61	32.33

DDR3 memory. The GPU device used was an NVIDIA GTX 280 with 1 GB of DDR3 memory and 240 cores.

6. Experimental User Study

We used a within-subjects design in which every participant performed the same tasks on the same set of terrains during a single session. Two within-subjects experiments were designed to determine the influence of the terrain synthesis framework and patch merging technique on the realism of the generated landscapes. Landscape realism was evaluated by asking 20 participants if the terrains (*stimuli*) presented to them were similar to real life landscapes. For each experiment, Shapiro Wilk tests [SW65] on the collected data revealed that the sample did not follow a normal distribution (p-value < 0.001). Thus, the Friedman test [Fri37] was used to test for statistically significant difference. When the Friedman test was positive, the Turkey honestly significant difference test (HSD) was used for post-hoc analysis.

6.1. Experiment 1—Comparing the realism of terrains from the proposed framework against deformed terrains [GMS09] and real landscapes

This experiment asked users to compare three types of terrains randomly presented to them. Each participant was asked to performed three subtasks, each consisting of ranking a set of three terrains or stimuli. We refer to a real landscape as a terrain of type T_{real}, a terrain from our system is of type T_{sys} and a terrain deformed in the original sketching interface [GMS09] is of type T_{def}. The hypotheses tested in Experiment 1 are the following:

- A T_{real} terrain sourced by scanning real landscapes is superior to T_{sys} or T_{def} procedurally generated terrains.

- A T_{sys} terrain created from patches of real terrain appears more realistic than a T_{def} landscape.

For each subtask, a T_{real} landscape was obtained from a database of real heightmaps [US11] and the T_{def} terrain was obtained by drawing curves in the sketching interface to deform a flat terrain. The T_{sys} landscape was generated by our patch-based terrain synthesis framework using the same curves as the T_{def} heightfield and the T_{real} heightfield as the exemplar. The terrains were presented in a random order and participants were asked to rank them in ascending order of realism. An example of the terrains viewed by users is presented in Figure 8. The data collected from this experiment are a list of 60 (20×3) ratings of terrains of type T_{real} T_{sys} and T_{def}.

Statistical tests confirm a significant difference between the three groups (p-value < 0.001). Post-hoc analysis indicates that at a 5% level of significance, real terrains and landscapes synthesized by our system are more realistic than deformed terrains (p-value < 0.001). However, there is no significant evidence that real terrains are superior to landscapes generated by our terrain synthesis (p-value = 0.981). We conclude that the realism of the terrains generated by our framework is not dissimilar from that of real landscapes.

6.2. Experiment 2—Comparing the frequency of artefacts in patch-based terrains generated using Poisson seam removal, Shepard Interpolation and our proposed merging method

We denote landscapes generated with our proposed merging technique by the term M_{New}, terrains synthesized using Shepard Interpolation as $M_{Shepard}$ and those generated with Poisson seam removal are of type $M_{Poisson}$. The hypotheses tested in Experiment 2 are:

- M_{New} terrains have the least number of artefacts compared to $M_{Shepard}$ and $M_{Poisson}$ terrains.
- $M_{Poisson}$ landscapes have the highest rate and the most severe artefacts.

During this experiment, 18 terrains were presented to participants, consisting of six sets of M_{New}, $M_{Shepard}$ and $M_{Poisson}$ heightmaps. Terrains within a set were generated using the same input data (the example terrain and the target terrain) and only differed by the patch merging process used. Figure 9 shows a set of terrains presented to participants in this experiment. None were deformed after patch-based texture synthesis. All 18 terrains were presented in a random order and participants were asked to mark regions they believed that had artefacts and specify the severity of each artefact as slightly severe, moderately severe or very severe. Figure 7 shows the means plot of the artefacts frequencies of M_{New}, $M_{Shepard}$ and $M_{Poisson}$.

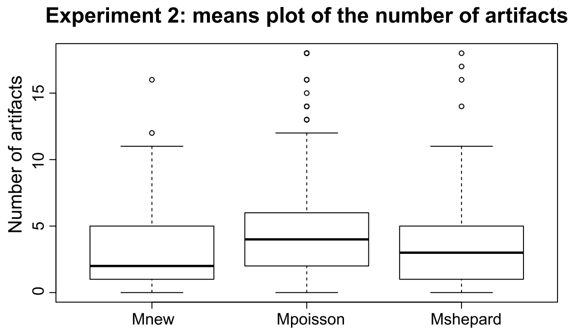


Figure 7: *Box plot of Experiment 2. The plot depicts the minimum, lower quartile, median, upper quartile and the maximum of artefact frequencies in each group.*

Statistical tests indicate a significant difference in the artefact frequencies of the three groups (p-value < 0.001). Post-hoc analysis reveals that $M_{Poisson}$ terrains have a higher artefact frequency than both M_{New} and $M_{Shepard}$ terrain. However, there is no significant evidence on the relationship between $M_{Shepard}$ and M_{New} terrains. Similarly, an analysis of the artefact severity level shows that there is a significant difference in the severity level: $M_{Poisson}$ terrain artefacts are worse than the artefacts in the other two terrain types. There is no statistically significant information indicating that the severity levels of $M_{Shepard}$ and M_{New} terrains are different.

An analysis of which artefacts the participants most agreed on reveals that their selection was dependent on the overall quality of the terrain. Terrains of type $M_{Poisson}$ whose artefacts were mostly lines on some areas of the landscape were more noticeable and participants easily agreed on those. Artefacts in $M_{Shepard}$ terrains on the other hand were spread on the landscape and participants opinions diverged over whether these were indeed artefacts or simply characteristics of the terrain topography. Terrains of type M_{New} had the least average number of artefacts but whenever an artefact was visible, due to the bad matching, it was clearly noticeable.

Bad matching occurs when a selected patch differs significantly from already placed pixels on the overlap region. For example, a combination of a high overlapping area cost and low scores in other matching criteria may place a patch with large height values next to an area with low elevation values. These situations are very rare but when they do occur, the large discrepancies in the pixel values cannot be completely removed by patch merging. A potential solution to this problem will be to deform already placed regions and the new patch such that patches with large height values and patches with low elevations can be brought down or raised up to an average level.

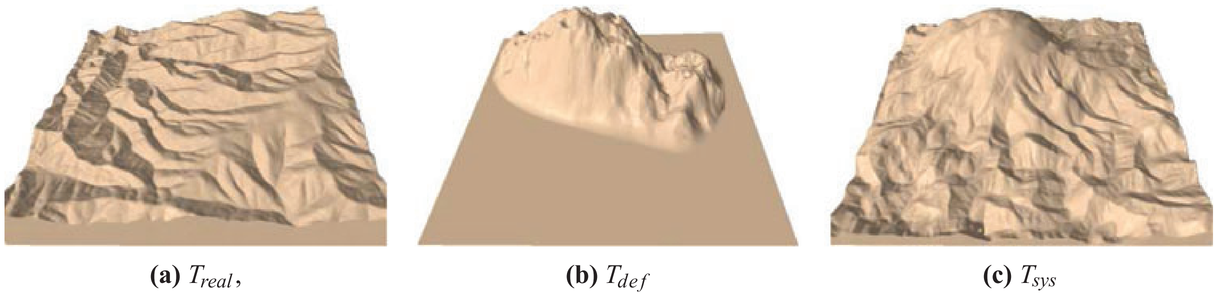


Figure 8: *A set of T_{real}, T_{def} and T_{sys} heightmaps presented to participants in Experiment 1. Twenty users were asked to rank them in ascending order according to their realism. Eight participants ranked the T_{real} terrain first, while 10 thought that the T_{sys} heightmap was the most realistic.*

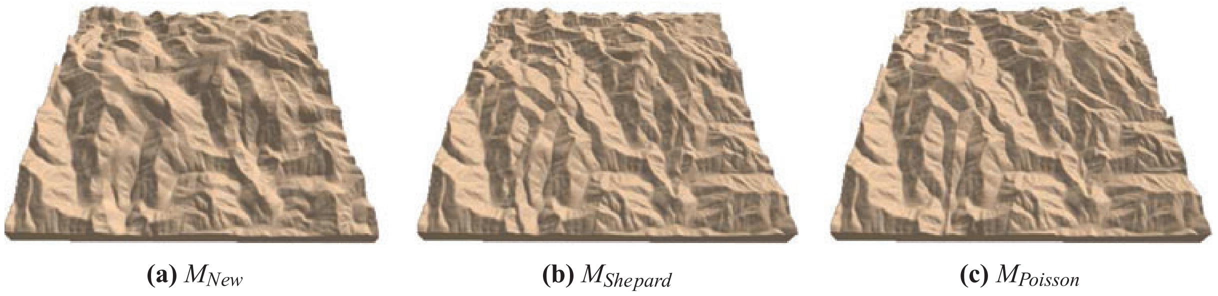


Figure 9: *A set of M_{New}, $M_{Shepard}$ and $M_{Poisson}$ heightmaps presented to participants in Experiment 2. Note how the boundary artefacts (straight lines) in the $M_{Poisson}$ terrain are easy to pinpoint, while the artefacts in $M_{Shepard}$ can be mistaken as part of the terrain surface. For this set for terrains, the average artefact frequencies recorded by the* 20 *participants were* 3.73 *for M_{New},* 5.93 *for $M_{Shepard}$ and* 14.44 *for $M_{Poisson}$.*

7. Visual Assessment

This section presents a visual assessment of the terrains generated by our system, as well as a comparison against previous patch-based terrain synthesis and a discussion of the limitations of our framework.

7.1. Comparison with Zhou *et al.* [ZSTR07]

We compare landscapes generated by our framework against the results of Zhou *et al.* [ZSTR07], using the same exemplar and target terrains. Figure 10 shows a terrain generated with our framework, using Mount Jackson as the exemplar and the “half-life” symbol as the target. The result is a better match of the target. Our framework is able to match the hooks of the half-life symbol and the circle around it so that they are well-defined in the synthesized heightmaps.

Figure 11 shows a failed case appearing on [ZSTR07]'s website. The output terrain is generated from patches of Cape Girardeau (KY, USA). It fails to match the target features due to poor feature matching. The output of our framework successfully matches the sketch map and regions where no features were specified having a lower level of detail. Because the feature dissimilarity cost takes into account the height values along the target features as well as the height profiles along paths perpendicular to those features, our system offers better feature patch matching.

7.2. Terrain results with user-sketched curves

When combined with the terrain sketching interface, our framework supports multiple sketched curves and deformations. Figure 12 shows a terrain deformed by four sets of constraint curves before a patch-based synthesis is applied to enhance realism. Instead of a smooth terrain, the final landscape is less homogeneous and exhibits erosion effects such as drainage patterns.

Combining a sketching interface [GMS09] with an example-based method allows users to benefit from intuitive and precise control (adding features such as volcanoes or removing the side of a mountain) along with higher quality terrains.

Our framework supports a variety of target terrains and its enhanced patch matching generates landscapes that match

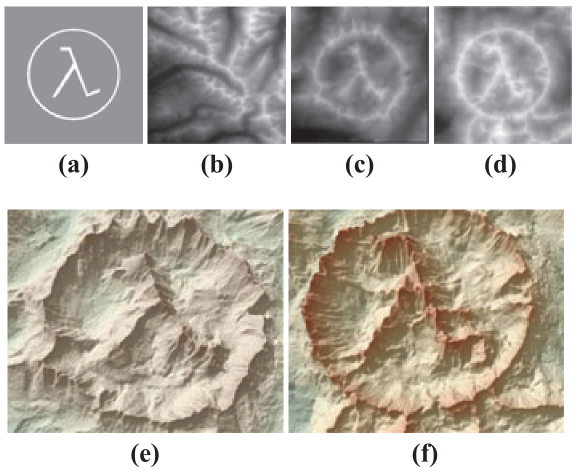


Figure 10: *Mount Jackson synthesis result. (a) Target terrain* (1000 × 1000). *(b) Mount Jackson exemplar* (1620 × 1620). *(c) Terrain generated by Zhou et al. algorithm [ZSTR07]. (d) Landscape generated by our framework in 1 min. (e) A 3D rendering of Zhou et al. output [ZSTR07]. (f) A 3D rendering of our framework's output.*

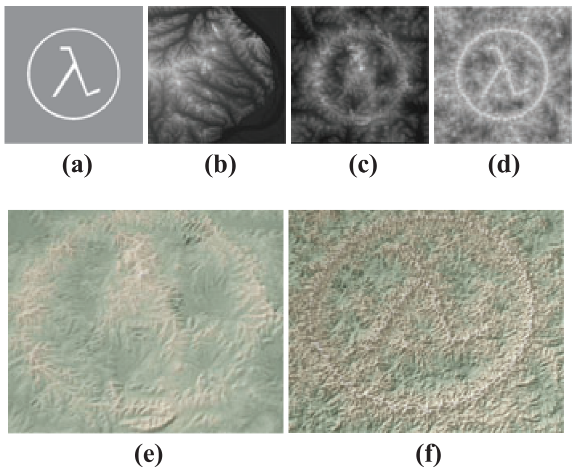


Figure 11: *Cape Giradeau synthesis result. (a) Target terrain (*2000 × 2000*). (b) Cape Girardeau exemplar (*1200 × 1200*). (c) Failed result generated by Zhou et al. algorithm [ZSTR07]. (d) Landscape generated by our framework in* 5.6 *min. (e) A 3D rendering of Zhou et al. output [ZSTR07]. (f) A 3D rendering of our framework's output.*

the user constraints as closely as possible. The synthesis illustrated in Figure 13 is heavily constrained by a concentric circles that cover a large portion of the target. The generated mountain range exhibits the same concentric circles in the form of ridges copied from a real landscape.

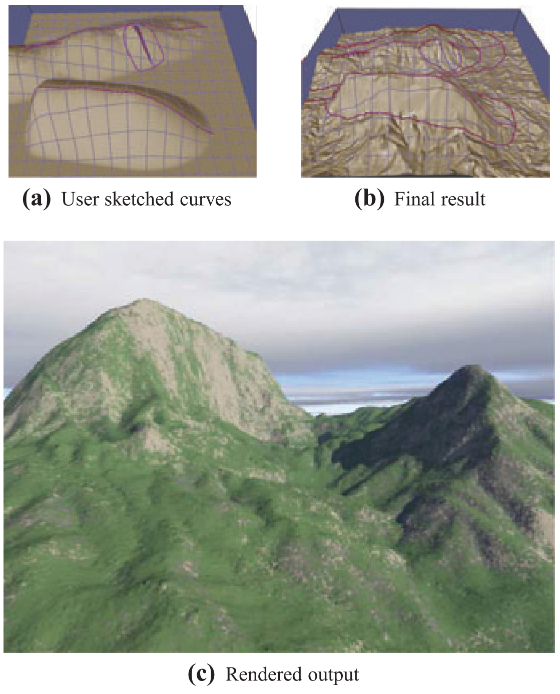


Figure 12: *Multiple sketched-curves. Both target and example terrains are* 512 × 512. *Generation time:* 20 *s.*

7.3. Limitations

The patch-based terrain synthesis framework presented in this paper improves on previous texture-based landscape generation methods and becomes more effective when combined with a terrain sketching interface. However, the framework fails to produce a terrain that is both realistic and fits user constraints in the following three cases:

- The features requested by the user are not present in the exemplar, and thus, cannot be placed in the output. Areas in the target with features that are not available in the example terrain are treated as regions with no dominant features. If a sketch interface is used for user control, the interface will deform the generated terrain by the patch-based synthesis to fit user constraints. If deformed regions are initially flat, then they will appear smooth and diminish the realism of the overall landscape.
- The patch size is too small and thus introduces very small random features in the output or the patch size is too large and fails to find good matches from the exemplar that capture the features specified in the target.
- During feature patch matching, similar or identical patches from the exemplar are placed in the output in close proximity, making the repetition noticeable if the features they are matched against are very similar or identical. This occurs when the target contains a long

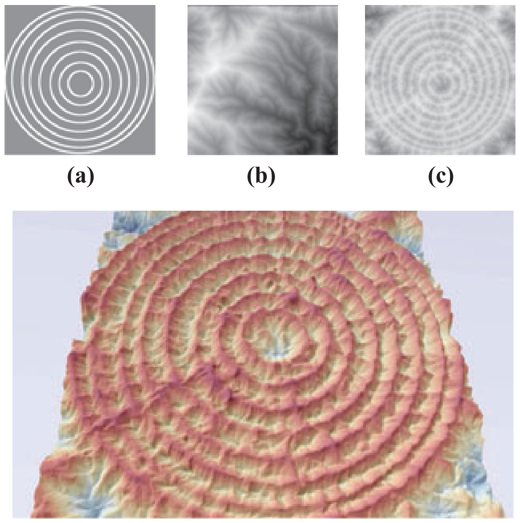


Figure 13: *Heavily constrained terrain synthesis. (a) Target terrain (*1000×1000*). (b) Exemplar terrain (*512×512*). (c) Landscape generated in 46 s. (d) A 3D rendering of the generated terrain.*

line with identical features, as the case in Figure 14. This can be fixed by keeping track of the most recently placed patches and ensuring that none of those patches are placed in the next iteration.

The shortcomings of our framework are fundamentally related to the input provided by the user. Although little can be done about real landscapes that do not have a specific feature (ridge or valley) desired by the user, an automatic search of the optimal patch size can be used to address the limitations related to patch size in future extensions of patch-based synthesis.

8. Conclusion

This paper presents a texture-based terrain synthesis, controlled by drawing $2\frac{1}{2}D$ curves in a sketching interface or supplying a 2D feature map. Terrain synthesis is a feature-guided patch-based texture synthesis process that extracts features from a supplied real landscape and a target terrain. The target heightfield is a terrain deformed by user-sketched curves in an interface or a provided feature map. Feature extraction is followed by a patch matching process. A novel patch merging technique is proposed to remove boundary artefacts created by overlapping patches. This paper also presents a parallel implementation on graphics hardware that speed-up patch matching by a ratio of 6 to 225 times. Finally, user experiments show that terrains generated by our system are as realistic as real landscapes and our patch merging technique is more successful than the state-of-the-art Poisson seam removal. Our patch-based terrain synthesis is useful for many applications including virtual environments, entertainment and flight simulation systems.

The patch-based synthesis presented in this paper could be extended by using multiple example terrains to produce landscapes with a visual appearance combining all the exemplars. This will increase the range of possible output terrains. The quality of user control can be further improved by extracting new user-controllable curves from the deformations introduced during non-feature patch matching. This gives users more control over small deformations on the terrain

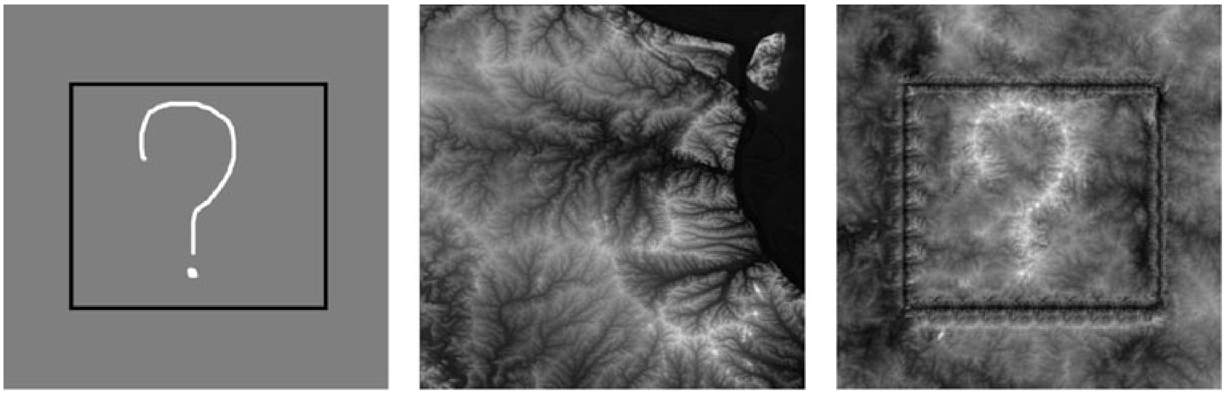

(a) Target (1000×1000)
(b) Exemplar 1025×1025
(c) Final output

Figure 14: *Synthesizing a terrain with a group of identical features. Features along the top, left and bottom sides of the rectangle, representing the terrain valleys, have similar or identical matches from the exemplar. The patch merging deforms each patch placed in the output according to already filled regions, and thus identical patches that are placed far apart from each other are difficult to spot. However, these repetitions are more visible in the feature matching step, due to their locality.*

surface. The proposed framework could be further accelerated by implementing patch matching on multiple GPU devices. The use of multiple GPU will also increase the supported maximum terrain sizes. Finally, automatically selecting the optimal patch size in a pre-processing step will not only reduce parameter manipulation but also improve the quality of the terrain synthesis.

References

[BdBG10] Bangay S., de Bruyn D., Glass K.: Minimum spanning trees for valley and ridge characterization in digital elevation maps. In *Proceedings of AFRIGRAPH '10* (New York, NY, USA, 2010), ACM, pp. 73–82.

[BSS06] Brosz J., Samavati F. F., Sousa M. C.: Terrain synthesis by-example. In *Proceedings of GRAPP '06* (Setúbal, Portugal, 2006).

[CPT04] Criminisi A., Perez P., Toyama K.: Region filling and object removal by exemplar-based image inpainting. *IEEE Transactions on Image Processing 13*, 9 (2004), 1200–1212.

[CSH98] Chang Y.-C., Song G.-S., Hsu S.-K.: Automatic extraction of ridge and valley axes using the profile recognition and polygon-breaking algorithm. *Computer Geosciences 24* (1998), 83–93.

[Dac06] Dachsbacher C.: *Interactive Terrain Rendering: Towards Realism With Procedural Models and Graphics Hardware*. PhD thesis, University of Erlangen-Nuremberg, 2006.

[EF01] Efros a. A., Freeman W. T.: Image quilting for texture synthesis and transfer. In *Proceedings of SIGGRAPH '01* (New York, NY, USA, 2001), ACM, pp. 341–346.

[EL99] Efros A. A., Leung T. K.: Texture synthesis by non-parametric sampling. In *Proceedings of ICCV '99* (1999), vol. 2, pp. 1033–1038.

[Fri37] Friedman M.: The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association 32*, 200 (1937), 675–701.

[GMS09] Gain J., Marais P., Strasser W.: Terrain sketching. In *I3D '09: Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2009), ACM, pp. 31–38.

[GN09] Garcia V., Nielsen F.: Searching high-dimensional neighbours: Cpu-based tailored data-structures versus gpu-based brute-force method. In *Computer Vision/Computer Graphics Collaboration Techniques*, vol. 5496. Springer, Berlin/Heidelberg (2009), pp. 425–436.

[KSE*03] Kwatra V., Schödl A., Essa I., Turk G., Bobick A.: Graphcut textures: Image and video synthesis using graph cuts. In *Proceedings of the ACM SIGGRAPH 2003 Papers* (New York, NY, USA, 2003), ACM, pp. 277–286.

[Lew87] Lewis J. P.: Generalized stochastic subdivision. *ACM Transactions on Graphics 6* (1987), 167–190.

[LH05] Lefebvre S., Hoppe H.: Parallel controllable texture synthesis. In *Proceedings of the ACM SIGGRAPH 2005 Papers* (New York, NY, USA, 2005), ACM, pp. 777–786.

[LLX*01] Liang L., Liu C., Xu Y.-Q., Guo B., Shum H.-Y.: Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics 20* (2001), 127–150.

[LZPW04] Levin A., Zomet A., Peleg S., Weiss Y.: Seamless image stitching in the gradient domain. In *Proceedings of EECV '04* (Berlin Heidelberg, 2004), Springer-Verlag, pp. 377–389.

[Man82] Mandelbrot B. B.: *The Fractal Geometry of Nature* (1st edition). W. H. Freeman, New York, USA, 1982.

[MJBF02] Milliron T., Jensen R. J., Barzel R., Finkelstein A.: A framework for geometric warps and deformations. *ACM Transactions on Graphics 21* (2002), 20–51.

[MKM89] Musgrave F. K., Kolb C. E., Mace R. S.: The synthesis and rendering of eroded fractal terrains. *SIGGRAPH Computer Graphics 23* (1989), 41–50.

[NVI10] Nvidia Corporation: *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*. NVIDIA Corporation, Santa Clara, CA, USA, 2010.

[NWD05] Neidhold B., Wacker M., Deussen O.: Interactive physically based fluid and erosion simulation. In *Proceedings of EGWNP '05* (2005), pp. 25–32.

[PGB03] Pérez P., Gangnet M., Blake A.: Poisson image editing. In *Proceedings of the ACM SIGGRAPH 2003 Papers* (New York, NY, USA, 2003), ACM, pp. 313–318.

[PGGM09] Peytavie A., Galin E., Grosjean J., Merillou S.: Arches: A framework for modeling complex terrains. *Computer Graphics Forum 28*, 2 (2009), 457–467.

[She68] Shepard D.: A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM National Conference* (New York, NY, USA, 1968), ACM, pp. 517–524.

[SW65] Shapiro S., Wilk M.: An analysis of variance test for normality (complete samples). *Biometrika 52* (1965), 591–611.

[US11] US Geological Survey: http://seamless.usgs.gov/, Accessed on 23 March 2011.

[WL00] WEI L.-Y., LEVOY M.: Fast texture synthesis using tree-structured vector quantization. In *Proceedings of SIGGRAPH '00* (New York, NY, USA, 2000), ACM Press, pp. 479–488.

[XLYD10] XIAO C., LIU M., YONGWEI N., DONG Z.: Fast exact nearest patch match for patch-based image editing and processing. *IEEE Transactions on Visualization and Computer Graphics 17*, 99 (2010), 1122–1134.

[ZG02] ZELINKA S., GARLAND M.: Towards real-time texture synthesis with the jump map. In *Proceedings of EGRW '02* (Aire-la-Ville, Switzerland, 2002), Eurographics Association, pp. 99–104.

[ZSTR07] ZHOU H., SUN J., TURK G., REHG J. M.: Terrain synthesis from digital elevation models. *IEEE Transactions on Visualization and Computer Graphics 13* (2007), 834–848.