EUROGRAPHICS 2015 / O. Sorkine-Hornung and M. Wimmer
(Guest Editors)

Volume 34 (2015), Number 2

Parallel, Realistic and Controllable Terrain Synthesis

J. Gain[1,2], B. Merry[1,2] and P. Marais[1] †

[1] Computer Science Department, University of Cape Town
[2] Centre for High Performance Computing, South Africa

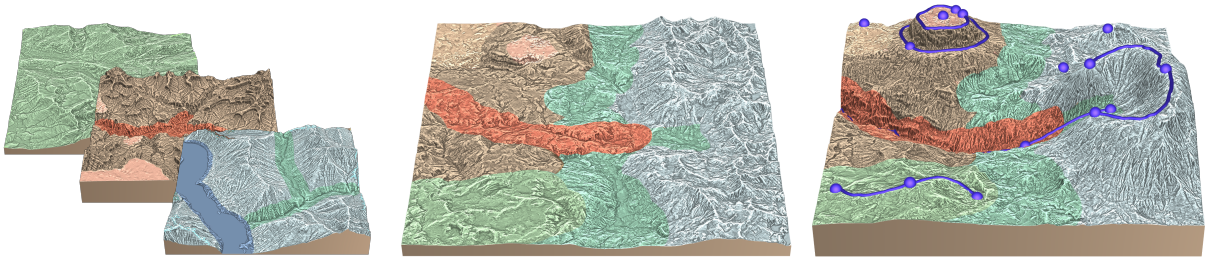


Figure 1: *Source heightfield exemplars (a) contribute to a synthesized terrain that can be controlled using: (b) a brush interface for painting terrain characteristics, and (c) point and curve constraints to shape landforms.*

Abstract

The challenge in terrain synthesis for virtual environments is to provide a combination of precise user control over landscape form, with interactive response and visually realistic results.
We present a system that builds on parallel pixel-based texture synthesis to enable interactive creation of an output terrain from a database of heightfield exemplars. We also provide modelers with control over height and surrounding slope by means of constraint points and curves; a paint-by-numbers interface for specifying the local character of terrain; coherence controls that allow localization of changes to the synthesized terrain; and copy-paste functionality to directly transplant terrain regions.
Together these contributions provide a level of realism that, based on user experiments, is indistinguishable from real source terrains; user control sufficient for precise placement of a variety of landforms, such as cliffs, ravines and mesas; and synthesis times of $165ms$ *for a* 1024^2 *terrain grid.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Three-Dimensional Graphics and Realism—Texture

1. Introduction

Virtual terrain is an important component in the representation of natural environments for computer games, film, simulation and training. In some applications, such as flight simulators, it can even be a dominant element.

From an artist's perspective, the process of creating digital terrain should ideally provide: (1) selective control, that ranges from the general (overall terrain character) to the specific (local placement of features such as ravines, cliffs, and ridges); (2) interactivity, with immediate feedback of design changes to support fine tuning; (3) realistic results, that are visually consistent with real landscapes.

So, how do existing approaches measure up to these goals? There are currently three broad strategies for terrain generation (ranked by increasing realism and attendant

† jgain@cs.uct.ac.za, bmerry@gmail.com, patrick@cs.uct.ac.za

computation cost): procedural noise, erosion simulation, and texture synthesis. Noise-based heightfields [Man83, Lew87, LLC*10] can be generated in real time from a small set of seed values, demonstrating considerable database amplification. They are also effective in capturing self-similarity across different scales and can be locally adapted to create terrain heterogeneity. On the downside, they do not adequately represent drainage and weathering, with the exception of work [GGG*13] that explicitly models river networks using principles derived from hydrology.

Another way to improve realism is by a simulation of hydraulic and thermal erosion [MKM89], a process that can be made interactive by a combination of localized updates and graphics hardware [NWD05, KBKv09, VBHS11]. Of necessity, however, these simulations simplify the complex physical processes involved, with a resulting loss of fine detail. For instance, these approaches cannot support the rapid interactive creation from scratch of continent-sized terrains, such as appear in Figure 12, where the erosion detail is fine relative to the scale.

More realistic terrains are generated by texture synthesis [ZSTR07, BSS07, WSG07], since this enables the use of real-world exemplars, such as the digital elevation models available through the U.S. Geological Survey [GOG*02]. Unfortunately, this realism currently comes at the expense of user control and interactivity, allowing only limited feature placement and requiring several minutes per synthesis.

Our focus is on bringing interactivity and user control to texture-based terrain synthesis. To achieve interactivity, we build on existing parallel pixel-based texture synthesis [LH05, LH06, HRRG08]. A parallel patch-based approach is less suitable because it would be roughly an order of magnitude slower [LL12].

We adapt synthesis (§4) to source more features by matching on relative rather than absolute height and by incorporating larger exemplars into GPU memory. To achieve user control, we introduce a variety of constraints (see Figure 1) specified through a mix of sketching, painting and widget interface elements (§3). Geometric constraints (§5.1) allow the user to shape the terrain by placing points and curves, with additional controls for height, slope and the envelope of surrounding influence. Type constraints (§5.2) are used to tag exemplar regions that share a common character (canyons, flat lands, coastline, hills, and so on), and this can then be transferred to the terrain under synthesis in a paint-by-numbers fashion. Finally, coherence constraints (§5.3) address the problem of local changes spreading globally by adding a per-pixel coherence term to the synthesis process that weights synthesis towards retaining existing values. This also enables copy-paste functionality, in which demarcated regions are interactively transplanted to a new location and elevation and then frozen by enforcing the maximum coherence.

Our key technical contributions are thus:

1. A hybrid interface for interactively specifying terrain constraints that makes use of sketching, painting and manipulator widgets.
2. Extensions to parallel texture synthesis to enable far larger exemplars and relative rather than absolute height matching, which are necessary to support terrain synthesis.
3. A set of novel mechanisms built into the synthesis matching process that satisfy point, curve, coherence and copy-paste constraints without degrading terrain fidelity.

The resulting system evinces (§6): high fidelity, based on a forced-choice user experiment in which we compared synthesized against real elevation data; control, which enables the placement of landforms at both a coarse and fine levels; and interactivity, with synthesis times on the order of 68ms for a 512^2 patch and 165ms for a 1024^2 patch.

2. Related Work

In the area of texture synthesis, landscapes are often used as illustrative examples [HJO*01, LH05, HRRG08]. Mostly, this is with the intention of demonstrating generality rather than a particular adaptation to terrain. In contrast, Zhou et al. [ZSTR07] provide terrain-specific synthesis by considering curvilinear features such as ridges and valleys as part of a patch-based approach. They choose patches from a single exemplar based on rough feature placement and then employ image warping to further align features, and graph-cut seams and poisson merging to hide patch boundaries. User control is by means of a 2D sketch map, which enables in-plane placement of features but no specification of height. Tasse et al. [TGM12] improve on this with a GPU implementation, better patch blending using Shephard Interpolation of the gradient field along seams and deformation to match height profiles. Texturing-inspired methods have also been used to add detail to low-resolution heightfields [BSS07] and patch holes in digital elevation models (DEMs) [WSG07]. While these methods are highly realistic they are far from interactive, requiring several minutes per synthesis. Furthermore, patch-based approaches often apply a post-hoc deformation after synthesis in order to match curvilinear constraints. This alters the direction of river courses and the slope of mountains in a fashion insensitive to the underlying geomorphology. For instance, when slope changes the character as well as inclination of erosion patterns may change significantly. In contrast, such constraints are built directly into the synthesis process in our system.

Much of the recent work on interactive terrain modeling has focused on sketching interfaces to procedural noise-based heightfield generation. Typically, the user sketches a characteristic curve, and an interpolating terrain is generated. These methods differ in the nature of the constraint curves: straight base lines [WI04]; $2\frac{1}{2}$D silhouette, shadow and footprint features [GMS09]; curves with side profiles [RME09]; and with attached control points to con-

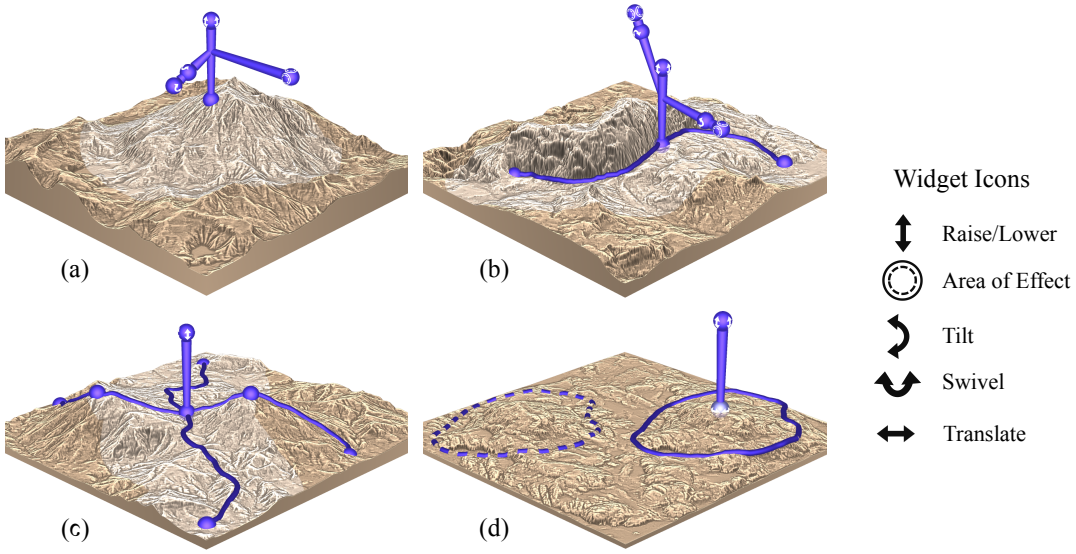


Figure 2: *Four types of constraint widgets provide different forms of control over height, gradient, area of influence and region copying: (a) free-standing, (b) curve-located, (c) junction-located, (d) centroid-located.*

trol elevation, orthogonal slope and noise [HGA*10]. They also display a variety of interpolation schemes, from multi-resolution deformation of wavelet noise to multigrid diffusion, even avoiding smooth interpolation altogether in order to support overlaying vector elements with distinct boundaries, such as rivers and roads [BN08]. The first person sketching interface of Tasse et al [TEC*14] is notable in that it builds on existing terrain data rather than noise. A user is able to draw complex silhouettes from a chosen viewpoint and ridge lines on the source terrain are raised or lowered to conform through diffusion-based warping. This has the advantage of building on real-world data but suffers from the same issues as patch-based texture synthesis in applying post-hoc deformation.

There are, of course, other options for terrain design besides sketching. De Carpentier and Bidarra [dCB09] employ a collection of circular brushes to paint different forms of procedural noise onto a heightfield, an approach with antecedents in early procedural noise control [PV95]. Overcoming the $2\frac{1}{2}$D restrictions of heightfields, Peytavie et al. [PGGM09] enable 3D structures, such as overhangs, arches and caves, by hybridizing implicit convolution surfaces and a grid of layered material stacks. Naturally, a sculpting interface metaphor is more appropriate in this instance and the system incorporates carving and accretion tools by using implicit blending. In general, these approaches are strong in user control and interactive feedback but fail to produce truly realistic terrains, since they rely on noise-based methods incapable of reliably modeling complex drainage patterns.

Our general approach is to provide local controls for texture synthesis specialized to terrain editing. The notion of local constraints on texture synthesis is, of course, not new. For instance, Lefebvre and Hoppe [LH05] provide drag-and-drop constraints, which locally suppress coordinate jitter within a circular region to enable feature placement. Our coherence constraints are a more general route to a similar goal. Likewise, Ashikhmin's [Ash01] interface allows users to paint from a color palette, with textures formed to match the overall image coloring. In a heightfield context, this would equate to setting regions of constant elevation, which is less useful than being able to paint terrain character.

3. Interface

From a user perspective, our system provides controls for modifying both the shape and character of a terrain, ultimately passing point, curve, type and copy-paste constraints to the synthesis process. These work best in concert, with users locally modifying the underlying landforms with geometric (point and curve) constraints as well as painting on type constraints to match the intended overlying landscape character. If required, terrain features can also be copied directly from source. In practice, users typically adopt a hierarchical approach: they begin with geometric constraints and, if these do not provide the desired outcome, then additional types constraints are painted in. Finally, if necessary an exact feature is copy-pasted. In this way, a balance between control and economy of expression is achieved.

At an interface level we rely on a combination of sketching, painting, and 3D widgets depending on the context. Point constraints have associated free-standing widgets (Figure 2(a)) with handles for height, slope direction, slope angle, and radius of effect. For curve constraints, an initial course is laid out by sketching onto the terrain, with alterations supported through oversketching. Curve attributes are accessible through curve-located widgets (Figure 2(b)) placed by the user. These are aligned with the local tangent direction and offer height control as well as left- and right-facing handles for gradient and area of effect. In this way, each side can be controlled separately [HGA*10]. For example, a cliff-top curve constraint would require a flat gradient on one side and a steep downward gradient on the other. We interpolate the widget values to derive attributes elsewhere on the constraint curve, using Hermite curves.

Unlike previous systems, we chose not to drive the interface entirely through sketching, because this does not handle the number of constraint parameters and required accuracy as gracefully as a hybrid approach. However, the underlying curve constraint values can be set along the curve as finely as required and so there is nothing to prevent interface alternatives, such as the sketching of a height profile. At curve intersections (including self-intersections) we automatically place junction-located widgets (Figure 2(c)). To avoid confusion over sidedness and a potential proliferation of handles, we limit these widgets to controlling only height. Together these geometric constraints support shaping the terrain both locally and globally.

To specify terrain character, we utilize a simple type painting interface. Users can pick a terrain type (representing semantically homogeneous terrain, such as canyons, mountains, or swampland) from an exemplar and paint it onto areas of the synthesized terrain. Ultimately, types are simply designated regions in the exemplars, so there is considerable flexibility in their choice. A palette entry is also reserved for locking down terrain changes, a useful function that prevents areas overlaid with the coherence-locking type from shifting during synthesis (as detailed in §5.3)

Copy-paste functionality represents the final core interface component of our system. Here, the user outlines a contiguous area by sketching (or oversketching) a loop on the synthesized or exemplar terrain and then shifts this as a whole to a new location and elevation using a centroid-located widget (Figure 2(d)). This supports several operations: repositioning a section of the synthesized terrain including raising or lowering its overall height; copying and transplanting an exemplar region; or even pasting in an entire exemplar, thus enabling the editing of existing terrains.

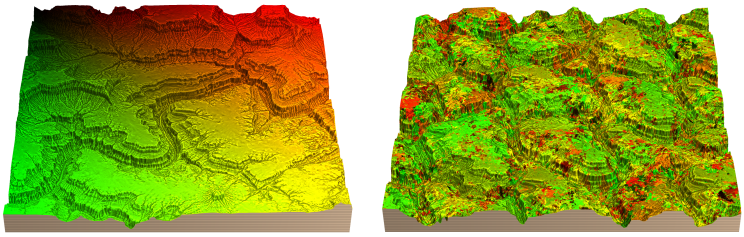
On the rendering side, our main difficulty lies in effectively portraying scale for blocks of isolated terrain, where it is easy to lose judgement of absolute height. Unfortunately, our interface imposes some restrictions. Using hypsometric tints to color elevation bands would interfere with type-coloring. We also found that fixing widget size to create a familiar reference damages usability across different zoom distances. In the end, we fell back on traditional contour and grid lines along with radiance scaling [VPB*10], a fast detail enhancement technique that accentuates high curvature features such as ridge lines and erosion patterns. An additional cue is provided by shading red those regions with elevation falling outside the maximum range of the input exemplars.

4. Synthesis Framework

Underpinning the interface for specifying geometric, type and coherence constraints is a set of modifications to existing parallel hierarchical texture synthesis [LH05] (see Figure 3). There the general strategy is to pre-generate a coarse-to-fine pyramid of exemplar data and then at run-time synthesize an image in parallel using a sequence of upsampling, jitter and correction phases. Upsampling moves synthesis towards finer resolutions, jitter introduces coordinate variation for the sake of visual interest and correction nudges coordinates towards a better match with exemplar neighborhoods using several passes. We also adopt the appearance space and multiscale extensions of Lefebvre and Hoppe [LH06] and Han et al. [HRRG08] to improve matching quality and support multiple terrain exemplars. Where we depart from these approaches it is to better exploit current graphics hardware, fit more terrain exemplars into graphics memory, and allow flexibility in matching coordinates with different heights but similar neighborhood shape.

Our first significant change is to replace the Gaussian image stack, introduced by Lefebvre and Hoppe [LH05], with a traditional image pyramid. Being able to capture larger terrain features requires larger exemplars, and while a stack supports greater detail at coarser levels, it requires $O(N \log N)$ memory, compared to $O(N)$ for a pyramid. Lefebvre and Hoppe [LH05] note that using integral coordinates to address pixels in coarse levels of the pyramid can cause features to snap to this grid, but this can be circumvented by using real-valued coordinates and bilinear interpolation when sampling the images. We store all the exemplars for a given pyramid level in a single OpenCL image, since the synthesis process needs very little modification to handle such an atlas. Also, in the interests of compacting the exemplar database, we drop the PCA approach to neighborhood matching [LH05], while retaining the PCA required for appearance space transformation.

As in Lefebvre and Hoppe [LH06], PCA reduction of Gaussian-weighted 5×5 neighborhoods defines an appearance space exemplar pyramid. We use four modes, the first of which is set to the weighted mean height. This gives access to slightly smoothed heights without incurring additional storage costs. The modes are quantized to 16 bits each and stored in an OpenCL image. Quantization reduces memory requirements, but more importantly, it reduces data transfer, which is critical to performance, because neighbor-



(a) (b)

Figure 3: *A patch view of synthesis: (a) single exemplar with coordinates mapped to red (x-axis) and green (y-axis), (b) resulting synthesis colored according to exemplar coordinates. This demonstrates the formation of small irregular patches from the input as is typical of parallel texture synthesis.*

hoods are sampled from non-coherent regions resulting in frequent cache misses.

The exemplar database is labelled from coarsest to finest level as $E^L, E^{L-1}, \ldots, E^0$, where E^0 is the original terrain input and E^{i+1} is E^i downsampled by a factor of two. We typically set $L = 6$. The corresponding appearance-space versions are denoted as $A^L, A^{L-1}, \ldots, A^0$. During real-time synthesis, a sequence of progressively finer synthesis images $S^M, S^{M-1}, \ldots, S^0$ is generated whose coordinates index the corresponding exemplar, except where $M > L$. Level M is initialized with the coordinates at the center of the first exemplar.

Our second major point of departure lies in how elevation is incorporated into synthesis. Alongside synthesis coordinates $S^i(p)$, we store a height offset $V^i(p)$ that is added to the original exemplar when it is indexed. This allows exemplar terrain to be raised or lowered to produce better matches, and is similar to the transfer function in Han et al. [HRRG08].

Upsample and jitter Each pixel at p in level i is used to initialize a 2×2 neighborhood in level $i-1$:

$$S^{i-1}(2p \pm \tfrac{1}{2}) = 2S^i(p) \pm \tfrac{1}{2} + rH(2p \pm \tfrac{1}{2}) \quad (1)$$

$$V^{i-1}(2p \pm \tfrac{1}{2}) = V^i(p) \quad (2)$$

Here r provides user control over the amount of jitter (with a default of $r = 0.4$), and H is a hash function that maps coordinates to pseudo-random offsets in $[-1,1]^2$. (Note that all samples are located at half-integer coordinates, for consistency with OpenCL image functions.)

Correction For synthesis at level L and below, coordinates are repeatedly altered to make neighborhoods in the synthesized result better match neighborhoods in the exemplar. We use two passes of correction, each split into four subpasses, as in Lefebvre and Hoppe [LH05].

At a given synthesis image position p, a number of potential replacements are considered for the exemplar coordinates $S^i(p)$:

$$Q = \left\{ C^i_j(S^i(p+\Delta) - \Delta) \;\middle|\; 1 \le j \le 2, \Delta \in \{-1,0,1\}^2 \right\}, \quad (3)$$

where C is a pre-computed lookup table of similar neighborhoods. For each candidate $q \in Q$ with height offset h, we gather the exemplar neighborhood N_E consisting of the four diagonal neighbors in appearance space, and also gather a synthesis neighborhood N_S, defined by

$$N_E(q,h) = \left\{ A^i(q+\delta) + ha \;\middle|\; \delta \in \{-1,1\}^2 \right\} \quad (4)$$

$$g^i(u,v) = A^i(S^i(u+v) - v) + V^i(u+v)a \quad (5)$$

$$N_S = \left\{ \tfrac{1}{3} \sum_{M \in \mathcal{M}} g^i(p + \delta, M\delta) \;\middle|\; \delta \in \{-1,1\}^2 \right\}, \quad (6)$$

where $\mathcal{M} = \{ \left(\begin{smallmatrix} 0 & 0 \\ 0 & 0 \end{smallmatrix}\right), \left(\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right), \left(\begin{smallmatrix} 0 & 0 \\ 0 & 1 \end{smallmatrix}\right) \}$ and $a = \left(\begin{smallmatrix} 1 & 0 & 0 & 0 \end{smallmatrix}\right)^T$. The score for a candidate q is then computed as:

$$\min_{h \in \mathbb{R}} (\kappa \| N_S - N_E(q,h) \|^2), \quad (7)$$

where $\kappa = 1$ if $j = 1$ (in the definition of q) and 2 otherwise. This weighting term has been shown to favor patch formation [LH05]. The optimal q and h become the new values of $S^i(p)$ and $V^i(p)$.

In common with previous work, a lookup table of similar neighborhoods is used to accelerate synthesis. For each pixel $A^i(p)$, we precompute the $k = 2$ most similar 5×5 neighborhoods in the exemplars at the same pyramid level, denoted by $C^i_j(p)$, with $1 \le j \le k$. There is the additional restriction that $C^i_j(p)$ cannot be on the edge of an exemplar, to avoid some out-of-bounds accesses that would otherwise occur during synthesis. Best matches are found using PatchMatch [BSFG09], executed on the CPU with multiple threads.

5. Control

5.1. Geometric Constraints

Point and curve constraints are the primary mechanism for shaping terrain in our system, and, as such, must satisfy

several requirements: constraint height and gradient should be interpolated, locally modifying the surrounding terrain within a prescribed region of influence, while avoiding exaggerated or unnatural effects.

To achieve this, we take a multi-resolution approach, with constraints enforced ever more locally as we move from coarser to finer synthesis. For point constraints, a pixel-specific target height h_t is determined from the specified height and gradient of the constraint.

It might seem sufficient to linearly blend the height of the pixel's best correction candidate h_c with the target h_t, and adjust the height offset accordingly. Unfortunately, the results tend to be either too flat-topped or too peaked and no single blending weight works in all instances.

Instead, we use a non-linear blending function, such that the final height is $h_c + \phi_d(h_t - h_c)$, where

$$t(d) = sd, \quad b(d) = \frac{-\lambda}{3t(d)^2}, \quad c(d) = b(d)t(d)^3 + \lambda t(d) \quad (8)$$

$$\phi_d(x) = \begin{cases} \lambda x + c(d) & \text{if } x < -t(d) \\ \lambda x - c(d) & \text{if } x > t(d) \\ -b(d)x^3 & \text{otherwise,} \end{cases} \quad (9)$$

with d being the distance between the current pixel and the constraint site. This is a C^1-continuous curve with slope 0 at $x = 0$ and slope λ at $x = \pm t(d)$, with default values of $s = 0.5$ and $\lambda = 0.5$, as shown in Figure 4. Informally, pixels for which the error in height is small are nudged only very slightly, while those with a large error receive a more disproportionate adjustment. The tolerance on this error depends on distance, so that pixels farther from the constraint site can vary over a larger height range than those nearby. Also, the area of effect is modulated on a multiresolution basis, to progressively localize the constraint at finer synthesis levels using $\min(\text{radius}, 4 \times 2^i)$.

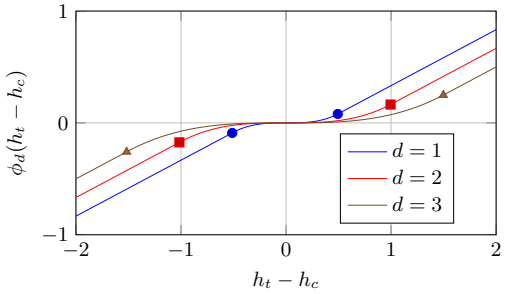


Figure 4: *Graph of ϕ_d for several values of d. The markers correspond to $x = \pm t(d)$.*

Applying these corrections across all synthesis levels tends to locally overconstrain curves, making their placement visually intrusive. This is solved by disabling geometric constraints on the finest two levels of synthesis, which fortuitously is also where they would be most computationally expensive.

This approach serves to attach a synthesized pixel to a point constraint (encompassing a ground-plane location, elevation, gradient vector, and radius of effect). To generalize from point to curve constraints, we create a curve parametrisation of constraint properties (location, elevation, left and right area of effect, and left and right gradient) using Hermite interpolation of the curve- and junction-located widget handles (§3). A pixel within the curve's envelope of effect is attached to the nearest point on the curve $C(t)$ and the properties at parameter t are fed to the same process as before (Equations 8, 9). The assigned properties also depend on whether the pixel falls to the left or right of the curve constraint. To cap the curve we simply use angular Hermite interpolation of the left and right properties at the endpoints.

Closest-point queries are accelerated using a distance field that is recomputed only when the horizontal shape of the curve changes, such as after a sketching operation.

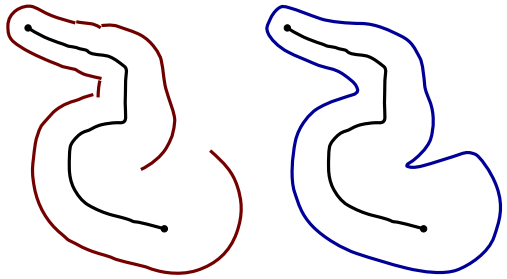


Figure 5: *Boundary discontinuities due to varying area of effect along a constraint curve: (a) original area of effect boundary with C^0 discontinuities and (b) mitigated by isocurve smoothing.*

There is, alas, one complication: while a distance field is C^0-continuous in the distance values themselves, this is not the case for the t parameter values of the closest points, which exhibit C^0-discontinuities. This can cause unintentionally irregular boundaries that are difficult for a user to anticipate or control. A classic example occurs along the medial axis of an arc where the parametrisation jumps from one endpoint to the other. More generally, wherever the area of effect is greater than the radius of curvature, we can expect such a discontinuity on the concave side of the curve (as evidenced in Figure 5(a)). This is resolved by an isocurve extraction followed by smoothing the area of effect envelope, which is then used to update the area of effect boundary and hence the overall extent of the deformation (Figure 5(b)). We do not otherwise need to modify the distance field because the constraint satisfaction and synthesis matching process is

robust to discontinuities, particularly when they occur further away from the curve.

A single constraint (point or curve) is applied to each pixel within its region of influence, but the influences of different constraints may well intersect. To accommodate such situations we combine overlapping constraints by blending the values of h_t and a, and recomputing t and c, using Equation (8). In the blend, these values are weighted proportionately to the inverse of the distance to the corresponding constraint. For each pixel, we maintain the sums $\sum b^i a^i$, $\sum b^i h_t^i$ and $\sum b^i$, where i runs over the constraints and b^i is the blend weight of the ith constraint. This has the advantage that a constraint can be modified by subtracting it, altering its properties, and then adding it back into the sums, without having to reevaluate all the overlapping contributors.

5.2. Type Constraints

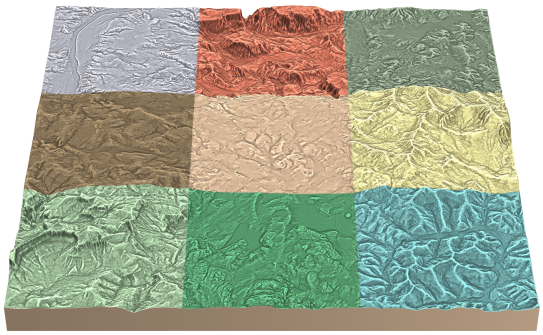


Figure 6: *A 3×3 quilt of type constraints.*

The goal of type constraints is to allow users to specify the regional characteristics of the landscape by painting a type tag (represented by a unique color in the interface) onto the terrain and then ensuring that the synthesis draws from exemplar coordinates with the same tagging (see Figure 6).

On the database side, all pixels p in the base exemplars are tagged with a terrain type $T^0(p)$. A coarser level i in the type pyramid $T^i(p)$ is derived by selecting the most frequently occurring type in the corresponding $2^i \times 2^i$ region of T^0. This coarsening takes place during database generation on the CPU. We briefly experimented with a machine learning approach to tagging the exemplars but this was not as effective as we had hoped, so instead the exemplars are entirely hand painted.

On the synthesis side, instead of a single type, each synthesis pixel p in the pyramid has a *s*et of target terrain types $R^i(p)$. The constraint is satisfied if $T^i(S^i(p)) \in R^i(p)$, that is, the type of the selected exemplar coordinate appears in the target set. This asymmetric matching enables unconstrained pixels that are allowed to take on any type.

The user paints R^0, as a single type per pixel, directly onto the terrain. Where no constraint is painted, we default to $R^0(p) = \mathcal{T}$, the set containing all terrain types present in the database. For $i > 0$, $R^i(p)$ is computed as the union of the corresponding 2×2 target sets at level $i-1$. However, any sets equal to $\mathcal{T}$ are excluded from this union, since otherwise narrow bands of constraints tend to be washed out at coarser levels. These calculations are performed on the GPU, updating a region locally at each level as the user modifies R^0.

During correction passes, candidates that fail to meet the constraints are rejected out of hand. We initially experimented with heavily penalizing such candidates instead, but found that with rare types, the constraint might never be satisfied because none of the candidates were suitable. If all possible matches are rejected, a seeding approach is applied to guarantee the constraint: for each type in $R^i(p)$, nine new candidates are picked at random (based on a hash of the synthesis coordinates) and evaluated as per Equation 7. For efficiency, we randomly preselect and store a list of coordinates for each terrain type.

Finally, as with geometric constraints, we found that the overall appearance was improved by disabling type constraints when synthesizing the finest two levels.

5.3. Coherence

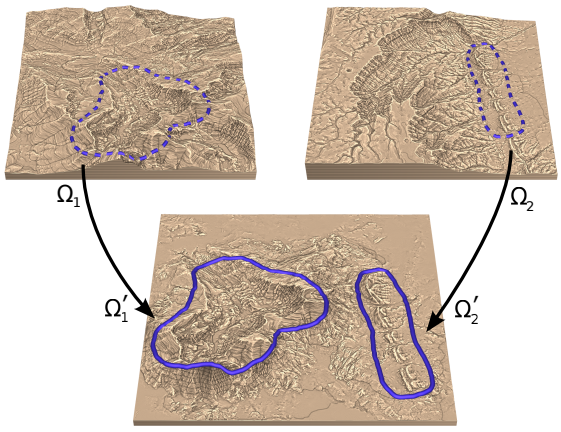
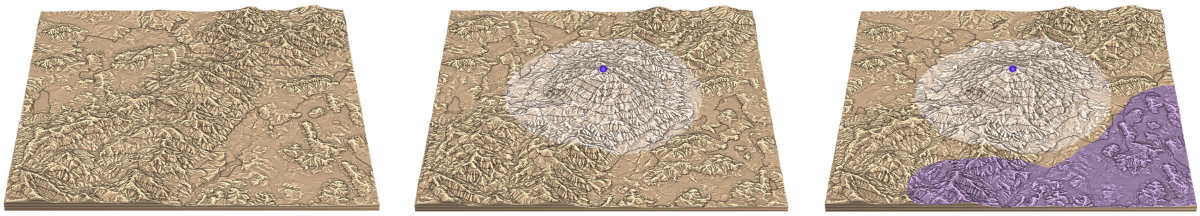


Figure 8: *Copy and paste operations. [top] Regions (Ω_1, Ω_2) copied from exemplar sources and [bottom] pasted into a synthesized terrain (Ω_1', Ω_2').*

Modifying constraints in real time introduces an additional complication: small changes do not necessarily exhibit either temporal coherence or spatial locality, and can cause unexpected popping artifacts. This is exacerbated by multi-resolution synthesis, where a small change in the input constraints can propagate from the coarser levels and have a large and abrupt impact on the final terrain.

To address this we explicitly incorporate change control in the form of a per-pixel coherence weighting term,



(a) (b) (c)

Figure 7: *Impact of coherence: (a) an initial terrain with no geometric constraints, (b) a point constraint causes small shifts even outside the area of effect (shown in a lighter shade), but this can be fixed (c) by applying coherence and locking constraints (the latter shaded in blue).*

$W^i(p) \in [0,1]$, where values close to zero enforce stasis and values close to one freely allow adaptation. This is implemented by recording the result of each correction pass. Then, in the subsequent frame this result is presented as an extra matching candidate, with its cost multiplied by $W^i(p)$. Weights towards zero thus bias the synthesis towards repeating the previous result, except where this would provide an unacceptably poor match.

To guarantee that resynthesis with the same constraints yields a repeatable outcome, it is not sufficient to record just the final corrected pixel for each level. The results for all sub-passes are needed, since otherwise the matches can diverge during earlier sub-passes, causing oscillation even for unchanged constraints.

In practice, we use three different weight values in W^0: unweighted $W^0(p) = 1$ inside the area of influence of manipulated constraints to encourage free alteration to meet the constraints; coherent $W^0(p) = 0.25$ over the remainder of the terrain to preserve locality and reduce popping; and locked down $W^0(p) = 0$ in regions that the user has painted over with a 'freezing' brush. To derive $W^i(p)$, for $i > 0$, we choose a zero (frozen) value if any pixel of the associated 2×2 block at level $i-1$ is zero; otherwise we simply select the most frequently occurring weight. This ensures that region freezing is properly dominant.

Region-based copying (see Figure 8) builds on this coherence architecture. To copy a region from an exemplar or the synthesized terrain we create a mask for the source region in question (Ω) and store the relevant coordinate values $S^i(\Omega)$ for all levels and sub-passes, using the same dilation procedure as for coherence freezing. These coordinates are then written to the target region (Ω') and the weight values in the target are locked to prevent changes $W^0(\Omega') = 0$. This enables not only translation from source to target but also, by adding a consistent offset to the base heights $V^i(\Omega') + \delta$, raising or lowering of the pasted region. Since the area around the target is unweighted the pasted region blends well with the surrounding synthesis.

6. Results and Discussion

The performance of our system was tested on a 3.2 GHz quad-core Intel Xeon, with 6 GB RAM, and an NVIDIA GTX 680 with 2 GB; the results appear in Figure 9. Of the three resolutions tested, the system performs interactively for 512^2 and 1024^2 with synthesis times averaging 63ms and 151ms, respectively. However, for curve constraints, creating the distance field during initial embedding and isocurve extraction during area of effect changes introduce additional overheads. The relatively high rendering costs are a result of preparing the terrain for radiance scaling, but this rendering mode could easily be dropped during active constraint manipulation.

In terms of memory usage, the database requires 32 bytes per pixel at the base level, which equates to $42\frac{2}{3}$ bytes per pixel for the entire pyramid. In contrast, a 7 level image stack would require $7 \times 32 = 224$ bytes. Synthesis is more memory intensive at 84 bytes per pixel, with the increase mostly due to the extra constraint storage. Further reduction is certainly possible by, for instance, using a single byte each for types and weights.

As an indication, our test hardware can load a 19-element exemplar database (each with 1024^2 base resolution) while synthesizing a 2048^2 terrain. For the same amount of memory, an image stack would only be able to fit 3 exemplars.

The allowable control and range of achievable landscapes are demonstrated in Figures 11–14. While most of the terrains in this paper are at a heightfield resolution of 1024^2 and a scale of 10m per pixel, Figure 12 shows continental-scale results (2048^2 resolution at 0.5km per pixel) and Figure 14 demonstrates higher resolution at a fine scale (4096^2 resolution at 10m per pixel).

One use-case is to broadly replicate existing landforms (Figure 11), which might require key alterations or for which there is no scanned data. However, more exaggerated fantastical terrains are also achievable (Figure 13), as are a wide variety of landforms (Figure 1).

In order to test terrain realism, we conducted a user experiment. This was designed as a two-alternative forced

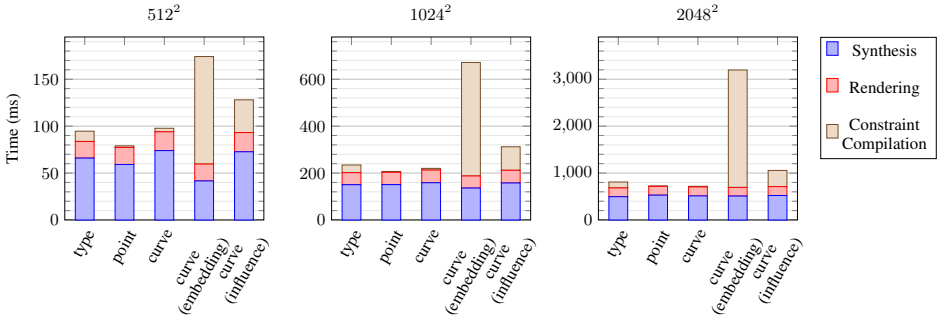


Figure 9: *Computation cost for synthesis at three resolutions (512^2, 1024^2 and 2048^2) for different constraint forms. Embedding curves and changing their area of influence have additional overheads and are graphed separately.*

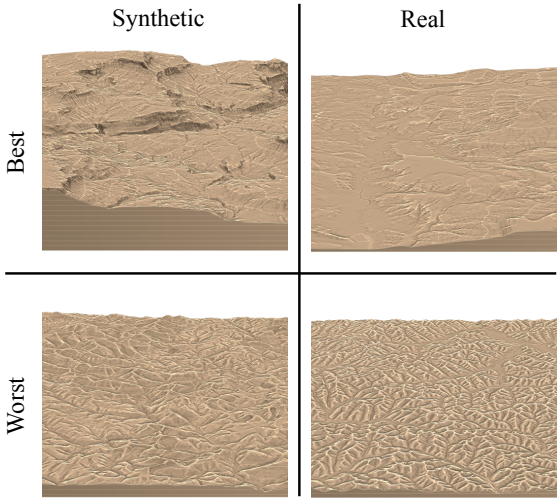


Figure 10: *Best- and worst-performing real and synthetic terrains from the user experiments. Each terrain was chosen as more realistic (clockwise from top left):* 84%, 80%, 4%, *and* 0% *of the time, respectively.*

choice (2-AFC) task [CW11], in which subjects were presented with a sequence of 60 pairs of terrain images, one real (sourced from the U.S. Geological Survey) and one synthetic, and asked to select which they considered more realistic.

The synthetic terrains were created by two expert users with our system using a database of 19 exemplars. They spent less than a day at the task, averaging about 20 minutes per terrain and employing only geometric and type constraints: no copy-pasting was used at all. All terrains were then rendered with the same settings from two viewpoints: overlooking and close-up. We independently permuted the order of both the real and synthetic images, as well as the option ('A' or 'B') under which each appeared. The experiment was run with 25 subjects, most of whom had a background in computer science but no tertiary training in geology or geography. For illustration, Figure 10 shows the best- and worst-scoring real and synthetic terrains.

If users are truly unable to tell the difference then we can expect the results to obey the same Bernoulli distribution as a sequence of coin flips. Surprisingly, in 54.6% of cases subjects chose the synthetic terrain over the real, and the coin-flip hypothesis can be rejected with a binomial test significance of $p = 0.0002$. Based on a post-experiment questionnaire, we tentatively attribute this paradoxical result to subjects misidentifying prominent features (such as sharp ridges and river bends) as unrealistic, while in the synthetic case the designers likely subconsciously avoided such cases.

Of the six subjects who identified more than half correctly, one subject, with extensive mountaineering and map making experience, chose the real option 78% of the time. Anecdotally, it would thus seem likely that subject experts would perform far better at identification, but this would need to be tested.

As part of a process of user-centered design, we conducted a separate informal usability study with three computer animators — an approach adopted in some other work on sketch-based interfaces and terrain modeling [WI04, GMS09, TEC*14]. In fact, it was in response to the artists' request for mechanisms to move and lock down sections of the landscape that we prioritised development of our coherence controls (§5.3).

During the usability study, users were given a short introduction and left to experiment with the system for up to an hour. Each user was then allowed 45 minutes to model an exaggerated fantastical landscape (see Figure 13). The users all settled on a common workflow: painting types over broad swathes, then applying curve constraints to shape the landscape, followed by detailed edits using a combination of localised type, curve and point constraints. In general, their use of curve constraints significantly outweighed point constraints.

Our system is not without limitations. It is impossible to reliably reproduce a distinctive curvilinear feature if it is oriented significantly at odds with entries in the database, such as a canyon type-painted east to west when source examples run only north to south: having sufficient space to incorporate features in a range of orientations is one reason we focus on careful memory reduction. There is no explicit consideration of geomorphology, which may lead to unrealistic global drainage patterns. For instance, a curvilinear feature, such as a canyon, may appear as a short isolated segment, albeit plausibly tailing off into side canyons and ravines using matches from the exemplars. The responsibility for detecting and correcting such situations is placed on the user. Finally, modeling finely inscribed detail in nearly flat areas, such as a shoreline with a small step in elevation, is time consuming and frustrating. This is because height and gradient changes in such cases are too local and too fine to be effectively specified by geometric constraints.

7. Conclusions

In summary, we specialize existing parallel pixel-based texture synthesis for terrain generation so as to support greater feature variety and more flexible matching by reducing the memory footprint of exemplars in the database and adding a height offset term. We also offer user control through a system of point, curve, type and coherence constraints that combine to enable the local placement of characteristic landforms. The resulting system is interactive, requiring an average of 165ms when updating a 1024^2 terrain, and visually realistic, in the sense that non-experts are unable to distinguish between real and synthesized terrains.

There are several avenues for future work. Supporting curvilinear features (such as rivers, ravines, and ridge lines) in different orientations by placing rotated versions of an exemplar into the database is rather memory inefficient. One way of overcoming this would be to build rotation into the matching process, by attaching a rotation term to synthesis coordinates.

We envisage improving the portrayal of surface detail in nearly flat areas by layering a flow field over exemplars to indicate the directionality of features and having users generate directionality constraints during painting. Another possibility would be to incorporate the image hybrid improvements of Risser et al. [RHDG10]. They improve the semantic fidelity of parallel texture synthesis using a multiscale descriptor and structure-preserving jitter applied in the appearance domain. This has the additional advantage of potentially improving overall terrain synthesis speeds.

Our early experiments with automatically classifying types using machine learning trained on a small set of hand-painted samples and the terrain's noise characteristics were promising enough to warrant further investigation.

Acknowledgements

Funding for this research was provided in part by the National Research Foundation and the Council for Scientific and Industrial Research, South Africa. DEM exemplar data is courtesy of the U.S. Geological Survey. Some computations were performed using facilities provided by the University of Cape Town's ICTS High Performance Computing team. Finally, we would like to thank Simon Anderson, Johanny Anderson and Kwegyir Lwabona for assistance with the images and valuable usability feedback.

References

[Ash01] ASHIKHMIN M.: Synthesizing natural textures. In *Proceedings of the 2001 symposium on Interactive 3D graphics* (2001), ACM, pp. 217–226. 3

[BN08] BRUNETON E., NEYRET F.: Real-time rendering and editing of vector-based terrains. *Computer Graphics Forum 27*, 2 (2008), 311–320. 3

[BSFG09] BARNES C., SHECHTMAN E., FINKELSTEIN A., GOLDMAN D. B.: Patchmatch: A randomized correspondence algorithm for structural image editing. In *ACM SIGGRAPH 2009 Papers* (New York, NY, USA, 2009), SIGGRAPH '09, ACM, pp. 24:1–24:11. 5

[BSS07] BROSZ J., SAMAVATI F., SOUSA M.: Terrain synthesis by-example. In *Advances in Computer Graphics and Computer Vision*, Braz J., Ranchordas A., Araújo H., Jorge J., (Eds.), vol. 4 of *Communications in Computer and Information Science*. Springer Berlin Heidelberg, 2007, pp. 58–77. 2

[CW11] CUNNINGHAM D., WALLRAVEN C.: *Experimental Design: From User Studies to Psychophysics*, 1st ed. A. K. Peters, Ltd., Natick, MA, USA, 2011. 9

[dCB09] DE CARPENTIER G. J. P., BIDARRA R.: Interactive GPU-based procedural heightfield brushes. In *Proceedings of the 4th International Conference on Foundations of Digital Games* (New York, NY, USA, 2009), FDG '09, ACM, pp. 55–62. 3

[GGG*13] GÉNEVAUX J.-D., GALIN E., GUÉRIN E., PEYTAVIE A., BENEŠ B.: Terrain generation using procedural models based on hydrology. *ACM Trans. Graph. 32*, 4 (July 2013), 143:1–143:13. 2

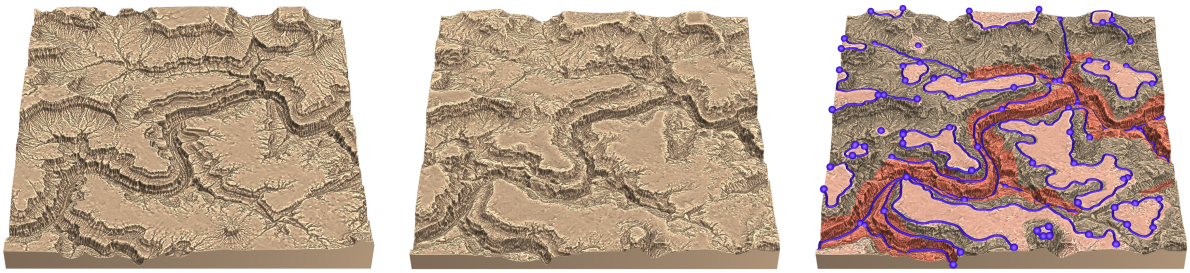
[GMS09] GAIN J., MARAIS P., STRASSER W.: Terrain sketching. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2009), I3D '09, ACM, pp. 31–38. 2, 9

[GOG*02] GESCH D., OIMOEN M., GREENLEE S., NELSON C., STEUCK M., TYLER D.: The national elevation dataset. *Photogrammetric Engineering and Remote Sensing 68*, 1 (2002), 5–11. 2

[HGA*10] HNAIDI H., GUÉRIN E., AKKOUCHE S., PEYTAVIE A., GALIN E.: Feature based terrain generation using diffusion equation. *Computer Graphics Forum 29*, 7 (2010), 2179–2186. 3, 4

[HJO*01] HERTZMANN A., JACOBS C. E., OLIVER N., CURLESS B., SALESIN D. H.: Image analogies. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2001), SIGGRAPH '01, ACM, pp. 327–340. 2

[HRRG08] HAN C., RISSER E., RAMAMOORTHI R., GRINSPUN E.: Multiscale texture synthesis. *ACM Trans. Graph. 27*, 3 (Aug. 2008), 51:1–51:8. 2, 4, 5



(a) (b) (c)

Figure 11: *Replicating the structure of existing terrain: (a) a section of the Grand Canyon, (b) broadly recreated in our system, and (c) the geometric and type constraints applied. In total three distinct types,* 27 *curve constraints and two point constraints were used in a modeling session lasting approximately* 2 *hours. (Note: no copy-paste functionality was used).*

Figure 12: *Large-scale landmasses, synthesised from exemplars of the west coast of North America at a resolution of* 2048^2 *and sampling of* 0.5*km per pixel (hence approximately* $1,000$*km across) and rendered using hypsometric tints. Each was created in less than half an hour by an experienced user, primarily employing type painting.*

[KBKv09] KRIŠTOF P., BENEŠ B., KŘIVÁNEK J., ŠT'AVA O.: Hydraulic erosion using smoothed particle hydrodynamics. *Computer Graphics Forum 28*, 2 (2009), 219–228. 2

[Lew87] LEWIS J. P.: Generalized stochastic subdivision. *ACM Trans. Graph. 6*, 3 (July 1987), 167–190. 2

[LH05] LEFEBVRE S., HOPPE H.: Parallel controllable texture synthesis. *ACM Trans. Graph. 24*, 3 (July 2005), 777–786. 2, 3, 4, 5

[LH06] LEFEBVRE S., HOPPE H.: Appearance-space texture synthesis. In *ACM SIGGRAPH 2006 Papers* (New York, NY, USA, 2006), SIGGRAPH '06, ACM, pp. 541–548. 2, 4

[LL12] LASRAM A., LEFEBVRE S.: Parallel patch-based texture synthesis. In *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics* (Aire-la-Ville, Switzerland, Switzerland, 2012), EGGH-HPG'12, Eurographics Association, pp. 115–124. 2

[LLC*10] LAGAE A., LEFEBVRE S., COOK R., DEROSE T., DRETTAKIS G., EBERT D., LEWIS J., PERLIN K., ZWICKER M.: A survey of procedural noise functions. *Computer Graphics Forum 29*, 8 (2010), 2579–2600. 2

[Man83] MANDELBROT B. B.: *The fractal geometry of nature*. Macmillan, 1983. 2

[MKM89] MUSGRAVE F. K., KOLB C. E., MACE R. S.: The synthesis and rendering of eroded fractal terrains. In *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1989), SIGGRAPH '89, ACM, pp. 41–50. 2

[NWD05] NEIDHOLD B., WACKER M., DEUSSEN O.: Interactive physically based fluid and erosion simulation. *NPH 5* (2005), 25–32. 2

[PGGM09] PEYTAVIE A., GALIN E., GROSJEAN J., MERILLOU S.: Arches: a framework for modeling complex terrains. *Computer Graphics Forum 28*, 2 (2009), 457–467. 3

[PV95] PERLIN K., VELHO L.: Live paint: Painting with procedural multiscale textures. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1995), SIGGRAPH '95, ACM, pp. 153–160. 3

[RHDG10] RISSER E., HAN C., DAHYOT R., GRINSPUN E.: Synthesizing structured image hybrids. *ACM Transactions on Graphics (TOG) 29*, 4 (2010), 85. 10

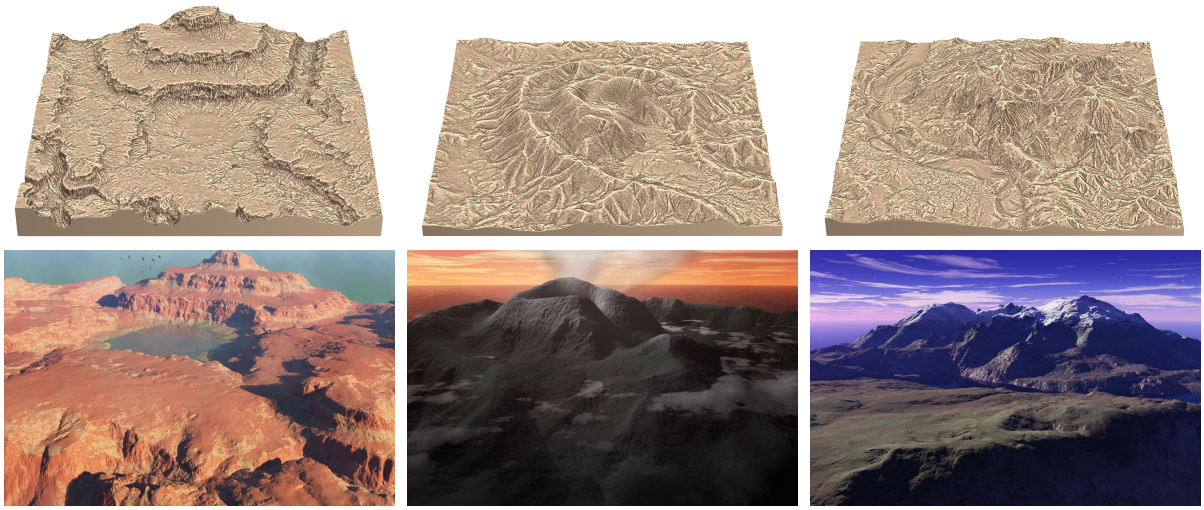
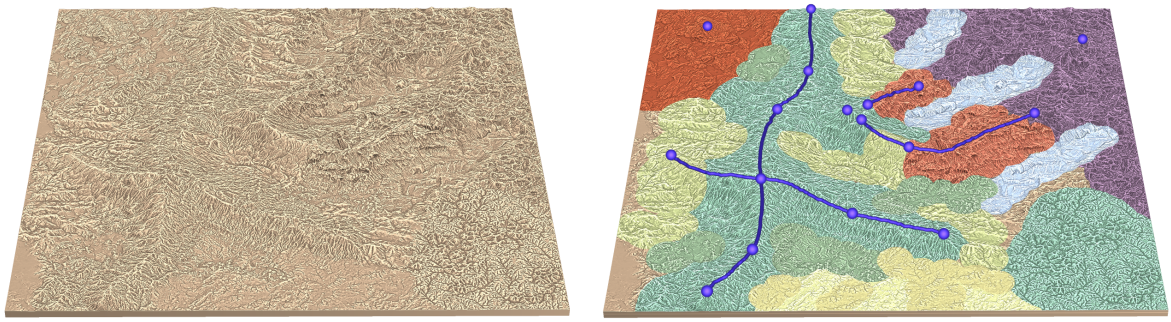


Figure 13: *Fantastical landscapes at a resolution of* 1024^2 *and sampling of* 10*m per pixel: [top] each modeled in under* 45 *minutes by artists working with our system, and then rendered externally using Terragen software [bottom].*



(a) (b)

Figure 14: *A large, detailed landscape with* 4096^2 *resolution and sampling of* 10*m per pixel: (a) without and (b) with geometric and type constraints displayed.*

[RME09] RUSNELL B., MOULD D., ERAMIAN M.: Feature-rich distance-based terrain synthesis. *The Visual Computer 25*, 5-7 (2009), 573–579. 2

[TEC*14] TASSE F. P., EMILIEN A., CANI M.-P., HAHMANN S., BERNHARDT A.: First person sketch-based terrain editing. In *Proceedings of the 2014 Graphics Interface Conference* (Toronto, Ont., Canada, 2014), GI '14, Canadian Information Processing Society, pp. 217–224. 3, 9

[TGM12] TASSE F.-P., GAIN J., MARAIS P.: Enhanced texture-based terrain synthesis on graphics hardware. *Computer Graphics Forum 31*, 6 (2012), 1959–1972. 2

[VBHS11] VANEK J., BENES B., HEROUT A., STAVA O.: Large-scale physics-based terrain editing using adaptive tiles on the GPU. *Computer Graphics and Applications, IEEE 31*, 6 (Nov 2011), 35–44. 2

[VPB*10] VERGNE R., PACANOWSKI R., BARLA P., GRANIER X., SCHLICK C.: Radiance scaling for versatile surface enhancement. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2010), I3D '10, ACM, pp. 143–150. 4

[WI04] WATANABE N., IGARASHI T.: A sketching interface for terrain modeling. In *ACM SIGGRAPH 2004 Posters* (New York, NY, USA, 2004), SIGGRAPH '04, ACM, p. 73. 2, 9

[WSG07] WECKER L., SAMAVATI F., GAVRILOVA M.: Contextual void patching for digital elevation models. *The Visual Computer 23*, 9-11 (2007), 881–890. 2

[ZSTR07] ZHOU H., SUN J., TURK G., REHG J.: Terrain synthesis from digital elevation models. *IEEE Transactions on Visualization and Computer Graphics 13*, 4 (July 2007), 834–848. 2