Proxy-Guided Texture Synthesis for Rendering Natural Scenes

Nicolas Bonneel[1,2], Michiel van de Panne[2,1], Sylvain Lefebvre [1,3], George Drettakis[1]

[1] REVES/INRIA Sophia-Antipolis [2] University of British Columbia [3] ALICE/INRIA Nancy

Abstract

Landscapes and other natural scenes are easy to photograph but difficult to model and render. We present a proxy-guided pipeline which allows for simple 3D proxy geometry to be rendered with the rich visual detail found in a suitably pre-annotated example image. This greatly simplifies the geometric modeling and texture mapping of such scenes. Our method renders at near-interactive rates and is designed by carefully adapting guidance-based texture synthesis to our goals. A guidance-map synthesis step is used to obtain silhouettes and borders that have the same rich detail as the source photo, using a Chamfer distance metric as a principled way of dealing with discrete texture labels. We adapt an efficient parallel approach to the challenging guided synthesis step we require, providing a fast and scalable solution. We provide a solution for local temporal coherence, by introducing a reprojection algorithm, which reuses earlier synthesis results when feasible, as measured by a distortion metric. Our method allows for the consistent integration of standard CG elements with the texture-synthesized elements. We demonstrate near-interactive camera motion and landscape editing on a number of examples.

1. Introduction

Modeling and rendering natural scenes using computer graphics remains a difficult problem. Creating such a scene using textured geometry involves a tedious modelling step, typically by an expert user, to create interesting geometry and then to create textures mapped to this geometry. The considerable effort that is entailed in each of these steps precludes the fast prototyping of natural landscapes, in particular by novice users. This constrasts sharply with the abundance of available images of natural scenes which contain a wealth of image detail, such as rock textures, and shape detail, such as the jagged silhouette of a craggy mountaintop. Our goal is to develop a method to exploit example images so that novice users can avoid the most painful steps in modeling a natural scene, i.e., modeling of detailed geometry and the creation and mapping of surface textures. This supports applications such as previsualization, requiring lightweight, easily modifiable models in order to support rapid exploration of possibilities. As an example (see Fig. 1 and video), imagine a director or game level designer wants to create a mountain scene with a lake. Using our system she can rapidly create a simple proxy and test the "look-and-feel" of two different example landscape styles, and walk through the scenes in 3D at near-interactive rate.

Three key challenges need to be overcome in realizing our proxy-guided rendering approach: (1) Complex natural scenes often exhibit visually detailed boundaries between texture regions. These need to replace the smooth bound-

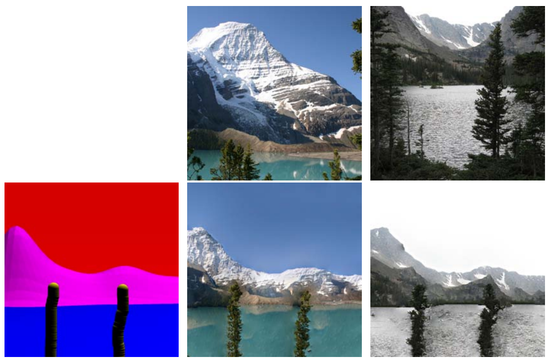
Figure 1: *Previsualization of a proxy (bottom left) using two different source images (top middle, top right). The renderings are shown below their respective source images.*

DOI: 10.2312/PE/VMV/VMV10/087-095

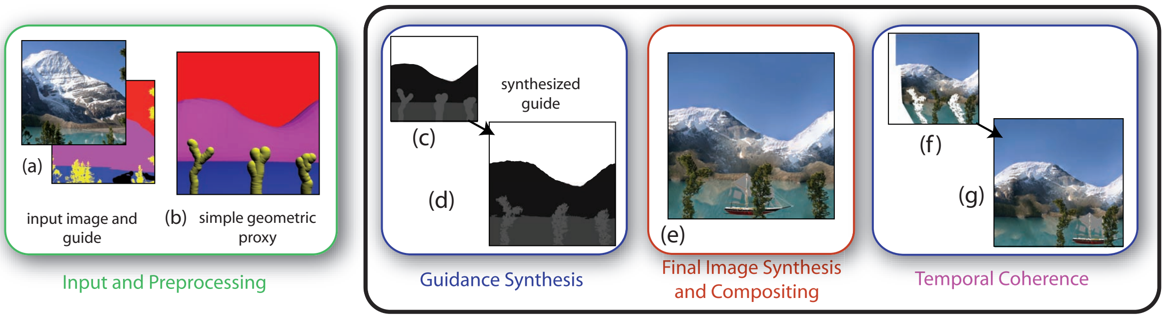

Figure 2: *Rendering rich visual depictions from simple proxy geometry using texture synthesis. Given a source photo and rapidly-modeled proxy geometry the goal is to produce depictions of the 3D proxy rich in detail and that mask the simple nature of the proxy. Temporal coherence is achieved using reprojection and texture resynthesis for disoccluded/distorted areas.*

aries that arise from using simple proxy geometry. (2) Proxy-guided texture synthesis for rendering should be fast. However, state-of-the-art parallel texture synthesis techniques do not directly work with guidance maps or with metrics such as Chamfer distance. (3) Temporal coherence is required as the camera moves through the scene. However previous texture synthesis approaches do not provide temporal coherence.

In our pipeline (Fig. 2), the user first selects a source photo, (a), from a pre-annotated set, having the desired detailed visual style. The associate guide image provides category annotations, such as sky, mountain etc. The user then creates a simple 3D model of the desired geometry (the *proxy*) (b). The proxy is rendered from a desired viewpoint to create a guidance map, (c). Detail is added to the silhouettes using texture synthesis to create a final guidance map, (d), based on the Chamfer distance as a principled way to treat labels. A fast texture synthesis stage, possibly composited with CG elements, produces the final image, (e). When the camera moves, temporal coherence is achieved using reprojection, shown in (f), followed by distortion flagging and guidance-based hole-filling resynthesis. The final image from the new view is shown in (g).

Our overarching contribution is the presentation of a new visual enrichment pipeline using the power of guided texture synthesis to render natural scenes with a crude proxy. To achieve this goal, we present the following contributions:

- An efficient guidance-map synthesis step to explicitly introduce silhouette detail where required. Because guidance maps involve discrete labels and not colors, we introduce a principled Chamfer distance metric and use this throughout our texture synthesis pipeline wherever guidance maps are involved.
- A fast guided texture synthesis approach. We substantially adapt the fast parallel approach of [LH05] to our hard-to-handle non-homogeneous input images, by introducing patch-based initialization, carefully treating repetitions in synthesis as well as the processing of patch/label boundaries during gradient transfer.
- A reprojection-based solution for local temporal coherence, which reuses previous frames, resynthesizes a small number of pixels in parallel, based on a distortion metric and includes Poisson stitching to handle difficult seams.

We illustrate our approach on numerous examples with rich visual detail created from pre-annotated images, shown in Sect. 7, additional material and the accompanying video.

2. Previous work

Texture synthesis from an example image has attracted much interest over the past two decades (see survey in [WLKT09]). Our work is most closely related to guided texture synthesis [Ash01, HJO*01]. In particular, in [HJO*01] the idea of texture–by–numbers is introduced. The input to the system is a color image and a map segmenting its content through shades of basic RGB colors, called *labels* in the following. Given a guide — a new map using similar labels but a different layout — the algorithm synthesizes a new image with a corresponding layout. Synthesis is performed by matching square neighborhoods in a coarse to fine process. The similarity metric is an L_2 norm comparing both colors and labels. The algorithm produces impressive results on a variety of images. Global energy minimization methods have also been proposed, via graph-cut [Ash01] or dynamic programming [EF01], while still relying on L_2 norms on guides.

Much of the visual richness of natural scenes stems from the complex boundary shapes as well as the texture itself. We thus introduce a guidance synthesis step, which allows the enrichment of boundaries from a rendered proxy geometry using details from a source image. Aspects of guidance synthesis can be found in [ZZV*03], where binary guidance maps are created. The more recent work of Rosenberger [RCOL09] focusses on ordered layers of textures and [RLC*06] on binary masks. Our guidance synthesis works with unordered multi-label guidance maps and is orders of magnitude faster than the synthesis times of [RCOL09].

The *Deep Photo* pipeline [KNC*08] uses such a texture-by-number approach to synthesize surface texture for regions that are occluded when backprojecting from photographs, but requires precisely registered images and knowledge of the 3D geometry. We only know a labeling of the source image, without knowledge of its 3D geometry, allowing the creation of novel 3D geometry without a real world-reference. In *CG2Real* [PFA*09] renderings are enhanced with photos from large image collections. The difference with our approach is that neither camera motion nor enhanced silhouettes are provided. A detailed comparison is presented in the additional material. Ritter et al. [HRRG08] proposes temporally coherent zooming with unguided texture synthesis and multiple input images but no 3D information is used thus lacking parallax effects.

For fast high-quality texture synthesis we rely on a parallel algorithm [LH05], which exploits spatial coherence using only local information around each synthesized pixel during the neighborhood matching process (k-coherent candidates [TZL*02]). The algorithm proceeds in a coarse-to-fine fashion, by iteratively refining the solution of the synthesis. The refinement works by selecting among precomputed k-candidate neighborhoods of each pixel of the input texture for the one best matching the neighborhood of the current pixel. Adapting such an algorithm for guided synthesis is non-trivial since the guide precludes a local search approach, and has not been attempted before.

Reprojection is important to ensure temporal coherence in our approach. Previous reprojection approaches reuse expensive-to-generate pixels of previous frames by reprojecting them in the current view, thus avoiding expensive generation. Such techniques have been used in image-based rendering (e.g., [CW93]), ray-tracing (e.g., [AH95]) and in interactive rendering (e.g, [WDP99], [SW08]). In our case, the expensive process per pixel is texture synthesis; we use reprojection to limit the number of pixels resynthesized at a given frame.

3. Input and Preprocessing

As input, we require simple proxy geometry for scene elements shaded with texture-synthesis, a source image with the desired texture categories, and any desired 3D CG elements.

Proxy geometry: We provide a set of basic tools to quickly model an approximate scene. Since we focus our efforts on the rendering process, we restrict ourselves to the use of a simple *terrain tool* and *sphere tool* to create the proxy geometry for our examples (see video). This limited toolset could be augmented or replaced by many other alternatives. The terrain tool simply pushes and pulls vertices of a heightfield with a Gaussian region of influence with the distance to the cursor. The sphere tool instances spheres along a given path, thereby enabling the easy creation of topologies that are not possible with the heightfield model, such as the arches shown in Fig. 9 (top row).

Source photo: Given the geometry, the user selects a photo providing rich details for texture-synthesis shading. The photo should have the suitable texture categories and have a roughly similar point of view as in the proxy scene. The texture categories in the source photo need to be labeled, which we accomplish through a segmentation process. In our case this involves about 30min of manual work, which is a one-time preprocessing overhead per source image. In large scale application, we envisage users selecting from a library of pre-segmented images. While a semi automatic labelling approach can be envisaged [RKB04], in our experience fully automated solutions do not always adequately capture the semantics. Although the accuracy of the segmentation is not critical, details in the boundaries will be transfered to a synthesized smooth guide. Thus details should be present in the input segmentation but do not always need to represent the boundary between regions with great accuracy.

Proxy guide: An image of texture category labels is obtained by rendering the 3D proxy geometry into a *proxy guide*, with no lighting computation and with the appropriate texture label associated with each component of the proxy. The labels should match those assigned to the segmented source image. In the illustrated images of the proxy guide, e.g., Fig. 2(a), we visualize the discrete texture category labels using distinctive colors. However, all processes in the shading pipeline will treat labels as having strictly discrete semantics, rather than continuously-valued colors. A proxy depth map (Fig. 5, mid right) is created at the same time as the proxy guide, and will be leveraged later to allow for depth-consistent compositing into the rendered scene.

4. Guidance synthesis

At this stage, the user has created a proxy guide with simple geometry; however this guide has smooth, unnatural borders and silhouettes. We now add rich border structure from the source photo via the source guide. We have chosen to use guided texture synthesis; however, the standard L_2-norms commonly used are ill-adapted to this problem because the image pixels represent discrete texture category labels, and not continuously-valued colors. Addressing this issue requires a number of modifications to state-of-the-art texture synthesis algorithms. The guidance synthesis phase also needs to produce a depth map corresponding to the synthesized guide, used to integrate CG elements into the scene and to move the camera through the scene. Both stages employ texture synthesis using guides consisting of labels, not colors. As we describe next, we propose methods based on Chamfer distances to address this issue.

4.1. Chamfer distance

A Chamfer distance computes a generalized distance between edges, or more generally, sets of points, and it is applied in many variations in shape matching applications [BTBW77, Bor88]. It is given by the mean of the minimal distances of each point in one set to the closest point in another set. Here, we define it as the sum over each pixel in neighborhood A of the L_∞-distance between the pixel in A and the closest pixel in neighborhood B sharing the same label. If no pixel in B shares the same label, a distance of twice the neighborhood size is given, which penalizes this matching. The symmetric Chamfer distance that we use is the sum of the Chamfer distance between neighborhoods A and B, and the distance between neighborhoods B and A. This allows us to account for differences in the geometry of the labels and for the fact that classes are intrinsically discrete.

The Chamfer distance provides a measure of proximity while the L_2 metric with colors to represents labels provides a measure of overlap, and is thus less meaningful. When details contain narrow features, overlap becomes a poor proxy for measuring similarity: a one-pixel wide vertical line will have a zero overlap with the same line displaced one pixel left or right. A second problem is that colors-as-labels allow only three texture labels to be conveniently represented independently in RGB. However, the majority of our examples have more than three texture labels. Any choice of colors to represent labels will thus lead to some pairs of dissimilar labels to be arbitrarily more distant than other dissimilar pairs. Finally, blending of color-based texture labels, in texture synthesis pyramids, creates mixed labels that can be problematic. We present an example in Fig. 3 (simplified to 3 labels for clarity) where the extra land that appears with the L_2 metric and colors (bottom) is an artifact of values color bleeding at coarse scales during multiresolution synthesis.

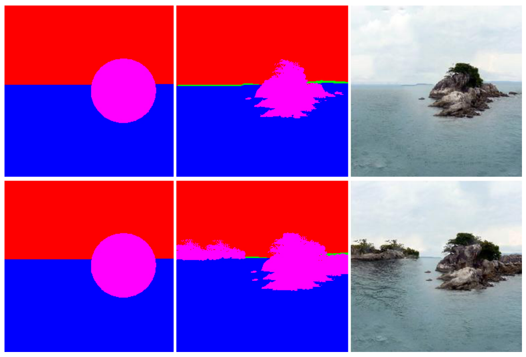
Figure 3: *Comparing a Chamfer distance on IDs with a voting pyramid (top) vs an L2 distance on colors during multiresolution synthesis (bottom). The latter creates new undesired magenta regions between the red sky and the blue sea. Left to right: Proxy guide, Synthesized IDs, Synthesis result.*

4.2. Synthesis process

Our guidance map texture synthesis uses an approach similar to [LH05]. However, the use of discrete identifiers precludes building a multiresolution Gaussian stack or a Gaussian pyramid [HJO*01]. We thus build a *voting stack* from which we can extract a *voting pyramid*. This step is performed by keeping the identifier which is the most present in a $N \times N$ kernel around each pixel. The kernel size is scaled by a factor of two at each level of the stack. The pyramid is extracted from the stack by sampling the stack every 2^l pixels, where l is the stack level. This filtering process corresponds to median filtering when we only have two labels.

We initialize the synthesis at a level which is not too coarse in order to preserve the large scale structure of the proxy guide. To synthesize a 256×256 guide, we start the synthesis at the 64×64 level. Starting at a coarser resolution gives more freedom in the silhouettes, but can result in a loss of semantic meaning. We do not use the jittering step of [LH05], to enforce coherence. We use 5×5 neighborhoods and $k = 5$ coherent matches, for a good tradeoff between quality and speed. These k coherent matches are found by choosing the k pixels that are furthest apart in image space to avoid repetitions, and which are closest in feature space to the current pixels in a set of $4k$ candidates. This is performed using a Hochbaum-Shmoys heuristic [HS85].

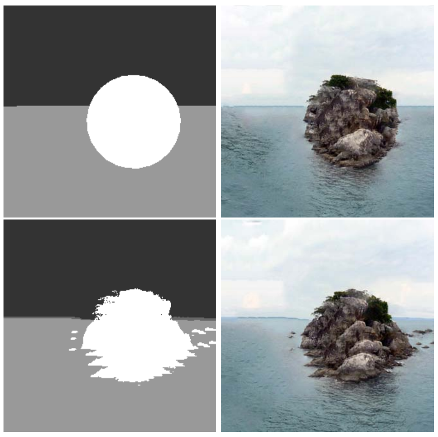
Figure 4: *Guidance synthesis. Guide and final result without (top) and with (bottom) enriched silhouettes. The input annotated photograph can be seen in Fig. 5 of the additional material.*

Fig. 4 illustrates the resulting image synthesis with and without the guide. The smooth silhouette of the proxy guide remains evident when guide synthesis is not used. The guide synthesis adds the necessary detail, resulting in the addition of trees and small rocks surrounding the main island.

4.3. Depth Synthesis

Consistent depth values are needed for all pixels to enable temporal coherence (§6) and compositing of 3D CG elements (§7). We use the available depth map for the proxy guide (Fig. 5(far left)) to develop a depth map corresponding to the synthesized guide. First, the pixels in the synthesized guide that have the same labels as the corresponding pixels in the proxy guide are assigned the proxy depth. The remaining pixels are assigned the depth of the closest pixel having the same label both in the proxy and the synthesized guide. This results in an extended depth map (Fig. 5 far right).

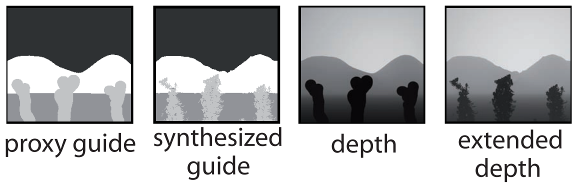

Figure 5: *Left to right: proxy guide without detail; synthesized guide; initial depth; extended depth.*

5. Fast Guided Texture Synthesis

We now have synthesized a detailed guide containing detailed silhouettes visually similar to the example, following the layout of the proxy rendering. The second challenge we address is developing a fast and high-quality *guided* texture synthesis approach, similar in spirit to [HJO*01]. To provide fast feedback and state-of-the-art synthesis quality we use a parallel texture synthesis algorithm [LH05], which uses local neighborhood information.

However, it is not straight-forward to use the algorithm for *guided* synthesis. A first difficulty is that in an area with poor label matching the local search will only find neighborhoods with incorrect labels. To overcome this, we initialize synthesis with an approximate result already enforcing labels. This ensures that, locally, neighborhoods with appropriate labels are found, while synthesis essentially improves colors. A second difficulty is that, contrary to the textures typically used in texture synthesis approaches, our images have non-homogeneous regions. This requires specific treatment to avoid repetitions, and careful processing of boundaries when applying gradient transfer.

Our fast texture synthesis algorithm proceeds in three steps: We initialize the synthesis process by growing color patches (Fig. 6(c)), perform parallel neighborhood matching using the Chamfer distance (Fig. 6(e)) and finally remove seams (Fig. 6(g)). We describe each step in detail below.

Similarly to [LH06] we enrich the neighborhoods with the distance of each pixel to the closest contour in the label map, computed using [Dan80]. This helps synthesis better capture the image appearance around boundaries in the label map. The distance map is used in neighborhood comparisons.

Step 1: Patch growth for initialization The purpose of this step is to grow patches on top of the synthesized guide (B in Fig. 6(d)). We randomly pick a pixel to grow in the synthesized guide and find its closest match in the original guide (A), using a weighted combination of the Chamfer distance between labels and the distance map. We use distance d:

$$d(p_A, p_B) = ||\mathcal{N}_{D_A}(p_A) - \mathcal{N}_{D_B}(p_B)||_2 + w\, C(p_A, p_B) \quad (1)$$

where $D_{A,B}$ are the distance maps, $\mathcal{N}_{\mathcal{D}_A}(p_A)$ the 5×5 neighborhood around p_A in D_A, and $C(p_A, p_B)$ the Chamfer distance between neighborhoods around p_A and p_B in the label maps A and B, in image space. w is a weight enforcing the respect of the guide, dependant of the neighbor size. We set $w = 60$ to largely enforce silhouettes to be respected, thus reducing flickering of silhouettes during camera motion. We use the best match as a seed to perform a flood fill in both A and B which stops either at already covered pixels or when d is larger than a given threshold. We use a threshold of 25% more than the distance of the best match to the seed + 5, to avoid very small incoherent patches at boundaries. This gives us a patch around the uncovered pixel in B. Each patch defines a mapping between pixels in B and pixels in the example image (color and labels). This process iterates until all pixels in B are covered.

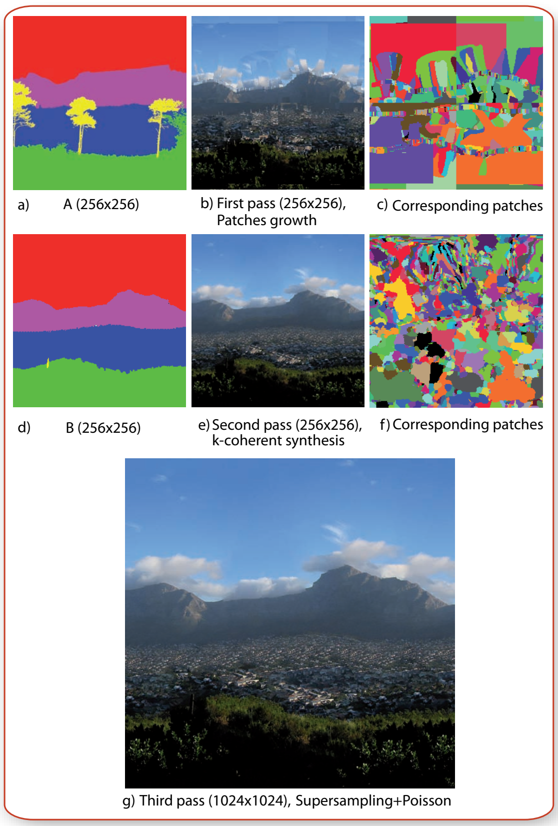
a) A (256x256)
b) First pass (256x256), Patches growth
c) Corresponding patches
d) B (256x256)
e) Second pass (256x256), k-coherent synthesis
f) Corresponding patches
g) Third pass (1024x1024), Supersampling+Poisson

Figure 6: *Our parallel three-step guided synthesis pipeline. Image A represents the labels in the input photo, while B are the synthesized labels from the proxy guide. The input photograph can be seen in Fig.7 of the additional material.*

The result is a set of patches as shown color-coded in Fig. 6(d). These patches define an image correct in terms of labels, but with many artifacts in the color channels (Fig. 6(c)). This first step is performed at an intermediate resolution, typically 256x256.

Step 2: Pixel-Based Guided Synthesis The second step performs guided synthesis, similar in spirit to [HJO*01] but using a parallel algorithm. We synthesize colors (Fig. 6(e)) using k-coherent synthesis, following [LH05]. Similarly to guidance synthesis, we use a voting stack for the Chamfer distances to perform multi-scale synthesis.

The distance metric is the same as in Eq. 1, augmented with the RGB color channels:

$$\begin{aligned} d(p_S, p_E) = \quad & ||\mathcal{N}_{F_S}(p_S) - \mathcal{N}_{F_E}(p_E)||_2 \\ & + w\, C(p_S, p_E) \end{aligned} \qquad (2)$$

where p_S is the pixel being synthesized and p_E the candidate in the example image, $F_{S,E}$ the feature vector containing the distance map and RGB pixel values and $\mathcal{N}$ are 5×5 neighborhoods. Synthesis starts at a coarse resolution of 32x32.

As mentioned above, the non-homogeneous nature of our images can result in repetitions. To overcome this limitation we reject candidates which are already present in a 9x9 neighborhood around the current pixel in the image being synthesized. To further limit repetitions, we also reject the candidate if any of its neighbors is present in a radius of 2^{l-1}, where l is the current synthesis level. We use 12 correction subpasses at each level of the pyramid, except the highest resolution where only 6 are performed for efficiency.

Step 3: Gradient transfer In the first two steps, we synthesize images at a resolution of 256x256 for efficiency. In the final pass we perform a synthesis magnification to the resolution of the input image (typically 1024x1024) by upsampling patches without correction passes. We then perform a final Poisson synthesis step to attenuate remaining artifacts.

The non-homogeneous nature of our input images can result in visible seams after texture synthesis. This may happen if a global gradient is present in the example image, for instance in the sky. In an approach inspired by [ADA*04] we transfer gradients instead of colors by solving a Poisson problem. We magnify the synthesized guide (Sec. 4.2) and ensure that boundaries between regions of different labels are left unchanged by using them as Dirichlet boundary conditions. We thus solve independently for each RGB channel:

$$\begin{aligned} \Delta u &= \Delta f \circ s && \text{in } \Omega \\ \nabla u &= \vec{0} && \text{in } \Gamma_1 \\ u &= f && \text{in } \Gamma_2 \end{aligned}$$

where u is the final image we solve for, f the input image, s the mapping giving the synthesized pixels location in the original image, Γ_1 the boundary between patches, Γ_2 the boundary between different IDs, and Ω all remaining pixels.

6. Temporal coherence

Full temporal coherence with camera motion is difficult to achieve using our approach, as for all texture synthesis based algorithms. As a first solution, we propose a reprojection-based method, which provides local temporal coherence, and limits the cost of generating new frames.

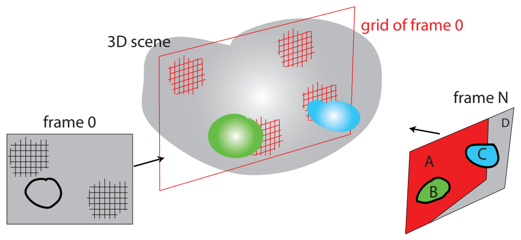

Figure 7: *The four categories of pixels: (A) Unoccluded projections of the grid; (B) Occluded by the same proxy as in frame 0; (C) Occluded by a new proxy; (D) Disoccluded. In A and B, we can reuse the depth synthesized in frame 0.*

To generate the current frame N, we use two sources of information for reprojection: the initial frame (frame 0), to provide a detailed guide and extended depth, and frame $N-1$ to reproject colors from the previous camera position.

In contrast to previous methods [WDP99,SW08], who reproject 3D points, we do not have full 3D information everywhere. The first step when moving the camera is to create consistent depth, reusing as much information as possible from the synthesized initial frame 0. In a second step, we reproject frame $N-1$ to the current view using the consistent depth, and decide which pixels need resynthesis. The third and last step is a modification of our texture synthesis algorithm which is run on pixels flagged for resynthesis. We now describe each step in more detail.

6.1. Creating Consistent Depth

We first create a mesh, which consists of a grid with each pixel of the initial frame as a separate small quad. We disconnect the mesh along a seam at label boundaries or when a large depth gradient is detected. . Mesh elements can thus overlap in depth, and thus occlude each other appropriately. Each node has coordinates (x', y', d) where x', y' are the coordinates of pixel (x, y) and d is the depth generated by the synthesis step in the first camera frame. We project this grid into the screen space coordinate system of current frame N. Note that the proxy is also present in the scene, and can occlude the grid (see Fig. 7).

After this operation, the current frame N contains four different kinds of pixels, as shown in Fig. 7. First, there are pixels not covered by the grid (grey area D) and pixels covered by the grid, which are are either of three cases: unoccluded by the proxy (red area A), occluded by the proxy and that were previously occluded (green area B), occluded by the proxy and that were previously not occluded (blue area C). In categories A and B, we use information from the grid which contain relevant details for depth and label. In categories C and D we use the proxy again to provide the missing depth and label values.

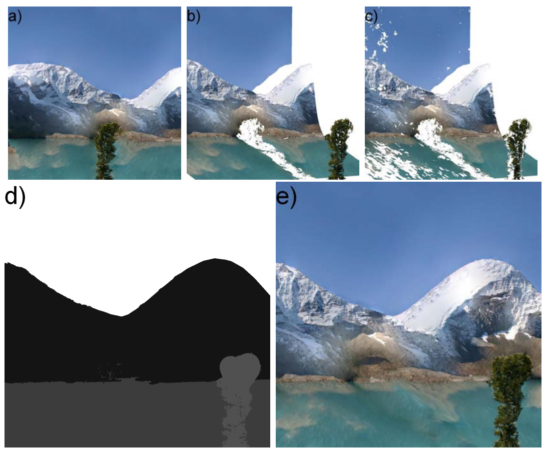

Figure 8: *Reprojection pipeline. (a) Initial frame. (b) Advected frame. (c) Distortion measure with noise. (d) new IDs. (e) new view. Regions to be resynthesized are shown in white.*

Each pixel now has a valid depth and label. Next, we use the depth to perform color reprojection.

6.2. Reprojecting Color and Distortion Computation

We project the 3D coordinates of a given pixel in the current frame (N) to the camera position of the previous frame ($N-1$); we copy this color to the pixel of the current image. Evidently, there will be no values available for pixels which became disoccluded in the current view; these pixels are flagged for resynthesis.

We next compute pixel distortion, which will determine when reprojected pixels are no longer reliable. We compute the determinant of the Jacobian J of the reprojected pixel coordinates used to access frame $N-1$, with respect to the current x, y screen coordinates [Hec89]. We define distortion as $D = ||det(J)| - 1|$ which we accumulate across frames. If D becomes greater than a given threshold, D is reset to 0 and the pixel is flagged for resynthesis. We perturb D with random noise to avoid a coherent "wave" of pixels requiring resynthesis. This is similar to the reprojection method of [SW08] which uses a noise function to render different fragments of consecutive LODs to avoid blending artefacts.

In addition to distorted and disoccluded pixels, pixels previously visible and now occluded by objects (category C in Fig. 7) are also tagged for resynthesis.

We now have a new frame in which a subset of pixels are flagged for resynthesis, i.e., disoccluded pixels, pixels newly occluded and pixels with excessive distortion. Numerical imprecision may result in isolated pixels being incorrectly flagged for resynthesis; these produce noticeable flickering. We remove these isolated pixels in 5x5 neighborhoods. We dilate remaining flagged regions to allow more freedom for the texture synthesis to correct them.

6.3. Final Synthesis

In areas flagged for resynthesis due to excessive distortion, texture synthesis is initialized with the existing distorted pixel colors. In the remaining areas flagged for resynthesis, floodfill is used (§5). The rest of the texture synthesis process remains the same, except the Poisson solve being restricted to the newly synthesized areas with Dirichlet boundary conditions in order to avoid global changes in the image.

Fig. 8 shows the entire process: the reprojection, the effect of distortion with noise, the new guide and the final view.

7. Implementation, Results and Discussion

Poisson solve is performed with a GPU multigrid method similar to [MP08]. GPU jump flooding [RT06] is used for depth extension, with a base metric similar to the distance transform but including an additional constant term to favour seeds with the same label as the area to be filled in. We use a 2+JFA+1 scheme [RT06] which gives accurate results in our test scenes with negligible overhead. The reprojection step as well as the dilation are also performed on the GPU.

We allow the integration of 3D CG elements, and interactive editing of parameters. The details of these features are presented in the additional material.

7.1. Results

We demonstrate the method on a number of example scenes. The results are best viewed in conjunction with the video[†]. Fig. 9 shows three examples; a further 11 examples are shown in the additional material[†]. Two synthesized views are shown for each scene exhibiting temporal coherence. The scenes appear rich in geometric detail and do not betray the coarse nature of the underlying proxy. In the additional material, we present results for non-photorealistic rendering and time-lapse based dynamic lighting.

7.2. Quality and Performance

Our method represents a significant departure from traditional modelling/texturing/rendering paradigms. As a consequence, many technical problems were encountered and solved to achieve our goal of proxy-based, texture-synthesis rendering. However, there are still some artifacts in terms of visual quality, and some issues with performance. Some of these are simply related to our implementation while others require future research.

Quality: There are two sources for the remaining flickering observed during camera motion. Popping in the interior of

[†] `https://www-sop.inria.fr/reves/Basilic/2010/BVLD10a/`

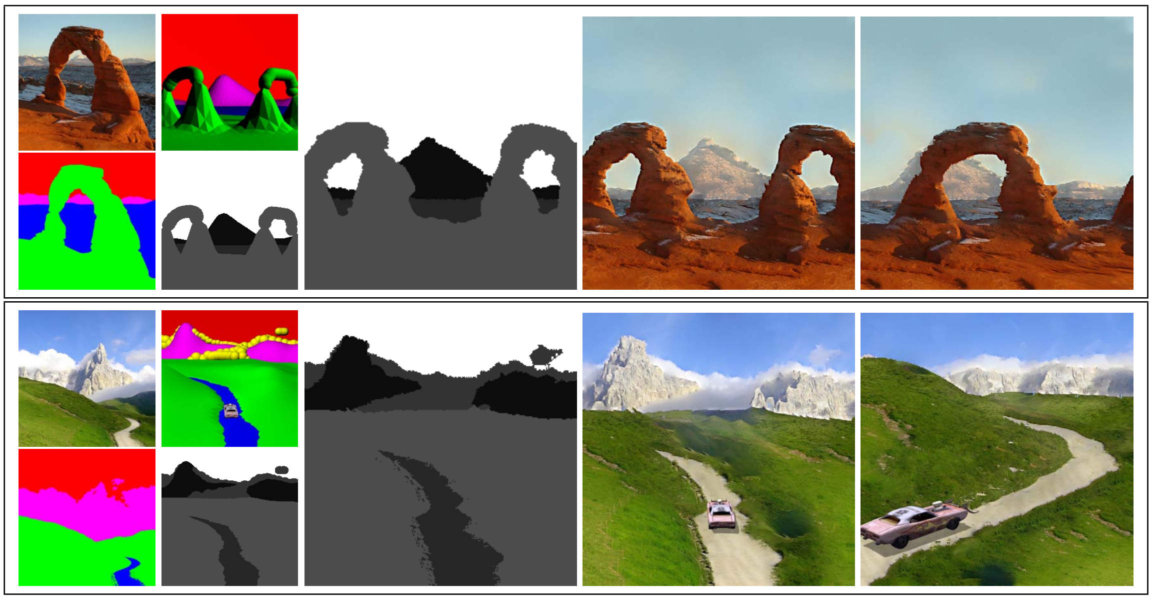

Figure 9: *Results, including integration of 3D objects. Further results provided in the supplemental material and video.*

the textures is due to the high rate of distortion, and the random noise modulating the distortion metric to avoid "waving" patterns (§6.2). Flickering around silhouettes is due to the dilation of synthesized regions (§6.2), giving leeway to the synthesis but resulting in the appearance of 2-3 pixel wide resynthesized regions.

The detailed boundaries synthesized in the first frame are only partially reused in subsequent frames: no new details are added to disoccluded regions. For large camera rotations, boundary detail is lost (see additional material).

The guide synthesis can lead to unnatural results such as trees with two trunks, given that the method does not model the abstract constraints needed to prevent this, due to the lack of semantic information. Artifacts can appear along structured border regions such as the horizon line when placed at a different angle in the proxy guide and original image. To ameliorate this problem, we specify that ID boundaries along horizon lines should remain unaltered so as to preserve the original structured boundary during the guidance synthesis process. When the refinement process corrects these regions, the original IDs are left unchanged and uncorrected.

We are currently limited to a single source image. An exciting direction for future work is to investigate the use of texture categories from several source images. Particular texture labels could be declared as being semantically equivalent, for example. This would provide much wider classes of texture-synthesis shading, and indexing based on other attributes such as normals or lighting conditions.

Performance: The first frame is fully synthesized and detailed silhouettes are created, thus taking longer to render. A full 1024×1024 image needs approximately $7s$ to render including $2s$ for the guide synthesis, on a QuadCore 2.4GHz with a GeForce 9800 GT. As a comparison, the guide synthesis-based approach in [RCOL09] requires 30 minutes per layer and an additional 5 hours to run Image Analogies. For all subsequent frames, only an update of the reprojected scene is needed, reducing the rendering cost. We summarize the performance of our implementation in Table 1. On a higher-end 8-core machine and a FX5800 graphics card, we update frames in approximately 1.2 seconds.

We accelerate Chamfer computation with a partitioning of space in hyperspheres. We compared the speed of this acceleration structure with the optimized ANN library [MA10] which uses an L2 norm on RGB neighborhoods and a *kd*-tree, on 7 scenes. Our method is 20% slower on average, although the ANN library performance varied significantly across scenes. In particular, our method varied from 8 times slower to 3 times faster. If visual artifacts such as those of Fig. 3 are acceptable for a given application, speedup can thus be expected by using the L_2 distance. Our approach will still work in such a setting, albeit with lower quality. We prefer to use the Chamfer distance since it is a principled solution for labels, and offers better quality.

Further discussion of performace, comparisons with a trained modeller and with Image Analogies are presented in the additional material.

8. Conclusions

We have introduced a proxy-guided texture-synthesis rendering technique, allowing the creation of visually rich images from a simple 3D proxy geometry. Three core issues are addressed to make this new class of depiction algorithm pos-

Op.	R&D	Feature	FF	Cor.	Poisson
Time (ms)	405	156	73	296	292
#threads	GPU	GPU	8	8	GPU

Table 1: *Speed on 8-cores and FX5800 GPU. Reprojection and dilation step (R&D), ID maps, distance transform, and voting pyramids (Feature), floodfill patch initialization (FF), corrections refining patches (Cor.) and Poisson synthesis.*

sible. First, guidance synthesis adds detail to texture boundaries, which would otherwise betray the simple nature of the underlying proxy geometry. We introduce the use of the Chamfer distance metric to deal in a principled way with the discrete semantics of labels. Second, we introduced a new patch-based initialization which enables a fast parallel algorithm to perform guided texture synthesis on the kind of input photographs we target. In addition we address limitations of previous methods related to repetitions and gradient transfer for this class of images. Third, we introduce a solution for local temporal coherence, thus enabling camera motion using a reprojection technique. We demonstrate the method on 13 different scenes, showing the power of our approach, for applications such as 3D previsualization.

Acknowledgments

We thank the reviewers for their detailed comments. We thank Fredo Durand, Kavita Bala and Olga Sorkine for a number of useful comments and suggestions.

References

[ADA*04] AGARWALA A., DONTCHEVA M., AGRAWALA M., DRUCKER S., COLBURN A., CURLESS B., SALESIN D., COHEN M.: Interactive digital photomontage. *ACM Trans. Graph. 23*, 3 (2004), 294–302. 6

[AH95] ADELSON S. J., HODGES L. F.: Generating exact ray-traced animation frames by reprojection. *IEEE Computer Graphics and Applications 15*, 3 (May 1995), 43–52. 3

[Ash01] ASHIKHMIN M.: Synthesizing natural textures. In *ACM I3D '01: Proceedings of the 2001 Symposium on Interactive 3D Graphics* (2001), pp. 217–226. 2

[Bor88] BORGEFORS G.: Hierarchical chamfer matching: A parametric edge matching algorithm. *IEEE Trans. on pattern analysis and machine intelligence 10*, 6 (1988), 849–865. 3

[BTBW77] BARROW H. G., TENENBAUM J. M., BOLLES R. C., WOLF H. C.: Parametric correspondence and chamfer matching: Two new techniques for image matching. In *Proc. 5th Int. Joint Conf. Artificial Intelligence* (1977), pp. 659–663. 3

[CW93] CHEN S. E., WILLIAMS L.: View interpolation for image synthesis. In *Computer Graphics (SIGGRAPH '93 Proceedings)* (Aug. 1993), Kajiya J. T., (Ed.), vol. 27, pp. 279–288. 3

[Dan80] DANIELSSON P. E.: Euclidean distance mapping. *Computer Graphics and Image Processing 14* (1980), 227–248. 5

[EF01] EFROS A. A., FREEMAN W. T.: Image quilting for texture synthesis and transfer. In *Proc. ACM SIGGRAPH '01* (2001), pp. 341–346. 2

[GDB08] GARCIA V., DEBREUVE E., BARLAUD M.: Fast k nearest neighbor search using gpu. In *CVPR Workshop on Computer Vision on GPU* (2008), pp. 1–7. 11

[Hec89] HECKBERT P.: *Fundamentals of Texture Mapping and Image Warping*. M.sc. thesis (TR. UCB/CSD 89/516), Univ. of California, Berkeley, 1989. 7

[HJO*01] HERTZMANN A., JACOBS C. E., OLIVER N., CURLESS B., SALESIN D. H.: Image analogies. In *Proceedings of ACM SIGGRAPH 2001* (Aug. 2001), Computer Graphics Proceedings, Annual Conference Series, pp. 327–340. 2, 4, 5, 10

[HRRG08] HAN C., RISSER E., RAMAMOORTHI R., GRINSPUN E.: Multiscale texture synthesis. *ACM Trans. Graph. (Proc. SIGGRAPH) 27*, 3 (2008), 51. 3

[HS85] HOCHBAUM D., SHMOYS D.: A best possible heuristic for the k-center problem. *Mathematics of Operations Research 10*, 2 (1985), 180–184. 4

[KNC*08] KOPF J., NEUBERT B., CHEN B., COHEN M. F., COHEN-OR D., DEUSSEN O., UYTTENDAELE M., LISCHINSKI D.: Deep photo: model-based photograph enhancement and viewing. *ACM Trans. on Graph. 27*, 5 (2008), 116. 2

[LH05] LEFEBVRE S., HOPPE H.: Parallel controllable texture synthesis. *ACM Trans. Graph. (Proc. SIGGRAPH) 24*, 3 (2005), 777–786. 2, 3, 4, 5, 6

[LH06] LEFEBVRE S., HOPPE H.: Appearance-space texture synthesis. *ACM Trans. Graph. (Proc. SIGGRAPH) 25*, 3 (2006), 541–548. 5

[MA10] MOUNT D., ARYA S.: Ann: A library for approximate nearest neighbor searching, 2010. 8

[MP08] MCCANN J., POLLARD N. S.: Real-time gradient-domain painting. *ACM Trans. on Graphics (SIGGRAPH 2008) 27*, 3 (Aug. 2008). 7, 11

[PFA*09] PFISTER H., FREEMAN W. T., AVIDAN S., DALE K., JOHNSON M. K., MATUSIK W.: *CG2Real: Improving the Realism of Computer Generated Images using a Large Collection of Photographs*. TR Report MIT-CSAIL-TR-2009-034, 2009. 3, 11

[RCOL09] ROSENBERGER A., COHEN-OR D., LISCHINSKI D.: Layered shape synthesis: automatic generation of control maps for non-stationary textures. *ACM Trans. Graph. (Proc SIGGRAPH Asia) 28*, 5 (2009). 2, 8

[RKB04] ROTHER C., KOLMOGOROV V., BLAKE A.: "grabcut": interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph. 23*, 3 (August 2004), 309–314. 3

[RLC*06] RITTER L., LI W., CURLESS B., AGRAWALA M., SALESIN D.: Painting with texture. In *17th EG Workshop on Rendering* (June 2006), pp. 371–376. 2

[RT06] RONG G., TAN T.-S.: Jump flooding in gpu with applications to voronoi diagram and distance transform. In *I3D '06: Proc. Symp. on Interactive 3D Graphics and Games* (2006), ACM, pp. 109–116. 7

[SW08] SCHERZER D., WIMMER M.: Frame sequential interpolation for discrete level-of-detail rendering. *Computer Graphics Forum (EGSR 2008) 27*, 4 (June 2008), 1175–1181. 3, 6, 7

[TZL*02] TONG X., ZHANG J., LIU L., WANG X., GUO B., SHUM H.-Y.: Synthesis of bidirectional texture functions on arbitrary surfaces. In *Proc. SIGGRAPH* (2002), pp. 665–672. 3

[WDP99] WALTER B., DRETTAKIS G., PARKER S.: Interactive rendering using the render cache. In *10th EG Workshop on Rendering* (June 1999), pp. 19–30. 3, 6

[WLKT09] WEI L.-Y., LEFEBVRE S., KWATRA V., TURK G.: State of the art in example-based texture synthesis. In *Eurographics STAR Report* (2009). 2

[ZZV*03] ZHANG J., ZHOU K., VELHO L., GUO B., SHUM H.-Y.: Synthesis of progressively-variant textures on arbitrary surfaces. *ACM Trans. Graph. 22*, 3 (2003), 295–302. 2