Assisted Texture Assignment

Matthäus G. Chajdas[1,3] Sylvain Lefebvre[1,2] Marc Stamminger[3]
[1]REVES / INRIA Sophia-Antipolis [2]ALICE / INRIA Nancy [3]University of Erlangen

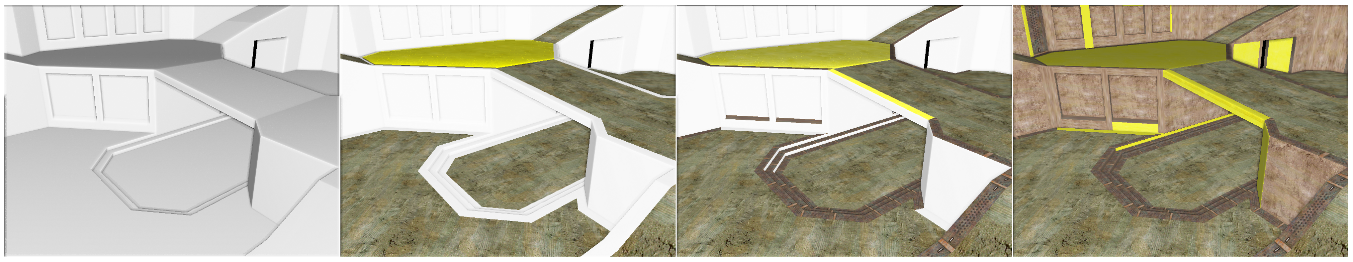

Figure 1: *We present a novel algorithm to assist a user assign textures on the surfaces of virtual environments.* (a) *Initial scene.* (b) *The user selects a texture for a floor (hilighted in yellow), all floors are automatically assigned by our system.* (c) *The user selects a texture for a border, all borders are automatically assigned.* (d) *Result after 10 selections. Surfaces touched by the user are highlighted in yellow. Texturing information is propagated throughout the entire scene at once.* Scenes from Quake 4, © Id Software.

Abstract

Virtual environments are typically textured by manually choosing an image to apply on each surface. This implies browsing through large sets of generic textures for each and every surface in the scene.

We propose to facilitate this long and tedious process. Our algorithm assists the user while he assigns textures to surfaces. Each time an image is chosen for a surface, our algorithm *propagates* this information throughout the entire environment.

Our approach is based on a new surface similarity measure. We exploit this measure in an algorithm ranking all possible textures for a given surface. Hence, we do not simply assign a texture to the surface but also propose an ordered list of choices for the user. In the unavoidable event of an ambiguous choice, the user can quickly make a decision and select the best texture. Our algorithm is fast enough to allow for interactive feedback. Applications range from assisted interactive texturing to fully automatic initial texturing solutions.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture; I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques

Keywords: Texture mapping, automatic texture assignment

1 Introduction

Virtual environments are typically textured by selecting materials for each surface within a database of pre–designed textures. Such textures encompass images of wood, concrete, grass, brick walls, but also nearly–flat architectural details such as windows, door frames, signs and control panels. Typically the geometry of the scene is first designed, the textures being later manually chosen.

The textures used for environments are very different from the textures applied onto specific objects or characters. They are not specialized to a particular geometry instance and are typically applied to surfaces through simple planar mappings. Most often, environmental textures are created to be cyclic or are procedurally generated allowing reuse by surfaces of various sizes. These specificities let texture artists pre–design databases of images for a given theme, later used by level–designers to decorate their creations.

Textures play an essential role in defining the look and feel of an environment. Unfortunately the task of selecting one texture for each surface is also very tedious, requiring the artist to click on many surfaces and to browse through large sets of images. This fact is worsened by the increasing use of automated processes to produce complex scenes. For instance details are generated within the modeling tool by physically simulating the behavior of many objects: Falling rocks, smashed walls, deforming or breakable objects. After this process, the artist is left with the tedious task of choosing appropriate textures for each object, for instance distinguishing rocks at the bottom or top of a pile.

Our goal is to simplify this task. We propose an approach to help the selection of the most appropriate texture for a surface, learning from already textured geometry. For instance, using our tool the artist starts by selecting textures for a few surfaces and this information is automatically propagated throughout the scene (see Figure 1). In another example, the user selects in just two clicks different textures for rocks in a pile (see Figure 7). We thus *amplify* user input, avoiding most of the repetitive and tedious work. Our algorithm can also transfer textures from one geometry to another, letting the user interactively improve the result with few interactions (see Figure 8). Throughout this work we assume that the scene comes pre–segmented into surfaces, and that destination surfaces already have proper texture coordinates (only used for display).

Assigning a texture to a particular surface is a seemingly easy to solve problem for a human user. We easily identify floors, walls, stairs from their shape and *purpose*. Nevertheless, ambiguities remain and the 'right' choice is often a matter of individual appreciation. Doing the whole process automatically is thus extremely challenging: We do not want to require semantic information about the scene – adding it would be as tedious as texturing – and we often have very little example data to work from. The shapes of polygons themselves are often not relevant for texturing purposes: Very different polygons use the same texture. The naive approach of finding a texture by matching polygons is thus bound to fail. Finally, since only the user can resolve ambiguities in the texture assignments – such as choosing among various floor textures – we can only hope to provide an efficient selection method.

Faced with these challenges we cannot reasonably hope for a fully automatic system providing perfect results. Instead, we seek to design an algorithm providing a good initial guess to work from, enabling texture artists to gain a significant amount of time and to focus on the important choices. In particular, rather than selecting a single texture for a surface our system ranks all possible choices, letting the user quickly select another texture among the good choices (please refer to the accompanying video).

Our main contributions are:

- A new surface similarity measure specifically designed for matching surfaces in a texturing context. The key idea of our measure is to check whether a surface can be *described* by the features of another. This abstracts the actual shape of the polygon while retaining information relevant for texturing.
- A ranking mechanism scoring each texture according to whether it is a likely candidate for a given surface. It relies on example surfaces and surface similarity. Neighboring surfaces with already assigned textures are used to reduce ambiguities.

We demonstrate these contributions in a prototype tool to assist manual texture assignment in large virtual environments. The following sections detail our approach and results.

2 Previous work

To the best of our knowledge there has not been previous work addressing precisely our problem. Nevertheless, several works investigate related issues and provide valuable insights on how to approach the texture assignment problem.

Guided texture synthesis and texture transfer Guided texture synthesis [Ashikhmin 2001; Hertzmann et al. 2001] produces a new texture following a guidance field. These approaches have been successfully used to transfer the texture of an object onto another using a 3D–scan as example [Mertens et al. 2006; Lu et al. 2007]. The goal is related to ours, however these works focus on smooth curved objects such as sculptures and deal essentially with stochastic textures. In contrast, we deal with complete environments and select textures for surfaces rather than synthesizing new ones. Nevertheless, a key insight is that local geometric properties such as curvature, accessibility and orientation are strongly correlated with surface appearance.

Shape matching and instancing A key component of our approach is to identify similar surfaces. Most shape matching techniques are designed to quickly search object databases by computing global shape descriptors [Tangelder 2004]. Partial shape matching [Gal and Cohen-Or 2006] is closer to our needs since it focuses on identifying matching sub–parts within meshes. Similarly, the work of [Martinet 2007] groups connected geometry together and automatically finds similar instances. The approach of [Pauly et al. 2008] detects repeating structures in a scene. While this could be very useful to pre–cluster similar objects, these approaches are difficult to apply on the surfaces we manipulate: Many are flat or exhibit identical sharp corners, which would lead to a large number of ambiguities. A key inspiration from these approaches, however, is the use of local geometric descriptors randomly spread within the scene to avoid dependency to tessellation. Finally, the work of [Anderson et al. 2000] identifies geometric structures in a crude block based modeling of a scene. It relies on logical predicates identifying symbols in the block layout to produce detailed textured geometry.

Machine learning We considered using machine learning techniques [Bishop 2006] to label a surface knowing the examples. However, per–surface descriptors are not sufficient for texturing purposes: The particular shape or size of a polygon is often not strongly related to the texture it is using. In addition, the ambiguities between textures exclude most direct clustering approaches. Finally, machine learning does not adapt well to the progressive texturing approach where surfaces are added one by one.

Automatic texturing of architectural scenes Architectural scenes are typically described by simple geometries and complex texturing. The work of [Legakis et al. 2001] is dedicated to the generation of complex cellular textures – such as brick walls – on architectural models. The choice of textures is however left to the user. Our work is complementary since it would help the user annotate the input. The work of [Cabral et al. 2009] creates new environments by sticking together pieces of existing buildings. The textures already in place are reused and automatically resized to adapt to geometric changes.

3 Our approach

The input to our approach is a scene composed of a set of *surfaces*. Each surface is made of polygons, not necessarily planar, meant to share a *continuous region* of a same texture. This is for instance a floor polygon, the cylindrical side of a barrel, or the steps of a stair. Some surfaces – example surfaces – are already assigned a texture while others – target surfaces – are not yet determined.

Our approach assigns textures by testing a target surface against all possible textures of the example. This results in an ordering of the textures by decreasing score of being a good choice. Some textures may be discarded as clearly bad choices and will not be ranked. As an initial guess we simply assign the first texture to the target. The user quickly resolves any wrong choice by exploring the sorted textures.

The score of each texture comes from two sources: First, a surface matching step computes how well the target surface matches against all the example surfaces using the texture (see subsection 3.1). Second, the already textured neighbors are used to influence the choice of texture for the surface (see subsection 3.2). In the end, both are merged to compute a final ranking. On scenes made of a few tens of thousands surfaces both steps are fast enough to be performed interactively while the user is assigning textures: We can propagate the texture assignment as the user is choosing textures for surfaces in a new scene. Applications are described section 4.

3.1 Texture ranking with surface similarity

Given a target surface, we need to score each texture according to whether it is a good candidate. The only information available comes from the surfaces already using this texture.

Our approach abstracts away from the notion of surfaces by introducing local descriptors: The scene is randomly sampled with uniform density by *probes* describing local properties of the underlying geometry. The probes are completely independent, and even on a same surface they are likely to contain different values. Eventually, each surface is represented by a group of probes and the actual geometry is discarded.

We use the probes to define our surface similarity. Our key idea is to compare two surfaces by counting how many probes of the first surface match with at least one probe of the second surface, and vice–versa. In other words, we verify that the second surface *describes well* the first surface, and that the opposite is also true. This notion of 'being good descriptors of each others' has also recently been introduced for image summarization [Simakov and Irani 2008].

The following sections describe these ideas in more detail.

3.1.1 Surface probing

We generate the probes by uniformly and randomly sampling the scene geometry. In a pre-processing step, we compute a voxelization of the scene, independent of the tessellation. For each surface inside a voxel we create a sample point. This is the probe location. The result of the probe sampling can be seen Figure 2.

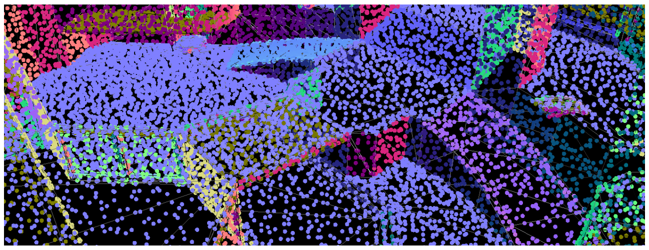

Figure 2: *Scene covered by probes. Colors outline the geometry.*

The probes capture a number of local geometric properties. These properties must be relevant in terms of finding an appropriate texture for a surface: i.e. they must help distinguish between surfaces having different roles (floor, wall, door, etc.). Some of these properties depend only on the surface shape, while others depend on the context around the surface. We use the following properties:

- *Local anisotropy* captures whether the probe is on a thin strip or close to an extremity of the surface.
- *Curvature* captures the local non–planarity of the surface.
- *Distance to edge* captures how far we are from the surface closest border. This gives some indication of the surface size.
- *Accessibility* captures whether the surface is in a concavity or a cavity of the scene.
- *Local orientation* captures the angle of the surface with respect to the scene.

These properties are visually illustrated Figure 3. Accessibility, orientation and curvature were previously identified as good texture discriminants [Mertens et al. 2006], while the other properties are well suited for architectural settings. We describe their computation in more details below. Anisotropy, curvature and accessibility require a scene–dependent scale factor.

Local anisotropy We define anisotropy by considering a disc around the probe with a fixed, scene-dependent radius. The radius must be chosen to capture thin features of the scene. We uniformly sample the disc and test all pairs of opposite points to check whether they fall outside the surface supporting the probe (see Figure 4). If both fall outside the anisotropy level is increased by one. This lets us efficiently characterize the local shape of a surface.

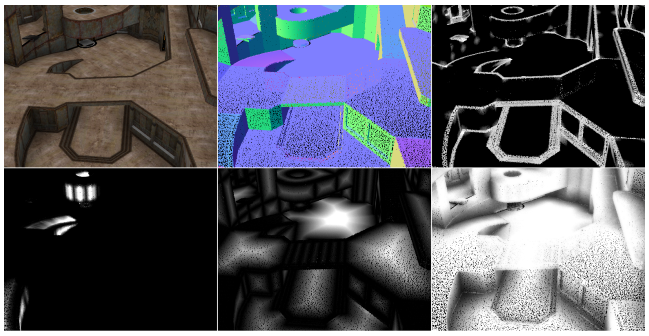

Figure 3: From left to right, top to bottom*: Example scene, local orientation (normals), anisotropy, curvature, distance to edge and accessibility.*

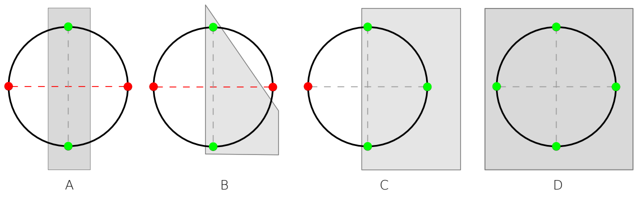

Figure 4: *Anisotropy estimation using four samples.* Cases A and B: *We increase the anisotropy value since two opposite points fall outside the surface.* Cases C and D: *These cases are isotropic.*

Local curvature estimate We extend the anisotropy computation to also estimate curvature, inspired by estimation schemes such as solid angle curvature [Mertens et al. 2006].

For each sample point on the circle, we compute the distance to the closest point on the geometry (see Figure 5). This comes at no additional cost: During anisotropy computation we simply retain distances to hit–points. We sum up the (unsigned) distances to measure how much the surface deviates from a plane in the local neighborhood. We typically use 32 samples around the circle.

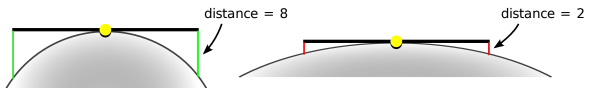

Figure 5: *Curvature estimation (side view). We consider a circle in the tangent plane around the probe and compute the distance to the surface at a few sample points.*

Distance to edge For fast approximation of the distance to edge we rely on the voxelization. For each surface, we compute the set of *edge voxels*: Voxels either containing another surface or having less than four defined neighbors in the 6–voxels neighborhood. (The 6–voxels neighborhood is made of the direct neighbors along the x, y and z axis). For each probe, we compute the distance to the closest edge voxel, and keep the minimum value.

Accessibility We determine an accessibility factor by tracing a number of rays in the hemisphere around each probe. Accessibility is checked within a user specified distance. Each ray hitting an object before the distance threshold increments the occlusion value. The final occlusion is the ratio of hitting rays to the number of total rays traced. We typically use 64 rays per probe. The noise resulting from the low number of rays does not impair our probe matching thanks to thresholding (see subsubsection 3.1.2).

Local orientation We add the smoothed normal vector to the probe attributes in order to capture the local orientation of the surface. This implies that our probes are not invariant under rotation.

While crucial to distinguish floors from walls, this could be relaxed for rotations around the up direction.

3.1.2 Bi–directional probe matching

We now have covered all surfaces in the scene with probes. Our next goal is to use the probes in order to measure how similar two surfaces are for texturing purposes. We will see afterwards how this surface similarity measure is used for texture assignment.

Let $\mathcal{P}$ be the set of all probes in a scene, $P_S \subset \mathcal{P}$ be the set of all probes for a given surface S and $P_c \subset \mathcal{P}$ the set of probes for all surfaces using texture c. We note $(A_1^p, A_2^p, \ldots, A_n^p)$ the attributes A_i^p of a probe $p \in \mathcal{P}$.

Our similarity measure is based on the key idea of being able to *describe* a surface with another. That is, for any given piece of surface A, it must be possible to find a similar piece in surface B. In our case, the 'pieces' being compared are the probes. This abstracts the size and layout of two surfaces, allowing them to be quite different *as long as the same features are found in both*. This measure enforces the property that a surface must be similar to itself.

We first define a binary matching function as:

$$\delta(x, y, t) = \begin{cases} 1 & \text{if } |x - y| \leq t \\ 0 & \text{else} \end{cases}$$

where x, y are values to be matched and t a threshold. We consider that two probes $p = (A_1^p, A_2^p, \ldots), q = (A_1^q, A_2^q, \ldots)$ are matching if all attributes are matching under a per–attribute threshold:

$$Match(p, q) = \begin{cases} 1 & \text{if forall } i,\ \delta(A_i^p, A_i^q, thres_i) = 1 \\ 0 & \text{otherwise} \end{cases}$$

The per–attribute thresholds $thres_i$ are computed automatically as discussed below.

We define the asymmetric similarity between surfaces S, T as:

$$SimilarTo(S, T) = \frac{1}{|P_S|} \sum_{p \in P_S} \begin{cases} 1 & \text{if } \exists q \in P_T \text{ st. } Match(p, q) = 1 \\ 0 & \text{otherwise} \end{cases}$$

This counts the number of probes in S finding a match in T. This similarity returns 0 if S is very dissimilar from T and 1 if they match well. Finally, we obtain the bi–directional similarity between surfaces as:

$$Similarity(S, T) = SimilarTo(S, T) \cdot SimilarTo(T, S)$$

We use a binary matching between the probes rather than a continuous distance for two reasons. First, this avoids having to scale the attributes with respect to each others: When computing a distance, it is not obvious whether orientation or accessibility should matter more than the distance to edge. Second, the binary matching allows for significant performance optimization, since we can exit the comparison when a first match is found. This is key in achieving interactive feedback.

Threshold for probe matching When comparing two surfaces we match attributes up to a threshold $thres_i$. This threshold is critical in order to achieve proper results. A tight threshold would only allow for perfect matches, while a very large threshold would over–generalize by matching very dissimilar probes. We define our threshold to allow slightly different attributes to match, relying on standard deviation for automatic adjustment. In addition, it takes into account the variability within the surfaces using a same texture: If two very different surfaces use a same texture we allow for a larger threshold, accounting for the fact that some attributes are less discriminant for this texture.

More precisely, given a surface S using a texture c we obtain $thres_i$ as follows. First, we compute the standard deviation of each probe attribute A_i over the entire set of probes $\mathcal{P}$. This results in vector $\text{STD}_{\mathcal{P}} = (\sigma_1, \sigma_2, \ldots, \sigma_n)$. Second, we compute the standard deviation of each probe attribute A_i over the set of probes P_c of the surfaces using texture c. This results in a second vector $\text{STD}_c = (\alpha_1, \alpha_2, \ldots, \alpha_n)$. For the threshold, we use $thres_i = max(\gamma\sigma_i, \tau\alpha_i)$, where γ and τ are user specified factors. The global threshold is controlled via γ, while τ controls the within–texture threshold. Typical values for γ and τ are 0.1 and 1.0. Increasing the values lets more surfaces fall within range of a texture, while decreasing them requires more precise matches and a larger number of example surfaces per texture. Please refer to section 4 and Figure 9 for more details.

3.1.3 Texture ranking with probes only

We now seek to score each texture according to whether they are good candidates for a given target surface. Our intuition is that a texture is a good candidate if the surfaces already using it are matching well with the target surface.

More precisely, if we note $\mathcal{S}_c$ the set of example surfaces using texture c, and T the target surface, the texture score is computed as:

$$Score_T(c) = \sum_{S \in \mathcal{S}_c} (Similarity(S, T))^2$$

Intuitively, we give more importance to well matching textures by squaring the similarity values, and give a higher rank to textures with many good matches.

We sort the textures according to their score. We refer to the set of ranked textures as the *probe candidate set*. If no surface matches the target across all textures, it is left unassigned (i.e. no good texture has been found).

Results with probe ranking only Texture assignment with probe ranking already performs well: Textures with high scores are often appropriate for the surface. However, on scenes with many ambiguous textures – such as several floor textures – the probes do not suffice to achieve satisfactory results. A key issue is the lack of consistency: Neighboring surfaces do not necessarily make the same choice in case of ambiguities. As explained in the next section, we need additional information to help reduce ambiguities.

3.2 Neighborhood–driven texture selection

Given a target surface we now have a set of candidate textures ranked using surface similarity. However, this information is not enough to obtain a consistent texturing of the scene: We need to take into account the already textured surfaces neighboring the target surface. For instance, some floor texture may only appear together with a specific wall texture. We need to capture these co–occurrences in our texture assignment algorithm. Ambiguities are reduced by this mechanism.

3.2.1 The texture graph

We start by capturing the neighboring relationship of the already textured surfaces. We consider as neighbors surfaces sharing at least one same voxel. We create a *texture graph*, in which each texture has a corresponding node. An edge is added in the graph between textures used by neighboring surfaces.

It is not enough, however, to simply record adjacency. In many cases, the texture of neighboring surfaces is determined by their spatial relationship. In our experiments we found that the length of the border between both surfaces is particularly relevant for the choice of texture. For instance a plinth surface will be neighboring other plinth polygons by a small edge length while it will neighbor wall polygons by larger, varying edge lengths. This also holds for vertical walls versus for instance door frames or window frames. Hence, we store along each graph edge the average and variance of the geometric border length between all surfaces using the two textures. We assume normal distribution of lengths. In practice we observed either a large variability of lengths, or very little variations around a single value.

3.2.2 Neighborhood influence

Our goal is now to compute a weight influencing the ranking obtained from the probes only. Our idea is to consider each already assigned neighboring surface, and see how likely it is to be the actual neighbor of each texture candidate.

To this end, we search in the texture graph to see if an edge exists between the candidate texture and the texture already assigned to the neighboring surface. If it does, we compute the probability that the neighboring relationship is indeed one observed in the example.

More precisely, noting $l_{T,N}$ the edge length between target surface T and one of its neighbors N, noting c the candidate texture and n the texture assigned to N, and finally noting $\gamma_{(c,n)}$ and $\sigma_{(c,n)}$ the average length and variance stored in the graph between textures c and n, we compute the probability of T using c next to N as:

$$P(T, c|N, n) = \frac{1}{\sigma_{(c,n)}\sqrt{2\pi}} exp^{-\frac{(l_{T,N} - \gamma_{(c,n)})^2}{2\sigma^2_{(c,n)}}}$$

The weight for candidate texture c from the neighborhood $\mathcal{N}_T$ of surface T is computed as:

$$w_{\mathcal{N}_T}(c) = \frac{1}{|\mathcal{N}_T|} \sum_{N \in \mathcal{N}_T} P(T, c|N, n)$$

Note that the neighborhood only contains surfaces with already assigned textures. Other neighbors are ignored.

In the end, we rank all texture candidates c for target surface T according to:

$$Score_T(c) \cdot w_{\mathcal{N}_T}(c)$$

The next section explains how this score is used to rank textures for each surface in the entire scene.

3.3 Global texture assignment

We have previously seen how to rank textures for a unique target surface T using both the probes and already textured neighbors. In this section we explain how to apply the texture ranking to all surfaces of a target scene.

Our algorithm iteratively improves the assignment for all surfaces. For maximum performance, we process all target surfaces in parallel within a same iteration. During the first iteration there are typically no textures assigned to any surface, and hence the texture score only takes the probes into account. During each subsequent iteration the score for a texture takes into account neighbors. This will improve the result by reducing ambiguities (see Figure 10). While there is no guarantee the result usually stabilizes after three or four iterations. Assigning textures to surfaces sequentially by decreasing probe scoring may lead to better result. However, this would be significantly slower than our current parallel scheme and would not allow for interactive feedback.

3.4 Adding the user in the loop

Our system provides initial guesses and lets the user add more information where required. The user typically performs three operations: Choosing a more appropriate texture within the set of candidates, confirming the choice made by the algorithm, or selecting a texture for a surface that could not be assigned. Please refer to the accompanying video for an illustration.

Choosing a better texture is done very quickly thanks to our ranking system. All ambiguous textures appear very early in the list of candidates, and the user does not have to browse long before finding a better choice. For instance to obtain the result in Figure 8 (bottom row) the user on average selected the texture at rank 2.2 – rank 1 being selected by default – out of 71 textures.

Confirming a choice lets the system add the target surface in the set of examples, reinforcing the knowledge about this particular texture. In fact it is a very similar operation than manually assigning a texture to a surface. In both cases the probes of the surfaces are added to the example database. The similarity between newly added surfaces and all other target surfaces is recomputed, updating all target surfaces throughout the scene. After a surface has been explicitly assigned by the user it is no longer considered as a target surface and will be left untouched by the algorithm.

3.5 Generating final texture coordinates

In this work we assume that the surfaces are already parameterized. We optionally rotate the texture coordinates by 90 degrees if we detect that the aspect ratio between the surface and the texture can be improved. However, we ignore the existing texture coordinates in the assignment process since this would give unfair knowledge about the target surface to our approach.

Generating proper texture coordinates is an area for future improvement: Even if in most cases texture coordinates are trivially computed at modeling time for environments, scaling factors and edge alignments should be taken into account. In addition techniques such as the one presented in [Cabral et al. 2009] could be used to adapt the texture to the surface and cancel any stretch.

In the case of solid or procedural textures (materials) we of course do not need to specify texture coordinates.

4 Applications and results

We tested our texturing application in a variety of situations. We first demonstrate our approach on small, controlled scenes in order to outline some of its properties. We then demonstrate the effect of the neighborhood driven texture assignment, and finally apply our method on entire game levels. We invite the reader to watch the accompanying video for a complete illustration of the interactive texturing interface.

Test scenes Our first test scene is a stair case surrounded by pillars, shown Figure 6. The user assigns textures for the entire scene by inspecting only seven surfaces. The stairs have varying lengths and the pillars have different heights. For easier visualization we do not display the textures but directly the texture ids as flat colors.

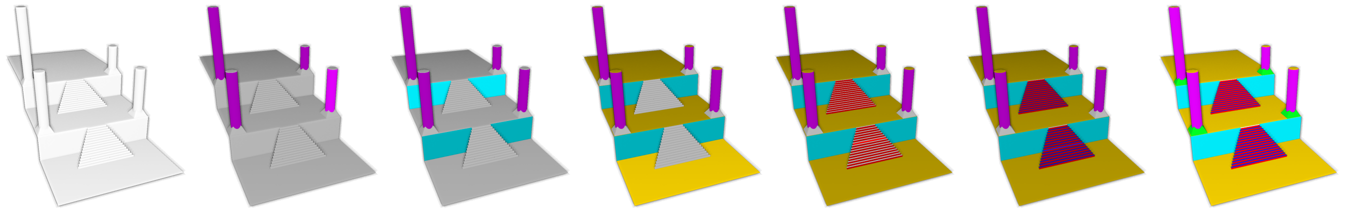

Figure 6: From left to right: *Initial temple scene and the 7 steps to texture all surfaces: Pillars, walls, floors, stairs. Surfaces are colored by texture id. Selected surfaces are highlighted.*

Our second test scene is a rock pile shown Figure 7. Each rock is obtained by adding a different random noise to a sphere. The user is able to assign different textures to the rocks outside the pile and inside the pile in just *two* clicks. The main discriminant is the accessibility factor.

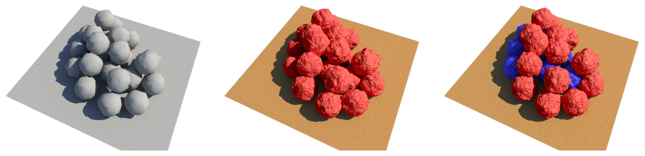

Figure 7: Left: *The initial rock pile. Each rock is different.* Middle: *A first rock is selected, all rocks are assigned the same material.* Right: *A rock down below is selected, all rocks partially hidden are assigned the same material.*

Finally, Figure 8 illustrates automatic transfer between game levels. The initial assignment is fully–automatic. The user then inspects a few surfaces to resolve ambiguities.

Thresholds Figure 9 illustrates the impact of the γ and τ thresholds in an extreme example. We illustrate the effect of γ by selecting a single surface and increasing its value progressively. This allows for more surfaces to be automatically assigned around the selected one, but soon includes surfaces which are too dissimilar. In general, γ is kept small and only accounts for small variations in the geometric data.

Similarly, we illustrate the effect of τ by selecting two different surfaces. τ has no effect with a single surface since per-texture variance is only defined when at least two surfaces use the texture. Here, increasing τ results in a proper automatic assignment of all other surfaces: The system properly interprets the variability in between the example surfaces. It is important to realize that τ only amplifies variations existing between the surfaces of a same texture: In this case there is no variation of the normal in the up direction, so increasing τ does not wrongly assign the texture to the floor surface. We typically use a large τ value, often 1.0. The only downside is that a texture used by very different surfaces – hence with large variance in all attributes – will appear as a good candidate for all target surfaces. One way to prevent this would be to rank higher textures with smallest variance in case of ambiguities. In most cases, the default settings of $\gamma = 0.1$ and $\tau = 1.0$ already work very well.

Neighborhoods influence As shown Figure 10, a few iterations of neighborhood fixup improves consistency of the texture selection. Nevertheless, some surfaces remain unassigned or ambiguous requiring the user to perform manual selection.

Noise resistance We illustrate the effect of noise Figure 11. The cubes quickly degenerate and are no longer recognized as such by the algorithm. The spheres are still captured since the right–most shape has a few probes matching the example sphere. However, the texture score dramatically decreases from $7.6 10^{-3}$ to $1.6 10^{-8}$.

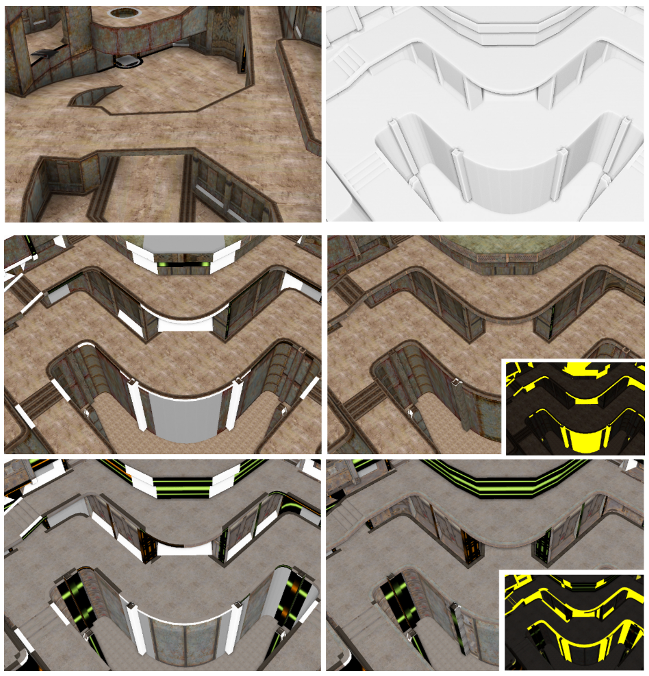

Figure 8: Top row: *A game level (left) is used as a source for assigning textures to a blank scene (right).* Middle: *The left image shows the automatic assignment. For each surface the algorithm ranked all textures and selected the first one. No match could be found for white surfaces. On the right the final images after the user manually selected textures for a few surfaces. Modified surfaces are shown in yellow inset. On average, the choice made by the user was at rank 2.2 affording for fast selection.* Bottom: *Same using a different source scene.* Scenes from Quake 4 © Id Software.

Performance The complete pre-processing time for a scene with 158464 triangles forming 16441 surfaces took 22.0 seconds on a dual Xeon 5160 3.0 GHz (4 cores). The ray-tracing part was done using the NVidia OptiX ray-tracer on a Quadro FX 5800, which resolved 43 million shadow rays.

The target scene shown in the top–right of Figure 8 is comprised of 26796 triangles grouped in 7062 surfaces. Generation of the 310669 probes took 7.8 seconds. The example scene at the top–left is comprised of 6497 surfaces and 182907 probes. A complete assignment from the example – including probe generation for the example – took 79.2 seconds. During interactive exploration, updating a single surface required 12 msec. on average.

5 Conclusion

We propose a first algorithm to assist texturing of large environments. Our method is interactive and immediately exploits user choices to assign textures to other surfaces. Rather than imposing an arbitrary best choice we design our method to rank all textures

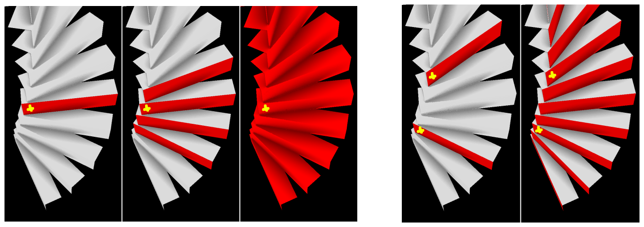

Figure 9: *In this example the user textures curved stairs. The surfaces selected by the user are marked by a yellow cross.* Left: *These three images illustrate how γ controls the global matching threshold ($\tau = 0$). On the leftmost image $\gamma = 0$ and no other surface than the selected one is assigned a texture. Increasing to $\gamma = 1.0$ allows for a few more steps to be assigned. However, in the third image increasing γ to 2.0 wrongly assigns the texture to all surfaces.* Right: *These two images illustrates how τ controls the per–texture matching threshold ($\gamma = 0$). On the first image $\tau = 0$. Two different surfaces are selected by the user, but nothing else is assigned. On the second image, $\tau = 1.0$ and all surfaces in between are automatically assigned: The system properly interprets variability between the two example surfaces.*

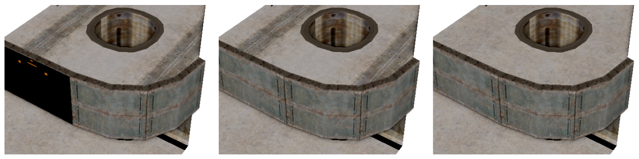

Figure 10: Left: *The initial result from the probes,* middle: *the result after one iteration (a wall gets fixed),* right: *the result after two iterations (the ground gets fixed).*

for a surface. This lets the user quickly select among the most appropriate choices.

A key contribution of this work is the probe–based surface similarity, measuring how well two surfaces describe each other by their probes. This lets us exploit geometric information to select a number of texture candidates for a surface. The use of probes not only enables a meaningful similarity measure for texturing purposes, it also abstracts away from polygons and avoids many of the typical robustness issues when dealing with vertices and triangles.

A major difficulty of the work was to define a robust and principled algorithm for automatic texture assignment. Throughout the work, our guiding principle has been to target simplicity – limiting the number of parameters – and practicality. We believe we succeed in this, even though our algorithm still cannot reliably texture a scene on its own and requires a number of surfaces to be manually inspected. Fortunately meaningful ordering of textures allows for fast selection.

For future work, we think there is room for improvement regarding the influence of neighbors. One aspect missing from our scheme is to consider the confidence of a neighbor – i.e. how high his own probe ranking is. In addition, an in–depth user study is required to measure the efficiency of the approach in a production setting. This work is only a first step and many interesting research directions are left over. We hope this will spark other approaches, as we believe computer assisted texturing will be a key element to create large virtual worlds of unprecedented visual richness.

Acknowledgements

We thank Michiel van de Panne, George Drettakis, Karan Singh, Nicolas Ray and the anonymous reviewers for insightful discussions, comments and proof reading. Thanks to NVidia for hardware, the OptiX SDK and for responsive and helpful support. This work was partially supported by Agence Nationale de la Recherche (ANR-08-CORD-021-03).

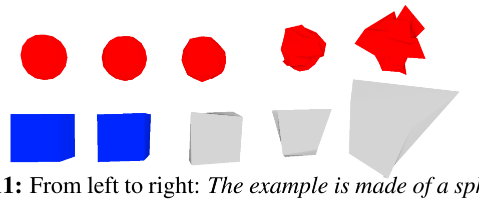

Figure 11: From left to right: *The example is made of a sphere and a cube textured differently. Different amounts of noise are added to the vertex positions.*

References

Anderson, D., Frankel, J. L., Marks, J., Agarwala, A., Beardsley, P., Hodgins, J., Leigh, D., Ryall, K., Sullivan, E., and Yedidia, J. S. 2000. Tangible interaction + graphical interpretation: a new approach to 3d modeling. In *Proceedings of ACM SIGGRAPH*, 393–402.

Ashikhmin, M. 2001. Synthesizing natural textures. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics*, 217–226.

Bishop, C. M. 2006. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Cabral, M., Lefebvre, S., Dachsbacher, C., and Drettakis, G. 2009. Structure preserving reshape for textured architectural scenes. *Computer Graphics Forum (Proceedings of the Eurographics conference)*.

Gal, R., and Cohen-Or, D. 2006. Salient geometric features for partial shape matching and similarity. vol. 25, 130–150.

Hertzmann, A., Jacobs, C. E., Oliver, N., Curless, B., and Salesin, D. H. 2001. Image analogies. In *Proceedings of ACM SIGGRAPH*, 327–340.

Legakis, J., Dorsey, J., and Gortler, S. 2001. Feature-based cellular texturing for architectural models. In *Proceedings of ACM SIGGRAPH*, 309–316.

Lu, J., Georghiades, A. S., Glaser, A., Wu, H., Wei, L.-Y., Guo, B., Dorsey, J., and Rushmeier, H. 2007. Context-aware textures. *ACM Transactions on Graphics 26*, 1.

Martinet, A. 2007. *Structuring 3D Geometry based on Symmetry and Instancing Information*. PhD thesis, INP Grenoble, 46, avenue Félix Viallet - 38031 Grenoble Cedex 1 - France.

Mertens, T., Kautz, J., Chen, J., Bekaert, P., and Durand, F. 2006. Texture transfer using geometry correlation. *Rendering Techniques*, 273.

Pauly, M., Mitra, N. J., Wallner, J., Pottmann, H., and Guibas, L. 2008. Discovering structural regularity in 3D geometry. *ACM Transactions on Graphics 27*, 3, #43, 1–11.

Simakov, D., C. Y. S. E., and Irani, M. 2008. Summarizing visual data using bidirectional similarity. In *CVPR*.

Tangelder, Johan W. H. Veltkamp, R. C. 2004. A survey of content based 3d shape retrieval methods. In *Proceedings of Shape Modeling International 2004*, 145–156.