Interactive Texture Synthesis on Surfaces Using Jump Maps

Steve Zelinka and Michael Garland
Department of Computer Science, University of Illinois at Urbana–Champaign, Champaign, Illinois

Abstract
We introduce a new method for fast texture synthesis on surfaces from examples. We generalize the image-based jump map texture synthesis algorithm, which partitions the task of texture synthesis into a slower analysis phase and a fast synthesis phase, by developing a new synthesis phase which works directly on arbitrary surfaces. Our method is one to two orders of magnitude faster than existing techniques, and does not generate any new texture images, enabling interactive applications for reasonably-sized meshes. This capability would be useful in many areas, including the texturing of dynamically-generated surfaces, interactive modelling applications, and rapid prototyping workflows.
Our method remains simple to implement, assigning an offset in texture space to each edge of the mesh, followed by a walk over the mesh vertices to assign texture coordinates. A final step ensures each triangle receives consistent texture coordinates at its corners, and if available, texture blending can be used to improve the quality of results.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Color, shading, shadowing and texture

1. Introduction

The synthesis of texture images from examples has been a well-studied problem in recent years, with numerous methods developed encompassing the spectrum of trade-offs between speed of synthesis and quality of results. The focus of this work is the synthesis of textures directly on 3D surfaces at interactive speeds. The ability to efficiently synthesize high quality textures on surfaces would be a useful tool for a variety of computer graphics tasks. Modelling systems can benefit from rapid feedback, allowing the interactive selection and fitting of textures to surfaces. Rapid prototyping systems may also benefit from the added realism textures can contribute, without any added impediment to the speed of development. And, of course, any system which generates geometry on the fly can be made more interesting by similarly generating textures on the fly.

Current methods for texture synthesis on surfaces may generate high quality results, but each of these methods generally has its own set of drawbacks (as discussed in Section 2). Our method is designed with a number of goals in mind, foremost among them being high quality synthesis results at interactive rates. In addition, since texture memory remains a scarce resource on graphics hardware, and the texture is assumed to be adequately described by the sam-

Figure 1: 40000 face model textured in 0.15 seconds.
patch pasting methods were first generalized by Praun et al. 5, who iteratively paste an overlapping irregular patch on a surface, and use texture blending to help hide boundary artifacts. More recently, Soler et al. 6 assign a consistent patch in texture space to hierarchical face clusters, refining as necessary to minimize boundary mismatch. Since only texture coordinates are generated, no additional texture space is required. While generating excellent results, the method requires significant computation time, despite accelerated boundary matching in Fourier space.

3. Texture Specification

A certain degree of user interaction is required in order to specify how the texture is to be mapped onto the surface. First, a scale factor for the texture must be given by
Figure 2: Left: Orientation vectors (blue) within the local tangent plane specify “up” in texture space. This field was generated using only sinks and sources at extremities. Right: Textured result.

Figure 3: Left to right: textured results; sample texture; patches visualized by texturing with a colour ramp.

the user which essentially fixes the size of each mesh triangle in texture space. The rotation of each mesh triangle in texture space may also need to be fixed. Like previous approaches\cite{10,8}, we use a vector field defined at each vertex of the mesh, the vectors of which correspond to the “up” direction in texture space, as illustrated in Figure 2. The user typically supplies a few “anchor” vectors at key points of the vector field, and these anchors are relaxed over the mesh to assign orientation vectors to the remaining vertices. We have found it especially useful to allow the specification of sinks and sources within the vector field, giving the user a greater degree of control over where these singularities occur, which allows their visual impact to be minimized. This is especially important as the genus of the mesh increases, since higher genus surfaces inherently require more singularities. Note that a full orientation field is primarily necessary for anisotropic textures; as previous researchers have noted\cite{10}, optimizations of the relaxation process exist if the texture only requires two- or four-way symmetry, and it may be skipped altogether (assigning a random orientation field) if the texture is isotropic.

4. Texture Synthesis on Surfaces

In this section, we develop our new method for texture synthesis on surfaces at interactive rates, by generalizing the image-based jump map texture synthesis algorithm. We first (§4.1) show a simple approach to selecting a vertex ordering for synthesis which may be pre-computed in fractions of a second. We then develop the process for choosing texture coordinates at a vertex (§4.2). Our probabilistically good matching algorithm allows a wide range of textures to be used, while avoiding the cost of run-time neighbourhood comparisons. Finally, we derive consistent texture coordinates at triangle corners based on these vertex texture coordinates (§4.3), optionally employing texture blending for improved quality (§4.4). Since our final output is a set (or sets) of texture coordinates, no new texture memory is required beyond that for the sample image, and the scale of the texture is not tied to the meshing of the surface.

4.1. Vertex Ordering

The order in which pixels are synthesized is key to generating high quality images using jump maps. A Hilbert path ordering was found to give highest quality results, as it allows an even distribution of directions from which patches may be extended. In contrast, we have achieved highest quality results on surfaces by maximizing the number of available neighbours from which to continue synthesis. We therefore adopt a simple region growing approach, iteratively selecting the unsynthesized vertex whose proportion of already-synthesized neighbours is greatest. As shown below (§4.2.5), selecting this vertex makes it more likely a good match is selected. This simple, greedy approach effectively grows the texture out from a seed point, producing almost surprisingly good results (Figure 3). Since this ordering is constant for a given mesh topology and seed vertex, it may be precomputed, typically in a fraction of a second.

It should be noted that recent work by Bagomjakov and Gotsman\cite{2}, which aims to optimize the use of vertex caches on recent graphics hardware, may also be of use here. As Hilbert paths optimize locality of reference, they develop a surface analogue called a Universal Rendering Sequence (URS), which reorders the faces of a mesh such that vertex accesses will exhibit high locality. A Hilbert path ordering provides high quality results for the image-based synthesis algorithm\cite{12}, and as might be expected, a URS similarly provides a good vertex ordering for our surface-based synthesis algorithm. In our experiments, the URS-based ordering and our above region growing approach yield comparably good results. There are trade-offs in deciding on which ordering to use: URS computation may require several seconds to minutes, but may also improve rendering speed on vertex caching graphics hardware; region growing, on the other hand, is extremely fast and simple to implement.
4.2. Texture Coordinate Assignment

Given the order in which vertices are processed, we now turn to how each vertex is processed. We first review the image-based algorithm in the next section, and develop our surface analogue to this algorithm in the following sections.

4.2.1. Review of Image-Based Texture Synthesis

The image-based jump map texture synthesis algorithm works by assigning an input image pixel address to each pixel of the output image. Each new output pixel is assigned an address by randomly selecting an already-synthesized neighbour to use as a source. The new pixel is then assigned the sum of the source’s effective address and the offset in the output image from the source to the new pixel. So, for example, if the source is the pixel above, and its effective address is \( \langle a, b \rangle \), the new pixel is assigned the address below \( \langle a, b \rangle \).

The source’s effective address is generated by another random choice: either its actual address in the input image, or a virtual address drawn from the jump map entry for its actual address. So, if the source’s actual address is \( p \), and the jump map entry at \( p \) lists \( a \) and \( f \) (meaning that that \( a \) and \( f \) are good matches for \( p \)), the source’s effective address is randomly chosen from \( p, a, \) and \( f \). The actual address is weighted much higher than the jump map entries, as jumps ought to be relatively rare events. As the actual address approaches an edge of the input image (within some fraction of the image size), its weight is linearly decreased to zero in order to encourage a jump away from the boundary; hitting the boundary may cause noticeable artifacts. The final step of the algorithm is to generate the output image using the assigned input image addresses at each pixel. To improve quality, blending may be applied by copying each pixel with a small blend kernel.

4.2.2. Neighbour Distances

On surfaces, the distance between two neighbours is not a simple unit offset in an axial direction. We address this by assigning a fixed distance in texture space to each edge of the mesh while respecting the user-defined texture orientation and scale. To avoid distortion, the texture should be mapped onto! the surface such that texture space locally corresponds to the tangent plane. As illustrated in Figure 4, each edge is projected into a tangent plane to determine its offset in texture space; the tangent plane for the edge is averaged from the tangent planes. First, we construct a basis for the tangent plane at each vertex, using the vertex’s tangent plane. The edge is then projected into this plane, and finally scaled according to the user-defined texture scale (given as texels per object space unit) to yield a 2D offset for the edge. Each vertex thus has a consistent view of the distance to each of its neighbours in texture space (note, of course, each endpoint of an edge sees the opposite offset from the other).

4.2.3. Floating Point Jump Map Look-ups

Since edge offsets are invariably floating point, vertices may no longer be assigned integer texel addresses. This complicates jump map look-ups for a vertex; previously, each pixel address corresponded to one particular jump map entry. As shown in Figure 5, we define a floating point jump map look-up to return a list of all jumps stored in the 4 neighbouring
new vertex. Instead, as shown in Figure 6, we construct a
randomly select a neighbour from which to continue synthesis,
for a particular vertex. Thus, we can no longer just ran-
As just discussed, jump map entries are not necessarily valid
for any offset actually used. We thus com-
image-based algorithm, as the offsets used at synthesis time
were known
to the vertex. This means the sum of the vertex's texture address
of a vertex may be textured from the address assigned to a

Figure 6: The middle vertex is to be synthesized, with two of
its neighbours already synthesized; their assigned addresses
are listed above their corresponding jump lists. Each neigh-
bour’s jump list is added to a selection table, with their jump
destinations offset by the corresponding edge offsets. Note
that a continuation entry is added for each neighbour as
well, corresponding to “do not jump”, which simply adds the
edge offset to the neighbour’s assigned address. The middle
vertex is assigned texture coordinates by a weighted random
choice from the table.

to the sub-pixel offset of the address. The jump
destinations are also changed according to this sub-pixel off-
set. Thus, existing image-based jump maps may be used on
surfaces.

4.2.4. Invalid Texture Coordinates

In order to simplify the consistent texture coordinate assign-
ment below, we ensure that the entire local neighbourhood of
a vertex may be textured from the address assigned to a
vertex. This means the sum of the vertex’s texture address
and the edge offset must remain within the texture, for each
descriptive offset to the vertex. Note this was not an issue for
the image-based algorithm, as the offsets used at synthesis time
were known a priori, and every jump in a jump map could be
guaranteed valid for any offset actually used. We thus com-
pute a 2D bounding box at each vertex which encloses all of
the offsets incident to that vertex. An address is considered
valid for a particular vertex only if the entire 2D bounding
box remains inside the texture when centered at the address.

4.2.5. Probabilistically Good Matching

As just discussed, jump map entries are not necessarily valid
for a particular vertex. Thus, we can no longer just ran-
domly select a neighbour from which to continue synthesis,
as it may not be able to supply any valid addresses for the
new vertex. Instead, as shown in Figure 6, we construct a

list of potential addresses drawn from all available neigh-
bours of a vertex. Effectively, we concatenate the jump lists
for each neighbour’s texel address together, filtering out the
invalid addresses, and then generate one random to decide
the new vertex’s address. Note that we must add entries for
each neighbour corresponding to the “do not jump” decision,
since each of these continuation entries may be different. In
the rare event that there are still no valid entries, a random
valid address is assigned.

The weighting scheme is the same as that used in the
image-based algorithm. There, the weights of the jump map
entries at a pixel were normalized to sum to one, or a lower
value if the jumps at that pixel were not very good; the
weight for the continuation entry (“do not jump”) was then
simply the desired average number of pixels between jumps.
The weights were then modified, if necessary, to encourage
boundary avoidance, and a weighted random selection made.
The only change for synthesizing on surfaces is to explicitly
divide the continuation weights by the length of the corre-
sponding edge offset; with images, the offsets were all only
1 pixel, so this was done implicitly.

Note that this method produces “probabilistically good”
assignments for each vertex. If several neighbours suggest
common destinations, greater weight will be assigned to
these destinations, and it becomes more likely that these des-
tinations, which are likely good matches, will be chosen.
Outliers suggested by a single neighbour, on the other hand,
will become less likely to be chosen.

4.3. Consistent Texture Coordinates

So far, we have shown how the mesh vertices are traversed,
and how each is assigned a set of texture coordinates. How-
ever, this alone does not properly texture the mesh; every
time a jump is taken over an edge, the endpoints of the edge
belong to different areas of texture space, and interpolation
of these texture coordinates across the edge or neighbouring
triangles would be erroneous. We must assign texture coor-
dinates to each corner of each mesh triangle such that each
triangle is properly mapped into texture space.

Our basic approach is to choose one vertex of the trian-
gle as a base vertex, using the vertex’s coordinates for the
corner coordinates, and assigning the other two corners by
summing the base vertex address with the edge offsets along
the respective edges. We have not developed any particular
strategy for choosing a base vertex which yields results qual-
itatively better than a random choice for general situations.
However, as discussed below, texture blending capabilities
imply that such a strategy need not be necessary.

4.4. Texture Blending

If texture blending resources are available, highest quality
results may be obtained by generating three sets of texture
coordinates for each triangle, each with a different base vertex. We then use multi-texturing, with the sample texture and one of the three sets of texture coordinates in each texture unit. We have generated good results with a linear alpha ramp, from one at the base vertex down to zero at the other vertices. In absence of multi-texturing facilities, three passes with appropriate blending may be used instead. Note that if the texture scale is sufficiently large with respect to the surface, it may be necessary to blend boundaries over multiple triangles.

5. Results and Discussion

As shown in the figures, our method generates high quality results over a range of surfaces and textures. The synthesis time is independent of both surface complexity and texture complexity; generally, we process about 150,000 vertices per second on an Athlon 1.5Ghz machine. Models shown vary in size from 20,000-40,000 faces each, taking from 0.05-0.15 seconds each. Sample textures are 200×200 pixels or slightly larger. All results use texture blending and the region growing-based vertex ordering.

Like the image-based method, the quality of our results is best for stochastic textures, but very often acceptable even for quite ordered textures. Our “probabilistically good” approach to matching produces results whose quality meets or exceeds that of the previous image-based approach for most textures. The underlying assumption of our method is that if two pixel neighbourhoods are well-matched, those neighbourhoods will still be well-matched if they are offset a certain distance (i.e., across an edge). While this is not always true, the quality of our results have shown this assumption to be warranted in most cases. One would expect that as the texture scale becomes smaller, the quality of results would degrade; as edges become longer in texture space, this underlying assumption would become more unlikely. However, this effect appears to be masked in practice by the reduced size of texture features on the mesh. Further, since jump maps are constructed offline, once per texture, we can afford to use relatively large neighbourhoods, ensuring successful results over a wide range of texture scales. Figure 7 shows the an example of the range of texture scales that may be accommodated by a single mesh and jump map.

6. Conclusions and Future Work

We have presented an algorithm for texturing a surface from an example at interactive speeds. We have managed to meet all of the goals set forth in the introduction: high quality results, interactive synthesis speed, memory efficiency, and generality over both surfaces and textures. Our generalization...
tion to surfaces of the image-based jump map texture synthesis algorithm leverages existing jump maps to produce textures directly on arbitrary manifolds, while retaining the speed and simplicity of the original algorithm.

In future, we expect that further development of our probabilistic matching scheme will lead to even higher quality results. In particular, extra sources may be sampled along or within neighbouring already-synthesized edges or triangles, emphasizing the importance of matches which remain good over large spatial areas, and reducing the likelihood of selecting matches which are only good in one particular direction. Extra passes over the mesh, when all neighbours are available to suggest destinations, may further improve the quality of results. In this respect, our algorithm may also serve as a fast pre-conditioner or place-holder for slower, higher-quality texture synthesis algorithms, allowing them to use full neighbourhoods from the start, for example. Note that these ideas may also be applied to image-based algorithms, perhaps further improving their quality. More generally, we expect a hierarchical generalization of our algorithm is possible, allowing the interactive texturing of progressive and LOD meshes from an example texture.

Acknowledgements

This work was supported in part by a grant from the National Science Foundation (CCR-0086084). The textures on the cat and the upright dragon models are courtesy www.scenic-route.com; that on the bull from the Brodatz collection; those on the Santa, torso, and serpentine dragon are from the VisTex texture database; and the remainder are from www.gr-sites.com. Thanks to the anonymous reviewers for their many helpful comments.

References

Figure 8: Jump Map Texture Synthesis Results.