NORMAL SYNTHESIS ON RGBN IMAGES

Thiago Pereira, Luiz Velho
Vision and Graphics Laboratory, Instituto de Matematica Pura e Aplicada, Brazil
tpereira@impa.br, lvelho@impa.br

Keywords: RGBN, texture synthesis, normal map, filtering

Abstract: In this work, we synthesize normals and color to add geometric details to an RGBN image (image with a color and a normal channel). Existing modeling and image processing tools are not apt to edit RGBN images directly. Since high resolution RGBN images can be obtained using photometric stereo, we used them as full models and as exemplars in a Texture from Example synthesis.
Our method works on RGBN images by combining the normals from two bands: base shape and details. We use a high pass filter to extract a texture exemplar, which is synthesized over the model's smooth normals, taking into account foreshortening corrections. We also discuss conditions on the exemplars and models that guarantee that the resulting normal image is a realizable surface.

1 INTRODUCTION

We use texture synthesis to edit large regions of an RGBN image changing both color and normals. The RGBN image (Toler-Franklin et al., 2007) is a 2.5D data type since it is a photo containing, for each pixel, not only color but also geometry information (normal). Editing regions by individually addressing local changes can be painstaking to artists. For this reason this process needs to be made automatic. Two options arise. Procedural methods are very powerful but are in general difficult to control. We use a texture from example method that takes as input only a small sample of the desired texture and then reproduces it in a large region. Our system can synthesize normal textures on shapes represented by normals (RGBN images), never using positions. Regarding synthesis, an advantage of working with RGBN images is that the projective mapping distortions on the synthesized textures can be corrected by using information from the normals. The texture sticks to the surface.

However, we do not want the texture to follow every small surface irregularity. Therefore, we separate the RGBN image into macro and mesostructure bands. It is common to cluster different geometric scales in different levels (Chen et al., 2006). The macrostructure level is the general shape of the model as would be seen from a distance. The mesostructure level contains intermediate geometric details that are visible with a naked eye such as bumps and creases. Our synthesis method can respect the base shape, only changing the details.

Our method can be applied to RGBN images from many different sources. A high-resolution modeled mesh can be used to extract it and enhance models with fewer polygons (Cohen et al., 1998; Cignoni et al., 1998). Further processing of details of this model can proceed using the normal map and avoiding a huge mesh. An alternative to modelling is capturing real-world objects. Capturing normals with 3D scanning requires expensive equipment and lacks resolution, being far behind the cheaper digital cameras. Better results come from photometric approaches as in the digitization of Michelangelo's Pieta (Bernardini et al., 2002). A projective atlas (Velho and Sossai Jr., 2007) can be built from this dataset. The resulting charts are surfaces mapped by a camera transformation to the image plane. Since they contain normal and colors, each chart is in fact an RGBN and can be edited by our system.

Capturing a model's normals is easy using photometric stereo techniques (Woodham, 1989). It takes as input multiple images from the same view point, each illuminated with different known light positions. It then solves a least-squares problem to find the normal and albedo (color) of each pixel in the image. Since photometric stereo works on regular images, it allows the capture of very high resolution models. This is a powerful method to obtain full models but also to generate texture exemplars.

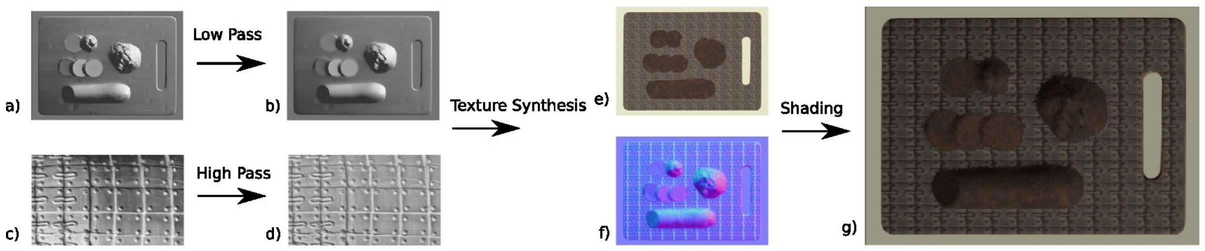
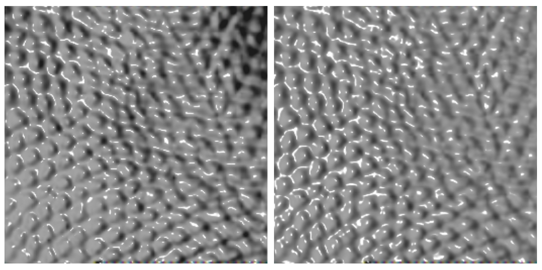


Figure 1: Overview of the method. Inputs: a) RGBN image, c) texture exemplar. Filtered inputs: b) smooth RGBN image, d) texture detail. Results: e) color, f) normal (represented in RGB colors) and g) shaded RGBN image

Our main contributions are:

1. Synthesizing normal vectors allowing relighting of the final model.
2. An RGBN image editing system implementing a frequency aware texture synthesis framework separating exemplars and models into base shape and details.

In section 3, we present an overview of our normal synthesis framework. In section 4, we discuss the work of (Zelinka et al., 2005) in color synthesis on RGBN images. In section 5, we extend their method for synthesizing normals. In section 6, we determine when the edited RGBN images correspond to a realizable surface.



(a) Original Exemplar (b) Filtered Exemplar

Figure 2: Removing the normal low frequencies of the exemplar only the desired details. We show shaded images of the normals under same lighting conditions.

2 RELATED WORK

Texture from example synthesis can be classified into two approaches. In pixel-based methods, new pixels are generated one at a time. In (Efros and Leung, 1999), the authors search for a best match between the neighborhood of the already synthesized texture and neighborhoods in the texture sample. There are also patch-based methods (Efros and Freeman, 2001) (Fang and Hart, 2004) where the algorithm step consists of iteratively synthesizing small regions at a time. This approach leads to a direct transfer of local statistics, but it has the drawback that seams between patches need to be handled.

Another classification of synthesis regards the domain. Texture synthesis can be parametric, volume-based or manifold-based. Volume-based methods will synthesize texture usually in R^2 or R^3. Parametric synthesis will target more general manifolds by working in the parametric domain (usually planar). There are also manifold methods which are non-parametric (Wei and Levoy, 2001) (Turk, 2001). These methods work directly in the manifold representation (usually a mesh) and use only local decisions.

In (Zelinka et al., 2005; Fang and Hart, 2004) the authors extend manifold techniques to work on RGBN images, using the normal as a local descriptor of shape. Both works use shape from shading to recover normals from photographs. Normals are used to guide distortions in the synthesized color texture. Zelinka (Zelinka et al., 2005) adapted jump map based synthesis which is a pixel based method to work on normals, while Textureshop(Fang and Hart, 2004) uses a patch based method. Both works allow for advanced editing of images. For example, object material can be replaced, while still respecting shape and shading. Unlike these works, in our method, both color and normals are synthesized on RGBN images. One way of looking at RGBN synthesis is as a quasi-stationary process (Zhang et al., 2003). Synthesis proceeds through the image but varies spatially depending on the normals.

In (Haro et al., 2001), the authors capture high resolution normals of small patches of face skin and then use texture synthesis to replicate these tangent space normals in parametric space. Adding detail to faces is an important application of our method. While in their approach, a normal map is synthesized from scratch, we can respect existing frequency bands in the normal map. Procedural synthesis was explored in (Kautz et al., 2001) where bump maps are created on the fly based on a normal density function.

3 METHOD

While the problem of synthesizing color textures on top of RGBN models has been studied in (Zelinka et al., 2005), we propose a method to synthesize normal textures. We start this section with an overview of our method (Figure 1). The method receives as input an RGBN image (a) and an RGBN examplar (c). It has three steps. First, in the frequency splitting step, we low-pass filter the normal image (b) removing details and we high-pass filter the examplar normals (d) removing the base shape. Second, in the texture synthesis step, we synthesize both the colors (e) and normals (f) of the examplar on the smooth normals (b) compensating for foreshortening distortions. Finally, in the combination step, we merge the synthesized normal details with the smooth RGBN image.

To build the smooth normals we use the RGBN bilateral filter (Toler-Franklin et al., 2007). It takes into account both normal and color differences and respects edges. In (Pereira and Velho, 2009), the authors have developed linear filters for normals that allows any kernel mask to be used. It builds on the one to one mapping between a normal field $N(x,y)$ and the gradient field (z_x, z_y) of an associated height function z given by:

$$N(x,y) = \frac{(-z_x, -z_y, 1)}{\sqrt{z_x^2 + z_y^2 + 1}}$$

To filter normals, they first convert it to a height gradient. By noting that derivatives and convolution (g kernel) commute, the gradients can be filtered as if we had the height function z at hand.

$$(z * g)_x = \frac{\partial(z * g)}{\partial x} = \frac{\partial z}{\partial x} * g = (z_x * g)$$

During splitting, we use their high pass kernel on the texture sample (Figure 2). As an alternative to the bilateral filter we could build the smooth normals using a low pass kernel. The filters in (Pereira and Velho, 2009) have the advantage that they guarantee that the resulting normals correspond to a surface.

To assist the user in defining the editing region, we use a segmentation method (Felzenszwalb and Huttenlocher, 2004) which was extended (Toler-Franklin et al., 2007) to handle RGBN images. The procedure is fast and is well suited to interactive applications. It also takes into account the normal and color channels, using all available information to improve results. It is also possible to segment objects based solely on geometry. To avoid over-segmenting, a bilateral filter should be used prior to segmentation to remove noise while preserving edges.

For replicating the RGBN sample on a base RGBN, we extended jump map-based texture synthesis (Zelinka and Garland, 2004), more specifically jump maps on RGBN images (Zelinka et al., 2005). This method is non-parametric and pixel-based, avoiding the need of a local parametrizations on normal maps. It does not produce the best quality results but it is the only interactive time technique, allowing for easy user experimentation. This strategy also handles base geometry distortions by varying image edge lengths during synthesis as a function of the normals.

During texture synthesis, colors are defined for each pixel. However the synthesized normals (high frequency) still have to be combined with the filtered input normals (low frequencies). We want to combine normals controlling which frequency bands we are replacing or editing. We follow the approach of (Pereira and Velho, 2009), where two combination schemes were proposed: the linear model and the rotation model. The linear model combines normals by looking at them as gradients. This strategy is simpler and has the advantage that the resulting normals are guaranteed to correspond to a real surface, but the equivalent position displacements are restricted to the z-direction. On the other hand, the rotation model deforms the details to follow the base shape before combining. We use this method for all examples.

4 JUMPMAPS ON RGBNS

We begin this section by reviewing jump map-based texture synthesis on a regular image. We also describe a few principles that makes the method better suited to our application. We then discuss (Zelinka et al., 2005) the extension to color synthesis on RGBN images.

The simple example-based synthesis algorithms (Efros and Leung, 1999; Efros and Freeman, 2001) exhaustively search the input for a best match. This search is done for each pixel or patch being synthesized. In (Zelinka and Garland, 2004), Zelinka splits

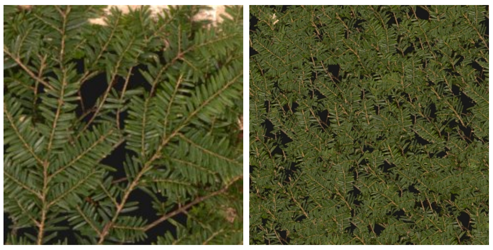


Figure 3: Jump Map synthesis results.

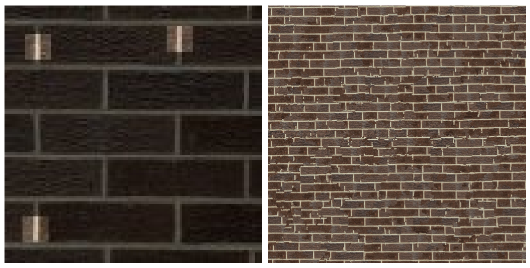


Figure 4: The highlighted neighborhoods are similar, but a jump between them introduces offsets, breaking brick alignment. A bigger mask size (19x19) improves results.

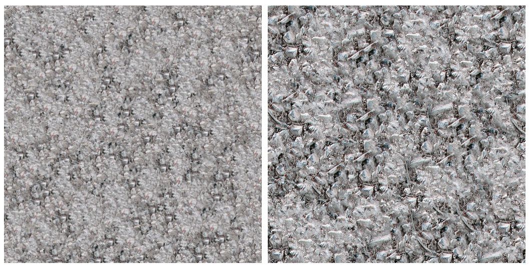


Figure 5: In both images, analysis was done with a 100x100 sample, but the second was synthesized with a 500x500 higher resolution sample, which would be hard to analyze.

texture synthesis in two phases: analysis and synthesis. Note that in previous methods analysis was being done online and would use an already synthesized neighborhood as a query. Zelinka's insight is that while this neighborhood is only known at synthesis time it will resemble an existing one in the texture example. As such, we can precalculate and store the similarities between all neighborhoods in the examplar. The Jump Map data structure holds for each examplar pixel a list of *jumps* to similar pixels.

During synthesis, the output texture is traversed and pixels are sequentially copied from the example. Eventually the input texture border will be reached, that is when the jump map comes in hand. As the border approaches, a jump is randomly selected from the Jump Map. As such, there is an infinite amount of texture to be copied.

The analysis phase is cast as a nearest-neighbor (NN) problem in a neighborhood space. Each pixel's square neighborhood is encoded in an M^2 feature vector, using the L_2 norm for comparisons. Notice that two parameters influence performance M and N (input sample size). Since analysis with small values of M won't represent the texture appropriately we are easily led to a high dimensional NN problem which would be too slow to handle directly. Instead, the authors use an approximate nearest neighbors (ANN) data structure (Mount, 1998) that allows very fast queries. For further optimization, PCA is used for dimensionality reduction of the feature vector.

As Zelinka notices, jump map synthesis works best for stochastic textures and weak structured textures. While quality results can be obtained with highly structured textures (Figure 10), these are exceptions. The reason can be seen in Figure 4. We see very similar neighborhoods, but if a jump is taken between them offsets will be introduced in the synthesized bricks. This problem happened because a small neighborhood was used for analysis. While increasing M improves results, it also increases analysis time.

For highly structured textures it is more important

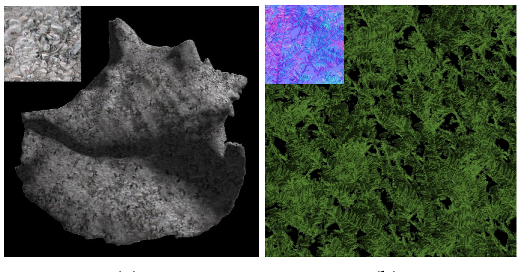

(a) (b)

Figure 6: a) Due to foreshortening, texture scale varies spatially according to the normals. b) The leaves were replicated generating a normal map. We show here a shaded version of the synthesized map.

to match structure than to transfer details. We use multiresolution to ignore sample details during analysis. We notice that it is possible to use different values of N (different resolutions) for analysis and synthesis, virtually allowing larger masks. We use small samples for fast analysis, while we use detailed samples for quality synthesis (Figure 5). Zelinka uses multiresolution but only to improve PCA compression.

To extend this algorithms to RGBN images, two problems have to be solved: synthesizing normals and synthesizing **on** normal images. The next section presents our solution to the first problem. To handle the second one (Zelinka et al., 2005) adapted jump map-based color synthesis to normal images. On RGBN images, we need to vary the scale of the synthesis spatially due to foreshortening (Figure 6-a). A unit pixel offset in the output image induces a displacement in the represented surface, which in turns induces an offset in texture space. These parameters may be global or may vary smoothly over the surface (Zhang et al., 2003). By approximating the surface locally by a plane orthogonal to the normal n, we can obtain the displacement d_t in texture space as a function of the displacement d_i in output image space and of the projection p of d_i onto the surface $d_t = \frac{|d_i|}{|n_z|}\frac{p}{|p|}, p = d_i - <n, d_i> n$.

5 NORMAL TEXTURES

We have extended the analysis and synthesis steps of the jump maps method to synthesize normals. In the analysis phase, we take a normal map sample representing our desired texture and we are asked to build a jump map for it. Since the basic primitive in jump map analysis is comparing neighborhoods, we must settle on which metric to use. In the color setting, the sum of the euclidean metric for each pixel neighborhood was used. However normals are in the unit-sphere S^2 and as a consequence each neighborhood feature vector is in the cartesian product $S^2 \times S^2 \times ... \times S^2$. Therefore, the sum of the pixel normal metrics will induce a natural definition of a neighborhood metric, we only have to choose a normal metric. We analyze three different metrics: euclidean, geodesic and dot product based.

The geodesic distance obtained from the angle $cos^{-1}(< n_1, n_2 >)$ is a natural choice (intrinsic distance) and it would be easy to adapt the $O(N^2)$ solution to the NN problem in the analysis phase. On the other hand, since cos^{-1} is a non-linear function, it is very hard to adapt either PCA or ANN, the fundamental algorithms that allow building jump maps in reasonable time. Linearity could be obtained by using a dot product based distance $1- < n_1, n_2 >$, but it falls far from the geodesic distance (Figure 7) specially for small values. In NN problems, these are the exactly the ones we are most interested in. This leads us to our last metric alternative: Euclidean. The Euclidean metric is a very good approximation for the geodesic metric for small distances since their derivatives at zero agree. Therefore, the Euclidean distance gives us a good compromise between simplicity and precision, so this is our choice for texture analysis.

As for synthesis, the order in which pixels of the output image are synthesized is important to reduce directionality bias. This happens because each pixel is synthesized based on only one of its neighbors. Zelinka found that following Hilbert curves produces much better results than scan-line or serpentine traversals. However, we do not use Hilbert traversal. The reason is we look for synthesis in subsets of RGBN images, which could be defined by either user or automated segmentation. Since these regions may have very complex shapes and topologies, it would be complicated to use a synthesis order based on Hilbert curves. Instead, we used a depth-first search (DFS) using a 4-connected pixel neighborhood. We recursively visit each pixels and its neighbors. The naive DFS algorithm produces unpleasant results with a directionality bias. This is a consequence of always visiting a fixed neighbor first, for example the north neighbor. A simple alternative is to use DFS, proceeding to each cardinal direction in random order. This breaks directionality and generates results almost as good as Hilbert traversal ones, as demonstrated by all results in this manuscript. The DFS traversal has two advantages. First, it only processes the pixels being synthesized. Second, it can respect discontinuities in the normals. For instance, in Figure 6-a we would not want synthesis to cross from the upper part of the shell to the lower part, we would like them to be in different connected components in the graph. This is easy to handle, if two neighboring pixels have different normals (defined by a threshold), they are simply not connected in the graph.

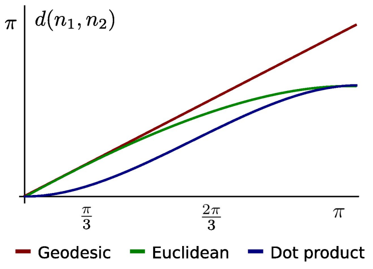


Figure 7: Different normal distances as function of the angle between the normals.

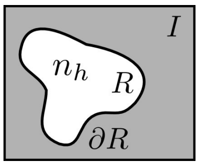


Figure 8: The detail normals n_h defined in R are combined with the image normals.

6 INTEGRABILITY ANALYSIS

In this section, we discuss the conditions for the resulting normals to be integrable, that is, for it to correspond to a surface. For simplicity, we only analyze the linear combination method (Section 3) and disregard the foresight distortions introduced by scaling.

Given a base normal field n_b defined in the entire image I (Figure 8) and a synthesized detail normal field n_h defined in $R \subset I$, the problem of combining normals is generating a new normal field which agrees with n_b everywhere, but is influenced in R by n_h. We refer to their respective height functions as b and h.

We can look at a normal image n as a 2D vector field w (Section 3). If this field is the gradient of a height function, we say it is a conservative vector field. This means our normal image does correspond to a surface.

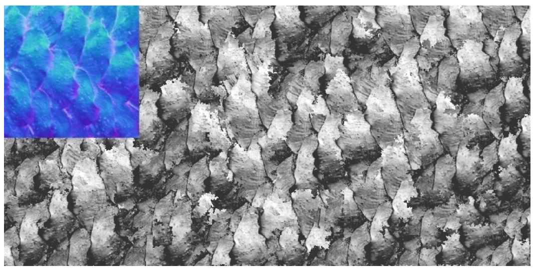


Figure 9: When the low frequencies of the exemplar are not removed, it is harder to infer pattern. This generates an uneven result.

The authors of (Pereira and Velho, 2009) show that conservative results can be guaranteed when combining normals by adding their vector fields counterparts w_h and w_b. Three conditions must be satisfied. First of all, they require w_h and w_b to be conservative. Second, since h is only defined in R, we must extend it as zero outside the edited region. This requires w_h to fall smoothly to zero close to ∂R and be equal to 0 outside. However this is not enough the added detail must not introduce level changes outside R or it will create discontinuities in height.

What restrictions do we have to make to guarantee that normal synthesis satisfies the three requirements above? To begin with, w_h might not fall smoothly or not even fall to zero in ∂R, which would result in discontinuities in w. In many applications (Figure 10), w_b is already discontinuous in ∂R so that new discontinuities are not created. Second, synthesis will not introduce level changes if w_h only contains high frequency content which tends to oscillate and cancel itself. Hence, the restriction on w_h is enforced with the high-pass filter on the exemplars. In addition, texture synthesis methods in general, and jump maps in particular, do not introduce repetitions and so no low frequencies are created.

It seems very unlikely that non-parametric texture synthesis methods can guarantee the generation of conservative fields in arbitrary domains, since only local operations are performed. On the other hand, local operations seem enough to generate curl-free fields that guarantee that w is conservative under the additional hypothesis that R is simply connected (does not contain holes). Traditional techniques aim at generating texture such that each pixel's neighborhood closely resembles a neighborhood in the exemplar. We argue that the curl is also similar in this neighborhood. This means given curl-free exemplars, the synthesis will generate approximately curl-free textures. As Figure 10 shows, quality results can be obtained even when R is not simply connected.

7 RESULTS

In this section, we will discuss some of the results obtained. Shaded images were produced with one light source. In some examples, uniform albedo is used to better highlight shape. In Figure 12-b, only the normals of the shell were changed, colors were unaffected. The base geometry was combined with high frequency normals extracted from rust. These new normals retain the original shape of the shell but give the appearance of a new material. It is true that with some trial and error rust could be generated with a procedural noise. On the other hand, structures synthesis from real objects on the shell (Figure 12-c, 13) can only be accomplished with texture from example methods. These structured examples show how synthesized details follow the base geometry.

In Figure 6-b, we can see a normal map sampled from scanned leaves and synthesized normals. Uniform albedo was used for shading. There are some aliasing artifacts which could be handled by synthesizing in a higher resolution and down-sampling. This is possible, since synthesis time scales linearly.

The original model contained a wooden board with real vegetables (see Figure 10, upper left). Segmentation was able to separate the objects. We added the relief and color of the armor of a stone Chinese warrior to the board. The complex topology of this object was not a problem and DFS was able to guide synthesis around it. Rust was synthesized on the vegetables (normal and color). The fine scale normals on the normal map are hard to spot on the shaded version.

In Figure 9, a sample of the normals of a pine cone was used for synthesis. Low frequencies were not removed. This can be seen as the exemplar varies smoothly from light (top) to dark blue (bottom). It has two consequences, first it is harder for the analysis phase to infer pattern. Second the final result is uneven. The shaded image shows some darker regions which are not facing the light.

As in Textureshop (Fang and Hart, 2004), shape from shading can be used to recover normals from photographs. We have also used it to obtain the texture exemplar (Figure 11). A swatch was extracted from the lizard's arm (b) and combined with smooth hand normals (d). The final image (e) was shaded under diffuse lighting, ignoring the lizard's skin reflectance properties. For more realistic results, a synthesis method that can handle morphing between textures is required. Notice how the texture of the fingers, the hand and the arm of the lizard are different. Very high frequency detail like human hand lines could be recombined in the final result.

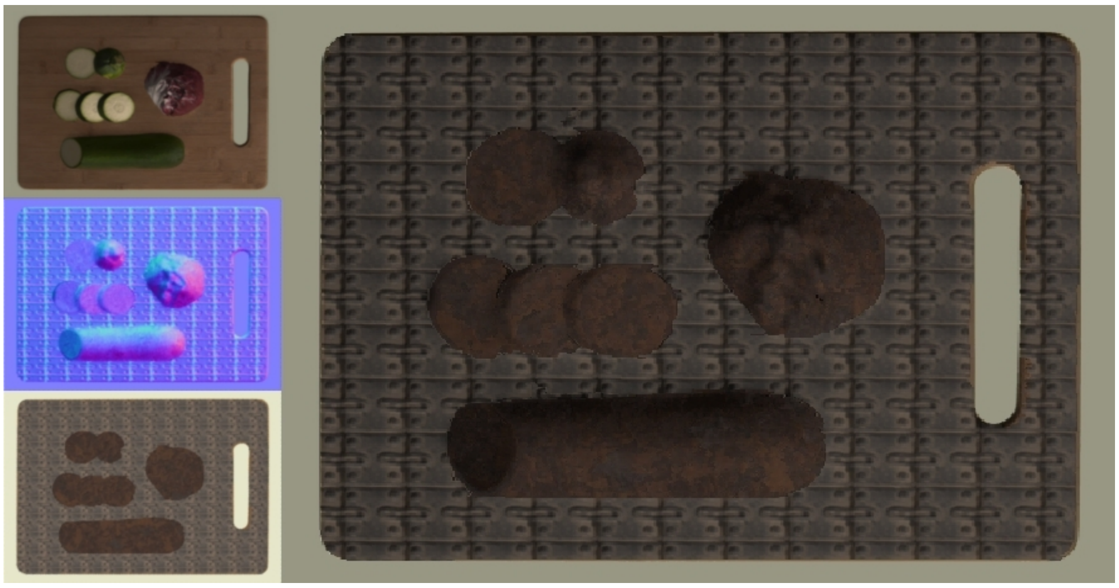


Figure 10: A shaded image of edited vegetables. Automatic segmentation followed by texture synthesis is a powerful tool for material replacement. The vegetables were turned into rust metal.

8 CONCLUSION

We have developed a method to synthesize normal textures on RGBNs. Our pipeline includes normal filtering to control which frequency bands we edit or replace. We discuss conditions on the exemplars and models that guarantee that the resulting normal image is a realizable surface.

Requiring the deformations to extend to 0 outside R is a limitation of our method, blending techniques should be investigated in the future to enforce this property on any synthesized deformation. A limitation of jump map-based synthesis is that being pixel-based, it has problems with very structured textures. On the other hand, it is harder to adapt less local methods to RGBN images, since the metric of bigger neighborhoods is non-trivial. In future work, we would like to use normal synthesis in the context of inpainting normals to fill missing regions. Another direction is to use a projective atlas (Velho and Sossai Jr., 2007) to extend this work to a surface. This new synthesis would take into account not only macrostructure (mesh) but also mesostructure in the form of existing normal maps.

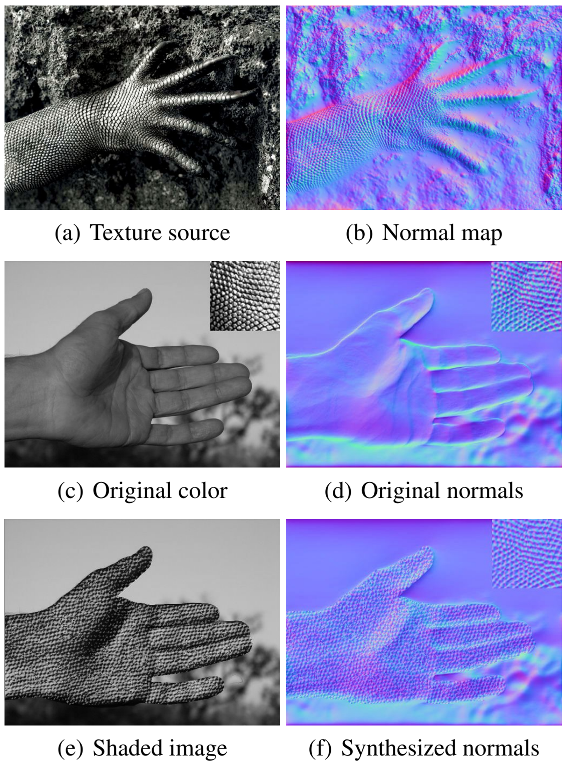


Figure 11: Shape from shading was used to estimate normals (b,d). A small sample of the lizard's arm was filtered and synthesized over the hand (e). Lighting (e) was set similar to the original. Only the normals were used to produce the final result.

REFERENCES

Bernardini, F., Rushmeier, H., Martin, I. M., Mittleman, J., and Taubin, G. (2002). Building a digital model of michelangelo's florentine pieta. *IEEE Computer Graphics and Applications*.

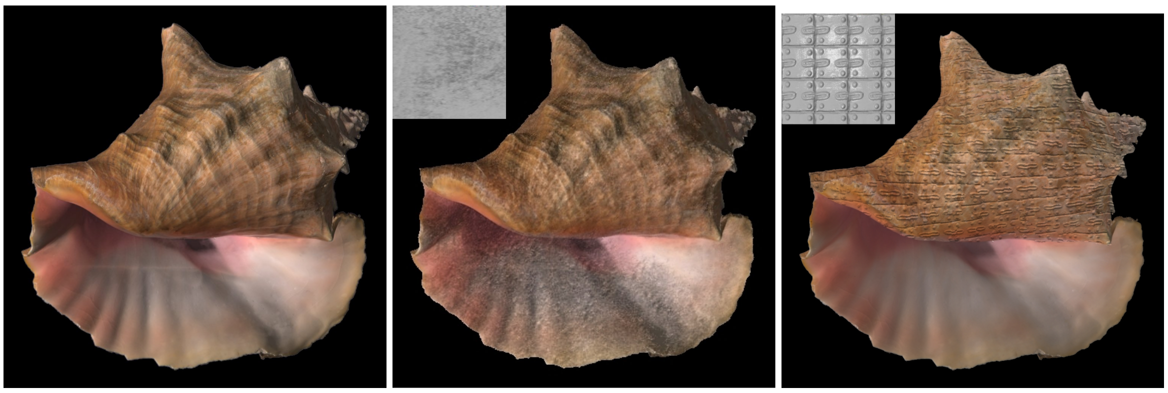

(a) Original Shell (b) Rust Normals (c) Armor Normals

Figure 12: The small images show the exemplars used under a direct light source. Images are embedded in high resolution.

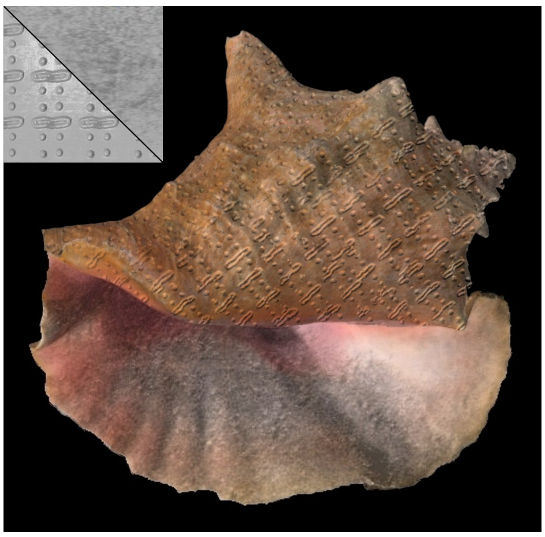


Figure 13: Structured texture synthesis.

Chen, T., Goesele, M., and Seidel, H.-P. (2006). Mesostructure from specularity. In *IEEE CVPR '06*.

Cignoni, P., Montani, C., Scopigno, R., and Rocchini, C. (1998). A general method for preserving attribute values on simplified meshes. In *VIS '98: Proceedings of the conference on Visualization '98*. IEEE Computer Society Press.

Cohen, J., Olano, M., and Manocha, D. (1998). Appearance-preserving simplification. In *SIGGRAPH '98*. ACM.

Efros, A. and Leung, T. (1999). Texture synthesis by non-parametric sampling. In *ICCV '99*. IEEE Computer Society.

Efros, A. A. and Freeman, W. T. (2001). Image quilting for texture synthesis and transfer. In *SIGGRAPH '01*. ACM.

Fang, H. and Hart, J. C. (2004). Textureshop: texture synthesis as a photograph editing tool. In *SIGGRAPH '04*. ACM.

Felzenszwalb, P. F. and Huttenlocher, D. P. (2004). Efficient graph-based image segmentation. *Int. J. Comput. Vision*, 59(2):167–181.

Haro, A., Essa, I. A., and Guenter, B. K. (2001). Real-time photo-realistic physically based rendering of fine scale human skin structure. *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*.

Kautz, J., Heidrich, W., and Seidel, H.-P. (2001). Real-time bump map synthesis. *HWWS '01: EUROGRAPHICS workshop on Graphics hardware*.

Mount, D. M. (1998). Ann programming manual. Technical report.

Pereira, T. and Velho, L. (2009). Rgbn image editing. *Proceedings of SIBGRAPI*.

Toler-Franklin, C., Finkelstein, A., and Rusinkiewicz, S. (2007). Illustration of complex real-world objects using images with normals. In *NPAR*.

Turk, G. (2001). Texture synthesis on surfaces. In *SIGGRAPH '01*. ACM.

Velho, L. and Sossai Jr., J. (2007). Projective texture atlas construction for 3d photography. *Vis. Comput.*, 23(9):621–629.

Wei, L.-Y. and Levoy, M. (2001). Texture synthesis over arbitrary manifold surfaces. In *SIGGRAPH '01*. ACM.

Woodham, R. J. (1989). Photometric method for determining surface orientation from multiple images. pages 513–531.

Zelinka, S., Fang, H., Garland, M., and Hart, J. C. (2005). Interactive material replacement in photographs. In *GI '05: Proceedings of Graphics Interface 2005*.

Zelinka, S. and Garland, M. (2004). Jump map-based interactive texture synthesis. *ACM Trans. Graph.*, 23(4):930–962.

Zhang, J., Zhou, K., Velho, L., Guo, B., and Shum, H.-Y. (2003). Synthesis of progressively-variant textures on arbitrary surfaces. In *SIGGRAPH '03*. ACM.