Scene-Based Geometry Synthesis

Jonathan Bronson*
UMBC

Abstract

Digital artists have increasingly been required to design and create ever larger and more detailed virtual environments. The purpose of these environments ranges mostly between cinema and game development, but the challenges are the same in both; to create large amounts of detail in an unrealistically short timescale. Fractals are one instance of a tool being utilized to speed up production time, mostly for landscape generation.

We present an alternative approach to generating unique and believable large-scale outdoor scenes. Through the utilization of texture synthesis and geometry synthesis techniques, it is possible to synthesize large natural environments based on a small predefined subregion. In this manner, an artist is only required to create a small portion of the product, and the algorithm will generate a full-scale scene with the same properties as the example. If built robustly enough, this system could dramatically improve production time of immersive movies and video games.

Keywords: texture synthesis, geometry synthesis, scene modeling, terrain generation, scene generation

1 Introduction

With the increase in users of high end graphics systems, the demand for ever large and more complex virtual environments also grows. Similarly, box office hits are increasingly taking advantage of graphics systems to place viewers into imaginary worlds of increasing beauty and realism. Both of these products are under the real-world time constraints for their development cycles. There are a fixed number of artists with a fixed amount of time to work. Allowing artists to focus on the style and important detail that is necessary will ultimately result in better movies and games.

The field of terrain generation has grown considerably, and is now quite believable in most aspects. One of key areas which is lacking however is the relation between these maps and the elements within them. Artists are forced to either hand place rocks, plants, and structures, or use some large scale randomized algorithm. The first choice is not only tedious but sometimes impossible due to sheer magnitude. The second choice works well for features with a simple distribution, such as flower fields, but fail miserably when complex environments arise. In such models, there is no way to account for flowers which might only grow on the edge of water. Similarly, there may be combinations of plants which grow well together and poorly apart.

*jonbronson@umbc.edu

We present a system which allows artists the ability to specify the relation between all models in a scene. Varying sizes of input can be used depending on the complexity of the environment being generated. For example, a large input area with minimal features would be used to generate regions of low and high density. Conversely, a small area would synthesize a more uniformly dense region of elements.

One could argue that an algorithm could also produce specific patterns of objects on a large scale. While this may sometimes be true, the relationship between a function and the output is not as clear as a picture. An artist cannot be expected to anticipate the outcome of a complex fractal equation before rendering it. By basing our generation off input scene geometry, we can create our new scenes based on human intuition. Artists can create the exact atmosphere they wish, and then carry it through to a much larger scale. The same cannot be said for other approaches.

Patch based synthesis techniques in particular lend themselves very well to this task, because they are particularly good at maintaining structure within the image. Therefore, the majority of work in this approach is in defining a clear mapping of the 3D scene, to a reasonable texture and back. Our work focuses on what is a good mapping, and what type of environments we could synthesize. The transition from 3D to 2D could be made trivially easy, however, if we do not take extra effort to define this direction properly, we may leave ourselves with too little information to accurately reconstruct the 3D equivalent.

2 Related Work

The methods used in this paper derive from traditional texture synthesis techniques. After a review of a range of synthesis techniques, it became clear that a patch-based method would be the most appropriate. These methods tend to be much faster than per-pixel approaches [Efros and Freeman 2001; Praun et al. 2000], and it is not vital to our goal that the result be as visually coherent and new. As long as the extraction algorithm is able to successfully reconstruct the output geometry, some disparity is acceptable. These patch-based methods are also particularly good at capturing structures, which is something that we absolutely want in our system.

In order for some scenes to be fully synthesized, it will be necessary to synthesize geometry appropriately. There have been two main approaches to this. One of them is to define some input texture which you wish to permute the surface of a model [Lai et al. 2005; Bhat et al. 2004]. This approach would be good if landscape is already defined, but the artist wants specific control in key areas. An alternative method is to take input geometry and interconnect it in shellspace over some surface [Zhou et al. 2006]. This approach will be more useful to us if we wish to generate new landscape ourselves.

There has been some very interesting work in giving artists more control over how synthesis proceeds, and this could lend itself well to the toolset. [Hertzmann et al. 2001] allow the user to paint key areas in what will become the new image, to direct the flow of the texture synthesis. This would be ideal for the type of work artists will use this for.

3 Approach

In order for us to take advantage of texture synthesis, we need to represent our input in the form of a texture. This is an interesting switch since our desire is to begin with some 3D modeled scene. Therefore, our first major task is to define a mapping into 2D space. We will test three different methods: Predefined Glyph Mapping, Generative Glyph Mapping, and Projection Rendering.

The first method, Predefined Glyph Mapping, will define a clear 1-to-1 mapping for each model and glyph. These objects will be centered in coordinates of the texture corresponding to coordinates of the object in the input scene. This is a very limited approach but its simplicity will make it a good benchmark. If we take this one step farther, we can generate a composite glyph to store more information. An N by N square glyph can store N^2 parameters if we store them naively. We could increase the capacity three (four) fold by exploiting RGB(A). In fact, we need not limit ourselves at all if our implementation is stand alone. Since we never lay eyes on this intermediate image, it need not be a legitimate image file. We may define an arbitrary number of fields, and store an arbitrary amount of data. However, by spreading the data across multiple pixels we provide the chance that features from different objects will cross-pollinate their attributes.

The final method, Projection Rendering, will render a top down orthographic view of the input scene into a texture. This approach will offer the most informationally detailed view of the scene. At the same time, it is also the most problematic. Without a uniform glyph shape we no longer have a clear origin point for the object. We also no longer have a clear way to scan the glyph to read data. In this type of mapping, it appears necessary to repeat all data across all pixels of an object. Unfortunately, this will remove the chance of crossing object attributes. For all three approaches, a separate texture will be used to represent the ground. Height map data is inherently different in that it is continuous across a scene block.

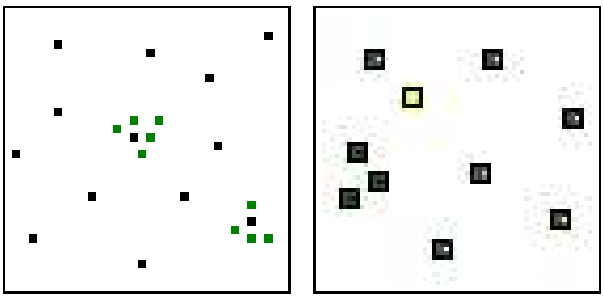
Once a suitable input texture has been created we can proceed with a chosen synthesis method. The result of this need not be beautiful, but must allow our system to reconstruct from each glyph certain key information. We must be able to detect at a minimum, model id and possibly directional orientation. Once such information is determined, we can reconstruct the final scene in several manners.

For terrain, we can use the heightmap data to reconstruct our terrain into whichever format we wish to render. For the purposes of this project, we will keep it in heightmap form. For geometry, we can either use orientation given by the synthesis, or randomly perturb it. The elements of the scene will dictate which is more appropriate. To get the variety of models we truly want, some of our parameters can be scaling and skewing factors. It might also be desirable to define groups of objects, and randomly pick between a subset. This will offer more variety across large areas such as forests. In this way, an artist does not have to place all models of a particular tree in their sample input to guarantee variety.

3.1 Seed Scene Creation

The method used to create the seed sample scene is irrelevant to this algorithm. We will restrict our scenes to the simple case for the sake of clarity. That is, input scenes will be in the format of an NxN area of arbitrary height. The texture which will represent this scene will be proportional in size. For our implementation, we define a sample as a simple scene graph. Locations and orientations of all models within the sample are stored numerically. A heightmap texture is stored to represent ground detail if needed.

To generate the sample texture from this scene, we iterate through all models within the scene graph. Horizontal coordinates relative to the scene block are used to place the corresponding glyph in the texture relative to the size of the texture. The larger a glyph is to be, the larger the sample texture will need to be to ensure no two glyphs overlap and obscure one another.



(a) (b)

Figure 1: Image (a) is an example of a 1-to-1 object model mapping. No extra information is stored other than ID. Image (b) is an example of a Generative Mapping. Each Glyph within the texture contains multiple pixels of varying data.

Figure 1 shows sample input textures for the first two glyph mappings. Texture (a) is a 1-to-1 Mapping, where color of the glyph represents model ID. Texture (b) is a Generative Mapping, where each glyph contains a number of pixels, each of which stores some parameters for the object.

3.2 Scene Synthesis

Once we have transformed our sample scene into an input texture, normal Texture Synthesis may proceed in its standard way. We implemented both pixel-based and patch-based techniques to compare their success under our restrictions. The two approaches have various strengths and weaknesses which we will not going into unless they apply directly to our method.

The advantage of doing a pixel-based approach is in the accuracy of any given pixel. It would seem that this would provide the smoothest transition across an image. However, we are forced not to implement an Image pyramid. The reason for this is that any change to color information will ultimately change the data stored and may misrepresent the reconstruction. This restriction means that we must capture all image structure using neighborhoods. This also means neighborhoods must become extremely large and our synthesizing time quickly becomes unwieldy. The whole point of this approach is to generate extremely large environments, and we see a pixel-based approach would be unreasonably slow.

A patch-based method will also capture large scale structures while still providing us with the speed we desire. The remainder of the scene synthesis consists of observations and results taken from performing a patch-based synthesis on the mapped glyphs. We define the output size of the synthesis to be proportional to the 3D area we wish to synthesize our geometry over. While there is no restriction on the output size, computational time will increase linearly with respect to each dimension.

3.2.1 Full Scene Reconstruction

Once the final scene texture has been synthesized, we can begin to see the strengths and weaknesses of each glyph mapping. All three

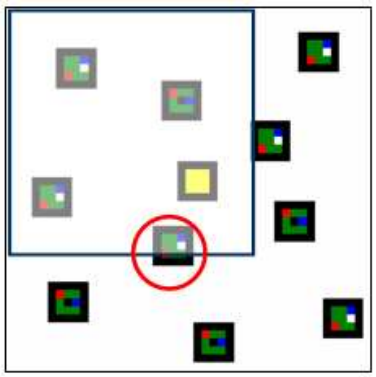


Figure 2: During patch-based texture synthesis no guarantee can be made that the best chosen patch does not intersect and chop off a piece of a multipixel glyph. The problem is less severe for sparse scenes and unavoidable for dense scenes. The glyph circled in red will be cut and not appear in whole in the synthesized image.

mappings will have a similar structure for reconstructing the output scene. The synthesized texture will be read in scanline order. Whenever a feature is detected, (by color variance), it will be examined and the appropriate geometry will be placed into the new scene graph. The manner in which the data is extracted from the glyph is what differs between these methods. A 1-to-1 Mapping needs only examine one pixel, at which point it can tag the rest as already reconstructed. The Generative Map must examine each pixel within the glyph, in scan-line order relative to the glyph. Once all parameters of the object are known, it too can place the correct model into the scenegraph with the appropriate properties. The Projection Mapping will read a single pixel, similar to the 1-to-1 mapping, but rasterization might make it difficult to tell if there are more than one object. A closing convolution in the original input texture might alleviate this problem.

While the 1-to-1 mapping seems to fair off well no matter which synthesis method is used, the Generative Mapping is shown to be problematic. The problem comes in several forms but is all based on the same principal. In order for our reconstruction to be successful, we must be able to read the entire glyph's properties. Unfortunately, the patch-based sampling cannot guarantee these glyphs stay whole. The first issue comes up when we pick a close match patch from the input image. The best match is only computed for the overlapping portions of the patch. This match may cause the chosen patch to cut off glyphs on the right and bottom sides, as shown in Figure 2.

The second scenario which hurts these multipixel glyphs occurs after the best patch is chosen. A best-cut is performed to provide as seamless a transition from patch to the next. This cut path is chosen by pixels which differ by the least amount of error. This means that even if we have perfectly overlapping glyphs, we will not have a cross pollination between them. The best cut will stay on the background surface and avoid any cut at all, choosing all of one glyph. Furthermore, this overlap region puts a restriction on the size of these glyphs. As Figure 3 shows, if a glyph is equal to or wider than the size of the overlap, it will be cut in half no matter what. This only works in our favor if we happen to have perfectly overlapping glyphs, both of which are equal to the overlap size. This is a very unlikely event, and we can see that the multipixel glyph is not a strong candidate for use in scene synthesis.

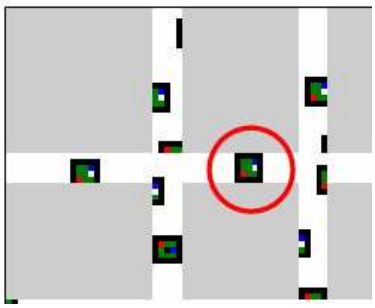


Figure 3: This figure shows how the glyph width can be problematic during path-based texture synthesis. The grey areas are those not overlapped, while the colored regions are those which exist before an overlap occurs. The glyph circled in red takes up the entire overlap region and is guaranteed to be cut by the "best cut" of the patch-based synthesis.

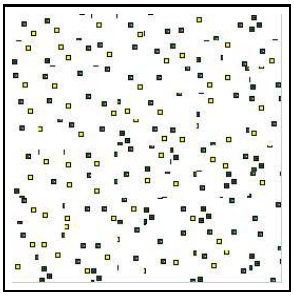


Figure 4: The image shows the output of a patch-based synthesis on the sample input provided from Figure-1(a). The implementation is not complete but it is clear that structure is being maintained.

4 Results

Preliminary results so far are encouraging. At first attempt pixel-based synthesis was used to generate new output textures. An Image Pyramid was not used so as to not change any of the data. However, without this tool to capture large structures, extremely large neighborhoods are needed to synthesize a good output. Since the original Wei and Levoy [2000] paper used both an Image Pyramid and clustering techniques to provide fast synthesis, it was apparent that this method should be abandoned.

An attempt using a patch-based synthesis worked much better. It also revealed that the multipixel generative glyph is not fit for use in this method, though it is still clear from figure 4 that structure is being maintained. The 1-to-1 pixel glyphs suffer from no such problem. Figure 5 shows its success in texture synthesis.

The results when using this 1-to-1 mapping with geometry data was very encouraging Figure 6 shows an example patch of forest given as input. The result is a much larger synthesized forest across the surrounded landscape. The overall structure of the patch is maintained by the synthesis and the extension is seamless. By providing more data to the glyphs we can achieve better results such as color variations among trees. Figure 7 shows a different sample scene generated with varying degrees of colors on the trees.

The two examples given so far of a relatively moderate degree of density. We start to see very different effects as we go in either direction. As one might expect, a less dense sample patch leads to more repetative synthesis. There is simply not enough variation for the patch matching to go by. We must artificially add more ran-

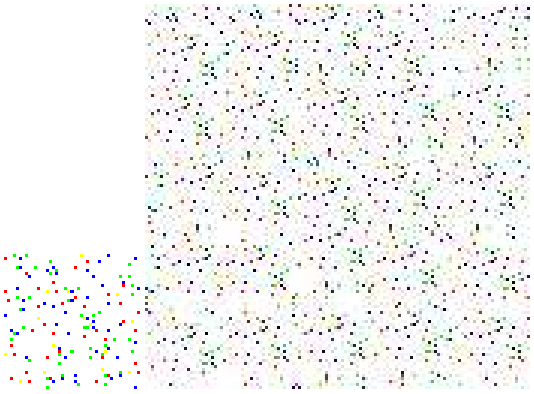


Figure 5: The image shows the success of a single pixel representation of a scene. In this naive mapping, each color corresponds to different input model, with no additional information stored. An arbitrary amount of information can be placed in a single pixel by extending the RGBA channels to N-dimensions.

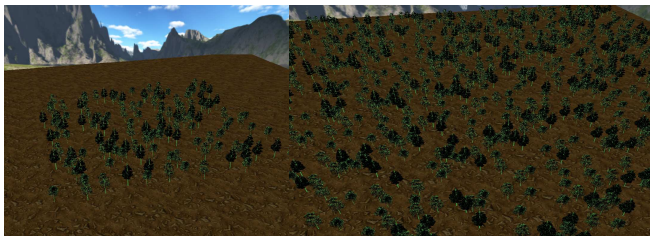


Figure 6: The image on the left is the input patch which would be provided by an artist. This patch is meant to be an accurate description of what an arbitrary piece of the scene would look like. To the right is a synthesized extension of this patch across the surrounding landscape.

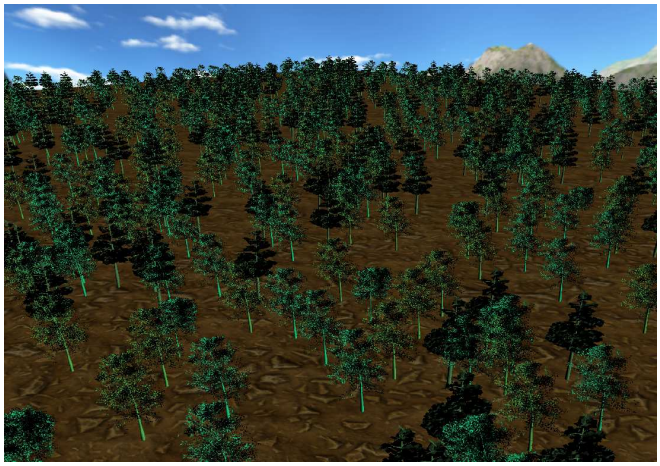


Figure 7: The scene above was synthesized from a sample patch packed into a 64x64 pixel texture. Color variations are added to provide improved visual impact. The structure of this forest is also clearly different than the one synthesized in figure 6

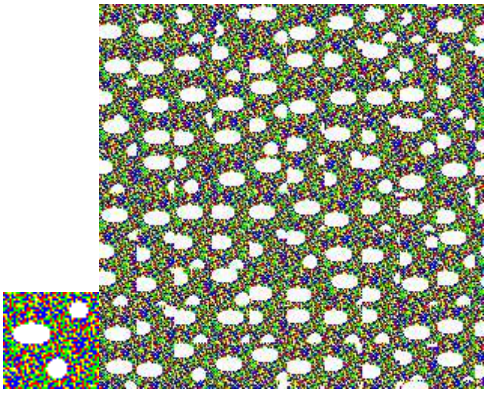


Figure 8: The sample input provided is a very dense forest area. We pack them as tightly as possible into a texture map and generate a synthesized texture map. The texture output very accurately portrays the characteristics of the input.

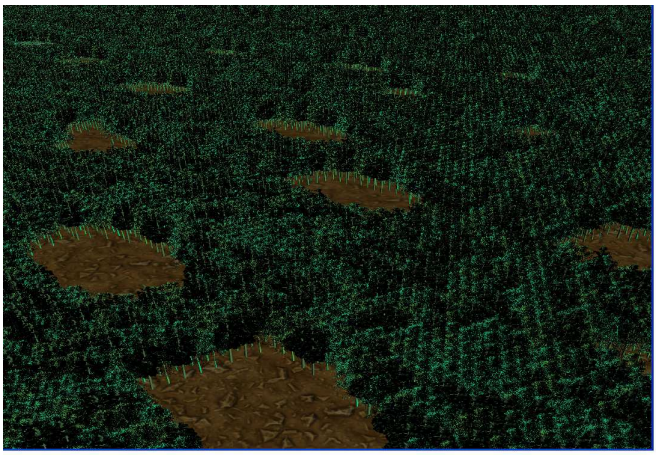


Figure 9: The synthesized texture from Figure 8 is used to generate the scene shown. The discrete distances of the pixels are clearly visible in the placement of the tree bases.

domness to to counteract this affect. What occurs when we travel in the other direction is not so obvious. The precision of the measure of relationship between models in the scene is defined by the scale of the texture and the disceteness of the pixels that the texture is comprised of. In order to capture the idea of open fields within forests, we provide an input sample with as high density as possible, to ensure the patch selection properly handles these open areas. The result of this texture synthesis is shown in Figure 8. As we follow this reconstruction through to the geometry stage, we see in Figure 9 that the same structure is visible. However, it is very obvious that our trees are now on a gridlike pattern formed by the pixels. To avoid such scenarios, we can either enforce the use of larger textures to capture finer granularity, or use composite models. An example of the use of composite models is shown in figure 10. Here we again have a completely dense image except it is no longer apparent from the view of the scene.

5 Conclusion & Future Work

As we have shown through the work presented, there is clearly a place for such synthesized scenes. The use of texture synthesis as

Figure 10: In this image a composite model is used to counteract the discrete pixel sampling issue. In this image there is no visible pattern of model placement.

a means will require more extensive research. The main reason the synthesis behaves so unexpectedly is we are trying to map a discrete quantity to a continuous texture, and then map that continuous texture back onto a discrete sampling of pixels. Through this layered process we lose a lot of information and a lot of flexibility.

If this technique can be implemented fast enough it would be an excellent tool to use in the background of a 3D editing tool. It could behave like a copy/paste tool in which an input region is highlighted, and then "pasted" over a larger output area. Except, in this case, the pasted output would be the synthesized scene. Future work includes the investigation of this technique with overlapping geometry as well as the further improvement of the structure capturing. It seems evident from the results so far that the approach of standard texture synthesis does not exactly match the goals of this scene synthesis.

References

BHAT, P., INGRAM, S., AND TURK, G. 2004. Geometric texture synthesis by example. In *"SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing"*, ACM Press, New York, NY, USA, 41–44.

EFROS, A. A., AND FREEMAN, W. T. 2001. Image quilting for texture synthesis and transfer. In *"SIGGRAPH '01: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques"*, ACM Press, New York, NY, USA, 341–346.

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

LAI, Y.-K., HUA, S.-M., GU, X., AND MARTIN, R. R. 2005. Geometric texture synthesis and transfer via geometry images. In *"SPM '05: Proceedings of the 2005 ACM symposium on Solid and Physical Modeling"*, ACM Press, New York, NY, USA, 15–26.

PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. 2000. Lapped textures. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 465–470.

WEI, L.-Y., AND LEVOY, M. 2000. Fast texture synthesis using tree-structured vector quantization. In *"SIGGRAPH '00: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques"*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 479–488.

ZHOU, K., HUANG, X., WANG, X., TONG, Y., DESBRUN, M., GUO, B., AND SHUM", H.-Y. 2006. "mesh quilting for geometric texture synthesis. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, ACM Press, New York, NY, USA, 690–697.