

# Fast Hybrid Approach for Texturing Point Models

Haitao Zhang      Feng Qiu      Arie Kaufman

Center for Visual Computing (CVC) and Computer Science Department  
Stony Brook University  
Stony Brook, NY 11794-4400, USA  
haitao | qfeng | ari@cs.sunysb.edu

---

## Abstract

We present three methods for texturing point models from sample textures. The first method, the point parameterization method, uses a fast distortion-bounded parameterization algorithm to flatten the point model's surface into one or more 2D patches. The sample texture is mapped onto these patches and alpha blending is used to minimize the discontinuity in the gaps between the patches. The second method is based on neighborhood matching where a color is assigned to each point by searching the best match within an irregular neighborhood. The hybrid method combines the former two methods, capitalizing on the advantages of both. The point parameterization method is used first to color most of the points, and the point neighborhood matching method is then applied to the points belonging to the gaps between the parameterized patches to minimize the discontinuity. We opt for fast texture generation, while some discontinuities may appear in the gaps of anisotropic textures.

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

---

## 1. Introduction

Coloring an arbitrary surface from a sample texture can be achieved by existing texture synthesis algorithms [GIS01, Tur01, WL01, YHZ01]. These algorithms use neighborhood matching to find a best match among all pixels of the sample texture. Speed is a concern for this brute force search although there are some methods to speed up the search process while compromising the synthesized texture quality. We call these algorithms neighborhood matching methods. Another kind of methods for texturing arbitrary surfaces is to cover the surfaces with patches taken from the sample texture [PFH00, SCA02]. A local parameterization is created for each patch on the surfaces and the texture is mapped directly onto that parameterized patch. Distortion and discontinuity are the main artifacts that should be minimized. These methods are referred to as texture placement [Tur01] or pattern mapping [SCA02]. We call these approaches parameterization methods.

With the development of 3D scanning technologies, representing an object by a set of surface points has become an important 3D model format. Levoy and Whitted [LW85]

have proposed the use of points as a display primitive and as a universal meta-primitive for objects modeling. Grossman [Gro98] has presented a point-based rendering system for sampling and rendering. Surfels [PZvBG00] and QSplat [RL00] use multi-resolution data structures to organize the point samples and devise rendering algorithms based on their data structure. Alexa et al. [ABCO\*01] use point sets to represent shapes and apply local parameterization for multi-resolution rendering. Zwicker et al. [ZPvBG01] have used EWA splatting for rendering. This technique can be accelerated by hardware [RPZ02]. Pauly and Gross [PG02] have introduced some surface simplification algorithms for point models. PointShop3D [ZPKG02] is a point model editing system which supports editing operations directly on the point models. Texture mapping over a surface patch is implemented by using a minimum distortion patch parameterization algorithm.

Currently, nearly all texture synthesis and texture parameterization methods for 3D models are applied to surface meshes. Alexa et al. [AKS03] have introduced a method for establishing local frames over point models, which is used



**Figure 1:** Left: original point model of Santa and four sample textures; Right: new textures synthesized over four parts of the model using the sample textures. The hat texture is synthesized using our hybrid method; the coat and trousers textures are synthesized using our point parameterization method; and the shoes texture is synthesized with our point neighborhood matching method.

for texturing point models with a neighborhood matching method. For a point model, one way to texture it is to triangulate it into a mesh and apply an existing method for meshes. Since point models have no coherence information, triangulation is not trivial. After triangulation, parameterization of the 3D mesh surface is still needed in texturing the surface. It is better to devise efficient methods to recover coherence information required for texturing directly from point models. Some operations needed in texturing, such as model hierarchy construction are much easier and more efficient when applied to point models. In this paper, we combine and extend existing texturing methods and apply them to point models. Given a point model and a sample texture, every point of the model is colored properly so that the surface of the point model shows a texture pattern similar to the sample texture. We focus on the techniques for fast texture generation over point models from isotropic sample textures. For the anisotropic textures, some discontinuities may appear in limited areas of the models.

We present a point parameterization method (Section 3), based on a fast distortion-bounded parameterization algorithm to flatten the point model’s surface into one or more patches. The sample texture is mapped onto these patches and alpha blending is used to fill in the gaps between patches. The drawback to this method is that discontinuity between patches is unavoidable. In Section 4 we present an alternative method, point neighborhood matching, which eliminates

this problem. In our point neighborhood matching method, an irregular neighborhood is created for every point from its neighboring points. This is used for finding the best match to color the point. A point model hierarchy is built for multi-resolution synthesis. This method yields seamless textures but is compute intensive. In order to achieve fast texturing, we present a hybrid method (Section 5) to capitalize on the benefits of both. The point parameterization method is used first to color most of the points, followed by the point neighborhood matching method for coloring points in the gaps between patches to get a seamless surface texture.

## 2. Previous Work

### 2.1. Surface Neighborhood Matching Methods

The surface neighborhood matching methods for 3D models have been inspired by the success of the 2D texture synthesis [Ash01, Bon97, EL99, WL00]. Based on Markov random field model, De Bonet [Bon97] has synthesized new textures by sampling successive spatial frequency bands from the sample texture. Efros and Leung [EL99] have simplified this pixel-by-pixel synthesis method by taking pixels directly from the sample texture. The region of neighboring synthesized points is compared with the similar regions in the sample texture, and the pixel is colored using the best match. The search for the best match is an expensive process whose run time is proportional to the size of the sample texture. Wei and Levoy [WL00] have used a similar idea for a

multi-resolution synthesis method to deal with textures with large patterns. The shape of the neighborhood is fixed so that a tree-structured vector quantization can be used to accelerate the neighborhood matching process for the price of a slight decrease in texture quality. Ashikhmin [Ash01] has proposed coherent synthesis, which restricts the search space for the neighborhood matching to the locations near the best matches of the already synthesized neighboring pixels. Thus, the computation is reduced greatly and it is rather successful in synthesizing textures with highly structured patterns, but discontinuity-free results cannot be guaranteed.

The 2D neighborhood matching methods have been extended to 3D mesh surfaces [GIS01, Tur01, WL01, YHZ01]. For each vertex, a fixed shape neighborhood is created by interpolating in the local parameterization near the vertex. The orientation of the local parameterization is determined by a vector field specified by the user or created randomly. A mesh hierarchy is built for multi-resolution synthesis. Similar to the 2D case, the neighborhood matching process is still the bottleneck. Ying et al. [YHZ01] have also extended the coherent synthesis method to 3D mesh. Tong et al. [TZL\*02] have used a neighborhood matching method to synthesize bidirectional texture functions over meshes based on surface textures. Alexa et al. [AKS03] have extended the neighborhood matching method to point models based on their techniques for generating direction fields over point-sampled geometry. In this paper, we extend the surface neighborhood matching methods to 3D point models by using an irregular neighborhood.

## 2.2. Surface Parameterization Methods

The textures generated by coherent synthesis [Ash01, YHZ01] are actually a group of patches taken from sample textures that are tiled together. Instead of synthesizing vertex by vertex (pixel by pixel for 2D), surface parameterization methods create textures patch by patch and minimize the discontinuities between patches. Efros and Freeman [EF01] have presented image quilting, a parameterization method for 2D texture synthesis. They find a minimum error boundary cut in the overlapping blocks to minimize the discontinuity. The patch-based sampling method proposed by Liang et al. [LLX\*01] tries to minimize the discontinuity by searching the best patch in the sample texture. Kwatra et al. [KSE\*03] have used a graph cut technique to stitch together irregular patches along optimal seams. Nealen and Alexa [NA03] have used a hybrid method for 2D texture synthesis by re-coloring the overlapping regions after the parameterization method. The lapped textures method proposed by Praun et al. [PFH00] covers a 3D mesh surface by pasting irregular texture patches onto it. Alpha blending is used in the regions where patches overlap. Soler et al. [SCA02] have presented a hierarchical method for texturing arbitrary mesh surfaces.

Patches are selected to minimize the fitting error with already textured neighboring patches. If the fitting error is too large, the patch is split into smaller ones. Dischler et al. [DMLG02] have presented an analytical extension of the lapped textures method. They first segment the sample texture image into elementary components called texture particles. These texture particles are then recomposed into surfaces according to spatial arrangements computed from the sample texture. In this paper, we present a fast parameterization method for point models. Furthermore, we extend the 2D hybrid method to point models in order to solve the discontinuity problem.

## 3. Point Parameterization Method

In this section, we describe our point parameterization method for texturing point models based on a fast distortion-bounded point surface parameterization algorithm.

### 3.1. Algorithm Overview

First, we introduce some definitions which are used throughout this paper. A point model is represented by a set of surface points :

$$PM_n = \{p_i | p_i = (d_i, n_i, c_i), i = 1 \text{ to } n\} \quad (1)$$

where  $n$  is the number of points,  $d_i$ ,  $n_i$  and  $c_i$  are the position, normal and color of the  $i$ th point  $p_i$ . We define  $N_k(p_i)$  as the  $k$ -neighborhood of point  $p_i$ , which is the set of the first  $k$  nearest points to point  $p_i$ . The points are organized by a block data structure. The bounding box of the point model is segmented into regular blocks and each block contains no more than 100 points. Thus, finding  $N_k(p_i)$  is fulfilled efficiently by searching in the blocks near  $p_i$ . The goal of texturing the point model  $PM_n$  is to assign the color value  $c_i$  for each point  $p_i$  according to a sample texture.

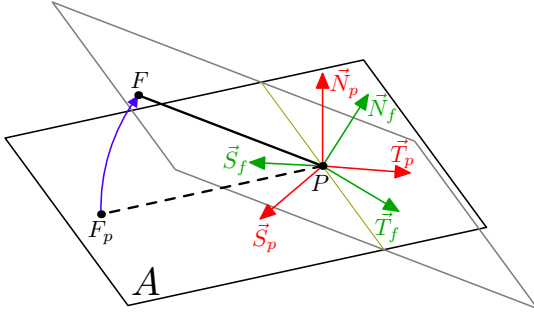
Our parameterization algorithm flattens the surface of the point model  $PM_n$  into one or more 2D patches. It is a region-growing method which expands the patches by adding neighboring points according to some criteria. There are three kinds of points: *free* points, *patch* points, and *gap* points. Initially, all points are *free* points. A patch is started from a seed point randomly chosen from the *free* points. Then, a *free* point near the patch is selected for evaluation. If the distortion caused by adding this *free* point to the patch is within a certain threshold, it becomes a *patch* point. Otherwise, it becomes a *gap* point. The region cannot be grown beyond these *gap* points. When no *free* points can be added to the current patch, the iteration for the next patch begins if there are still *free* points left. Once the iterations end, one or more patches have been created and every point is classified as either a *patch* point or a *gap* point. The *gap* points separate neighboring patches or two different parts of the same patch. The sample texture is mapped onto the patches and the *gap* points are textured by alpha blending. This parameterization algorithm is similar to the mesh parameterization

presented by Sorkine et al. [SCOGL02]. Since we are dealing with point models and the goal of our parameterization is for texturing, the two algorithms differ in major steps such as criteria for adding points, distortion calculation, and self-intersection prevention.

### 3.2. Patch Growing with Bounded Distortion

Each *patch* point  $p_i$  is associated with a position  $q_i$  on the 2D patch and the orientation  $\langle \vec{N}_i, \vec{S}_i, \vec{T}_i \rangle$  of the patch. When the patch is created from a seed point  $p_j$ , it is the tangent plane at point  $p_j$ . Thus,  $\vec{N}_j = n_j$  and  $\langle \vec{S}_j, \vec{T}_j \rangle$  is a pair of orthogonal vectors on the tangent plane which determines the orientation of the texture. We assign the orientation randomly or use the projection of the up vector  $(0, 1, 0)$  on the tangent plane as  $\vec{T}_j$ . A local coordinate system of the patch is established with the seed point  $p_j$  as its origin and  $\langle \vec{S}_j, \vec{T}_j \rangle$  as its axes.

A *free* point  $F$  with the maximum number of *patch* points in its  $k$ -neighborhood  $N_k(F)$  is selected as the candidate *patch* point. A priority queue of *free* points is maintained during the iterations, which uses the number of *patch* points in the  $k$ -neighborhood as the key for sorting. If  $p_i \in N_k(p_j)$ , it is very likely that  $p_j \in N_k(p_i)$ . We update the priority queue in this way: after adding a point  $p_i$  to the current patch, all the *free* points in  $N_k(p_i)$  increase their keys by 1. Thus, we can efficiently get the candidate *patch* point for the patch expansion.



**Figure 2:** Computing position and orientation of candidate patch point  $F$  according to a neighboring patch point  $P$ .

The position  $q_f$  and orientation  $\langle \vec{N}_f, \vec{S}_f, \vec{T}_f \rangle$  of the candidate *patch* point  $F$  can be computed from each of its neighboring *patch* points. Figure 2 illustrates how to compute  $q_f$  and  $\langle \vec{N}_f, \vec{S}_f, \vec{T}_f \rangle$  of  $F$  from one of its neighboring *patch* points  $P$ . The plane  $A$  is defined by the three orthogonal vectors  $\langle \vec{N}_p, \vec{S}_p, \vec{T}_p \rangle$  of  $P$  with normal  $\vec{N}_p$  and local coordinates  $\langle \vec{S}_p, \vec{T}_p \rangle$ .  $F_p$  is at the position  $q_f$  by projecting  $F$  to plane  $A$  followed by a scaling to make  $|F_p P| = |F P|$ . Then, plane  $A$  is rotated around  $P$  (the rotation axis goes through  $P$  and is orthogonal to  $F_p P$  and  $F P$ ) until  $F_p P$  overlaps with

$F P$ .  $\langle \vec{N}_p, \vec{S}_p, \vec{T}_p \rangle$  becomes  $\langle \vec{N}_f, \vec{S}_f, \vec{T}_f \rangle$  after the rotation. The formulas for computing  $q_f$  and  $\langle \vec{N}_f, \vec{S}_f, \vec{T}_f \rangle$  are given in Appendix A.

If there are  $m$  *patch* points in  $N_k(F)$ , we will get  $m$  positions and orientations:  $\{(q_f^l, \langle \vec{N}_f^l, \vec{S}_f^l, \vec{T}_f^l \rangle) | l = 1 \dots m\}$  for  $F$ . The final position and orientation for  $F$  is the weighted average of these values, where the weights  $w^l$  are inversely proportional to the distance between  $F$  and the corresponding *patch* points. The three vectors in the weighted average orientation  $\langle \vec{N}_f, \vec{S}_f, \vec{T}_f \rangle$  are nearly orthogonal since we restrict the distortion as discussed below.

There are two kinds of distortions in adding point  $F$ : position distortion and orientation distortion. We define the distortions using the average and maximum differences between the  $m$  values and the final value:

$$D_{pos}(F) = w_1 \frac{\sum_{l=1}^m \{w^l |q_f^l - q_f|\}}{\sum_{l=1}^m w^l} + \max_{l=1}^m \{w^l |q_f^l - q_f|\} \quad (2)$$

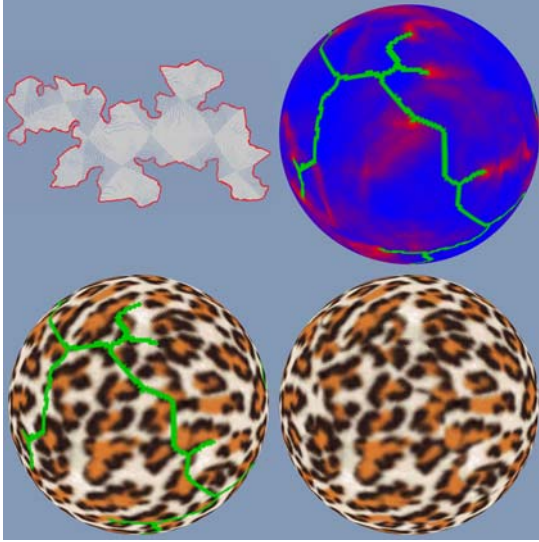
$$D_{ori}(F) = w_2 \frac{\sum_{l=1}^m \{w^l (2 - \vec{S}_f^l \cdot \vec{S}_f - \vec{T}_f^l \cdot \vec{T}_f)\}}{\sum_{l=1}^m w^l} + \max_{l=1}^m \{w^l (2 - \vec{S}_f^l \cdot \vec{S}_f - \vec{T}_f^l \cdot \vec{T}_f)\} \quad (3)$$

$$D(F) = w_3 D_{ori}(F) + w_4 D_{pos}(F) \quad (4)$$

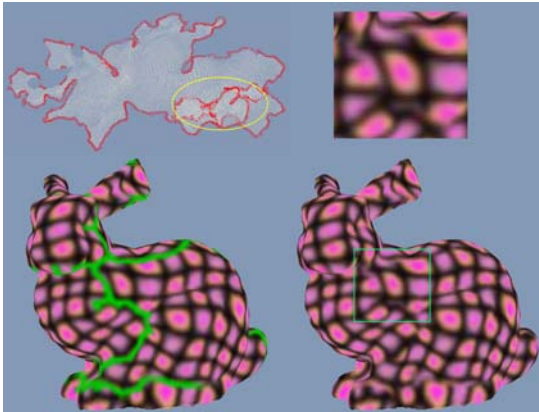
A unit length will be introduced later in Subsection 3.4, which is used for normalization of the position distortion to make it scale-independent. If the distortion  $D(F)$  is within a user-defined threshold,  $F$  becomes a *patch* point. Otherwise, it becomes a *gap* point. We use  $w_1 = w_2 = 1, w_3 = 3, w_4 = 2, threshold = 1.6$  without any noticeable distortions. Figure 3 and 4 show two examples of our parameterization method. We can see that this parameterization algorithm flattens the 3D surfaces by region growing and creating cuts in the regions where the distortion exceeds the threshold.

### 3.3. Self-intersection

There are two types of self-intersections: local self-intersections and global self-intersections. A local self-intersection occurs when a surface region folds over on the parameterized patch. When adding a point  $p_i$  to the patch, we only use those neighboring *patch* points  $p_j$  that satisfy  $n_i \cdot n_j > 0$ . No noticeable local self-intersection will appear in practice after adding this simple constraint. The global self-intersection occurs when different regions in the patch overlap. In Figure 4, there are global self-intersections in the parameterization of the bunny model. The overlapped regions do not overlap in the 3D model and they do not connect directly. The global self-intersections will not create any artifacts in the resulting texture. Therefore, in our method, we disregard global self-intersections.



**Figure 3:** Parameterization of a sphere point model. Top-left: parameterized patch (patch points near gaps shown in red). Top-right: color-coding of the parameterization distortion (red for high distortion and blue for low distortion). Bottom-left: before alpha blending (gap points shown in green). Bottom-right: after coloring the gap points.



**Figure 4:** Parameterization of a bunny point model. Top-left: parameterized patch (there are global self-intersections inside the yellow circle). Bottom-left: before alpha blending (gap points shown in green). Bottom-right: after coloring the gap points (there is discontinuity inside the green box, top-right is its close-up view).

### 3.4. Texturing

After the parameterization algorithm, points are classified into *patch* points and *gap* points. First, the *patch* points are colored by texture mapping. A unit length on the parameterized patch which is equivalent to the sidelength of a pixel in

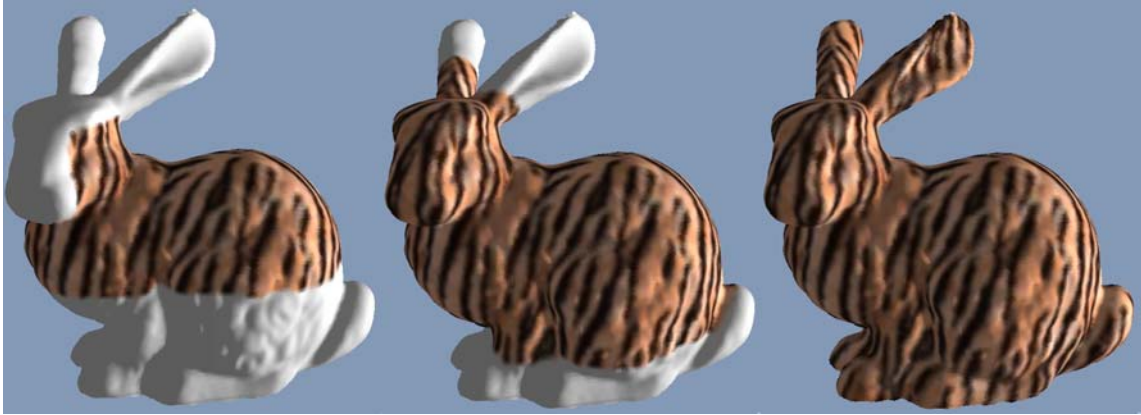
the sample texture should be determined. We make the point density on the patch roughly the same as the sample texture by taking the average distance between each point and its nearest point as the unit length. This unit length can be adjusted by the user to change the texture scale over the model. If the sample texture is tileable, the size of the parameterized patch is not restricted, which usually results in a single patch for the model. Then, a random position is chosen from the sample texture for the patch origin and the sample texture is tiled onto the patch. We need to restrict the size of the patches if the sample texture cannot be tiled. A size smaller than the sample texture is chosen as the maximum size a patch can grow to. In the parameterization algorithm, a *free* point cannot be added into the patch if it makes the patch size larger than the maximum size. Under this size restriction, more than one patch will be generated. A random patch within the sample texture is selected for each parameterized patch for the texture mapping.

The *gap* points are colored in an order similar to the candidate *patch* point selection of the parameterization algorithm: the *gap* point  $p_i$  with the maximum number of *patch* points in its  $k$ -neighborhood is the next point to color. For each neighboring *patch* point of the *gap* point  $p_i$ , a position on that patch for  $p_i$  can be calculated using the same method described above. The color in the corresponding sample texture is assigned to  $p_i$ . Alpha blending is used to blend all the colors that  $p_i$  gets. The weights used in the blending are inversely proportional to the distance from  $p_i$  to the corresponding neighboring *patch* points. The *gap* point becomes a *patch* point after coloring. As we can see in the result of bunny model in Figure 4, discontinuity in the textures with anisotropic patterns cannot be avoided by this parameterization method. This problem can be solved by our hybrid method discussed in Section 5.

### 4. Point Neighborhood Matching Method

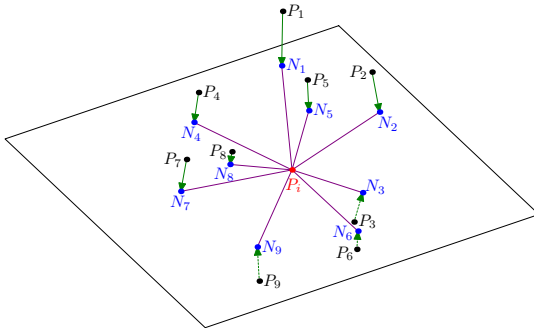
In this section, we describe our point neighborhood matching method for synthesizing textures over a point model. Wei and Levoy [WL01] and Turk [Tur01] use a regular square neighborhood in their neighborhood matching methods for mesh models, which is created by resampling in the local parameterization surrounding the vertex to be colored. Unlike the mesh model, where it is easy to find the three vertices of a triangle for resampling, the point model does not have such coherence information. It is a challenging task to create a regular neighborhood on the point model surface. Alexa et al. [AKS03] use a regular grid as the neighborhood in their texture synthesis method for point models, and the colors of close sample points are used for the neighborhood instead of resampling. We overcome this difficulty by using an irregular neighborhood and resample this neighborhood in the sample texture.

The irregular neighborhood is constructed by mapping the  $k$ -neighborhood points onto the tangent plane of the point to



**Figure 6:** From left to right: coloring 33%, 66% and 100% points of the bunny point model using a 25-neighborhood.

be colored, as shown in Figure 5. The mapping is exactly the same as computing the position of the *patch* point described in Section 3. Assuming the point density is roughly uniform



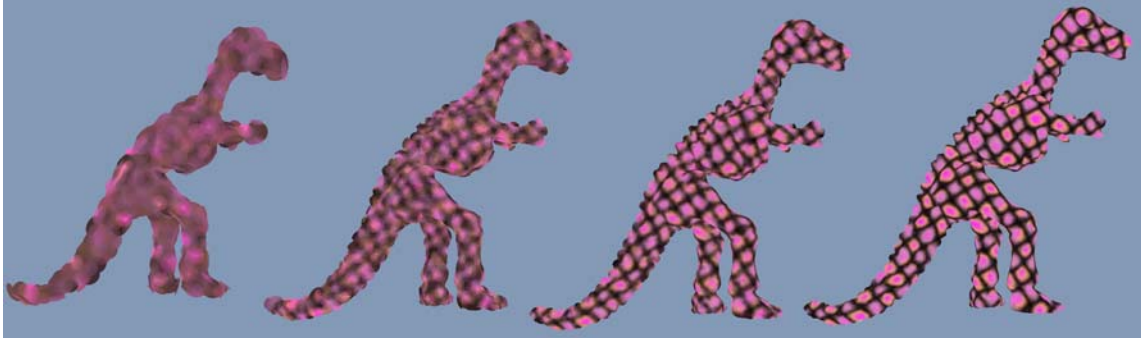
**Figure 5:** Irregular neighborhood of point  $p_i$  created from its  $k$ -neighborhood. In this example,  $k = 9$ ,  $N_9(p_i) = \{p_1 \dots p_9\}$  and  $\{N_1 \dots N_9\}$  is the irregular neighborhood of  $p_i$  over the model surface, which is a 2D manifold, the size of an irregular  $k$ -neighborhood is equivalent to the size of a  $\sqrt{k} \times \sqrt{k}$  square neighborhood. Since the points in the irregular neighborhood have in general non-integer coordinates, bilinear interpolation is needed to get the corresponding colors in the sample texture to be matched. For each point to be colored, interpolation is required for each pixel in the sample texture. This is time consuming compared to the regular neighborhood method with only one interpolation on the mesh per vertex. We supersample the sample texture of size  $A \times B$  using bilinear interpolation with scale  $m$  to create a new texture of size  $mA \times mB$ . In the neighborhood matching process, interpolation is avoided by using the nearest pixel in the new texture. Only those pixels corresponding to the original sample texture are searched for best match, so that the size of the search space remains the same although the texture size has been expanded. In practice,  $m = 8$  is enough to create the same quality texture as the result from using

bilinear interpolation in the sample texture. The texture orientation is assigned with a fixed or a random direction.

The synthesis order of the points is determined by maintaining a priority queue of uncolored points, which is very similar to the candidate *patch* point selection in our parameterization method. The key for sorting the priority queue is the number of the colored points in a  $k$ -neighborhood of the uncolored points. Figure 6 shows the synthesized texture after coloring 33%, 66% and 100% points of the bunny point model by using a 25-neighborhood.

#### 4.1. Multi-resolution Synthesis

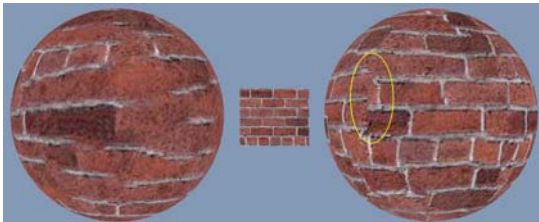
In order to use a relatively small neighborhood for sample textures with a large pattern size, and to capture patterns with different scales, a point model hierarchy should be created for multi-resolution synthesis [WL01, Tur01]. Unlike the mesh hierarchy construction, which needs to update the mesh connectivity and maintains surface consistency, constructing a point model hierarchy is simple. There are many algorithms to construct a point model hierarchy [PG02]. Since preserving accurate geometry is not an important issue for the hierarchy, which is created for texture synthesis, we use a fast and simple clustering method. Four near points are clustered together to form one point in the lower resolution so that the number of points in a lower resolution is one quarter of the number of points in the higher resolution. In addition, the point density in any area in the lower resolution is proportional to the point density in the corresponding area in the higher resolution. The un-clustered point with the maximum number of clustered points in its  $k$ -neighborhood is the next point for clustering. On some occasions, an un-clustered point cannot find three nearby un-clustered points. A new point is still created from this cluster although the number of points in the cluster is less than four. This occurrence is very rare because of the order of clustering. For example, a lower resolution model created from the Santa



**Figure 7:** Multi-resolution synthesis. The dinosaur point model is textured using 4-level hierarchy with a 30-neighborhood. The number of points in the four levels are 886, 3518, 14055 and 56194, from left to right, top to bottom, respectively.

model of 75781 points has 18949 points, among which only 6 points are from clusters with less than four points.

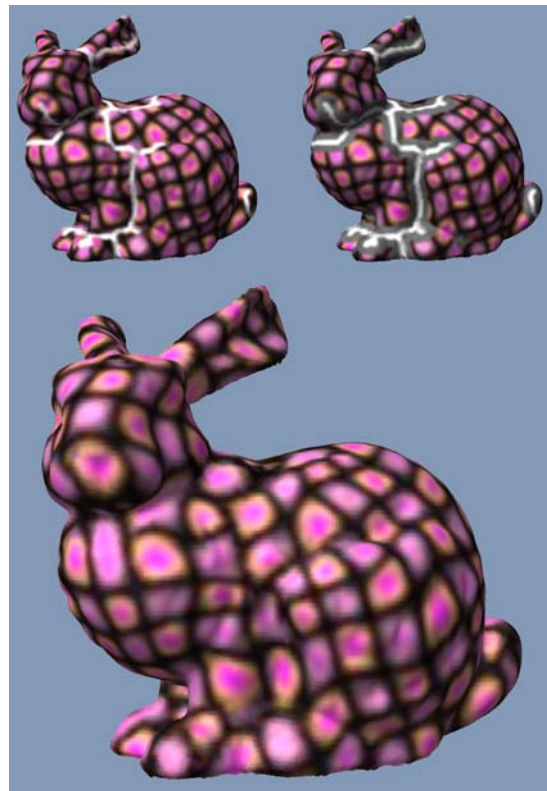
In the multi-resolution synthesis, a point model hierarchy and a corresponding sample texture pyramid are constructed with the same depth specified by the user. The texture synthesis starts from the lowest resolution to the highest resolution (i.e., the original model). All points in the lowest level in the model hierarchy are colored randomly. For all other levels, the neighborhood for matching consists of points from two levels, the colored points in the  $k$ -neighborhood of the current resolution and all the points in the lower resolution corresponding to the  $k$ -neighborhood of the current resolution.



**Figure 8:** Point neighborhood matching results using a 25-neighborhood on a sphere point model. Left: non-coherent synthesis using 4-level hierarchy, with an average search space size of 13568 (the same as the sample texture size). Middle: sample texture of size  $128 \times 106$ . Right: coherent synthesis, with an average search space size of 23 (discontinuity inside the yellow circle).

#### 4.2. Coherent Synthesis

In the neighborhood matching process, for every point to be colored, a matching error is computed for every pixel in the sample texture and the color of the pixel with the smallest matching error is used. This brute force search for finding the best match is a bottleneck in the neighborhood matching method. Coherent synthesis [Ash01, YHZ01] speeds up



**Figure 9:** Hybrid method. Top-left: after the point parameterization method (7% gap points). Top-right: after gap expansion (23% gap points). Bottom: after the point neighborhood matching method.

this process by decreasing the search space. For each colored point, it remembers the pixel in the sample texture where its color comes from. The search space for the best match for

a point  $p_i$  is restricted to the pixels near the corresponding pixels of the colored points in  $N_k(p_i)$ . Figure 8 is an example of a coherent synthesis, which can preserve highly structured patterns, but cannot guarantee discontinuity-free results.

### 5. Hybrid Method

Our point parameterization method is fast, yet highly structured patterns can be preserved. Since it cannot guarantee seamless connection between patches, discontinuity usually appears as a noticeable artifact. The point neighborhood matching method is slow, but there is no discontinuity in the synthesized textures. By extending the 2D hybrid method [NA03] to point models, our hybrid method makes full use of these two methods and capitalizes on the advantage of both. It applies the point parameterization method first to texture the *patch* points, followed by the point neighborhood matching method for coloring the un-colored *gap* points.

The gap width is approximately  $\sqrt{k}/2$  when  $k$ -neighborhood is used in the parameterization method. If this width is much smaller than the pattern size of the sample texture, discontinuity may still appear after the neighborhood matching step. In order to avoid discontinuity, gaps are expanded to the size of the texture pattern if necessary after the parameterization method. The gap width will increase by  $\sqrt{k}$  after all the *patch* points belonging to the  $k$ -neighborhood of any *gap* point are changed to *gap* points. If multi-resolution synthesis is used as the neighborhood matching step, a point model hierarchy is created from the result of the parameterization method. Points in the lower resolution of the hierarchy are colored points if and only if all the corresponding points in the upper resolution are colored points. The un-colored points are textured using neighborhood matching from lower to higher resolution. When using a large sample texture for the parameterization method, a smaller sample texture with all texture patterns (this can be a portion of the large sample texture) can be used in the neighborhood matching step. This can significantly reduce the best match search time since the neighborhood matching processing time is proportional to the size of the sample texture. Figure 9 shows the result of the bunny point model using our hybrid method.

### 6. Results

High speed has been achieved by our texturing methods. Table 1 shows the texturing speed using the methods presented in this paper. All the tests have been conducted on a 2.53 GHz P4 PC with 1G RAM. All processing steps such as finding  $k$ -neighborhood of each point are counted for the computation time. The point parameterization method can reach a speed above 26000 point/sec. Except for the coherent synthesis, the running time for the point neighborhood matching method is proportional to the sample texture size and the neighborhood size. In the hybrid method, 70% to 95% of the points are colored by the point parameterization method and the remaining points are textured by the

point neighborhood matching method. The average speed of the hybrid method is about 900 to 4000 point/sec. Our parameterization method and hybrid method for point models are much faster than the existing methods for mesh models [PFH00, SCA02, Tur01, WL01]. For example, Wei and Levoy [WL01] report that the synthesis time for a sphere mesh model with 24576 vertices using a sample texture of size  $64 \times 64$  is 695 seconds with exhaustive search or 82 seconds with TSVQ acceleration on a 450 MHz Pentium II machine, excluding the preprocessing time. For comparison purposes, we have used the same sample texture for a sphere point model with 32769 points using our hybrid method on a 500 MHz Pentium II machine. The running time is 41 seconds, including all the operations such as  $k$ -neighborhood search and point model hierarchy construction.

Figure 10 shows more results of our texturing methods for point models. We can see that the texture quality is comparable to the best results of existing methods [PFH00, SCA02, Tur01, WL01] for mesh models. The images that appear in this paper are all rendered by EWA splatting [ZPvBG01].

In the parameterization method, texture orientation is not considered in the patch expansion. For isotropic textures, this is not a problem. For anisotropic textures, inconsistency of texture orientation will appear in the boundaries of the patches (for example, the ball joint in Figure 10). The hybrid method uses the parameterization method as the first step, thus also bears the same limitations.

### 7. Conclusions and Future Work

We have presented several methods for texturing point models directly from sample textures, focusing on isotropic textures. The point parameterization method is fast while the point neighborhood matching method can create seamless textures. The hybrid method combines these methods together so that a seamless texture can be quickly generated.

One interesting issue for future work is to use the graphics hardware which supports fragment programming and floating point precision to accelerate the best match search in the neighborhood matching process. Given a sample texture  $I$ , a neighborhood containing  $m$  points with positions  $\{\text{offset}(i)|i = 1..m\}$  and colors  $\{\text{color}(i)|i = 1..m\}$ , the matching error for pixel  $i$  can be expressed by the following equation:

$$ERROR_i = \sum_{j=1}^m \{[I(i + \text{offset}(j)) - \text{color}(j)]^2\} \quad (5)$$

A fragment program is generated from the  $m$  position and color values while the sample texture is binded as texture. Five instructions are needed for one term in Equation 5: texture coordinate calculation, texture fetching, subtraction, dot product and addition. The matching error results for all pixels are saved in an off-screen floating point pixel buffer

**Table 1: Texturing Speed**

Point Parameterization Method					
Model	Number of Points	Sample Texture Size	Neighborhood Size	Total Time (seconds)	Speed (point/sec)
Sphere	16386	64 × 64	25	0.30	54620
Bunny	34834	128 × 128	36	0.96	26285
Santa	75781	128 × 128	25	1.83	41410
Dinosaur	56194	64 × 64	36	1.52	36969

Point Neighborhood Matching Method (test model: dinosaur with 56194 points)					
Sample Texture Size	Neighborhood Size	Hierarchy Levels	Coherent Synthesis	Total Time (seconds)	Speed (point/sec)
64 × 64	25	1	N	70	839
64 × 64	49	1	N	120	466
128 × 128	25	1	N	244	230
64 × 64	25	4	N	130	431
128 × 128	49	4	N	894	62
64 × 64	25	1	Y	1.8	31012
128 × 128	49	1	Y	4.5	12361

Hybrid Method								
Model	Number of Points	Sample Texture Size	Neighborhood Size	Hierarchy Levels	Coherent Synthesis	Gap Point Percentage	Total Time (seconds)	Speed (point/sec)
Bunny	34834	64 × 64	25	1	N	12	8.2	4254
Bunny	34834	64 × 64	25	4	N	13	13.6	2559
Bunny	34834	64 × 64	49	4	N	13	24.3	1433
Santa	75781	64 × 64	25	3	N	9.7	23.7	3203
Santa	75781	128 × 128	25	3	N	10	82.4	920
Dinosaur	56194	64 × 64	25	4	N	27	39.6	1417
Dinosaur	56194	128 × 128	49	1	Y	27	2.9	19133

without any loss of precision. We have successfully implemented it using GeForce FX 5800 Ultra graphics card. Currently the synthesis speed is slower than software implementation. We believe that this is a promising direction to explore.

There are several other possible directions for future work. It is possible to incorporate the method for assigning direction fields over the point model [AKS03] to make the texture orientation as consistent as possible. One challenge is how to maintain the assigned texture orientation consistency in the patch expansion of the parameterization method. It is more useful to use different textures in different regions of the model. We segment the model (for example, the Santa model in Figure 1) manually and apply the texturing method to each part. It is better to incorporate automatic or semi-automatic segmentation methods to make the process more flexible. Memory will become a bottleneck when the number of points in the model is very large. Out-of-core methods could be developed to deal with large point models.

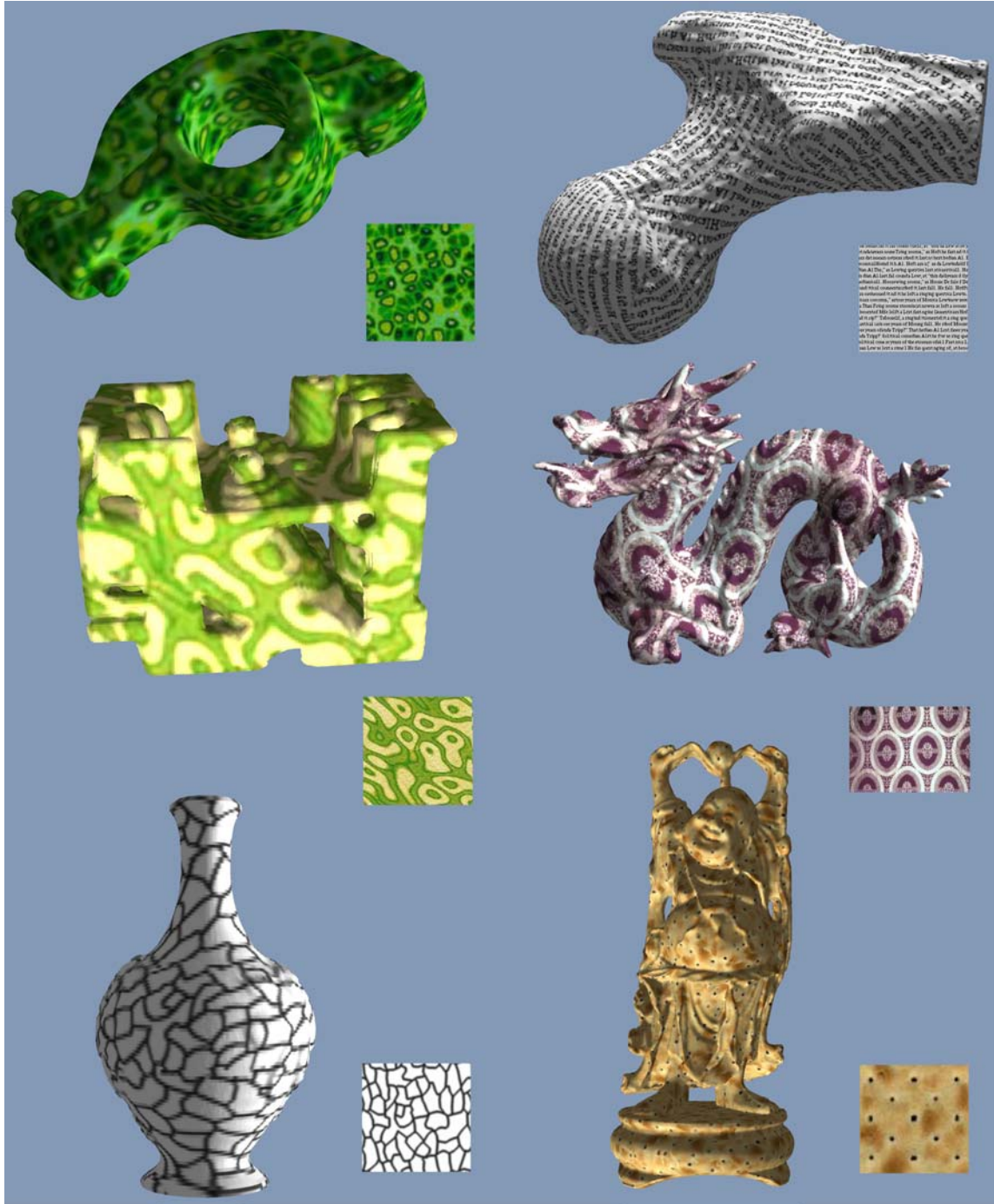
## 8. Acknowledgements

This work has been partially supported by a NSF grant CCR-0306438 and ONR grant N000140110034. The point mod-

els are courtesy of Cyberware and Stanford University Computer Graphics Laboratory.

## References

- [ABCO\*01] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C. T.: Point set surfaces. *IEEE Visualization* (October 2001), 21–28.
- [AKS03] ALEXA M., KLUG T., STOLL C.: Direction fields over point-sampled geometry. *Journal of WSCG 11*, 1 (2003), 27–32.
- [Ash01] ASHIKHMIN M.: Synthesizing natural textures. *Symposium on Interactive 3D Graphics* (2001), 217–226.
- [Bon97] BONET J. S. D.: Multiresolution sampling procedure for analysis and synthesis of texture images. *SIGGRAPH* (1997), 361–368.
- [DMLG02] DISCHLER J. M., MARITAUD K., LEVY B., GHAZANFARPOUR D.: Texture particles. *EUROGRAPHICS 2002 Proceedings 21*, 3 (2002), 401–410.



**Figure 10:** More texturing results: rocker arm (parameterization method), ball joint (parameterization method), engine part (hybrid method), dragon (hybrid method), vase (hybrid method) and happy buddha (parameterization method).

- [EF01] EFROS A. A., FREEMAN W. T.: Image quilting for texture synthesis and transfer. *SIGGRAPH* (2001), 341–346.
- [EL99] EFROS A. A., LEUNG T. K.: Texture synthesis by non-parametric sampling. *International Conference on Computer Vision* (1999), 1033–1038.
- [GIS01] GORLA G., INTERRANTE V., SAPIRO G.: Growing fitted textures. *SIGGRAPH* (August 2001), 191.
- [Gro98] GROSSMAN J. P.: *Point Sample Rendering*. Master’s thesis, Department of Electrical Engineering and Computer Science, MIT, August 1998.
- [KSE\*03] KWATRA V., SCHÖDL A., ESSA I., TURK G., BOBICK A.: Graphcut textures: Image and video synthesis using graph cuts. *SIGGRAPH* (2003), 277–286.
- [LLX\*01] LIANG L., LIU C., XU Y., GUO B., SHUM H.: Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics (TOG)* 20, 3 (2001), 127–150.
- [LW85] LEVOY M., WHITTED T.: *The Use of Points as a Display Primitive*. Tech. Rep. 85-022, Computer Science Department, University of North Carolina at Chapel Hill, January 1985.
- [NA03] NEALEN A., ALEXA M.: Hybrid texture synthesis. *EUROGRAPHICS Symposium on Rendering* (2003), 97–105.
- [PFH00] PRAUN E., FINKELSTEIN A., HOPPE H.: Lapped textures. *SIGGRAPH* (2000), 465–470.
- [PG02] PAULY M., GROSS M.: Efficient simplification of point-sampled surfaces. *IEEE Visualization* (2002), 163–170.
- [PZvBG00] PFISTER H., ZWICKER M., VAN BAAR J., GROSS M.: Surfels: Surface elements as rendering primitives. *SIGGRAPH* (2000), 335–342.
- [RL00] RUSINKIEWICZ S., LEVOY M.: QSplat: A multiresolution point rendering system for large meshes. *SIGGRAPH* (2000), 343–352.
- [RPZ02] REN L., PFISTER H., ZWICKER M.: Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering. *EUROGRAPHICS 2002 Proceedings* (2002), 461–470.
- [SCA02] SOLER C., CANI M., ANGELIDIS A.: Hierarchical pattern mapping. *SIGGRAPH* (2002), 673–680.
- [SCOGL02] SORKINE O., COHEN-OR D., GOLDENTHAL R., LISCHINSKI D.: Bounded-distortion piecewise mesh parameterization. *IEEE Visualization* (2002), 355–362.
- [Tur01] TURK G.: Texture synthesis on surfaces. *SIGGRAPH* (2001), 347–354.
- [TZL\*02] TONG X., ZHANG J., LIU L., WANG X., GUO B., SHUM H.: Synthesis of bidirectional texture functions on arbitrary surfaces. *SIGGRAPH* (2002), 665–672.
- [WL00] WEI L., LEVOY M.: Fast texture synthesis using tree-structured vector quantization. *SIGGRAPH* (2000), 479–488.
- [WL01] WEI L., LEVOY M.: Texture synthesis over arbitrary manifold surfaces. *SIGGRAPH* (2001), 355–360.
- [YHZ01] YING L., HERTZMANN A., ZORIN D.: Texture and shape synthesis on surfaces. *EUROGRAPHICS Rendering Workshop* (2001), 301–312.
- [ZPKG02] ZWICKER M., PAULY M., KNOLL O., GROSS M.: Pointshop 3D: An interactive system for point-based surface editing. *SIGGRAPH* (2002), 322–329.
- [ZPvBG01] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: Surface splatting. *SIGGRAPH* (2001), 371–378.

### Appendix A:

Formulas for computing the position  $q_f$  and the orientation  $\langle \vec{N}_f, \vec{S}_f, \vec{T}_f \rangle$  in the parameterization method (Figure 2) :

$$\begin{aligned}\vec{V} &= F - P \\ \vec{V}_0 &= \vec{V} / |\vec{V}| \\ q_f &= (\vec{V} \cdot \vec{S}_p, \vec{V} \cdot \vec{T}_p) \\ \vec{N}'_f &= \vec{N}_p - (\vec{V}_0 \cdot \vec{N}_p) \vec{V}_0 \\ \vec{N}_f &= \vec{N}'_f / |\vec{N}'_f| \\ \vec{T}_f &= (\vec{V}_0 \cdot \vec{T}_p) \vec{V}_0 + (\vec{V}_0 \cdot \vec{S}_p) \vec{N}_f \times \vec{V}_0 \\ \vec{S}_f &= \vec{T}_f \times \vec{N}_f\end{aligned}$$

Free point  $F$  should satisfy:

$$|\vec{V}_0 \cdot \vec{N}_p| \neq 1$$

Otherwise, free point  $F$  is in the direction of the normal of patch point  $P$ . Thus,  $F$  should not be added to the patch.