The Use of Subdivided Surfaces and Direct Texture Mapping in an Alternative Approach to Texture Synthesis on Surfaces

Chantelle Morkel*
Department of Computer Science
Rhodes University

Abstract

This paper explores the use of subdivision and direct texture mapping to synthesize textures on surfaces. The algorithm is not reliant on pyramids to maintain mesh hiearchies, nor does it use neighbourhood colour matching. We use a single, subdivided, model and a colour matching strategy that relies on the current vertex and it's neighbours being mapped into a sample texture image. The subdivided model is used for both synthesis and display. Several experiments have been conducted, using varying levels of subdivision. The results are not as expected, as the model resolutions are not high enough. The patterns from the sample texture images are not coming through clearly in the synthesized textures. This is attributable to the small number of subdivisions being performed on each model. The solution to this is to raise the level of subdivision. This solution is not without cost, however, as the more levels of subdivision performed, the larger the number of vertices within the model that need to be synthesized.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation - Display algorithmsI.3.5 [Computer Graphics]: Computational Geometry and Object Modeling - Curve, surface, solid and object representationsI.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism - Color, shading, shadowing and texture

Keywords: texture synthesis, subdivision

1 Introduction

1.1 Problem Statement

This paper looks at an alternative implementation approach to the texture synthesis strategy suggested by Turk [2001]. The alternatives suggested include:

- Using a subdivision scheme to create high resolution meshes (and only synthesizing texture on the highest resolution mesh);
- Using direct texture mapping to obtain colours for the vertices of the mesh; and
- Displaying the synthesized texture directly on the subdivided mesh.

1.2 Background

Texture synthesis is the process of creating a new texture by making use of a texture sample, or by using a procedural texture generation process such as noise or reaction-diffusion. Sample-based texture synthesis methods rely on the user providing a sample texture image which can be used in the synthesis process. The synthesized texture must be sufficiently different from the sample image, but resemble it closely enough to determine that it results from the sample texture image provided [Bonet 1997]. Common procedural texture generation regimes rely on a variety of schemes, but our discussion will be limited to reaction-diffusion [Turk 2001] and a noise function [Perlin 1985].

Texture synthesis can take one of two forms, texture synthesis on images or texture synthesis on a surface. Each type of texture synthesis can be done with or without an input sample texture. If no sample is used, then one of the procedural texture generation processes is used.

Texture synthesis on an image is the process of creating a new texture image (map) from a supplied sample image, or through the use of a procedural texture generation scheme. Texture synthesis on a surface synthesizes the texture directly on the surface of an object by assigning colours to the vertices of the mesh. These colours can either be assigned to the vertices of the current (highest resolution) mesh, or if the synthesis scheme relies on it, the colours can be assigned to the vertices of each mesh in the mesh hierarchy. If the latter approach is used, a combination of blending, upsampling and downsampling is used to get the colours emerging on the highest resolution mesh.

1.3 Overview

The rest of the paper is set out as follows. Section 2 discusses related work in the field of texture synthesis. Section 3 describes the the refinements introduced by the texture synthesis algorithm proposed by this paper. Section 4 discusses the implementation of the algorithm. Section 5 presents the results of the algorithm, and compares them to the results achieved by other texture synthesis approaches. Section 6 discusses the conclusions reached, and presents ideas for future work.

2 Related Work

This section will detail the texture synthesis methods mentioned in the previous section. The related work for each of the two texture synthesis strategies (on images and on surfaces) will be presented, with a separate discussion for the methods of colour matching, orientation and synthesis.

The procedural texture generation strategies are dealt with separately, as they are not reliant on a texture sample.

Several of the texture synthesis strategies rely on the use of multi-resolution hierarchies. A brief explanation of multi-resolution meshes will precede the texture synthesis related work.

*e-mail: g99m0761@campus.ru.ac.za

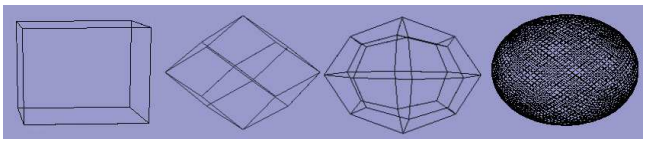

Figure 1: A cube at different resolution levels. The original mesh (far left) has 8 vertices, at level 1 it has 14 vertices, at level 2 it has 26 vertices, and at level 9 (far right) it has 3074 vertices

2.1 Multi-Resolution Meshes

Multi-resolution synthesis makes use of a set of successively refined surfaces or images (depending on the type of synthesis being performed). Each refined surface or image contains more detail than the preceding one. Figure 1 illustrates that each higher resolution mesh has more vertices than the preceding levels, it is more detailed.

Subdivision, however, is only one manner in which a set of multi-resolution meshes can be obtained. Another approach is to randomly place points on the surface of an object. Repulsion is used to spread these points evenly over the surface, and Delauney triangulation is used to connect these points (and the original vertices) into a connected mesh structure [Turk 2001].

2.2 Texture Synthesis Strategies

This section looks at the related work to texture synthesis on images and texture synthesis on surfaces respectively.

2.2.1 Texture Synthesis on Images

Heeger and Bergen [1995] use image pyramids to synthesize texture images. The process requires a sample texture and a noise image. The process relies on histogram matching, which in this instance is a form of histogram equalization. The pyramids (one for the sample texture and one for the synthesized texture) are matched layer by layer, until the process is complete. Once every layer of the pyramid has been synthesized, the pyramid is collapsed to create the texture image. Instead of providing a noise image, a block of noise can be provided to create solid texture.

De Bonet [1997] uses a two-phase algorithm to perform texture synthesis. The first phase, analysis, constructs a Laplacian pyramid by decomposing the sample texture into different resolutions. Each of these resolutions corresponds to a level in the pyramid. Once the analysis phase is complete, the synthesis phase begins. A synthesis pyramid is constructed by utilising the corresponding levels in the analysis pyramid. To begin the synthesis pyramid construction, the lowest resolution layer of the analysis pyramid is copied. The synthesis pyramid is constructed layer-by-layer from the corresponding analysis pyramid layers, with the already synthesized layers dictating the possible colours for the current pixel. If the top level of the synthesis pyramid is larger than that of the analysis pyramid, that specific layer in the analysis pyramid is copied repeatedly into the synthesis pyramid layer. The texture image is generated by collapsing the synthesis pyramid.

Textures are synthesized by Efros and Freeman [2001] using a quilting approach. The user supplies a sample texture which forms the basis for the new texture. To create the new texture random blocks of the sample texture are taken and pieced together to form the new texture. This is the naive approach. To refine the texture synthesis process, a matching mechanism is introduced. Texture blocks are no longer randomly selected, instead they are selected based on a sum of least squares approach. This produces slightly better results, but the seams between blocks are still unsightly. In a further attempt at refinement, a minimum error boundary cut is used. This results in the blocks having jagged edges, so there is no definite edge. This, in conjunction with blending allows for the creation of reasonable synthesized texture image.

Patch-based sampling allows Liang et al. [2001] to synthesize textures on images. The patches are portions of the input texture that are pasted together to form the new texture. New patches are selected (to be pasted into the texture) based on the patches that have already been added to the texture. These new patches are taken directly from the sample texture, and are matched according to the distance between the existing patches and the patch to be pasted. This distance is known as the boundary of a patch. Using this method, it is possible to perform two kinds of texture synthesis, namely unconstrained texture synthesis and constrained texture synthesis. Unconstrained texture synthesis takes a random patch to begin with, and finds all the patches that have a boundary that matches this random patch. A patch is then selected from this list. If the list is empty, the closest possible match is used. This process is repeated until the entire image has been synthesized. Constrained texture synthesis looks at patching holes and tiling patches in the new texture. Patch-based synthesis is very similar to image quilting.

Wei and Levoy [2000] use an input texture sample and a random noise image to synthesize new texture images. This method coerces the noise to resemble the sample texture by synthesizing the noise pixel by pixel (in raster scan order). The synthesis process relies on neighbours that have already been synthesized so, the neighbourhood is L-shaped to only encompass the previously synthesized pixels (because of the raster scan). The sum of squared difference is used to determine a match between neighbourhoods. This process can be extended to multi-resolution synthesis by using Gaussian pyramids. Two pyramids are constructed, one for the sample texture and one for the synthesized texture. The same matching technique is used, with the exception that the neighbourhood consists of pixels in the current resolution and in lower resolutions. This process creates a synthesized texture image, but can be extended to temporal synthesis.

The texture synthesis method proposed by Ashikhmin [2001] is loosely based on the texture synthesis method by Wei and Levoy [2000]. To cater for natural textures, a method that copies pieces from the texture sample for use in the synthesized texture is used. Matching is performed by creating neighbourhoods from preceding pixels that have already been synthesized. This texture synthesis technique allows for user interaction by allowing a user to specify where patterns in the synthesized texture should be created.

2.2.2 Texture Synthesis on Surfaces

Turk [2001] synthesizes textures onto surface through the use of a Gaussian-pyramid and neighbourhood colour matching. This process uses a sample texture image to synthesize textures. The mesh hierarchy is created from the initial model by adding $3n$ new points to the surface of the mesh at stage of refinement. Synthesize occurs over each of the meshes in the hierarchy, with the highest resolution mesh being the surface that the texture is synthesized for. The synthesized texture is converted into a texture map which is mapped to the lowest (user inputted) mesh for display.

Ying et al. [2001] make use of two colour matching strategies to

synthesize textures onto surfaces. The first method is based on the multi-resolution texture synthesis strategy suggested by Wei and Levoy [2000]. The process starts by synthesizing the texture on the coarsest mesh resolution and proceeds to synthesize the texture on the next coarsest resolution. Each repetition of the texture synthesis process is based on the previously synthesized mesh. This process continues until the texture has been synthesized on the finest resolution mesh. This implementation requires that the surface be covered by an atlas of overlapping charts. Two methods of creating sample neighbourhoods are employed, surface marching is used for the coarsest resolution mesh and chart sampling is used for all other meshes.

For each point within the coarsest mesh, a neighbourhood is constructed using surface marching. Each sample point within the neighbourhood is connected by to the center-most point by drawing a straight line between the two points. The resulting pattern is used to traverse the surface and values at each point are calculated through the use of bilinear interpolation from the texture sample.

Chart sampling is used to construct the neighbourhoods for the samples in the higher resolution meshes. Each chart corresponds to a mesh vertex, and overlaps with the surrounding charts.

The other neighbourhood creation strategy employed is coherent synthesis. For each point, consider it's neighbours that have already been synthesized. Each synthesized neighbour proposes a value based on it's location in the sample texture. The best match is computed using the sum of least squares.

Praun et al. [2000] make use of overlapping texture patches to create textures on surfaces. The process starts by selecting texture patches from an input sample texture. The orientation and scale of the lapped texture is controlled by a tangential vector field. The texture is grown on the surface until the entire surface is covered. The growing operation involves adding faces (which are themselves mapped into texture space so that the current point maps to the texture patch's center) to the surface.

2.2.3 Colour Matching Strategies

The colour matching strategies adopted by each of the texture synthesis methods are listed in table 1. Many of the colour-matching strategies rely on some kind of neighbourhood creation and influence.

2.2.4 Orientation

The texture synthesis methods that synthesize textures onto surfaces commonly use a tangential vector or orientation field that covers the surface of the object [Praun et al. 2000], [Turk 2001] and [Ying et al. 2001].

2.2.5 Synthesis

Many of the texture synthesis techniques (both images and surfaces) use multi-resolution synthesis. This is commonly in the form of a pyramid, or set of pyramids. The pyramids vary from Gaussian [Turk 2001], to Laplacian [Bonet 1997] and Steerable [Heeger and Bergen 1995].

2.3 Procedural Texture Generation Strategies

Apart from texture synthesis, another form of texture creation is through the use of procedural texture generation strategies such as reaction-diffusion, noise and solid textures. The main characteristics of procedurally generated textures are the following:

- The texture is synthesized based on an expression and does not require the use of an sample texture.
- Each point (or pixel depending on the synthesis) is catered for individually and is not reliant on it's neighbours to dictate what colour/texture values it is to assume.
- There is no problem mapping the resulting texture to a surface (is texture synthesis on a surface is performed), as the texture is generated directly on the surface. This alleviates the need of orientation/vector fields.

Reaction-diffusion describes the reaction between chemicals as they diffuse across a surface to form stable shapes and patterns on the surface [Turk 1991]. Both Turk [1991] and Witkin and Kass [1991] make use of reaction-diffusion to synthesize textures.

Noise [Perlin 1985] and solid texturing [Perlin and Hoffert 1989] are based on the same premise, a noise function. Noise is a seeded random number generator that consistently produces random values that generate patterns and shapes by assigning colours to points (or pixels if it is an image). Consistency is ensured by passing through integer values to the noise function. Solid texture performs the same operation, but instead of creating the texture on a surface, the texture occupies the full volume of the object.

3 Design

This experiment provides an alternative strategy for texture synthesis on surfaces to the one described by Turk [2001].

3.1 Texture Synthesis on Surfaces

To synthesize textures onto surfaces, Turk [2001] makes use of the following algorithm. Aspects of the algorithm that require further discussion will be highlighted in the subsections that follow.

- Load a texture sample
- Load a model
- Create a mesh hierarchy by creating successively higher resolution meshes and adding them to a Gaussian pyramid-like structure
- Allow the user to specify approximately a dozen orientations at the vertices of the highest resolution mesh
- Interpolate the user-specified orientations over the surface of the mesh to create a vector field, through the use of upsampling and downsampling techniques.
- Calculate sweep values for each vertex over the surface of the mesh, and order the sweep values in ascending order.
- For each mesh, traverse the vertices in sweep value order.
- For each vertex visited, create a neighbourhood to perform colour matching. Find the closest matching colour in the sample texture and use this to colour the current vertex.

	Colour Matching Strategy
Texture Synthesis on Images	Boundary Matching - boundaries of candidate patches are compared with against the patch already placed, with the patch having the closest matching boundary being selected. [Liang et al. 2001]
	Histogram matching is used to find the closest colour match. [Heeger and Bergen 1995]
	Column/row matching [Efros and Freeman 2001], which is very similar to boundary matching.
	Neighbourhood matching - in raster order, [Wei and Levoy 2000] construct causal (neighbours that have already been assigned) neighbourhoods
	Pyramid-based synthesis, where colours in the synthesis pyramid are obtained from the analysis pyramid according to a set of features and 'parental' constraints. [Bonet 1997]
Texture Synthesis on Surfaces	Neighbourhood matching - each vertex in a mesh has either a full or half neighbourhood constructed for it. The neighbours do not have to be assigned already. [Turk 2001] Neighbourhood matching - the neighbourhoods are constructed using either surface marching or coherent synthesis.

Table 1: The colour matching strategies employed in texture synthesis

- Display the texture by creating a two-dimensional texture map from the synthesized texture

3.1.1 The Mesh Hierarchy

The mesh hierarchy used by Turk [2001] is similar to a Gaussian pyramid structure, where the highest resolution mesh forms the tip of the pyramid and the lowest resolution mesh forms the base.

The mesh hierarchy is created by repeatedly refining the input model until the desired resolution level is obtained. The refinement process relies on the random placement of points ($3n$ points per iteration) on the surface of the model. Repulsion is used to evenly distribute the points over the surface of the model, and Delauney triangulation is used to connect these points (including the already existing points, which means that the new mesh contains $4n$ points) to form the next highest resolution mesh.

3.1.2 The Vector Field

Turk [2001] allows a user to specify a dozen or so orientation directions over the surface of the coarsest resolution mesh. These orientations are used to create a vector (orientation) field over the surface of the model. Since the orientation directions have been specified on the coarsest resolution mesh, downsampling and diffusion are used to get assign orientation vectors to the vertices in the coarser meshes. Once the coarser have vector fields, the vector field for the highest resolution mesh is created by using upsampling from the lower resolution meshes.

3.1.3 Surface Sweeping

Once the vector (orientation) field has been created, Turk [2001] performs surface sweeping over each mesh in the mesh hierarchy. This process selects an anchor vertex (which is randomly chosen), and uses the orientation vector associated with each vertex, to determine which vertices lie in front of and behind the anchor vertex. These sweep values are ordered in ascending order, and are used to traverse the mesh during the texture synthesis process.

3.1.4 The Colour Matching Process

Each mesh uses the same process to synthesize colours and patterns on the surface. Each vertex in the current mesh is visited in sweep value order. To synthesize colours and patterns at a vertex,

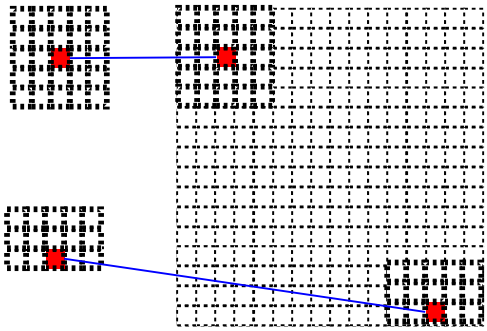

Figure 2: The neighbourhood matching process. Two types of neighbourhood are highlighted, the full neighbourhood (top left) and the half neighbourhood (bottom left). The per-pixel matching approach is used for both types of neighbourhoods

Turk [2001] uses a neighbourhood colour matching process. The neighbourhood for each vertex is constructed by considering points (that are not necessarily vertices) that are within a given radius of the current vertex. If a point does not have a colour to contribute to the colour matching process, a colour is found by using interpolation of the point's neighbours colours. Figure 2 illustrates how the neighbourhood is mapped into the sample texture to find a suitable matching colour. A neighbourhood for each pixel in the sample texture is constructed, with the best matching neighbourhood being selected to give the current vertex its colour.

The colour matching process relies on being able to find a closest match in the texture sample provided by the user. A neighbourhood is constructed for each pixel within the texture sample, and then compared against the neighbourhood constructed from the current vertex and neighbours. The sum of squared differences is used to compare each neighbourhood, the smallest (closest) match is provides the colour for the current vertex.

For multi-resolution synthesis, a current vertex's neighbourhood includes vertices from the next lowest resolution mesh.

3.1.5 Displaying the Synthesized Texture

Turk [2001] creates a two-dimensional texture map from the synthesized texture. Each triangle in the original mesh (the input mesh) is mapped onto a corresponding triangle within the texture map. The triangles in the texture map are uniform in size, and are coloured through the use of interpolation on the synthesized texture.

3.2 The Proposed Texture Synthesis Algorithm

The texture synthesis strategy we propose is is very similar to the algorithm described in section 3.1. The main differences between the two algorithms are different approaches to achieving a similar effect. The changes proposed include:

- The use of a single, subdivided, high-resolution mesh instead of a mesh hierarchy;
- The use of direct texture mapping instead of constructing vertex neighbourhoods; and
- The display of the texture directly on the surface of this high resolution mesh, instead of creating a texture map which is then mapped to the lowest resolution mesh.

3.2.1 The Mesh

Turk [2001] uses irregular meshes to synthesize textures, but notes that a regular mesh (such as a subdivided mesh) can also be used. We propose the use of the $\sqrt{2}$ subdivision scheme developed by Li et al. [2004]. A model is subdivided to create a high resolution mesh, which is then used for texture synthesis.

3.2.2 The Colour Matching Process

Turk [2001] uses neighbourhoods to perform colour matching. This process involves the construction of a $5x5$ (full neighbourhood) or $5x3$ (half neighbourhood) neighbourhood which comprises vertices and arbitrary points within a specified radius of the current vertex. This is performed once per vertex, but is performed for each pixel in the sample texture. Neighbourhoods in the sample texture are created in the same manner, with the exception that there is no need to specify a radius, as the grid pattern of the sample image lends itself to neighbourhood creation.

In contrast, we only use the adjacent vertices of the current vertex and the current vertex itself for colour matching in the sample texture. The current vertex and it's neighbours are mapped directly into the texture using the strategy outlined in section 4.6.

3.2.3 Displaying the Synthesized Texture

The subdivided mesh used during the synthesis process uses a per-vertex colour approach to display the colours assigned and patterns formed during the synthesis process.

Turk [2001] creates a texture map from the synthesized texture, which is then mapped to the lowest resolution mesh for display.

3.2.4 The Texture Synthesis Algorithm

Taking into consideration the alternative strategies suggested, we propose the following texture synthesis algorithm.

- Load a sample texture.
- Load a model.
- Subdivide the model to obtain the desired resolution level.
- On this subdivided mesh, specify several orientation vectors.
- Generate a vector field over the surface of the mesh by using interpolation and diffusion.

Algorithm 1 Texture synthesis algorithm

```
Load a sample texture
Load a model

Subdivide the model
Specify several orientation vectors

while NOT allOnSameLevel
   Generate vector field

while NOT allOnSameLevel
   Perform surface sweeping calculations

for each vertex
   Get neighbours
   for each pixel in the sample texture
      map current vertex and neighbours into
      sample to obtain a match value

   for all matches
      if (currentMatch < smallestMatch)
         smallestMatch = currentMatch

   Assign current vertex the colour associated
   with smallestMatch

Display the mesh
```

- Perform surface sweeping by calculating the sweep distance of each vertex based on its distance from the anchor vertex.
- For each vertex (ordered by sweep distances), get the adjacent vertices as neighbours, and map the current vertex and neighbours into the sample texture map. Assign the closest matching pixel's colour to the current vertex.
- Display the model

4 Implementation

Algorithm 1 outlines the steps required to synthesize textures onto a surface. Several of these steps warrant further discussion.

4.1 Operations

The operations, such as interpolation and diffusion, that we use in our implementation are the same ones used by [Turk 2001].

4.2 Subdivision

The subdivision scheme we use is the $\sqrt{2}$ subdivision scheme proposed by Li et al. [2004]. One of the main reasons for using this scheme is that at each subdivision step the mesh is only growing by a factor of 2. Another reason for using this subdivision scheme is that it can be used on a mesh of arbitrary topology, irrespective of whether the mesh uses triangles or quadrilaterals.

To begin the texture synthesis process, we subdivide the model provided by the user. The higher the level of subdivision, the more texture detail comes through on the surface of the object. We use our subdivided mesh for display.

Algorithm 2 The Vector Field Algorithm

```
for j = 1 to numberOfIterations
  while NOT allOnLevelJ
     for all vertices
        getNeighbours
        for all neighbours
           if neighbour assigned
              add to allowedNeighbours
        interpolateOrientation(allowedNeighbours)
        currentVertexLevel = j
        assigned = true
```

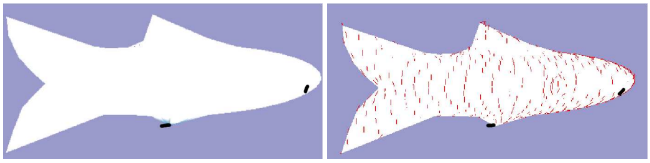

Figure 3: The original orientation vectors (left), and the interpolated vector field (right). The original orientation vectors or shown in black

4.3 Neighbours

The neighbours of a vertex provide a way of finding what the colour value of a vertex should be. We do not create neighbourhoods of neighbours, rather, we use the vertices directly adjacent to the current vertex. We can extend this to include the neighbours of the current vertex's neighbours. We have to be careful however, if we do not restrict the level of neighbours, we will end up with all the vertices in the mesh being neighbours of the current vertex.

4.4 Vector (Orientation) Field

Once we have a subdivided mesh, we assign orientation vectors to several (ranging from one to five) vertices. The vector (orientation) field is generated through the use of interpolation and diffusion.

To ensure that we obtain representative orientation vectors for each vertex, we restrict which neighbours may contribute to the orientation vector. This is done by only permitting a neighbour to contribute to the orientation vector if they have a non-zero length orientation vector themselves.

To obtain a stable vector field, the interpolation process is repeated a number of times. This will also allow the number of contributing neighbours to increase as more neighbours are assigned.

4.5 Surface Sweeping

We use the same approach as Turk [2001] to perform surface sweeping. To obtain sweep value stability, we employ the same strategy described in section 4.4, which the is the repetition of the surface sweeping process a number of times.The effects of the surface sweeping operations may be seen in Figure 4, where the dark spot at the bottom denotes the anchor vertex, and the varying shades of colour denote the distance of the vertices from the anchor vertex. The further away the vertex, the brighter the colour assigned to it.

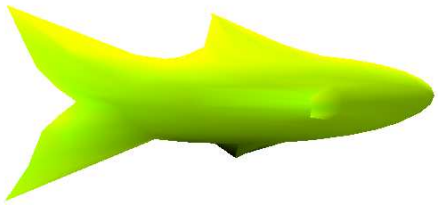

Figure 4: Surface Sweeping

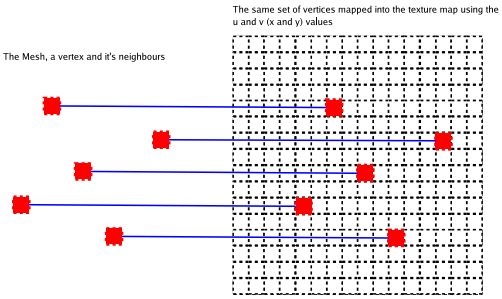

Figure 5: Mapping a vertex and it's neighbours into the texture map

4.6 Colour Matching

The colour matching process relies on the current vertex and its neighbours being mapped into the sample texture. To get the distances (and directions) of the neighbouring vertices (from the current vertex), we project the neighbouring vertices onto the orientation vector (the V axis) and the cross product of the orientation vector and normal (the U axis). To get a u and v value that will correspond to the x and y axes, we use expressions 2 and 1. The u and v values are calculated once, but are mapped to the texture sample each time the next pixel is selected.

$$v = (\mathbf{neighbour} - \mathbf{current}) \cdot (\frac{\mathbf{orientation}}{||\mathbf{orientation}||}) \quad (1)$$

$$u = (\mathbf{neighbour} - \mathbf{current}) \cdot (\frac{\mathbf{normal} \times \mathbf{orientation}}{||\mathbf{normal} \times \mathbf{orientation}||}) \quad (2)$$

Toroidal mapping is used. If a neighbour of the current vertex lies outside the boundaries of the sample texture, it is brought back within the boundaries through a series of additions and/or subtractions. Every vertex in the mesh, along with its neighbours, is mapped into the sample texture. For each vertex mapping that occurs, we calculate the sum of squared differences for the current vertex and its neighbours against their counterparts in the sample texture. This process is repeated for each pixel within the sample texture, ie if the dimensions of the sample texture are 64x64 pixels, the number of mappings will be 64*64. The set of pixels that have the lowest sum of squared differences are chosen to contribute a colour to the current vertex. Algorithm 3 shows the process required for finding the best possible colour match. Figure 5 illustrates the vertex to texture mapping process.

Algorithm 3 The colour synthesis process

```
public void colourSynthesis()
{
  for j = 1 to numberOfIterations
    while NOT allOnLevelJ
      for each vertex
        getNeighbours
        for each neighbour
          if neighbourColourAssigned
            calculate u value
            calculate v value
      mapIntoTexture(currentVertex,
                     validNeighbours)
  }

public Colour mapIntoTexture(currentVertex,
                             validNeighbours)
{
  for each pixel in Texture map
    currentPixel = currentVertex
    for each validNeighbour of the current
    vertex
      u = neighbour.getU
      v = neighbour.getV

      Scale u and v
      Ensure u and v are in bounds
      Calculate the sumOfSquareDifferences for
      the current vertex and it's neighbours

  currentVertex.colour = colourOfBestMatch();
}
```

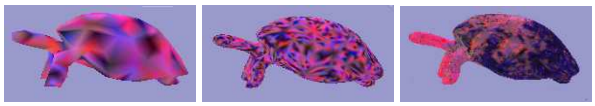

Figure 6: Tortoise - level 0 subdivision (left), level 3 subdivision (middle), level 6 subdivision (right). The sample texture used can be found in Table 2

5 Results

Table 2 summarises the results in terms of input textures, levels of subdivision and repetition, and time taken to subdivide the model and synthesize the texture respectively.For each texture synthesis experiment, the sample texture has the same dimensions - 64x64 pixels.

The results we have obtained using this texture synthesis method are not quite as expected. It is possible to associate the original texture with the synthesized texture, but it is not possible to distinguish specific texture features coming out in the synthesized texture.

There are several possible explanations for this observation.

- We have noted that the higher the level of subdivision, the more detail comes through in the synthesized texture, see figure 6. This can be attributed to the neighbouring scheme we are employing. The fewer vertices there are, the larger the spaces between neighbours. Figure 4.6 illustrates the mapping process, an ideal scenario, however is to have the vertices close together (which implies that we should use a higher level of subdivision) as illustrated in figure 10. In this manner it is possible to extract local features, whereas with the vertices far apart, we are extracting global features of the sample texture.

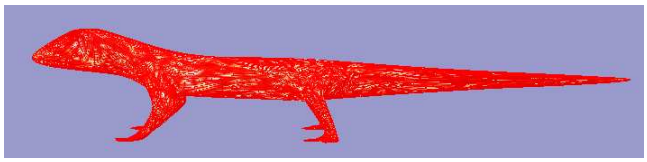

Figure 7: A gecko with 22850 vertices, using a brick texture

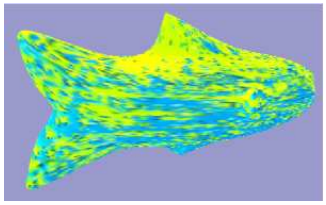

Figure 8: A fish with 23042 vertices, using a puzzle texture

- Very high levels of subdivision can, however, be an inhibiting factor, as the synthesis process has to work on more vertices, which requires large amounts of processing time. As a consequence of this, we tend to set the level of subdivisions at a maximum of 6. An indication of the additional processing time required for a highly subdivided model versus a non-subdivided model can be seen in Table 2. The highly subdivided (level 6) tortoise model takes almost 3 hours more to synthesize than the original tortoise model. The tortoise model has three different levels of subdivision applied to it, with the amount of time required to synthesize a texture onto it increasing as the level of subdivision increases. The amount of time to subdivide, relative to the texture synthesis time, is minimal.

- The number of repetitions for the vector field, surface sweeping and colour matching is also an inhibiting factor. The larges the number of repetitions, the longer it takes to synthesize the texture.

- An alternative approach to circular (toroidal) mapping is to use an offset when mapping into the texture which ensures that only pixels within the bounds of the sample texture are used for colour matching. This method has been successfully employed by Wei and Levoy [2000]. This change in colour-matching strategy might provide different results to the ones we are obtaining at present.

A short-coming of this texture synthesis algorithm is its inability to

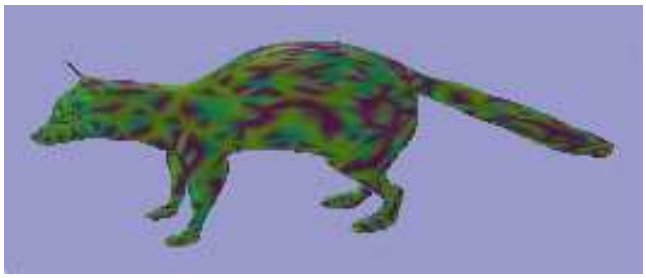

Figure 9: A raccoon with 5600 vertices, using a reptilian-like texture

	Levels of Subdivision	Vertices (End)	Time taken to Subdivide	Texture	Synthesis Time
Fish - Figure 8	6	23042	9m55.3s		2h20m54.5s
Tortoise - Figure 6	0 3 6	271 3230 25826	0.0 s 14.3 s 15m18.7s		1m27.7s 29m41.6s 2h56m19.4s
Raccoon - Figure 9	2	5600	31.5s		32m39.9s
Gecko - Figure 7	6	22850	11m53.7s		2h32m10.6s

Table 2: The details of the images found in the results section

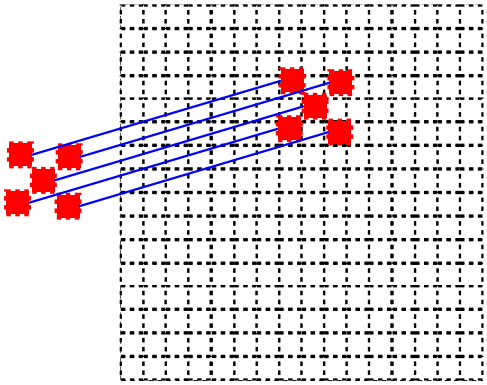

Figure 10: An ideal mapping scenario

synthesize feasible textures based on regular sample textures (such as the brick texture in table 2). This is an artefact of the low level of subdivision being performed. Similar to the shortcomings listed above, increasing the level of subdivision should make local features in the sample texture more prominent in the synthesized texture.

Apart from the actual synthesis of textures, the other aspects of the synthesis process (such as the creation of vector fields and sweep distances) appear to functioning satisfactorily. Figures 3 and 4 show an example of a vector field and surface sweeping respectively.

6 Conclusion

Several conclusions have been made regarding the proposed texture synthesis algorithm.

- Although not ideal, the texture synthesis strategy proposed produces feasible results. The sample texture patterns are not readily visible in the synthesized textures, but in many instances it is possible to ascertain that a texture is synthesized from a particular sample.
- The synthesis process is inhibited by the large amount of time and computation required to synthesize textures on highly subdivided surfaces. The amount of time varies from a couple of minutes to hours and days.

Possible extensions to this work include:

- The use of an optimised mapping algorithm. One optimisation is to use a pre-matching scheme that will identify a close to match to one of the vertex's neighbours, and map the remaining vertices, including the vertex to be matched, into the sample texture. This could perhaps be based on the darkest or lightest neighbour.
- The implementation of an offset scheme which only uses vertices from the valid region of the sample texture.

References

ASHIKHMIN, M. 2001. Synthesizing natural textures. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, ACM Press, 217–226.

BONET, J. S. D. 1997. Multiresolution sampling procedure for analysis and synthesis of texture images. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 361–368.

EFROS, A., AND FREEMAN, W. 2001. Image quilting for texture synthesis and transfer. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM Press, 341–346.

HEEGER, D., AND BERGEN, J. 1995. Pyramid-based texture analysis/synthesis. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, ACM Press, 229–338.

LI, G., MA, W., AND BAO, H. 2004. $\sqrt{2}$ subdivision for quadrilateral meshes. *The Visual Computer 20*, 2 (May), 180–198.

LIANG, L., LIU, C., XU, Y.-Q., GUO, B., AND SHUM, H.-Y. 2001. Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics 20*, 3, 127–150.

PERLIN, K., AND HOFFERT, E. M. 1989. Hypertexture. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, ACM Press, 253–262.

PERLIN, K. 1985. An image synthesizer. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, ACM Press, 287–296.

PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. 2000. Lapped textures. In *SIGGRAPH 2000: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 465–470.

TURK, G. 1991. Generating textures on arbitrary surfaces using reaction-diffusion. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*.

TURK, G. 2001. Texture synthesis on surfaces. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM Press, 347–354.

WEI, L.-Y., AND LEVOY, M. 2000. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press - Addison-Wesley Publishing Co., 479–488.

WITKIN, A., AND KASS, M. 1991. Reaction-diffusion textures. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, 299–308.

YING, L., HERTZMANN, A., BIERMANN, H., AND ZORIN, D. 2001. Texture and shape synthesis on surfaces. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, Springer-Verlag, 301–312.