2D Texture Refinement Using Procedural Functions

E. Clua[1] , M. Dreux[2] and M. Gattass[3]
Pontificia Universidade Católica do Rio de Janeiro – PUC-Rio
[1, 3] Computer Science Department
[2] Mechanical Engineering Department
[1]esteban@inf.puc-rio.br
[2]dreux@mec.puc-rio.br
[3]gattass@tecgraf.puc-rio.br

Abstract

In Computer Graphics, aliasing is a problem which is always present when discrete elements are mapped to continuous functions or vice-versa. Although there is no general solution for this kind of problem, there are many techniques that aim at reducing the effects of aliasing. This work first discusses how interpolation methods are usually applied in order to correct this problem and shows the limitations of this techniques. The article presents then another solution for this problem, that can be used together with the interpolation and consists at increasing details to the texture, making use of procedural functions.

Keywords: Procedural functions**,** texture-mapping, interpolation.

1 Introduction

During texture mapping, usually using an image, there are some problems related to the texture sampling due to the fact that a 2D discrete function is being mapped onto a 3D continuous surface.

In such cases there are aliasing artifacts caused by the image sampling. There are some antialiasing techniques that reduce this problem, since it is not possible to completely solve it. This problem can be stated as follows: given a set of points $P = \{ P_1, P_2, ..., P_i \}$, $i > 1$, belonging to a continuous 3D surface S, with all points within a sphere of radius ε, a discrete function f (e.g. an image mapping) returns the same result for all elements of P, producing jaggies. This problem increases when working with low-resolution mapping images.

A technique to reduce this problem makes use of interpolation methods. Several interpolation methods can be used, such as linear, spline, etc. This work also presents a new interpolation method that better suits the problem.

2 Interpolation

A texture function f to be mapped onto a surface has $U = (u_1, u_2, ..., u_i)$ as its input parameters, where i is the dimension of the domain of f, and in the case of a 2D image $i = 2$. The output value of f is the mapping value, e.g. the color. In the case of f being a discrete function within the domain, U has to be approximated to a valid number, e.g. only integer values. The linear interpolation averages the distance from a desired point to surrounding valid points, rather than simply using the output of the nearest valid point:

Let:

$U_1 = (\lfloor u_1 \rfloor, \lfloor u_2 \rfloor, ..., \lfloor u_i \rfloor)$

$U_2 = (\lceil u_1 \rceil, \lfloor u_2 \rfloor, ..., \lfloor u_i \rfloor)$

...

$U_k = (\lceil u_1 \rceil, \lceil u_2 \rceil, ..., \lceil u_i \rceil)$

and $m_1 = | U - U_1|$

$m_2 = | U - U_2|$

...

$m_k = | U - U_k|$

where $k = 2^I$.

Equation (1) is used in order to find the mapping of a given surface point U using linear interpolation:

$$C = f(U_1) \cdot \frac{m_1}{m_1 + m_2 + \ldots + m_k} + f(U_2) \cdot \frac{m_2}{m_1 + m_2 + \ldots + m_k} + \ldots + f(U_k) \cdot \frac{m_k}{m_1 + m_2 + \ldots + m_k} \tag{1}$$

where C is the interpolated result of the mapping function.

This work proposes an alternative method to evaluate C through a fractal interpolation. This technique introduces a noise generated by a fractal iteration of a given function. The previous equation would be modified to:

$$C = f(U_1) \cdot r(U_1) \cdot \frac{m_1}{m_1 + m_2 + \ldots + m_k} + f(U_2) \cdot r(U_2) \cdot \frac{m_2}{m_1 + m_2 + \ldots + m_k} + \ldots + f(U_k) \cdot r(U_k) \cdot \frac{m_k}{m_1 + m_2 + \ldots + m_k} \tag{2}$$

where r is a noise function with the following properties:

$$r(U_1) + r(U_2) + \ldots + r(U_k) = 1$$

and

$$0 \leq r(U) \leq 1$$

Although interpolation alone can reduce the jaggies of a mapped texture, the result consists of a blurred image, which in many situations does not improve the image quality. The proposed method combines patterns with the desired texture, including new data to the original texture and refining the final image. These patterns can be other images or procedural functions.

The use of procedural functions eliminates the aliasing problems, since they are continuous and, hence, for each input point there is a unique output point. Nevertheless, the functions that can describe the procedural textures are restricted. There are some cases in which these functions do not work, e.g. it is impossible to find a function that might describe an artistic painting to be mapped onto a frame.

3 Basic Procedural Functions

This work makes use of the noise function, initially described by [3]. This function has some relevant characteristics for this work:

1) It is a repeatable pseudorandom function of its inputs. This means that every time it is called with the same parameters it returns the same output value.
2) It is continuous in a given interval of R^n, and band-limited in the frequency domain. This means that the function has no sudden changes, returning similar values for similar parameters.
3) Statistical invariance under rotation and translation.

Basically, these properties guarantee that a sequence of rendered images in an animation are consistent, not changing the patterns for a specific region. Because of the continuity, there will be a coherence between neighboring points when the refinement is applied.

By using the noise function it is possible to derive some useful functions that will also have the above described properties.

One of these functions is the *fractal Brownian motion* (fBm) [2], which is a fractal iteration of the noise function over a point. Initially, a table of coefficients is built. The coefficients are used to limit the result obtained in each fractal iteration. The values must decrease and be between 0 and 1, the first value being equal to 1. The following pseudo-code shows how this table can be constructed:

```
frequency = 1.0
i = 1
While (i < table length) do
        Table[i] = $frequency^{-H}$
Frequency = frequency x 2
```

Any function that varies between 0 and 1, and is decreasing is a candidate function. Different results are obtained by simply changing the *H* exponents. Therefore, the user should be allowed to interactively modify these parameter.

Once the table is initialized, the fractal iterations are performed using a chosen function applied to the surface points. For each iteration the partial result is stored, and the function is called again to the same point, but this time with a small spatial displacement. This work uses the noise function during the iterations.

```
For i = 1 to (Number of Fractal Iterations) do
    result = result + Noise(point) * Table[i]
    point = point * spatial displacement
```

The coefficient *Table* limits the value of each iteration, as shown in the above pseudo-code. The point spatial displacement means that a new point sampling is made with a spatial resolution higher than the previous one. A greater number of fractal iterations produces a better refinement of the pattern being built.

This work applies texture refinement for natural phenomena, such as water and landscapes. The fBm function, described above, produces excellent results for water, with a few number of iterations (2 or 3). As fBm is a homogeneous function - equal distribution through all the space - and isotropic - equal in all directions-, it does not present excellent results for general landscapes. So, for applications that deal with such features, best results are obtained with an fBm variation: the multi-fractal functions. They are similar to fBm but have a heterogeneous distribution. [1] describes such functions, providing also implementation details. (See color plate - Figure 3.)

4 Texture Refinement Using Procedural Functions

In order to use the method, it is necessary to choose the procedural function that is adequate to the set of textures to be refined. There are situations with excellent results, and cases in which the method should not be used. That choice could either be made by the program, automatically, or by the user. The automatic choice, in most cases, is not so precise, but sometimes it is strictly necessary. For instance, a landscape scene could have many different components such as rocks, earth, water, etc, and several functions could be necessary to make the refinement for each component. Finding the appropriate functions is sometimes computationally expensive. An alternative solution is to search for a function, in a table, based only on the color of the pixel (blue could be a function for water, green for grass, and so on). Figure 3 from the color plate uses a appropriated function for sand, and figure 5 was built using a function for sand and another one for water.

As previously mentioned, the texture aliasing problem happens because, for a given set of surface points, the texture mapping value is the same. Since the domain of the used procedural functions is continuous, the returning values are not necessarily the same. So, for each point that is being visualized, the procedural function is called with the points coordinates as the input parameters. As each point of a continuous surface may only appear in a single pixel of the rendered image, different data can be included for each point of a surface that is receiving the same texture mapping value. Although the results may be different for each point at the surface, the function has to guarantee that close points will produce values that form the desired patterns. Once a value has been evaluated, by using a procedural function for a given point, it is necessary to blend it with the original texture.

It is important to remember that every time a function is called with the same parameters its return value is the same (see properties 1 and 3 of the noise function). However, if a surface is moving between frames, it is necessary to use local coordinates, otherwise the texture would have different results for the same surface point (to guarantee invariance under rotation and translation).

Blending

There are different ways to perform the blending. Ideally, the output value of the procedural function should be the final texture value. This is only the case in which the procedural function exactly describes the mapping texture. Suppose that a sand texture has to be mapped onto a surface. If there is a function than can simulate its appearance, the aliasing problem is completely eliminated. In most cases there is not such function, and it is necessary to combine a procedural function with the mapping texture. This blending could be achieved as follows:

$$T(P) = Texture(P) * (1 - coef) + f(P) * coef \qquad (3)$$

where *f* is the chosen procedural function, and *coef* is the blending coefficient, i.e., the percentage of procedural texture contribution.

It is also possible to include another procedural factor in order to achieve a non-uniform blending. This new factor allows the generation of non-uniform textures such as surfs, pieces of wood on a sandy soil, etc. The above equation would be modified to:

$$T(P) = Texture(P) * (1 - coef*f2(P)) + f(P) * coef* f2(P) \qquad (4)$$

where *f2* is the other procedural function applied to the blending.

The use of fractal interpolation during the blending process can produce impressive effects, especially when the textures represent noisy surfaces such as sand, water, marble, etc. (see color plates 3 and 5).

The general equation for the texture refinement, using the fractal interpolation for a given point *P*, is the following:

$$\begin{aligned} C = {} & T(U_1)\cdot r(U_1)\cdot \frac{m_1}{m_1+m_2+\ldots+m_k} + \\ & T(U_2)\cdot r(U_2)\cdot \frac{m_2}{m_1+m_2+\ldots+m_k} + \ldots + \\ & T(U_k)\cdot r(U_k)\cdot \frac{m_k}{m_1+m_2+\ldots+m_k} \end{aligned} \qquad (5)$$

where $T(U_1)$, $T(U_2)$, ..., $T(U_k)$ correspond to the results of the blending equation, with the values that surround the point *P*.

Optimization

The blending process sometimes becomes a lengthy task with calls to *f(P), f2(P),* and $r(U_l)$. Texture refinement can be optimized if these functions are pre-evaluated and stored in a table. On one hand, this optimization generates lower-quality images, especially for small tables, since there will be pattern repetitions. On the other hand, it speeds up the rendering, as it consists of a simple look-up table. The table´s input parameters are the spatial coordinates of the desired point. The images shown in color plates 3 and 5 were generated making use of that optimization.

5 Conclusion and Future Research

This work has presented a new method for reducing aliasing problems in texture mapping. It has introduced procedurally-created patterns that are similar to the original texture, and has proposed a method for blending them. Some functions that simulate natural elements, such as water, sand, grass, etc., have been explored, but other patterns could also be used. This work has applied the method proposed to the color attribute only, but this study could be extended to other attributes (e.g. bump-mapping).

Acknowledgements

The authors are grateful to TeCGraf, at PUC-Rio, where this research was developed, and to Flávio Szenberg, who made all his previous work available to the authors. The first author would also like to thank CAPES, whose grant partially sponsored this work. Also thanks to Carolina Alfaro for the text review.

References

[1] Davis Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, Steve Worley. Texture and Modeling: A procedural Approach. Academic Press, 1994.

[2] F. Kenton Musgrave, Craig E. Kolb and Robert S. Mace. The synthesis and rendering of eroded fractal terrains. *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pp. 41-50, July 1989.

[3] Ken Perlin. An image synthesizer. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pp. 287-296, July 1985.

[4] Steve Worley. A Cellular Texture Basis Function *(SIGGRAPH '96)*, volume 30, pp 291-294, August 1996.

Color Plates

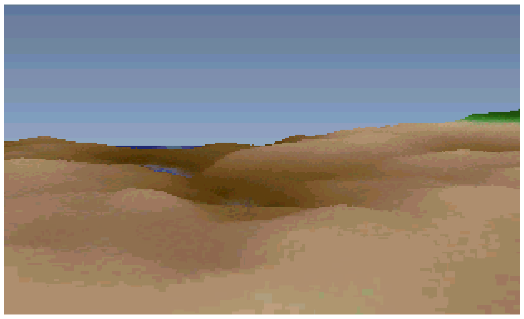
Figure 1- Original terrain, without interpolation and texture refinement.

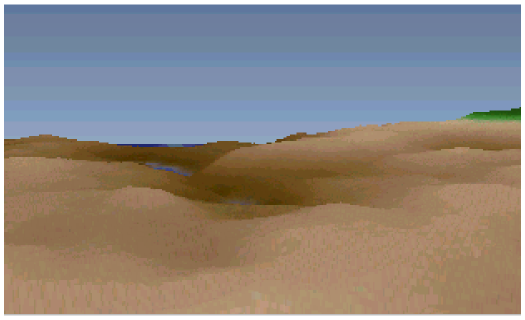
Figure 2- Terrain with fractal interpolation

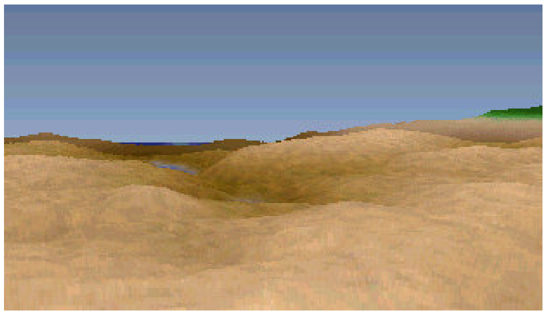
Figure 3- Terrain with fractal interpolation and texture refinement, using multi-fractal functions.

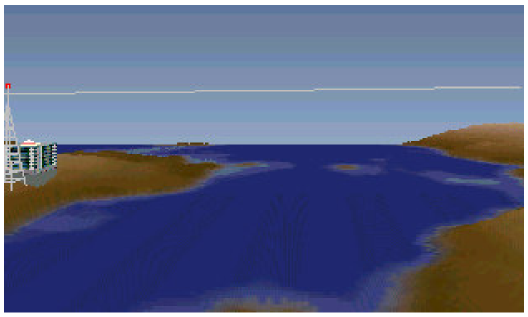
Figure 4- Texture of a lake in a terrain with linear interpolation.

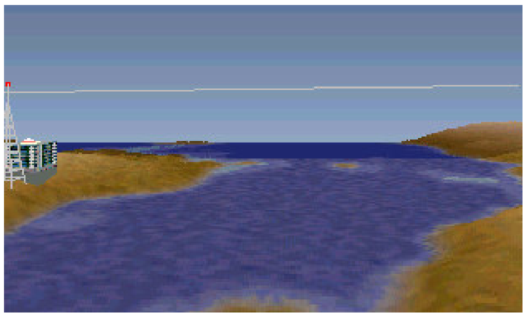
Figure 5- The same lake of figure 4, with fractal interpolation and texture refinement using fBm function.