The Visual Computer

A texture controller

G. Papaioannou, T. Theoharis, A. Boehm†

Department of Informatics, University of Athens, TYPA Buildings, Panepistimiopolis, 157 84 Athens, Greece

We propose an efficient method for creating accurate and controllable textures. In the past, procedural textures have been used to cover every point of an object uniformly, although their behaviour could not be controlled locally, as is frequently needed. In order to provide local control, texture-attribute control points are inserted in the model, and the behaviour of the texture at every point is defined through interpolation of the control-point attributes. The texturing algorithm proposed behaves as a texture controller and can be applied to any kind of procedural texture.

Key words: Texture mapping – Procedural texture – Texture control

Correspondence to: Georgios Papaioannou, Department of Informatics, University of Athens, TYPA Buildings, Panepistimiopolis, 157 84 Athens, Greece
E-mail: georgep@di.uoa.gr

1 Introduction

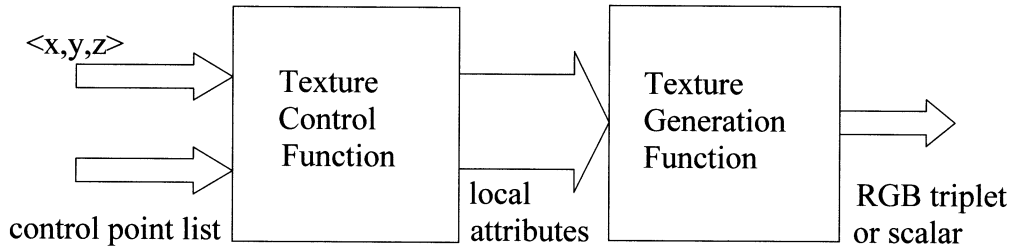
The motivation behind this paper was to implement an accurate representation of marble texture in such a way that a real marble object could be modelled and the marble veins could be placed upon the model as similarly to the real object as possible. The driving application is the accurate modelling of pieces of marble belonging to the Acropolis of Athens. They could not be scanned due to their size. In the past, procedural textures could be applied to entire object primitives (Watt and Watt 1993), but even if the textures themselves had an intrinsic variation in their pattern (Peachey 1985; Perlin 1985), their behaviour was random. The only way they could be controlled was either by texture-blending masks or by texture nesting. Both techniques are expensive, not always applicable, and often require cumbersome work before one can achieve the desired final texture.

A procedural or an image mapping texture is created by a *texture generation function*, which binds the object space with the texture space. This is accomplished via a mapping transformation and a series of geometric transformations that define the features of the texture. Such features include the texture orientation (an important aspect of many procedural textures), the texture scaling and perturbation, etc. In order to localise this behaviour, we need to define some control points in 3D space where the texture must behave in a certain, known way. Let us consider a black box (Fig. 1) that will represent the *texture control function* (TCF). This black box has as input the point vector, for which the texture must be calculated, as well as a list of structures describing the texture attributes for each predefined control point. The output of the black box is a set of local attributes at the input vector that will be used by the texture generation function to produce the final colour triplet (R, G, B), or a normal vector in the case of bump mapping. Both cases are treated in the same way and are application dependant.

2 The control-point texture structure

A required input to the texture control function is a set of control points. The control points indicate positions on the 3D model where the texture attributes are considered to be known (e.g. because these attributes can be measured on the real object). The behaviour of the texture at any other point in 3D space depends on the geometric and vi-

The Visual Computer (1998) 14:488–496



1

Control Point Texture Structure	
Texture ID	
Colour1 (R, G, B)	Appearance Attributes
Colour2 (R, G, B)	
Turbulence t	
Scaling Vector (S_x, S_y, S_z)	Positional Attributes
Rotation Vector (R_x, R_y, R_z)	
Position Vector (T_x, T_y, T_z)	

2

Fig. 1. The texture interpolation black box

Fig. 2. The proposed control-point texture structure for each texture control point. The bare minimum version of this structure could dispose of the colour attributes, assuming a greyscale or blending value between 0 and 1

sual attributes of the texture at the control points. Each control point is described by a *control-point texture structure*, as we will call it for the rest of this work (Fig. 2). The control-point texture structure contains information about the way the texture of the object should behave at the location of the control point, such as the position of the control point, the colour attributes of the procedural texture, the orientation, scaling and offset of the principal axis of the texture or any other variable attribute of a procedural texture (Perlin and Hoffert 1989). The list of control points, along with their attributes, is passed to the TCF. All control-point attributes must be defined externally via the user interface or an object database.

When modelling real-life objects, the control points should represent key locations on the object where the texture exhibits clear and distinct properties or is important for the description of the object. For example, the control points on a piece of marble must be set on locations where the marble veins have a well-defined direction.

The TCF, apart from being able to control the behaviour of a texture, can be extended to assign multiple textures to an object at specific locations via the incorporation of a *texture ID* in the control-point texture structure (Fig. 2). If multiple textures are assigned to an object, the resulting texture at a given point P will be interpolated from the respective procedural textures at each control point. For the rest of this paper, a single texture will be assumed for the entire object, although it may contain many other nested ones.

3 The texturing algorithm

Given any 3D-space input point $P(x, y, z)$, the texture attributes for this point are calculated and the texture for point P is computed with these attributes. We compute such a set of attributes by interpolating the attributes of the control points as discussed in Sect. 4. The interpolation scheme pro-

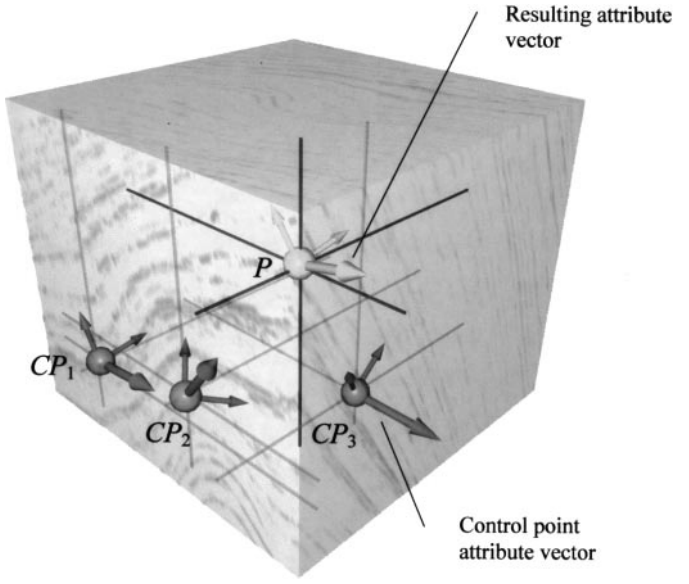


Fig. 3. Interpolating the control point attributes in 3D space. The local attributes of *P* are calculated through interpolation of the attributes of the control points

duces a set of attributes, which are referred to as *local attributes* and not colour values.
After deriving the local attributes from the interpolation stage, the input point has to be transformed according to the calculated local positional attributes (Fig. 2). The input point *P* is thus expressed in the procedural texture coordinate system. It is first translated, then rotated and finally scaled. The new, transformed, point is passed to the procedural texture generation function, which produces the final output. The texture generation function can be anything from an image mapping function or a procedural texture to a composite texture comprising many other procedural textures. Figure 3 demonstrates the texture control method. The algorithm for the controllable texture generation is presented here.

Preprocessing stage.
a. Define the control points CP_1, CP_2,..., CP_{Nc}:
For each control point
1. Define its position.
2. Assign a set of local attributes to the control point, including the desired orientation and size of the texture at the control point, the colours and the amount of spatial noise to use.

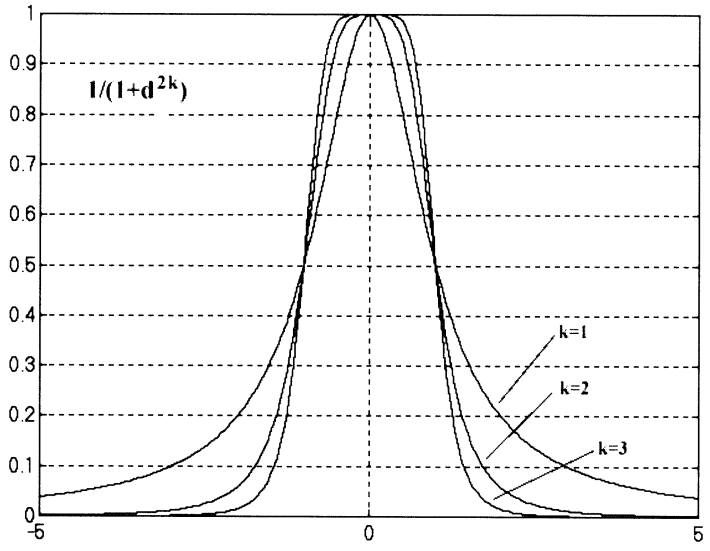
b. For every input point $P(x, y, z)$:
1. Calculate the local attributes at *P* by interpolating the attributes of all the control points.
2. Transform *P* according to the local positional attributes (Fig. 2), in order to move P to the texture generation function coordinate system:
- Translate *P* by ■ $(-T_x, -T_y, -T_z)$.
- Rotate *P* according to the texture rotation vector $(-\mathbf{R}_x, -\mathbf{R}_y, -\mathbf{R}_z)$ (rotation in degrees with regard to the three axes).
- Scale *P* by ■ $(1/S_x, 1/S_y, 1/S_z)$.

3. Use the transformed point *P*' and the local appearance attributes to specify the final colour for point *P* through the texture generation function:

$$Colour = f_{\text{texture}}(P', \{\text{appearance attributes}\}).$$

4 Interpolating the attributes

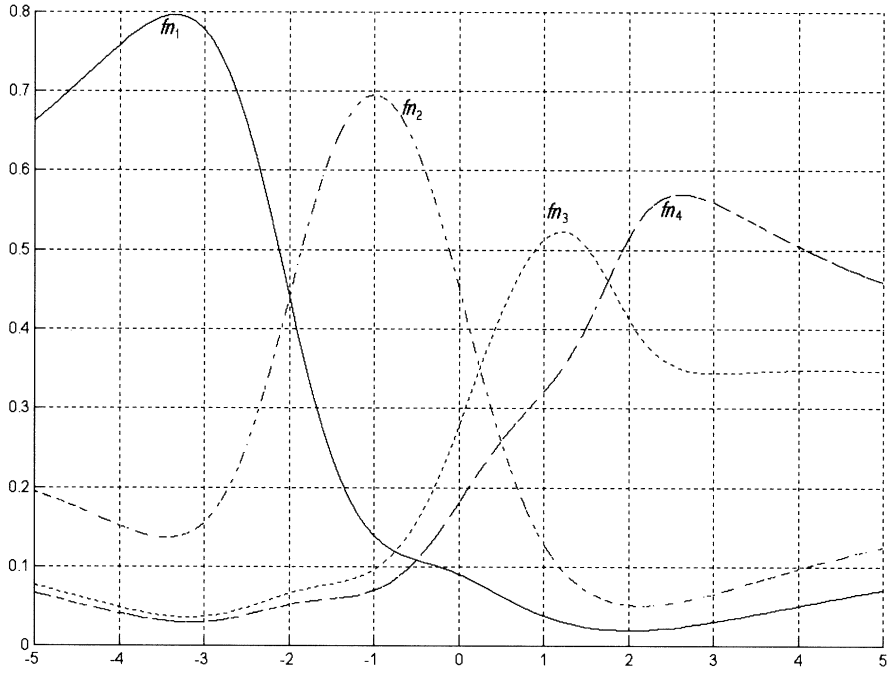
The interpolation scheme that we use in the texture control function has four main goals: (a) the entire 3D space must be addressable in order for the texturing function to act as a material generator, (b) attributes in regions between control points should change smoothly for small displacements of the input point, (c) attributes must converge to a constant value in regions outside the convex hull formed by the control points and (d) smoothness at the region boundaries must be ensured.



4

Fig. 4. The basis function associated with each control point

Fig. 5. An example of the normalised basis functions for $\mathcal{R}$. The control points are CP_1=–3, CP_2=–1, CP_3=1.5, CP_4=2



5

Before proceeding any further, the *control region* concept must be clarified. A control region is defined as the region whose attributes are biased towards the attributes of a certain control point. In every control region, there is one dominant control point, and every point within the region is closer to this particular control point than to any other defined control point. We therefore have a Voronoi tessellation of 3D space (Preparata and Shamos 1985).

In order to define the attribute values at an arbitrary point P, we express an attribute value as a

combination of a set of basis functions f_{Bi} defined for each control point CP_i.The basis function f_{Bi} must be chosen in such a way that its maximum value appears at CP_i and decreases monotonically as the distance from CP_i increases. It is apparent that:

$$f_{Bi}(CP_i) = 1 \quad \text{and} \quad f_{Bi}\left(CP_j\right) < f_{Bi}(CP_i), \quad \forall j \neq i.$$

Another requirement is that the basis function is at least C^1 in order to ensure that it varies smoothly over the entire 3D space. The function chosen, which complies with the restrictions given is:

$$f_{Bi}(P) = \frac{1}{1 + d_i^{2k}}, \tag{1}$$

where d_i is the euclidean distance between the arbitrary point P and the control point CP_i, and k is the order of f_{Bi}, controlling the locality of the basis function. The form of f_{Bi} is shown in Fig. 4.
Considering the need for short computational time in the texturing procedure, we must choose a blending function of low complexity. A simple, but rather effective method is to use the normalised basis function values as the interpolation coefficients for a given point P. An attribute value A for a point P is derived as:

$$A = \sum_{i=1}^{Ncp} \frac{A_i f_{Bi}(P)}{\sum_{j=1}^{Ncp} f_{Bj}(P)} \tag{2}$$

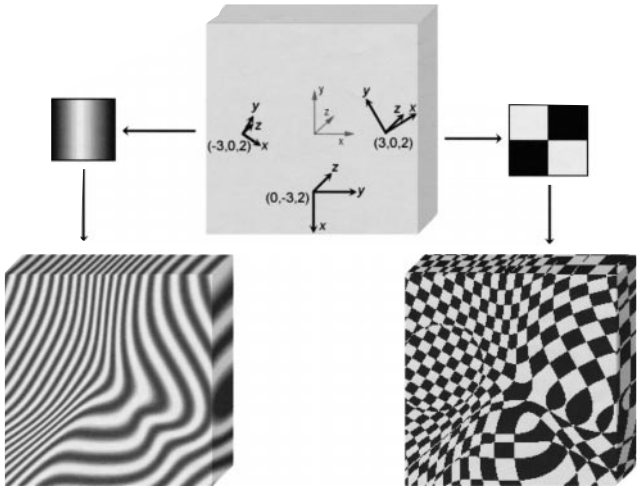
or

$$A = \sum_{i=1}^{Ncp} A_i fn_i, \tag{3}$$

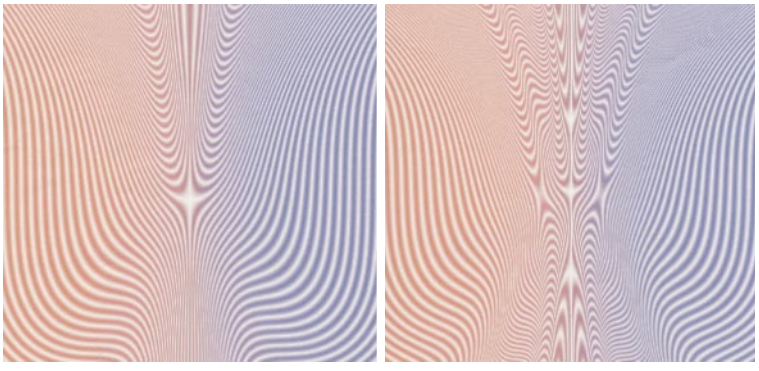
where fn_i is the value of the normalised basis function of control point CP_i at P, A_i is the attribute value at CP_i and N_{CP} is the number of control points.
It is obvious that: $\sum_{i=1}^{Ncp} fn_i = 1$.
Figure 5 presents a set of normalised basis functions in 1D space for the control points $\{-3, -1, 1.5, 2\}$. The closer the control points are to each other, the more their regions are influenced by the other control points. The peak of the functions of isolated control points is higher and tends towards 1.
One can see that the interpolation function defined by Eq. 2 and 3 complies with the four restrictions stated in the beginning of this section. The TCF is well defined, continuous and smooth at every point in $\Re^3$.
Using the interpolation scheme just described, all texture attributes of a material can be interpolated for each point in $\Re^3$ (Fig. 3). For points that are far away from the bulk of the control points, the attribute values are averaged because of the behaviour of the fn_i. As fn_i depends on the distance d_i of a point P from the control point CP_i, the influence of all control points is averaged for points far away from the convex hull of the control points. If this convergence of the attribute values is not desirable, one need only add extra control points on the boundaries of an object. As the complexity of the texture interpolation is low, additional control points do not pose any significant overhead.
It is also possible to apply the interpolation procedure, apart from the material characteristics, to the materials themselves. Every region can have its own texture. However, this can more easily be implemented with a layered texture blending function. The interpolation is required in those cases where the correct, continuous transition from one attribute value to another is needed in order to achieve the visual effect of a distorted texture. This restriction has no meaning when different textures are used; one can thus use a blending mask for such an operation instead of the TCF. After all, the TCF is a procedural texture itself and can be smoothly integrated in a complex texture hierarchy (including texture masks), in the same manner as any other texture.

5 Tests

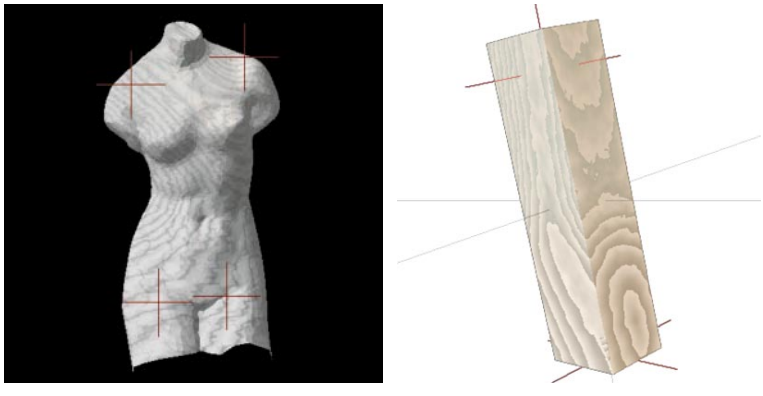
For the implementation of the TCF, a renderer was created by combining the speed of OpenGL polygon shading and a simplified ray-tracing scheme incorporating a two-level polygon cache for the calculation of the objects' texture.
In Fig. 6, three control points were defined in the space ([−5, 5], [−5, 5], [1.5, 2.5]). For control point CP_1=(−3, 0, 2), a scaling of the texture by 0.5 and a rotation by −30 degrees around the z-axis were supplied. For control point CP_2=(3, 0, 2) a rotation



6



7a 7b



8a 8b

Fig. 6. Two tests of the interpolation function with three control points. The sine wave and the checker patterns were used to demonstrate the results

Fig. 7. Using two control points to rotate a texture by 360º (**a**) and 720º (**b**)

Fig. 8a. A marble statue, complete with turbulence. Veins can be controlled so that they have the desired spacing, orientation, colour and amount of turbulence. **b** A piece of wood. Using the texture control function we can produce a more realistic texture with variable grain orientation, density and colour transitions

by 30 degrees around the z-axis was set without any scaling. For the last control point CP_3=(0, −3, 2), the texture was rotated by −90 degrees around the z-axis. Two totally different textures were used, a simple marble texture without spatial noise $f_{\text{marble}}(x, y, z) = \sin(2\pi x)$, and the checker function $f_{\text{checker}}(x, y, z) = (\lfloor 2x \rfloor + \lfloor 2y \rfloor + \lfloor 2z \rfloor) \bmod 2$, with period 1 along all axes. The texture at points close to CP_1 is bent by approximately 30 degrees around the z-axis, and its spatial frequency is twice that at regions closer to the other control points due to the texture scaling factor of 0.5 in the vicinity of CP_1.

Figure 7 demonstrates the ability of the TCF to bend a texture pattern using an angle greater than 360º. As the interpolation is applied to the rotation angles at the control points and not their absolute direction, it is possible to obtain a visual effect that corresponds to the rotation of the pattern by a full circle or more.

In Fig. 8a, the TCF was applied to a perturbed sine function (Perlin 1985), resulting in a marble-like texture with veins running along predefined vectors and a different turbulence at each control point. In this example, four control points were used in order to modulate the size, rotation angles, spatial noise and the colour palette of the texture. The output of the controlled texture was mixed with a heavily perturbed sine function in order to add some random red veins.

In Fig. 8b, a wooden block was rendered with a sawtooth function and two control points to specify the density, orientation, smoothness and hue of the wood grain. The original wood generation function produced concentric cylinders along the y-axis. Near the control point located at the top of the wooden block, the texture was rotated by 10 degrees clockwise around the z-axis, and the colours were chosen in such a way as to produce a greenish hue. The rotation angles for the control point at the base of the block were set to 0.0, 0.0 and −35 degrees for the texture rotation around the x, y and z-axes, respectively. We also scaled the texture by 2 to create a larger wood grain.

With our texture controller, it is easy to apply deformations to a texture pattern as shown in Fig. 9. In the example, we used three control points to locally deform the texture either by scaling the pattern or by rotating it.

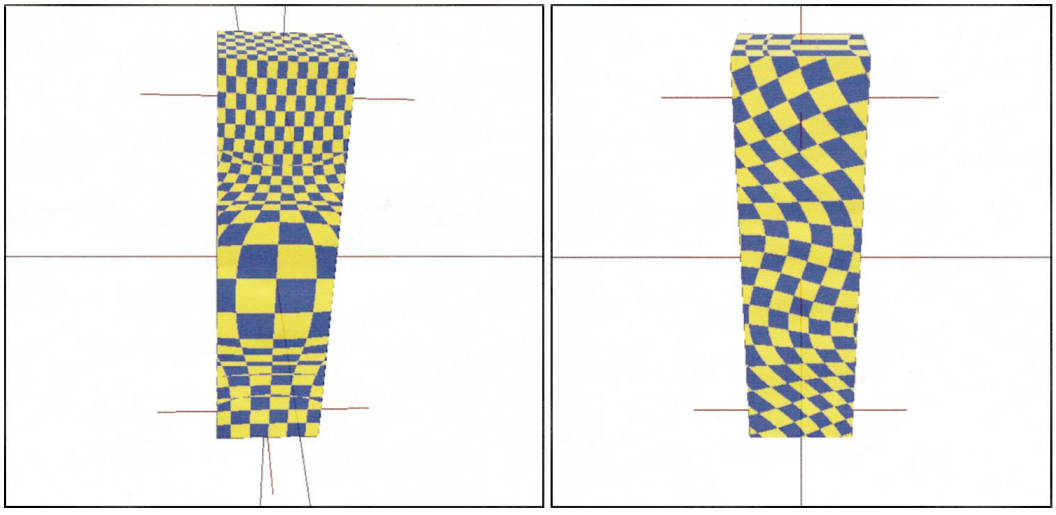
6 Applications of the texture control function (TCF)

The TCF was originally intended to be used to provide a more accurate simulation of real-life objects, such as marble and wooden objects (Fig. 8). In applications where precise control over the characteristics of a texture is important, the TCF can offer accurate texture modelling. For example, it used to be quite difficult to model the wood texture to be applied to a tree in such a way that the rings and cracks of the bark followed the branch orientation in a realistic and continuous way. This is very easily done with the TCF. The only requirements are a set of control vectors, which can automatically be obtained from the object mesh by following the curvature of the central line of the branches and a concentric or sawtooth procedural texture (Oppenheimer 1986). Figure 10 shows the application of the TCF on a tree branch with six control points (left frame). The control points were placed at locations where the curvature of the wood grain changes. In the same figure, the TCF was used to texture map a bent object (right frame).

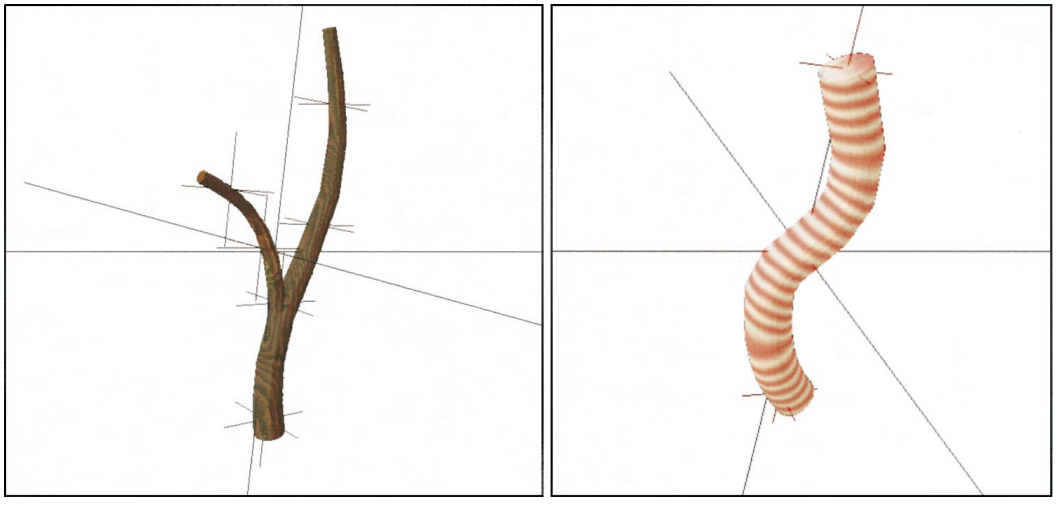
The complexity of the TCF grows linearly with the number of control points. A large number of control points can therefore be used on a textured object. Due to simplicity and low complexity, the TCF attributes can be altered interactively. The method is, therefore, ideal for fast preview during the texture modelling phase as well as for the final renderings. As mentioned earlier, we implemented the TCF in a mixed OpenGL – ray-tracing environment, producing a fast, interactive viewer. Being a well-defined texture generation function itself, it can be used as a procedural texture either to produce colour variations on rendered objects or to implement bump mapping. It can be incorporated in any renderer as a stand-alone texture or in a texture hierarchy (Abram and Westover 1985; Cook 1984; Perlin 1985). Of course, the control point texture structure may be extended to support an alpha channel, colour balance curves or any other application-specific attribute.

7 Conclusion

We have presented a method for controlling the attributes of a texture using an interpolation scheme



9



10

Fig. 9. The texture control function being used to control the scaling (**a**) and rotation (**b**) of the texture of a checkered box. The crosshair markers indicate the control point positions

Fig. 10a, b. Application of the texture control function in shape following: **a** the texture for a tree branch is guided with six control points; **b** a smooth stripe pattern follows the spine of the object using three control points

based on a number of texture control points. Our method can be considered as a procedural texture itself and can therefore be incorporated in any renderer. Because of the algorithmic simplicity and relatively low complexity, our method is suitable for interactive modelling and previewing, as well as for final demanding renderings. The ability of the TCF to modify the texture of an object at any given point in 3D space provides a means of accurately modelling real-life objects without using any cumbersome method to adjust multiple layered textures.

Acknowledgements. This work was supported in part by University of Athens Research Grant No. 70/4/3241.

References

Abram G, Westover L (1985) Efficient alias-free rendering using bitmasks and lookup tables. Comput Graph (SIGGRAPH '85 Proceedings) 19:53–59

Cook RL (1984) Shade trees. Comput Graph (SIGGRAPH '84 Proceedings) 18:223–231

Oppenheimer PE (1986) Real time design and animation of fractal plants and trees. Comput Graph (SIGGRAPH '86 Proceedings) 20:55–64

Peachey DR (1985) Solid texturing of complex surfaces. Comput Graph (SIGGRAPH '86 Proceedings) 19:279–286

Perlin K (1985) An image synthesizer. Comput Graph (SIGGRAPH '85 Proceedings) 19:287–296

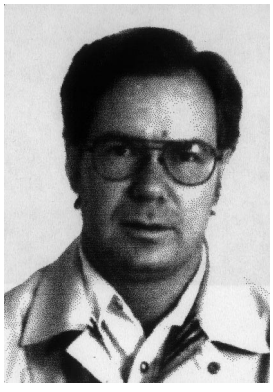
Perlin K, Hoffert EM (1989) Hypertexture. Comput Graph (SIGGRAPH '89 Proceedings) 23:253–262

Preparata FP, Shamos MI (1985) Computational geometry: an introduction. Springer, New York Berlin Heidelberg Tokyo

Watt A, Watt M (1993) Advanced animation and rendering techniques: theory and practice. Addison-Wesley, New York

GEORGIOS PAPAIOANNOU is a PhD candidate at the Department of Informatics, University of Athens, Greece. He received his degree in Computer Science from the same department in 1996. Currently, he is working on his PhD thesis in Computer Graphics and is also a partner of the 3DHeartView and PAROT ESPRIT programmes.

THEOHARIS THEOHARIS is an Assistant Professor in the Department of Informatics, University of Athens, Greece. He received his PhD degree in Computer Graphics and Parallel Processing from the University of Oxford, England. He served as a Research Fellow at the University of Cambridge. His main research interests are in the field of computer graphics and parallel processing.



ALEXANDER BOEHM was an Assistant Professor in the Department of Informatics, University of Athens, Greece. He received his PhD and BSc degrees from the Department of Mathematics, University of Athens, and a Diploma in Information Technology from the University of Karlsruhe, Germany. In the past, he has also served on the research team of Fried Krupp, Essen, Germany. His main interests were in the field of computer graphics and fractals. Unfortunately, Alexander Boehm died of a sudden heart attack on 1 May 1998.