Texturing Composite Deformable Implicit Objects

RUBEN ZONENSCHEIN[1] JONAS GOMES[1] LUIZ VELHO[1] LUIZ HENRIQUE DE FIGUEIREDO[2]
MARK TIGGES[3] BRIAN WYVILL[3]

[1]Instituto de Matemática Pura e Aplicada – IMPA
Estrada Dona Castorina 110, 22460-320 Rio de Janeiro, RJ, Brazil

[2]Laboratório Nacional de Computação Científica – LNCC
Avenida Getúlio Vargas 333, 25651-070 Petrópolis, RJ, Brazil

[3]Department of Computer Science, University of Calgary, Calgary, Alberta, Canada T1W 1H2

Abstract. In this paper we present a method for applying 2D textures onto composite and articulated objects defined by implicit functions. The method generates a particle system associated with the gradient vector field of an implicit function which acquires texture coordinates at a support surface. By extending this method to composite objects, an implicit surface may change its shape in time, while maintaining texture consistency. This approach prevents the appearance of undesirable effects such as ghosting and artifacts at the blending parts of an implicit object.
Keywords: texture mapping, particle systems, implicit surfaces.

1 Introduction

Texture mapping is a well-established technique in computer graphics. It maps a texture source to a surface, improving surface detail without changing the underlying geometry.

Texture mapping is closely tied to surfaces described parametrically, such as patches, since the mapping of two parametric spaces is usually straightforward. Implicit surfaces, i.e., those defined by an iso-contour of an implicit function [2, 14, 7], present a major difficulty in that implicit surfaces do not have a natural coordinate system defined on them.

Since an implicit surface cannot be easily parameterized, a common way to apply textures onto it is to use solid textures [8, 9, 15]. Although 3D texture mapping is a powerful technique, it is limited to materials that have a 3D structure, such as wood and marble; it cannot deal with textures that would wrap a surface, such as an image label.

Composite objects are those created by combining primitives via Boolean operators. This intuitive way to model complex objects can be found in various modeling schemes, such as CSG (Constructive Solid Geometry) and implicit blends. The problem of applying textures onto composite objects reveals the close relationship between texture mapping and parametric surfaces. Texture mapping a composite object based on parametrically defined primitives is possible; however, it demands special care at the intersections of the primitives. On the contrary, applying textures onto a composite implicit object highlights how unsuitable texture mapping techniques are to implicitly defined surfaces.

Even if we were able to texture map an implicit object, either via parameterization techniques or using solid textures, it is not obvious how the texture should behave as primitives blend with each other or as the object changes its shape in time. This is a result of computing texture coordinates (2D or 3D) from functions of the field value at some point in space. Problems also occur when combining primitives with different textures. Using the field to produce a weighted sum of texture values gives undesirable effects where objects blend (we refer to these as *ghosting effects*).

A new method of applying 2D textures onto implicit surfaces was introduced in [18]. It uses a force field derived from the gradient vector field of the implicit function to simulate a particle system that associates the implicitly defined model to a support surface for the texture. In this paper, we extend this method to cover composite implicit objects with moving parts, i.e., objects whose shape may change in time, as well as address the problem of combining primitives with different textures.

2 Previous Work

Perlin [9] and Peachey [8] introduced the idea of solid textures: a 3D texture space is embedded in the object space, and texture coordinates values are defined with no additional cost at any point on a surface. Although limited to materials that have a 3D structure, this technique is very useful for texturing implicit surfaces.

The use of solid textures on implicit surfaces was discussed by Wyvill et al [15], who were particularly interested in the characteristics of blobby objects whose shape changes in time, and how a 3D texture space should be defined in order to follow changes in the geometry of an implicit model. If separate 3D texture spaces are defined for each skeletal element of a blobby object, their method performs a weighted sum at the level of the texture space, to assign texture values to a point on the surface. Although they presented a method to keep the texture continuous while an object changes shape, the texture distortion does not always follow the geometric change, thus creating texturing artifacts.

One alternative for applying 2D textures onto implicit surfaces is via a projection [1]. Similarly to a slide projection, a 2D texture source can be projected onto a surface, by computing the intersection of rays that leave the texture in the direction of the surface. This approach presents a critical problem: the rays as a whole will never reach all 3D surface points from the outside, making impossible the idea of wrapping the texture around the surface. Even if only part of the surface is to be textured, a ray can intersect it more than once, giving the same texture attributes to different points, which may not be desirable.

Other strategies to apply 2D textures onto implicit surfaces focus on implicit-to-parametric conversion techniques. Although there is no general conversion method, it is reasonable to follow this approach, since applying 2D textures onto parametric surfaces is a simpler problem. Pedersen [10] introduced a method that estimates geodesics on an implicit surface and performs local parameterizations creating patches over it. His results are very effective, mainly regarding the interactive positioning of patches on the surface. The method allows one to even move, copy and paste textured patches along the surface, but does not indicate how the patches should be glued together to allow a unique texture to be applied to the whole surface. Pedersen's work does not address the problem of applying a texture to a composite implicit surface with moving parts.

Once the texture is defined on the implicit surface, another related problem is how to manipulate the texture such that it remains on the surface. This is important for texture editing and animation. Smets-Solanes [11] proposed a method that constraints a particle system to be on the implicit surface. Essentially, it restricts the motion of particles to a direction perpendicular to the gradient field of the implicit function. Motion equations are derived for various animation effects, such as texture sliding and gliding. Smets-Solanes approach assumes that the implicit model is already textured and they do not discuss how to establish the initial correspondence between the texture space and points on the surface.

The method presented here can be used for both texture placement and manipulation. It has the advantage of applying a 2D texture in a natural manner, using the gradient vector field, an inherent characteristic of implicit surfaces. Moreover, the method setup has the functionality of assigning a particular texture projection for each primitive of a composite implicit object, allowing an object to change shape in time and even break into separate parts while maintaining texture consistency.

3 Particle Texturing of Implicit Surfaces

An implicit surface is defined as the set of points x in 3D space that satisfy an equation $F(x) = c$, where $F: \mathbf{R}^3 \to \mathbf{R}$ and $c \in \mathbf{R}$. Thus, F defines a continuous family of *level surfaces*, one for each *isovalue* c. The gradient vector field ∇F has a close relationship with the level surfaces, which follow the gradient orthogonally, as can be seen in Figure 1. This observation is the starting point for a method for sampling implicit surfaces [4, 6] and for the texturing method described in this section.

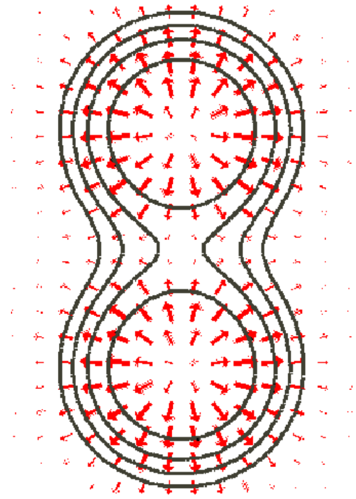

Figure 1: Level surfaces follow gradient vector field.

Let S be one of the level surfaces defined by F. We may assume that the isovalue corresponding to S is $c = 0$, i.e., that $S = F^{-1}(0)$. We employ the gradient vector field ∇F to generate a force field defined in the ambient space, and we use it to guide particles that are initially at rest on S. Our goal is a particle system that establishes a correspondence between points on the implicit surface S and points on a support surface T, where the texture is defined: the texture attribute for each point on S is taken from the intersection of the corresponding particle trajectory with T.

The support surface T should be simple enough so that it is very easy to define texture coordinates on it (e.g., a cylinder, sphere etc.). For this reason, we select a surface that has both a parametric and an implicit description. The parametric description allows us to define a texture on T. The implicit description is useful for detecting whether a particle is inside, outside, or on T.

We present two alternative stepwise methods to compute the trajectory of a particle.

3.1 Mechanical model

We consider two mechanical models: one kinematic and one dynamical. In the kinematic model [4, 6], the field describes velocity in terms of position, and the motion of a particle is governed by the following differential equation:

$$\frac{dx}{dt} + \nabla F = 0,$$

where x is the position of the particle, and t is time. This model was used in the original description of the texturing method [17, 18]. In the dynamical model [6], the equation of motion is

$$\frac{d^2x}{dt^2} + \gamma\frac{dx}{dt} + \nabla F = 0,$$

where γ is a viscosity constant.

In both cases, integration of motion equations using any suitable numerical technique (such as the classical methods by Euler and Runge-Kutta) provides the field lines along which particles are traced. See [6] for details and also for a discussion of the relative merits of the two models.

3.2 Electro-magnetic model

Using an electro-magnetic field analogy, if we consider the support surface T to be oppositely charged from the iso-surface S, then a field would exist that would provide a 1: 1 mapping of points on S with points on T.

To approximate such a field, we compute the linear combination of two forces, one repelling and one attracting:

$$\mathbf{N} = K_0 \cdot F_0 + K_1 \cdot F_1.$$

The weightings of the two vectors can be any smoothly varying values that sum to 1. It is convenient to use the field value for the current position of the particle, i.e., $K_0 = F(x)$ and $K_1 = 1 - K_0$.

The repulsive force F_0 is simply the gradient of F at x. The direction of the attractive force F_1 is the direction of the shortest path from x to the support surface T:

$$F_1 = \frac{x - m}{||x - m||},$$

here m is a *centre point* of T (i.e., a point on the axis of a cylinder, or the centre of a sphere, etc.).

3.3 Computing the texture mapping

We have implemented an algorithm for computing texture mappings using the particle systems described above in both polygon and ray tracing rendering systems. Figure 2 illustrates the results of the algorithm in 3D, using the polygonal rendering implementation, as described below.

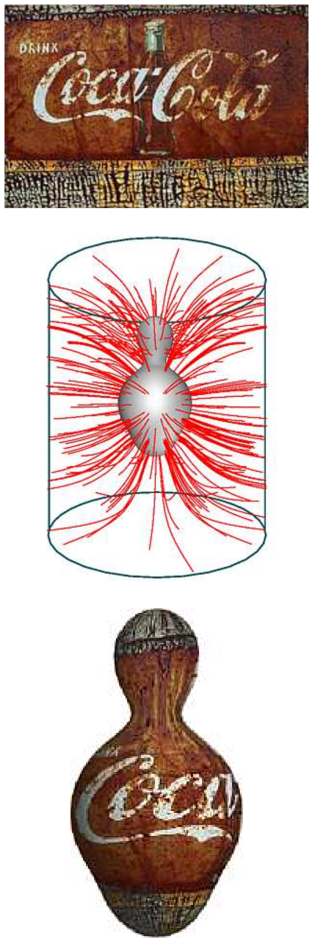

Figure 2: Texture source (top), particles trajectories (middle), and textured blobby object (bottom).

For the polygon based system (such as SIPP [16]), we start the simulation with particles placed at the vertices of a polygon mesh that approximates the implicit surface. A simple adaptive polygonization algorithm is used to construct the mesh [12]. Each particle follows a trajectory along the force field. At each step, the particle position is tested against intersection with the support surface T (using its implicit definition). Intersection with T is computed when the trajectory calculation terminates. Texture coordinates are then taken from the parametrically definition of the support surface at the intersection point. After the simulation has finished for all particles, texture coordinates are available for each vertex of the polygon mesh. Interpolation of texture coordinates inside each polygon completes the texturing of the implicit surface. (The interpolation can be done either by software, as in SIPP [16], or automatically in hardware. A polygonal approximation with texture coordinates at the vertices is commonly used on most graphics pipelines and can be rapidly textured using dedicated graphics engines.)

In the case of the ray traced solution, one particle is traced for each ray intersection with the implicit surface, the surface characteristics of the intersection point are then taken from the texture map.

4 Composite Implicit Objects

Modeling systems typically allow the creation of complex objects through the composition of hierarchies of simpler primitives. It is very important to be able to texture objects whose components may move and change shape in a particular manner with respect to the object as a whole.

In the case of implicitly defined objects, the basic primitives are defined by implicit functions of the form $F(x) = 0$. A combination operator C may then be applied to a set of implicit primitives resulting in new implicit function F_c, defining a composite implicit object:

$$F_c = C(F_1, F_2, \ldots, F_n).$$

A common combination operator is the union operator. To be able to glue together different implicit primitives, one should perform the operator given by:

$$C = \max(F_1, F_2, \ldots, F_n).$$

Implicit functions have the advantage of using a particular type of combination operator, called blending operators, which perform a smooth combination, creating seamless transitions between primitives. An example is a linear blend, given by

$$C = \sum_{i=1}^{n} F_i.$$

If the combination operator produces a differentiable function, a composite implicit object can be textured using the method described in section 3. The particle system method is suitable because the composite implicit object is itself defined by an implicit function F_c. The problem arises when we want to texture implicit objects whose geometry may change in time, in this case we would like texture space to reflect geometric change accordingly. If the support object for the texture is fixed, any change in the shape of an object will generate a sliding effect, which is not desired (see Figure 3).

We have investigated situations where a composite object changes in a global way. In those cases, the support surface can be transformed to follow the object change as can be seen in Figure 4. Although the result is good, this approach is limited to changes that affect the object as a whole, and is not applicable to objects with moving parts, in which case the texture sliding effect will still occur locally.

Figure 3: An object moving showing texture sliding.

Figure 4: $F_c = C(F_1, F_2)$. While F_1 and F_2 move in opposite directions, the support surface is scaled accordingly, maintaining texture consistency.

5 Method Setup

In the previous sections we discussed methods of applying 2D textures onto implicit surfaces. These methods are not applicable to composite implicit objects with moving parts. The main difficulty they impose is that they apply a texture in a global manner, without taking into account the possibility of local changes in geometry.

From this observation it became clear that a correct approach should set a texture mapping to each part of a composite implicit object. Also, the texture should be bound to the part, the texture follows the movement of the component it is texturing.

The method described in section 3 is modified to allow a separate support surface for each primitive of a composite object. The affine transformation for the primitive F_i is also applied to its associated texture support surface T_i.

A problem remains at the parts where a blend of primitives occurs. In fact, the blending parts of a composite implicit object may be a considerable part of it, and sometimes may correspond to the entire object.

To address this problem we recall the basic characteristic of composite implicit objects. F_c is a combination of various implicit functions $F_1, F_2, \ldots, F_n$. These implicit function values for each primitive can be used to weight the textures between blended components, this is the standard weighting of attributes used in [15] for solid texturing.

In the following subsections, we analyze two variants of the method that use the setup presented.

5.1 Color blending

The first attempt to texture the object is to use the weights mentioned above to perform a weighted sum of color values.

We compute the texture value at a point using the following steps:

1. Simulate the particle trajectory for each component of the object contributing to the surface location of the point.
2. Calculate the texture value at each intersection between a particle and its support surface T: $c_1, c_2, \ldots, c_n$.
3. The final texture value is then computed by a weighted sum of each value associated to that point, depending on the proportion to F_c: $c_p = \alpha_1 * c_1 + \alpha_2 * c_2 + \ldots + \alpha_n * c_n$.

This weighting of texture values is used for color, bump or other surface detail maps. Figures 5 demonstrates this approach.

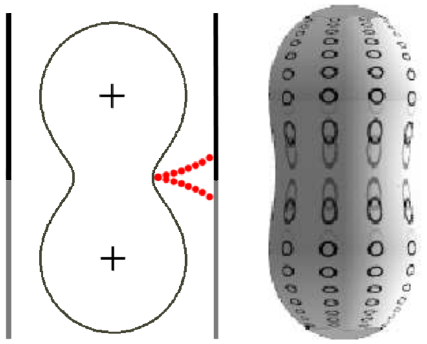

Figure 5: A particle at the blending part of a blobby object taking two trajectories, one for each primitive support surface.

Figure 6 shows a composite implicit object created using a blend of five primitives: one generalized cylinder and four spheres. Each object has a texture with the same structure (a checkerboard) but with different color attributes, Our blending method provides a way to combine the texture as depicted in the figure.

Figure 7 shows a composite articulated object generated from two implicit primitives. Each primitive has a different texture associated with it. In this example, as one of the primitives moves relative to the other, its texture remains consistent with the shape of the primitive.

Although the sliding effect disappears in the blended areas, a ghosting effect still occurs where the superposition of multiple textures is clearly visible. This is illustrated in Figure 8.

On the other hand, the texturing problem is now limited to the blending parts of the composite implicit object. Setting a separate support surface for each primitive allows us to move the implicit primitives in time. Aside from the blending areas, the texture follows the primitive in accordance to its changes.

5.2 Transformation blending

To solve the ghosting problem, we use the α_i weights to perform a weighted blend of support surface transformations, i.e. we perform a linear combination of the affine transformation of the support surface for each primitive used for each particle. The simulation is then run once for each particle (instead of multiple time as with 5.1), each of which with a specific support surface T transformed by a combined transformation M_c. The support surface T is common to all primitives involved in the blending.

The method consists of the following steps:

1. For each particle at rest at the surface, compute the contribution of each primitive to F_c: $\alpha_1, \ldots, \alpha_n$.
2. Simulate the motion of each particle and test its trajectory against a support surface T transformed by M_c, which is positioned using the proportion α_i of each primitive to F_c.
3. Texture coordinates are read at the intersection with the transformed support object M_c.

Figure 9 illustrates the method. If a particle is at a surface point where only one primitive contributes to the implicit surface iso-value, the trajectory is computed with respect to the transformation for that primitive. Particles which originate in a blend area of the surface use the linear combination of the transformations of the components contributing to the field value of the surface location. Note that the support surfaces T have to be the same (geometry and texture) in the blending parts.

Figures 10 and 11 demonstrate this method. In Figure 10 we have an articulated implicit arm with two links. Note that as the arm moves the texture deforms in a way that is consistent with the deforming shape. In Figure 11 we have a composite blobby model consisting of a skeleton with two point sources. In the initial configuration, the two sources are in the same position and, therefore, the entire surface is a blend of these sources. As the point sources move apart the object changes its topology, splitting into two separate connected components. Note that the texture follows the topology change and splits in the same way.

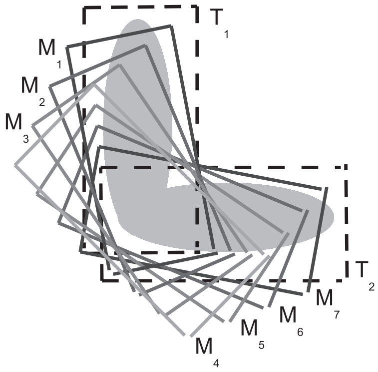

Figure 9: Transformations: M_1: $\alpha_1 = 0.9, \alpha_2 = 0.1$; M_4: $\alpha_1 = 0.5, \alpha_2 = 0.5$; M_7: $\alpha_1 = 0.1, \alpha_2 = 0.9$.

6 Conclusion

We have presented a method of applying 2D textures onto composite implicit surfaces with moving parts. The method has the advantage of maintaining coherence while the shape of an implicit object changes in time. Our method provides a natural initial texture placement based on the implicit object definition. It uses the geometric instance transformations applied to each primitive of a composite object in order to define a piecewise texture mapping with blending and solve the ghosting problem.

Our method provides several intuitive parameters that the user can control to determine how the texture is going to be applied to the implicit object. The variation of these parameters in time is a very effective way to animate the texture and produce special effects. These characteristics of the method have not been exploited in this paper.

In terms of future work, we are currently developing an interactive system for texture placement and animation. We are also working on examples with skin textures applied to articulated animals.

References

[1] A. Barr, *Decals*. In SIGGRAPH 83 State-of-the-Art of Image Synthesis Course Notes.

[2] J. F. Blinn, *A generalization of algebraic surface drawing*. ACM Transactions on Graphics, 1(3), pp. 235–256, 1982.

[3] J. Bloomenthal (ed.), *Introduction to Implicit Surfaces*. Morgan Kaufmann, 1997.

[4] J. Bloomenthal, B. Wyvill, *Interactive techniques for implicit modeling*. Proceedings of 1990 Symposium on Interactive 3D Graphics, pp. 109–116, 1990.

[5] L. H. de Figueiredo, J. Gomes, D. Terzopoulos, L. Velho, *Physically based methods for polygonization of implicit surfaces*. Proceedings of Graphics Interface '92, pp. 250–257.

[6] L. H. de Figueiredo, J. Gomes, *Sampling implicit surfaces with physically-Based particle systems*. Computer & Graphics, 20(3), 1996, pp. 365–375.

[7] H. Nishimura, M. Hirai, T. Kawai, T. Kawata, I. Shirakawa, K. Omura, *Object modeling by distribution function and a method of image generation*. Trans. IECE Japan, Part D J68-D(4), pp 718–725, 1985.

[8] D. Peachey, *Solid texturing of complex surfaces*. Proceedings of SIGGRAPH '85, pp. 279–286.

[9] K. Perlin, *An image synthesizer*. Proceedings of SIGGRAPH '85, pp. 287–294.

[10] H. K. Pedersen, *Decorating implicit surfaces*. Proceedings of the SIGGRAPH '95, pp. 291–300.

[11] J.-P. Smets-Solanes, *Vector field based texture mapping of animated implicit objects*. Eurographics '96 Conference Proceedings, Poitier, France, 1996.

[12] L. Velho, *Simple and efficient polygonization of implicit surfaces*. Journal of Graphical Tools, 1(2), 1996, pp. 5–24.

[13] L. Velho, L. H. de Figueiredo and J. Gomes, *A methodology for piecewise linear approximation of surfaces*. Journal of the Brazilian Computer Society, 3(3), 1997, pp. 30–42.

[14] B. Wyvill, C. McPheeters, G. Wyvill, *Data structures for soft objects*. The Visual Computer, 2(4), pp. 227–234, 1986.

[15] G. Wyvill, B. Wyvill, C. McPheeters, *Solid texturing of soft objects*. IEEE Computer Graphics & Applications 7(12), 1987, pp. 20–26.

[16] J. Yngvesson, I. Wallin, *SIPP – a 3D rendering library*. Available at ftp://ftp.isy.liu.se/pub/sipp.

[17] R. Zonenschein, J. Gomes, L. Velho, L. H. de Figueiredo, *Textura de superfícies implícitas com sistemas de partículas*. Proceedings of SIBGRAPI '95 pp. 305–306.

[18] R. Zonenschein, J. Gomes, L. Velho, L. H. de Figueiredo, *Texturing implicit surfaces with particle systems*. Visual Proceedings, SIGGRAPH '97, 172. http://www.visgraf.impa.br/Projects/dtexture/.

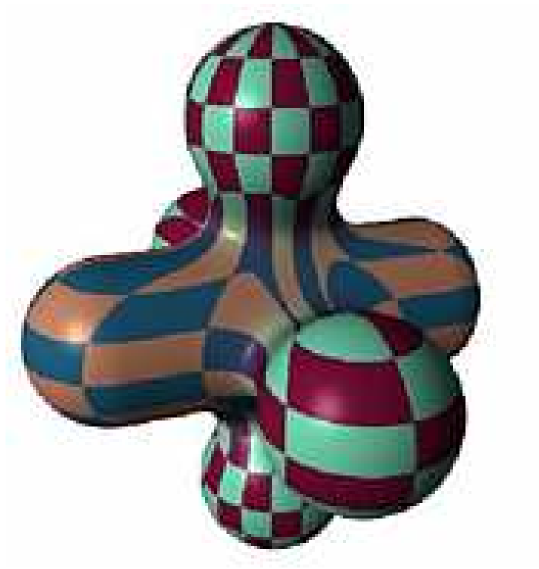

Figure 6: Combining different texture attributes using blending.

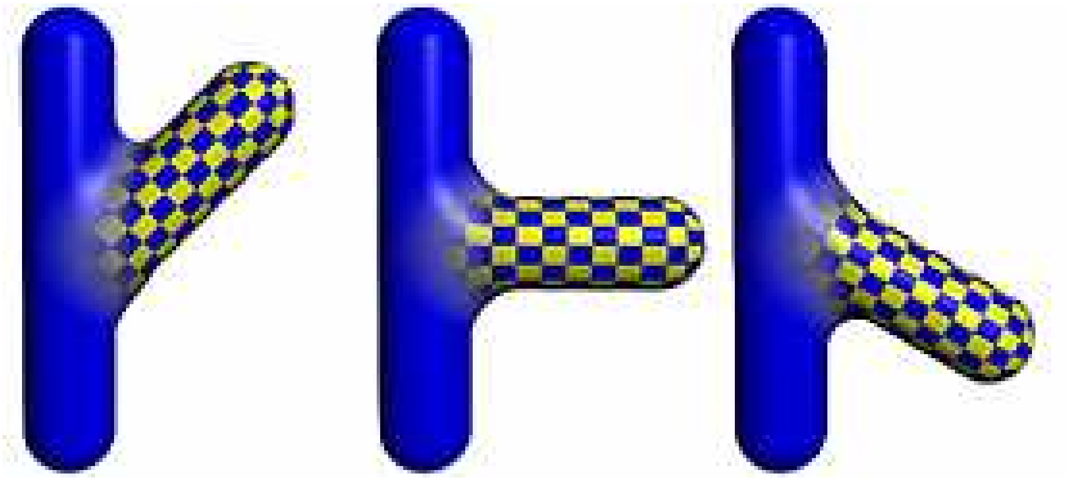

Figure 7: Blending of two textures in a moving articulated model.

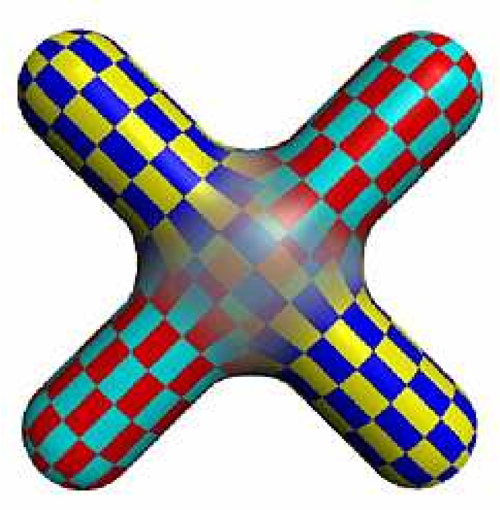

Figure 8: Ghosting effect caused by color blending of textures.

Figure 10: An arm maintaining texture consistency as the articulated joint rotates.

Figure 11: Two primitives keeping their textures while moving apart.