3D Texture Painting of Point Models

José Ricardo Mello Viana, Ricardo Marroquim, Cláudio Esperança
Universidade Federal do Rio de Janeiro - PESC/COPPE

Abstract

We propose a technique for painting 3D models based on a rendered image of the model and a 2D map of normal vectors. Texture generation and coordinate mapping is then produced on-the-fly, making it possible to paint in broad strokes or in small detail. The minimalistic requirements of this technique make it suitable for painting both regular meshes and point-based models.

1. Introduction

Most 3D painting systems follow the so-called *UV-mapping* paradigm. Thus, they require an initial step where an "unwrapping" map of the model surface is established. This is then followed by mapping the user's painting strokes on the obtained flattened canvas. Such a system is heavily dependent on the quality and resolution of the unwrapping map which ideally should contain as little angle distortion as possible [2]. The map should also minimize the extent of the texture seams which will be responsible for discontinuities on the map. Another problem is the fact that the method is not concerned with the areas of the model which will actually be painted: the texture resolution is fixed once the unwrapping map is established, making it hard to paint small detail in some parts of the model without making the texture bigger as a whole.

In this work we follow the ideas of the Chameleon system [1], where textures and maps are created dynamically, and only for affected parts of the model. Given a view of the model, the artist paints it as though it were a regular 2D canvas. Only after the view is changed – rotating or moving the model, for instance – is the painting transformed into a texture and a corresponding map.

2. Painting Interface

The painting interface is similar to that of other 3D painting systems. A 3D model is loaded and rendered, after which the user chooses a color for painting. The painting strokes are realized with the aid of a 3D cursor in the shape of a regular polygon with size and number of sides specified by the user. The location of the 3D cursor is specified with the mouse, but its projection on the screen follows the curvature of the model. By clicking and dragging the mouse over the model, the parts of the model traveled by the 3D cursor have their color changed. These paint strokes are in fact drawn on a temporary buffer which will later be used to build a proper texture.

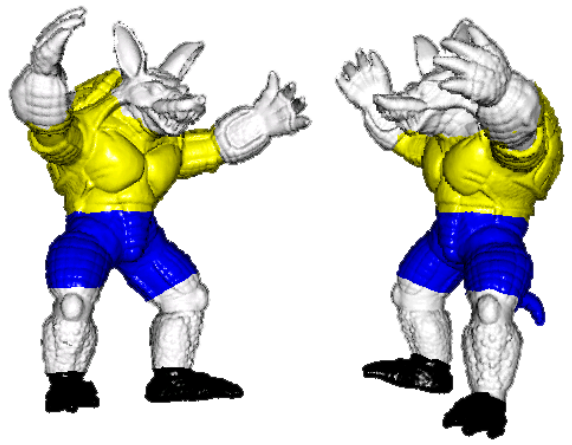
Figure 1. *Views of a model painted with our system.*

Despite the fact that no texture is generated at this stage, the user is shown a proper visual feedback by recomputing the illumination function of the screen pixels modified by the painting (see Figure 1). In order to compute the 3D cursor projection and the illumination function, a *Normal Buffer* is used, that is, a buffer which associates a normal vector with each color buffer pixel. The Normal Buffer also makes it possible to distinguish background pixels from pixels covered by the model (see Figure 2).

3. Texture Generation

Once the model view is zoomed, rotated or translated, a new texture is created containing all paint strokes input by the user since the previous view change. This corresponds to (1) a texture image and (2) a set of texture coordinates which will be assigned to the affected model points.

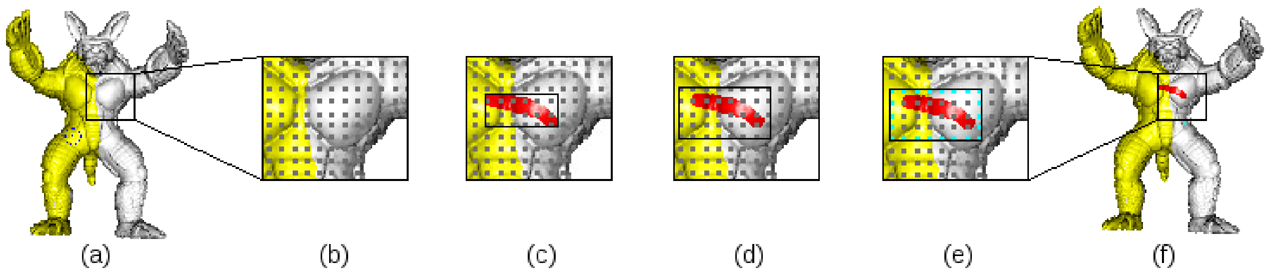

Figure 3. *Drawing texture passes: (a) input model for paint stroke, (b) points of affected area, (c) bounding box of new stroke, (d) expanded bounding box containing all points related with the stroke, (e) critical points (green) possibly associated with more than one texture, (f) model with new stroke.*

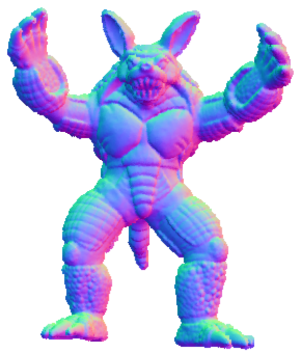

Figure 2. *Normal Buffer.*

The texture image should contain not only the newly painted strokes, but also the background pixels in the neighborhood of the painted areas. In fact, the texture image corresponds to all color buffer pixels inside a bounding box B large enough to contain all strokes, plus a safety margin. This safety margin is the maximum edge length in the case of a mesh model, or the maximum distance between neighbor samples in the case of a point-based model. For efficiency considerations, texture images are not handled individually, but rather copied as patches in a large texture atlas. This, however, makes it necessary to adopt some sort of management of the atlas space such as the one proposed by Igarashi [1].

For a point of the model to be assigned a new set of texture coordinates, it must satisfy the following conditions: (a) it must be visible in the current view, and (b) it must fall inside bounding box B. The selection of these points is performed with the aid of an *Id Buffer*, which maps screen pixels to model points. Notice, however, that a given point may participate in more than one texture image. This may occur for vertices close to the border of B. As a consequence, drawing the textured model may require several passes (see Figure 3).

4. Point-based rendering

As explained above, the system uses two auxiliary buffers (Normal and Id) which must be supplied along with a rendered view of the model. Although these could be easily obtained for regular mesh models, our proof-of-concept prototype implements texture painting on point-based models. This is accomplished with the help of the point-based renderization technique described in [3].

5. Conclusions

Although model rendering is mostly implemented in GPU, the texture painting algorithms are still processed entirely in CPU. We are currently at work implementing a better-performing prototype where texture painting is mostly implemented in GLSL, like [4].

An important topic not addressed by Igarashi is that of texture patch reuse. We are experimenting with an algorithm for counting texture patches in which each point participates and eliminating unused patches.

References

[1] T. Igarashi and D. Cosgrove. Adaptive unwrapping for interactive texture painting. In *I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 209–216, New York, NY, USA, 2001. ACM.

[2] B. Lévy, S. Petitjean, N. Ray, and J. Maillot. Least squares conformal maps for automatic texture atlas generation. *ACM Trans. Graph.*, 21(3):362–371, 2002.

[3] R. Marroquim, M. Kraus, and P. R. Cavalcanti. Efficient point-based rendering using image reconstruction. In *Symposium on Point-Based Graphics 2007, Prague-Czech Republic*, September 2007.

[4] T. Ritschel, M. Botsch, and S. Müller. Multiresolution gpu mesh painting. In *Eurographics 2006 Short Papers*, pages 17–20, 2006.