Adaptive Unwrapping for Interactive Texture Painting

Takeo Igarashi

Computer Science Department
Brown University
takeo@acm.org

Dennis Cosgrove

School of Computer Science
Carnegie Mellon University
dennisc@cs.cmu.edu

ABSTRACT

We present a method for dynamically generating an efficient texture bitmap and its associated UV-mapping in an interactive texture painting system for 3D models. Typical 3D texture painting programs require the user to explicitly define the underlying UV-mapping from 3D geometry to 2D bitmap prior to painting. This mapping is unchanged by the painting process. However, a predefined UV-mapping can cause distortion at arbitrary locations and waste bitmap memory in unpainted areas. To solve these problems, we propose an adaptive unwrapping mechanism where the system *dynamically* creates a tailored UV-mapping for newly painted polygons during the interactive painting process. This eliminates the distortion of brush strokes, and the resulting texture bitmap is more compact because the system allocates texture space only for the painted polygons. In addition, this dynamic texture allocation allows the user to paint smoothly at any zoom level. This technique can be efficiently implemented using standard 3D rendering engines, and the painted models can be stored as standard textured polygonal models. We implemented a prototype system, called Chameleon, and our users' experiences suggest that our technique is very useful for simple painting by casual users.

Keywords

Texture mapping, texture painting, interactive 3D graphics, multiresolution paint, zooming, 3D content creation.

1. INTRODUCTION

This paper addresses the problem of interactively creating and refining a hand-painted texture on a 3D polygonal model. Traditionally, the user first specifies the UV-mapping that maps 3D geometry onto the 2D texture bitmap (the process is called *unwrapping*), and then paints on the 2D texture bitmap using various paint tools. 3D painting systems make the process easier by allowing the user to paint the object surface directly in the 3D view. The system re-projects the painted strokes in the 3D view to the 2D bitmap according to the predefined UV-mapping, and instantly presents the result in the 3D view.

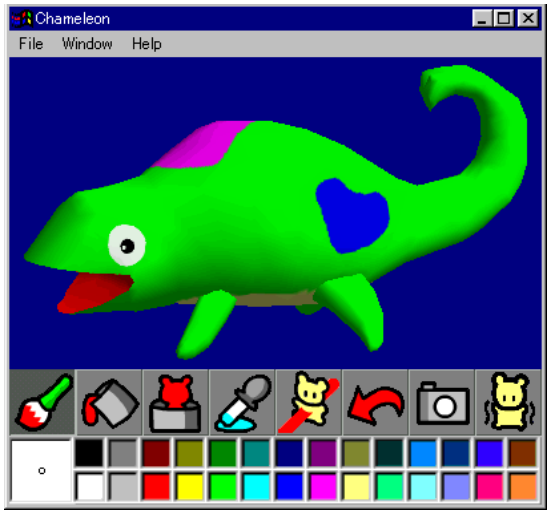


Figure 1: A screen snapshot of Chameleon. The user paints strokes on the 3D model directly without specifying the UV mapping beforehand.

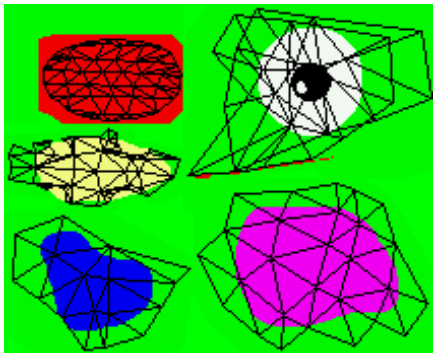


Figure 2: Automatically generated texture and the UV-mapping for the painted model in Figure 1. Each patch corresponds to a meaningful visual feature, and the bitmap memory is not assigned for the areas that are not touched by the brush strokes.

However, traditional 3D paint programs have a number of limitations. First, specifying the UV-mapping manually can be difficult and tedious. While manual unwrapping is the preferred solution for professional designers to obtain the best aesthetic results, this approach is too difficult for casual users. Mapping by

hand often takes longer than the painting itself. A standard set of predefined mappings, such as cylindrical and spherical, work well for simple shapes, but fail for models with extrusions and concavities. Several advanced automatic unwrapping methods are available in commercial systems [6], but they still require the user to manually adjust several parameters to get a customized mapping for a specific painting. Without explicit guidance from the user, the automatically generated mapping can place seams at important visual features and cause distortion. In addition, automatic methods usually distribute the bitmap evenly across the entire surface, but this can be wasteful when the user adds details to only a few polygons and fills the remaining areas with a solid background color. In such cases, it is desirable to allocate more bitmap memory for the detailed area, and a minimum amount for the remaining area.

Our approach is to perform unwrapping *on the fly* during the user's painting operation instead of constructing a static UV-mapping beforehand. The system assigns a new texture bitmap and UV-coordinates for newly painted polygons with each paint operation. This improves interactive 3D paint systems in several ways. First, the brush strokes are never distorted during the paint operation, and several useful behaviors can be added to the brush strokes such as the ability to paint both front and back surfaces simultaneously. Second, the resulting texture bitmap is compact and robust, because texture bitmap is assigned to painted areas only and most of the seams run through the gaps between these painted regions. Finally, dynamically allocating texture bitmap enables different levels of texture detail across the 3D surface (*multiresolution paint*). We built a prototype system, Chameleon, to show that our adaptive unwrapping method can be efficiently implemented using a standard 3D rendering engine to provide the user with immediate feedback during painting operations. We performed a qualitative user study using a prototype system, and observed that Chameleon is particularly suitable for simple paintings by casual users that involve a limited number of strokes.

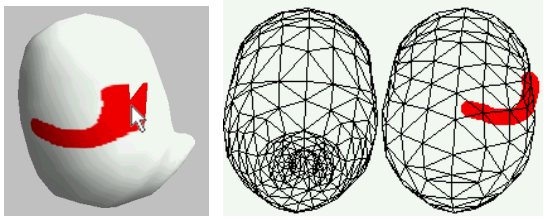
The rest of this paper is organized as follows: we first describe the background of this research. Second, we introduce the Chameleon system from the user's point of view. Third, we describe the implementation of Chameleon in detail. We then present some results from our user study. Finally, we discuss the limitations of our approach and our planned future work before concluding the paper.

2. BACKGROUND

Texture mapping applies a 2D bitmap image to 3D geometry to represent additional visual features without increasing the complexity of the underlying geometry [8]. A fundamental difficulty of texture mapping is finding a good surface parameterization (UV-mapping) that minimizes distortion. A typical approach is to define an energy function for the mapping and to minimize it through an optimization process [3][13]. The problem with these approaches is that the result is a general solution and may not be appropriate for individual paintings. Therefore, professional artists prefer to manually specify the UV-mapping to obtain the best aesthetic results. Our approach is to incorporate the user's painting operation as an input to generate a customized mapping tailored for the particular painting sequence.

3D texture painting was first introduced by Hanrahan and Haeberli [9], but they stored painted color information at the mesh vertices instead of using a separate texture bitmap. Today, many commercial programs provide 3D painting features, where the user can paint directly in a 3D view to edit a 2D texture bitmap [2][6][16]. In these systems, the user first prepares a UV-mapping manually or automatically (e.g. V.A.M.P. mapping in [6]), and the system subsequently re-projects the user's strokes in the 3D view to the corresponding position in the 2D bitmap based on the predefined UV-mapping. The problem is that the brush strokes are often distorted because of the underlying UV-mapping. If an area of the 3D surface is stretched out in the 2D bitmap, the brush stroke becomes small in the 3D view, and vice versa. If an area of the 3D surface is split in the 2D bitmap, the discontinuity becomes visible when the user's stroke runs across the seam (Figure 3). These problems can be mitigated by adaptively adjusting the brush size [12], or by drawing strokes on the screen first and then re-projecting them onto the texture bitmap on a per pixel basis (e.g. projection paint in [6]), but the problems cannot be easily solved comprehensively as long as the underlying UV-mapping remains unchanged.

Our work is inspired by the notion of 3D graphics for casual users (including children) introduced in [4][11]. In general, traditional 3D graphics tools and techniques are designed for professional users, and the quality of resulting imagery is more important than the usability of the system. However, the ease of use is most important for casual users, and the technological challenge is to produce the best possible results from minimum user effort. Based on this principle, our technique tries to generate ideal textures based on the user's natural painting operation.



3D view 2D bitmap

Figure 3: A problem with traditional 3D texture painting. The user's painting operation is re-projected onto the 2D bitmap, and the system draws a brush stroke on the 2D bitmap. As the result, a distorted stroke appears in the rendered 3D view.

3. THE CHAMELEON SYSTEM

This section describes the behavior of Chameleon from the user's point of view. We also clarify the benefits of our approach.

Basically, the Chameleon system works as a standard 3D paint program except that it does not require the user to specify the UV-mapping beforehand. The user loads a polygonal model into the system, specifies brush size and color, and draws strokes on

the 3D model directly as shown in Figure 1. Currently, the left mouse button paints strokes, and the right mouse button controls the camera. Dragging on the model rotates it [10], and dragging on the background moves the model parallel to the screen. Dragging immediately after a click by the right mouse button changes the zoom level.

The brush size and shape remain constant wherever the user paints. This is in contrast to traditional 3D paint systems where the effective brush size and shape can change in the 3D view because of a distorted UV-mapping [12]. In addition, the brush works intelligently based on the structure of underlying geometry. When a brush stroke paints near a boundary edge, the ink does not spill across the edge (Figure 4, left). The boundary edges can be given by the modeling system (for example, the Teddy system [11] creates boundary edges as a result of cutting or extrusion operation), or explicitly specified by the user. Chameleon also allows the user to paint areas that are partially hidden by other surfaces (Figure 4, right).

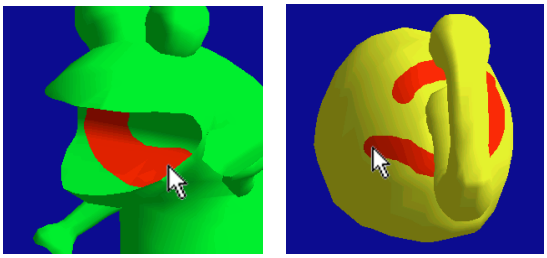


Figure 4: Smart brush. The brush does not spill across boundary edges (left). The brush remains on the surface being painted (right). The structural information can be automatically provided by a modeling system or explicitly specified by the user.

The buttons at the bottom of the panel (Figure 1) allow the user to choose brushes and to perform various operations. The second button from the left, flood fill, allows the user to specify the color for an entire area with a single step. The area is the set of polygons contiguous to the polygon the user clicks that are either enclosed by boundary edges or are all the same color (whichever is smaller). The third button, dunk fill, fills the entire model with the target color. The fourth button, eye dropper, picks a color from the model surface and sets it to be the current painting color. The fifth button is a laser brush. While this button is on, the user's strokes penetrate the model and paint both front and back surfaces. This makes it convenient to, for example, paint wings or a tiger's tail (Figure 5). The sixth button is undo. Our current implementation supports unlimited undo operations. The seventh button allows the user to rotate the model to a fixed viewing angle (we provide predefined front, left, right, back, top, and bottom views for convenience). The final button is not functional in the current implementation.

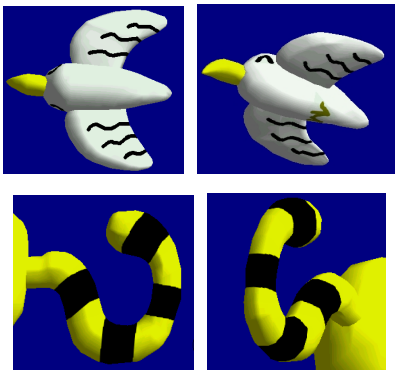


Figure 5: Laser paint. When the user paints strokes in laser mode (left), the strokes penetrate the model and are visible from both sides (right).

Zooming in our system works differently from standard 3D or 2D painting systems where the predefined texture resolution limits the ability to represent details. With these systems, as the user zooms in, the brush strokes exhibit aliasing. In Chameleon, the user can paint with constant smoothness even in an extremely zoomed-in view (called multiresolution paint [7]) (Figure 6). As a result, the user can add as much detail as is desired to specific areas of the model.

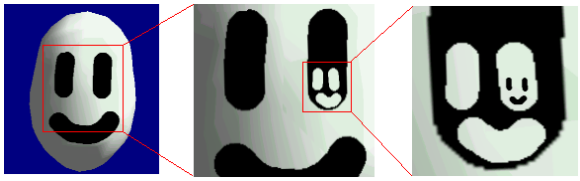


Figure 6: Multiresolution paint. In any zoom level, the user can always paint smooth strokes.

When the user finishes painting, the system stores the texture bitmap and the geometry with the assigned UV coordinates. The resulting bitmap has a number of good characteristics (Figure 2). The seams mainly run through the gaps between important visual features. This prevents distortion when reducing the size of the bitmap later (Figure 13). The bitmap is assigned mainly for the areas where the user painted with brush strokes, and the rest of the surface does not waste texture memory. In addition, the system assigns an appropriate amount of bitmap according to the visual complexity of each area (e.g. eyes typically consume a large area, while the identically colored belly consumes a smaller area).

4. IMPLEMENTATION

This section describes the implementation details of Chameleon. We emphasize how we achieve fast feedback to the user during the painting operations using a standard texture mapping engine. We wrote our prototype system in Java using Microsoft's DirectX libraries [14].

4.1 Painting Strokes

The essential idea is to construct the UV-mapping and the

corresponding texture bitmap on the fly during the user's painting operation instead of using a predefined UV-mapping throughout the painting process. The system identifies the painted polygons each time the user paints strokes, and assigns new UV-coordinates and a new texture bitmap to them. Figure 7 illustrates an overview of the process. Internally, the system uses the standard textured polygonal model, and renders it using a standard 3D rendering engine.

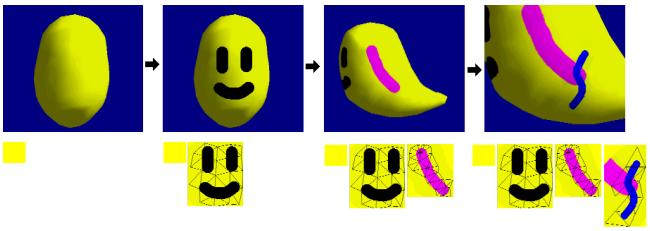


Figure 7: Overview of the painting process. As the user paints strokes (above), the system assigns new texture bitmap and new UV-coordinates to the painted polygons (bottom).

When the user loads an unpainted model or dunks a model, the system sets the UV-coordinates to (0.5, 0.5) and associates all polygons with an initial blank texture bitmap. The initial blank texture bitmap is of 1x1 pixels, and the system sets its color to the base color specified by the user. When the user paints strokes, the system stores these strokes as *incoming strokes* until the user rotates the model. The system represents the incoming strokes as independent 2D strokes on the screen. When the user starts rotating the object, the system projects the incoming strokes to the object surface (Figure 8). This idea is similar to projection paint in [6], but they create a 2D bitmap representation of incoming strokes, and re-project each pixel to the texture bitmap according to a predefined UV-mapping.

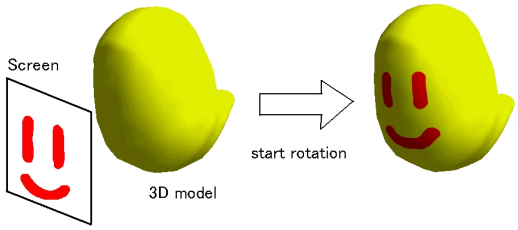


Figure 8: Projection of incoming strokes. Incoming strokes are represented as independent 2D strokes on the screen until the user starts rotation. When the user starts rotation, the incoming strokes are projected onto the 3D surface.

The projection is done by the following procedure (Figure 9). First, the system identifies the polygons that are painted by the strokes (Figure 9a). The system searches for the painted polygons starting from the polygon where each stroke starts, and recursively checking the distance between the adjacent polygons and the stroke in screen space. If the distance is shorter than the stroke radius, the polygon is identified as painted. This recursive search along the surface prevents the stroke from affecting irrelevant polygons, and the search stops at the boundary edges to prevent spilling. Second, the system generates a new texture bitmap, and re-projects any existing texture for the painted polygons onto the new texture (Figure 9b). The size of the new texture bitmap is identical to the bounding box of the painted polygons in the screen space. Third, the system paints the incoming strokes on the new texture using a standard 2D drawing procedure (Figure 9c). Finally, the system updates the UV-coordinates of the painted polygons and associates them with the new texture (Figure 9d). This projection is simple and fast because it does not require complicated pixel level operations. All operations are performed using a standard rendering engine and standard 2D brush drawing.

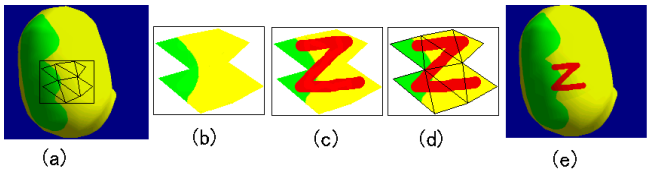


Figure 9: Projection of the incoming strokes onto the model surface.

The system re-projects the existing texture (Figure 9b) using a standard rendering pipeline. The system renders the painted polygons as a 3D scene to an off-screen image buffer with an ambient light of maximum brightness (called *bright light rendering*). In the actual implementation, we render additional polygons around the painted polygons to prevent the boundary from becoming visible in the 3D scene. This is also important to prevent aliasing when the bitmap is scaled down later (Figure 13).

In laser paint mode, the system applies the projection procedure to both the front and back facing polygons. Both polygons share an identical texture bitmap to minimize memory consumption. This causes problems when different images are already painted on the front and back facing surfaces. We could avoid this problem by assigning independent texture bitmaps for front and back facing polygons, but we chose to minimize bitmap usage in the current implementation.

We automatically achieve multiresolution paint in this framework because the resolution of the new texture bitmap is determined by the apparent screen resolution. However, the size of underlying polygons does impose a limitation. When the user zooms in too much and the strokes are much smaller than the size of the painted polygon, the system wastes a large amount of bitmap space, as each texture bitmap must be larger than the polygon. A potential solution is to sub-divide the polygon to generate a finer mesh, but we have not tested this yet.

4.2 Feedback for Incoming Strokes

The simplest approach for presenting feedback for the incoming strokes is to paint them as a 2D overlay on top of the rendered 3D scene. This is fast and easy to implement, but cannot reflect the characteristics of painted surfaces correctly (Figure 10, left). Because the users in our informal user tests universally responded negatively to this initial implementation, we implemented the following technique to provide accurate feedback. Using this technique, the incoming strokes are properly

shaded, and they do not appear outside the appropriate places.

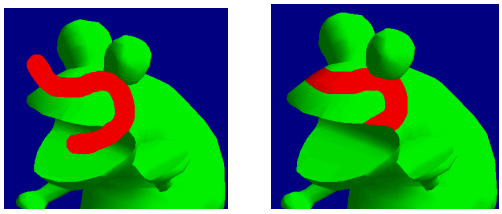


Figure 10: Feedback for the incoming strokes. The simplest approach is to paint the incoming strokes as 2D overlay over the 3D scene (left). We paint the incoming strokes on temporary texture to add appropriate effects (right).

When the user finishes rotation, the system prepares a temporary texture bitmap for the incoming strokes. The system temporarily associates all polygons with the temporary texture and assigns them temporary UV coordinates. The temporary texture is identical to the 3D view of the model on the screen except that the temporary texture is rendered using bright light rendering to suppress shading effects (Figure 11). The system assigns UV coordinates accordingly to make the resulting 3D image indistinguishable from the image rendered using the original textures.

The system then paints incoming strokes on this temporary texture bitmap using a standard 2D brush. As a result, the strokes appear as shaded strokes on the model surface, and they never appear outside the object in the 3D view (Figure 10, right). When the user starts rotation, the system discards the temporary texture and the temporary UV coordinates. This also allowed us to implement the eye dropping by picking the color from this temporary texture bitmap at the desired location.

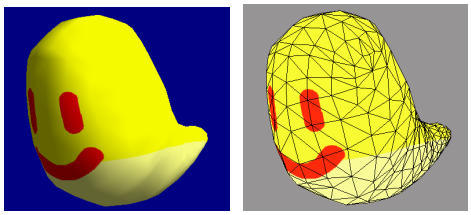


Figure 11: 3D view on the screen (left) and the temporary texture (right). The incoming strokes are painted on the temporary texture bitmap, and appear in the 3D view as shaded strokes.

One remaining problem is that this approach cannot give appropriate feedback when the user paints strokes across boundary edges or behind other parts (Figure 4). To solve this problem, the system generates multiple copies of the temporary texture bitmap, and assigns a copy to each part separated by boundary edges. When the user draws an incoming stroke, the system identifies the part to be painted and renders the incoming stroke on the appropriate copy of the temporary texture bitmap.

4.3 Packing

When the user finishes painting and attempts to save the result, the system generates a single texture bitmap (texture atlas [17]) by assembling the number of texture bitmaps created during the painting process. We call this process *packing*. The system adjusts the UV coordinates to match the packed texture, and stores the result in a standard textured polygonal model format, making the completed painted model readable by various graphics applications.

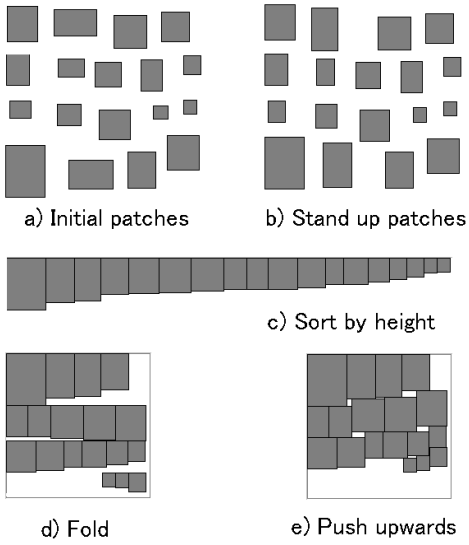
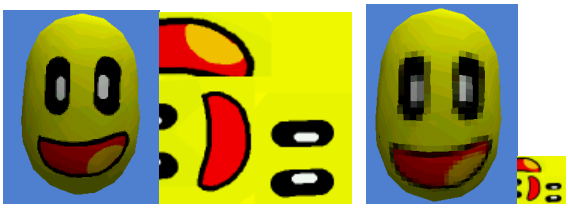
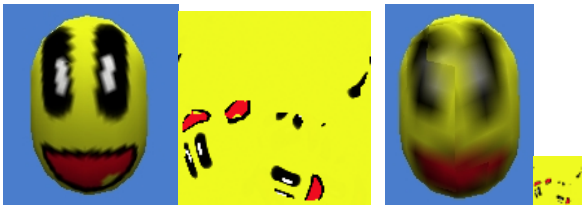


Figure 12: Packing algorithm.



A texture created using Chameleon.



A texture created using an automatic unwrapping [6].

Figure 13: Scaling the packed texture from 128x128 (left) to 32x32 (right). The texture created by Chameleon is robust against scaling because the texture bitmap is dedicated to the important visual features and the seams run through the gaps among the painted regions.

The difficult part is to arrange the small texture bitmaps (*patches*) inside the final texture compactly. This problem of packing a set of polygons into a given 2D domain is called “pants packing” [15] and is well studied in computational geometry.

Figure 14: Example 3D models painted by one of the authors using Chameleon, and resulting texture mapping. Chameleon is suitable for simple paintings with a limited number of strokes.

However, an exact solution cannot generally be computed because the problem is NP-hard. A typical approach is to ask the user to do the task manually when high quality packing is required.

In our current implementation, we use a simple heuristic algorithm. This algorithm is a temporary solution and we plan to investigate more sophisticated implementations in the future. However, the algorithm described below is easy to implement, fast, and produces reasonable results, which is sufficient for our prototype.

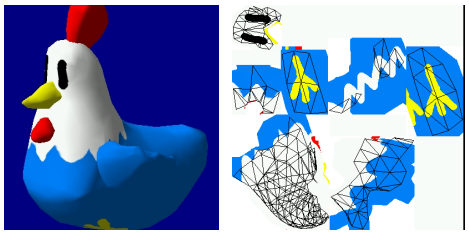
As a preprocessing step, the system obtains the bounding box of each patch, and handles each as a rectangular box throughout the packing process (Figure 12a). The system calculates the total area of the patches, and gets the square root of the total area. The system defines the target size (L) of the resulting texture as slightly longer than the square root (multiplied by 1.2). After this, the system first finds the patches whose width is longer than their height and rotates them 90 degrees. As a result, all patches are taller than they are wide (Figure 12b). Then the system sorts the patches by their height, and lines them up horizontally (Figure 12c). The system then folds the line as shown in Figure 12d to fit within a square whose side length is L, and pushes each patch upward until it hits another patch to minimize the gap (Figure 12e). We found that this simple algorithm works reasonably well for the typical sets of patches we see in our experiment (Figure 14,15).

The result of this straightforward packing algorithm can be too large for certain purposes. If the user wants to send the painted model over the network, a smaller texture bitmap would be preferable. In these cases, it is necessary to shrink the packed texture bitmap to a lower resolution. However, reducing the texture resolution can make seams visible, and this damages rendered images because patches start to be affected by the surrounding pixels. The texture atlas created by Chameleon is more robust against this aliasing problem because more seams are arranged in unpainted areas, unlike an automatically generated UV mapping in traditional 3D paint systems (Figure 13). It is possible to prevent the aliasing problem by implementing a dedicated scaling procedure that takes cares of the patch boundaries, but we have not worked on this yet.

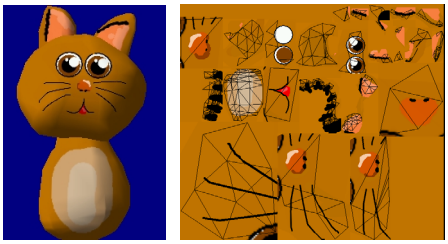
5. RESULTS

Figure 14 shows several models painted by one of the authors using Chameleon. It took less than a few minutes to paint each of them from scratch. The rendered images are clear and the texture bitmap is compact and efficient. We used a 500 MHz Pentium III PC without any hardware acceleration, and the computing time is almost unnoticeable during the painting operation. These models also represent the target class of paintings for Chameleon. Only a few areas are touched, and most areas are covered by the base color. Different areas require different levels of detail. They are informal paintings rather than professional models to be used in production.

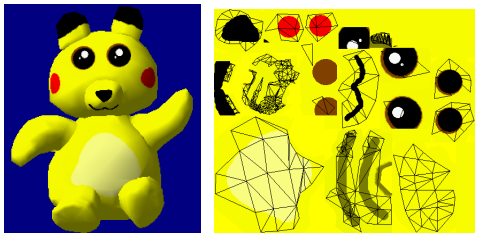
We performed a user study to observe how users react to the system, and to obtain insights about their paintings. Fifteen participants from various backgrounds took part in the study: four digital artists/designers, four computer scientists and seven casual users. We first asked them to try Chameleon on a practice model without any instruction. Then, we provided them with complete tutorial and asked them to freely paint five models (cat, chicken, elephant, monkey, and bear). As a result, we got 75 paintings in total. Average time for a subject to complete a painting was approximately 4.8 minutes. Figure 15 shows some of the paintings by the subjects. In general, the subjects found Chameleon intuitive, and painted the models without difficulty.



subject #0, 22% are painted, 4 minutes.



subject #3, 48% are painted, 11 minutes.



subject #10, 26% are painted, 6 minutes.

Figure 15: Examples of paintings by the subjects.

We are very encouraged that while those with experience in using commercial 3D painting tools are impressed with the various features of Chameleon, casual users simply take it for granted that one can interactively paint directly on a 3D model and receive the correct feedback in real-time. The laser tool turned out to be very useful. Several users struggled with repeatedly brushing and rotating to cover all sides of an extremity or complete a stripe on the tail of the practice model. They were all pleased when shown the laser tool in the tutorial portion of the study, and they proceeded to use it extensively throughout the remainder of the study.

In order to verify our assumption that the casual users paint only limited areas of the surface in informal paintings, we counted the number of polygons that were painted by brush strokes and calculated the percentage of the painted polygons for each painting. Polygons filled by flood fill and dunk fill are not included in the painted polygons. Figure 16 shows the histogram of the result. Although we did not provide any direction on how to paint the model, most users painted only a limited percentage of the entire surface (the average was 24.3%).

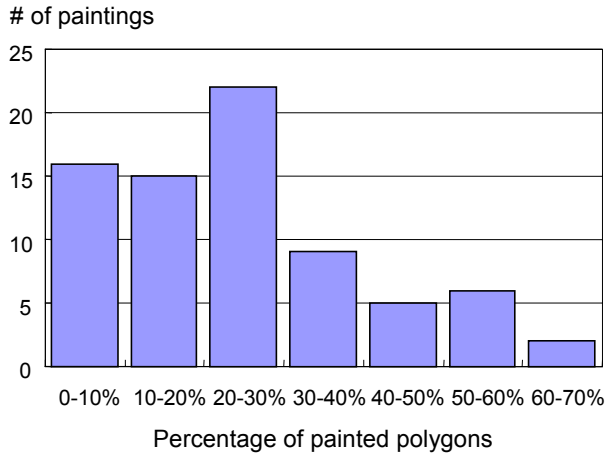


Figure 16: The percentage of painted polygons in the 75 paintings by the subjects. The average is 24.3%. In most cases, only 20-30% of the entire surfaces are painted by brush strokes.

6. LIMITATIONS AND FUTURE WORK

Chameleon is specifically designed for casual users to quickly paint simple paintings such as those in Figure 14. Chameleon is not appropriate for professional designers to paint highly detailed models. New paint strokes affect the appearance of previously drawn textures, and the resulting texture is not necessarily efficient if almost all surfaces are painted. Seams become visible when the user covers the entire surface with complicated patterns such as wood or bricks.

In addition to this fundamental limitation, there are many problems to be fixed in our current implementation. The biggest problem is with flood fill. Our current implementation simply applies standard 2D flood fill to the corresponding texture bitmap, but the result can be inconsistent in the 3D view because the region to be filled can be spread across multiple patches. We need to extend the flood fill algorithm to recursively search for adjacent patches on the polygonal surface.

A second problem is that repeated re-projection of existing texture gradually degrades the quality of the bitmap image. A possible solution is to apply multiple layered textures to the surface with transparency. When the user paints new strokes on already-painted area, the system could convert new strokes into a 2D texture bitmap with transparency, and add them to the surface on top of the existing texture. Limitations of this approach are that it requires a special 3D rendering engine with support for multiple layered textures, and that it consumes a lot of resources when the user paints strokes over and over.

There is also much room for improvement in our packing algorithm. We need an algorithm that can arrange polygons in a square more efficiently using various optimization techniques. In addition, we need to implement a dedicated scaling procedure to shrink the packed texture while taking care of the patch boundaries.

Another important future direction is to re-mesh the underlying geometry as the user paints. In the current implementation, a tiny stroke can affect a large polygon even when the stroke affects only a small area of the polygon. In this case, it would be desirable to sub-divide the polygon to minimize the area to be re-projected. Re-meshing could also make multiresolution painting work better in an extremely zoomed-in view.

Finally, it is possible to represent all strokes as 3D strokes on the object surface instead of using standard texture mapping methods [1][5]. This approach is superior in that there will be no sampling problem and the strokes are always smooth. We chose our current implementation because it is relatively easy to implement and runs robustly on a standard rendering environment, but we plan to investigate this approach in the future.

7. CONCLUSIONS

We have introduced a technique for creating an efficient UV mapping for interactive texture painting programs. While traditional 3D paint programs use a predefined UV-mapping, our system dynamically creates a tailored UV mapping for newly painted polygons during the painting process. As a result, the user can paint undistorted brush strokes all over the surface from any viewing direction. The resulting texture bitmap is compact because the bitmap is dedicated to the painted areas only, and it is robust against scaling because seams run through the gaps among the painted regions. In addition, dynamically assigning the texture allows the user to paint smooth strokes at any zoom level.

This technique can be implemented efficiently using standard 3D rendering capabilities without writing code for pixel level operations. The result of painting is stored as a standard textured polygonal model and is readable by a variety of graphics applications. Our informal user study showed that novice users easily understand the technique and that the system can generate appropriate UV mappings for their paintings.

References

[1] Akleman, E., Implicit Surface Painting, *Proceedings of Implicit Surfaces'98*, pp.63-68, 1998.

[2] AMAZON 3D Paint, Interactive Effects, Inc., www.ifx.com

[3] C. Bennis, J-M. Vezien, G. Iglesias, A. Gagalowics. Piecewise surface flattening for non-distorted texture mapping. *SIGGRAPH 91 conference proceedings*, pp.237-246.

[4] M. Conway, *et. al.* Alice: Lessons Learned from Building a 3D System for Novices. *Proceedings of CHI 2000*, pp.486 - 493, 2000.

[5] E.Daniels , Deep canvas in Disney's Tarzan, *SIGGRAPH 99: conference abstracts and applications,* pp. 200, 1999.

[6] Deep Paint 3D (Texture Weapons), Right Hemisphere Ltd., www.righthemisphere.com

[7] A. Finkelstein, C.E. Jacobs, D.H. Salesin. Multiresolution Video. *SIGGRAPH 96 Conference Proceedings*, pp. 281-290, 1996.

[8] P. Heckbert. Survey of Texture Mapping, *IEEE Computer Graphics and Applications.* Nov. 1986, pp.56-57.

[9] P. Hanrahan, P. Haeberli, Direct WYSIWYG Painting and Texturing on 3D Shapes, *SIGGRAPH 90 Conference Proceedings*, pages 215-224, 1990.

[10] J. Hultquist. A virtual trackball. *Graphics Gems* (ed. A. Glassner). Academic Press, pages 462-463, 1990.

[11] T. Igarashi, S. Matsuoka, H. Tanaka. Teddy: A Sketching Interface for 3D Freeform Design, *SIGGRAPH 99 Conference Proceedings*, pp. 109-116, 1999.

[12] D.E. Johnson, T.V. Thompson II, M. Kaplan, D. Nelson, E. Cohen, Painting Textures with a Haptic Interface, *Proceedings of Virtual Reality '99*, pp. 282-285, 1999.

[13] J. Maillot, H. Yahia, A. Verroust, Interactive texture mapping. *SIGGRAPH 93 Conference Proceedings*, pp.27-34, 1993.

[14] Microsoft DirectX APIs Supported by Java Classes, http://www.microsoft.com/Java/sdk/40/relnotes_sdk/directx.htm

[15] V.J. Milenkovic. Rotational polygon containment and minimum enclosure. *Proc. of the 14th Annual Symposium on computational Geometry*, 1998.

[16] Painter 3D, Metacreations, www.metacreations.com

[17] E. Praun, A. Finkelstein, H. Hoppe, Lapped Texture, *SIGGRAPH 2000 conference proceedings*, pp.465-470.

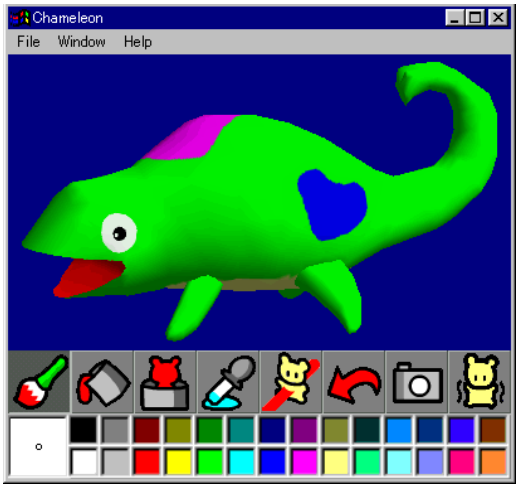


Figure 1: A screen snapshot of Chameleon.

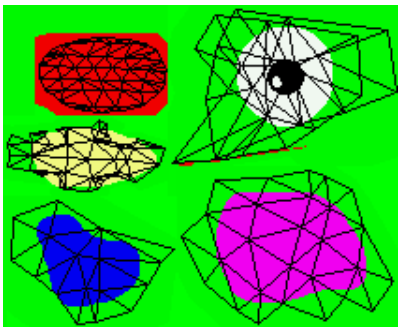


Figure 2: Automatically generated texture and the UV-mapping for the painted model in Figure 1.

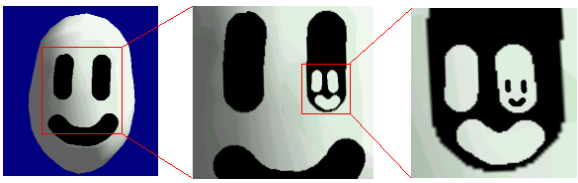


Figure 6: Multiresolution paint.

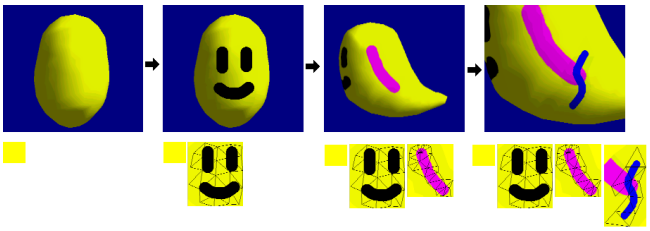


Figure 7: Overview of the painting process.

Figure 9: Projection of the incoming strokes onto the model surface.

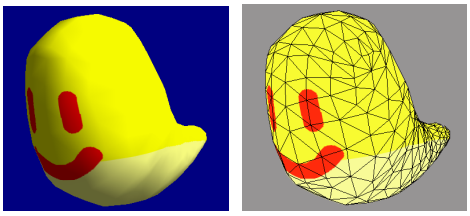
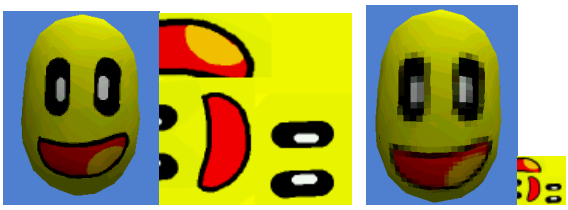
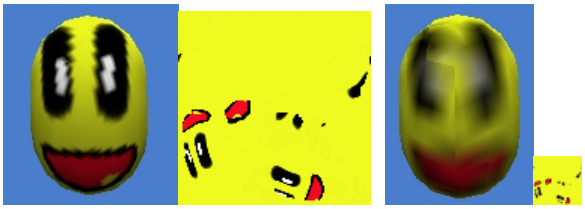


Figure 11: 3D view on the screen and the temporary texture.



A texture created using Chameleon.



A texture created using an automatic unwrapping [6].

Figure 13: Scaling the packed texture from 128x128 (left) to 32x32 (right).

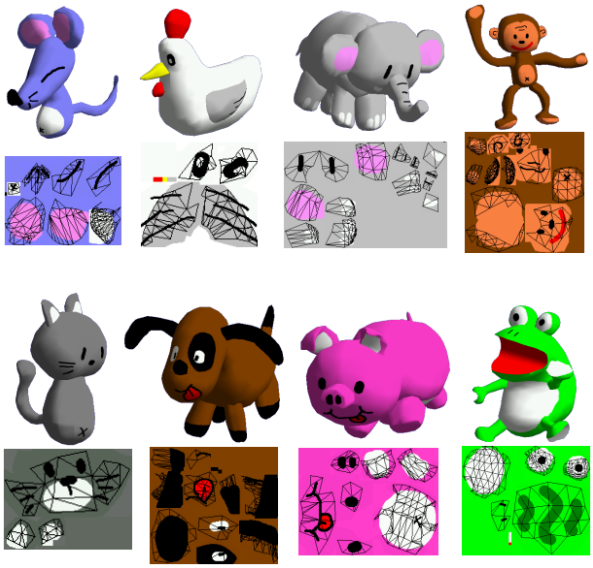


Figure 14: Example 3D models painted by one of the authors using Chameleon, and resulting texture mapping.