Specifying Interaction Surfaces Using Interaction Maps

Jeffrey S. Pierce Randy Pausch

January 2001
CMU-CS-01-100

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

This research was sponsored by the US Army Research Laboratory (USARL) under contract no. DAAD17-99-C-0061 and by a Microsoft Graduate Fellowship. Additional support was provided by Intel.

Keywords: HCI (human-computer interface), texture mapping, paint systems, curves & surfaces

Abstract

Defining how 3D models respond to user actions is a crucial step in building an interactive 3D world. Unfortunately, existing tools make it very difficult for interaction designers to assign responses to any part of a 3D model that is not a pre-defined group of polygons. This is particularly problematic for applications that use image-based models or models where most of the detail is in the model's texture map. We present a flexible and easy-to-use technique that overcomes this problem by allowing a designer to specify *interaction surfaces*, parts of the model he wants to respond to events, by painting them onto the model. We capture the painted areas by projecting them onto a 2D *interaction map*. An interaction map is similar to a traditional texture, but it specifies interaction surfaces instead of affecting a model's appearance. We allow designers to name interaction surfaces and then assign them responses to events both statically and at run-time. In addition, designers can modify the size and shape of interaction surfaces at run-time and can pass parameters to these surfaces' responses by encoding them in the model's interaction map.

1 Introduction

In recent years researchers have worked to simplify the process of building interactive 3D worlds by creating tools that allow users to focus on their particular task instead of its underlying implementation. Modelers can now create a 3D model by sketching it [11][19] instead of explicitly specifying its polygonal mesh, and artists can paint a texture on a model without explicitly specifying the underlying UV mapping [4][12]. Users can avoid traditional modeling and painting altogether by taking pictures of an existing object and constructing an image-based 3D model [3] from them. Animators can make a model move and turn without manipulating 4x4 homogeneous matrices [1][2][18]. These tools empower novices and make (expensive) experts more productive.

Unfortunately, the crucial step of defining how models respond to user actions remains a weak link in the process of building an interactive 3D world. Interaction designers currently assign responses to groups of a model's polygons: when a particular event happens to a polygon in a particular group, the model responds in a particular way. The problem with this method is that it ties a model's interaction semantics to its geometric structure: an interaction designer cannot assign a response to a part of the model that does not exactly correspond to a pre-defined group of polygons.

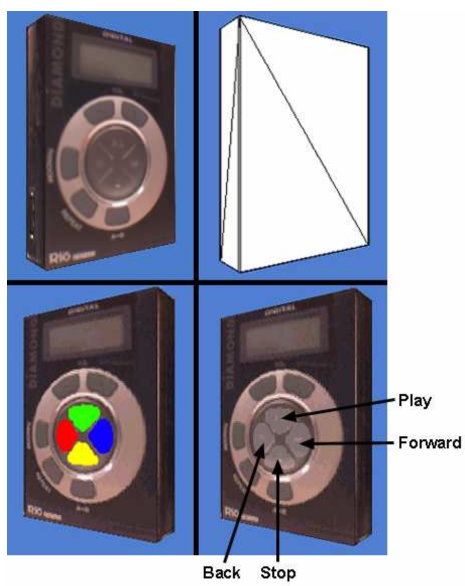

Figure 1: A 3D image-based model of an MP3 player with (top left) and without (top right) its original texture. The designer paints interaction surfaces for the play, forward, back, and stop buttons (lower left), and creates a new texture to provide feedback to the end user (lower right).

To illustrate this problem, consider constructing a 3D e-commerce application using the MP3 player model in Figure 1. We would like to specify that the model plays a song when a user left-clicks on the "play" button, but the model's polygons are unstructured because we created it using an image-based method. We cannot simply group a subset of the model's polygons because they do not provide enough resolution. We must first restructure the existing polygons to create a set that exactly matches the shape and position of the button, update the model's UV mapping so that the texture still projects correctly, and then explicitly group the polygons.

This problem is not restricted to image-based models. The current method of allowing designers to only assign responses to groups of polygons assumes both that a modeler explicitly creates and groups the polygons and that he can foresee what parts of the model the designer will want users to interact with. The first assumption fails when a tool, rather than a modeler, creates and groups the polygons (e.g. [3][4][12]). The second assumption fails when an artist paints details into the model's texture that are not reflected in its polygonal structure.

An ideal solution to this problem would allow interaction designers to easily and flexibly specify what parts of the model respond to events; we call these parts *interaction surfaces*. The designer has the best grasp on how he wants users to interact with the model, and an easy-to-use, flexible tool would allow him to rapidly prototype interactive behaviors. Modifying the model's polygonal mesh is not a viable solution because it is labor intensive and inflexible.

Our solution is to allow a designer to specify an interaction surface by painting it onto the 3D model. The designer can specify multiple interaction surfaces by painting each a different color. We capture the painted areas by projecting them onto a 2D *interaction map*. An interaction map is similar to a traditional texture, but its painted regions specify interaction surfaces instead of appearance: when the designer finishes painting we create an interaction surface for each distinct color in the interaction map. Designers can name these surfaces and assign them responses to different events. To create the interactive behavior in our MP3 player example, a designer would paint over the "play" button, name the resulting interaction surface PlayButton, and then specify that PlayButton responds to left-click events by playing a song.

We believe that interaction surfaces will be particularly useful for rapidly prototyping e-commerce applications. For existing products the designer can create an image-based model and paint in the desired interaction surfaces. For products under development, the designer can quickly prototype the interaction for different control layouts when all the detail is in the texture.

Figure 2: End user's view (left halves) and interaction designer's view (right halves) of a phone and a PDA.

2 Related Work

Allowing users to interact with sections of an object is a well established idea. Image maps [10] allow web pages to load different URLs when users click on different sections of a 2D image.

While image maps only support circular or polygonal "active areas", QuickTime VR [14] allows designers to paint irregularly-shaped active areas onto a 2D panoramic image. In addition to extending the idea of active areas from 2D to 3D, we allow non-contiguous active areas, and we allow active areas to pass parameters to their responses. We also allow designers to change the size and shape of active areas, as well as their associated responses, at run-time.

Our technique also draws on the idea of using false color as a per polygon or object ID (e.g. [17]). We use textures (interaction maps) to extend this idea to allow more than one ID (interaction surface) per polygon. We associate responses with these texture-based IDs to govern how sections of a 3D model respond to events.

We discussed our technique with game developers, who often use textures in novel (and unpublished) ways, but found that they usually constrain the development process so that modelers know all interaction surfaces in advance. Researchers and game developers have used textures for more than affecting the visual appearance of a 3D model [6][8], however, for example to associate "off limits" areas, paths for computer controlled characters, and the type and density of flora [5][13] with 3D terrain.

3 Interaction Surfaces and Maps

We now discuss interaction surfaces and interaction maps in more detail. To help ground our discussion we will use as an example a designer who wants the dog model in Figure 3 to respond when the end user clicks on top of the dog's head, under the dog's chin, or on the dog's tag.

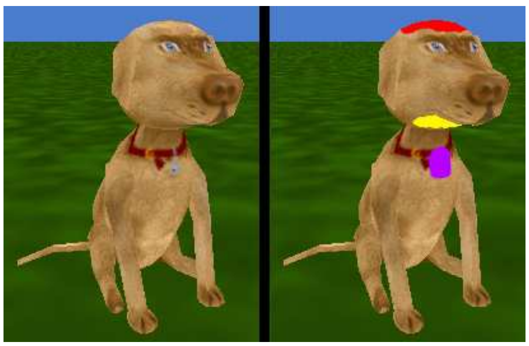

Figure 3: The end user's (left) and designer's (right) views. The designer has painted interaction surfaces on top of the dog's head, under the dog's chin, and on the dog's tag.

3.1 Creating an Interaction Map

A designer specifies interaction surfaces by painting an interaction map. The simplest method of creating an interaction map is to load the model's texture image into a 2D painting program that provides layers (we assume that the interaction map's resolution will not exceed the texture's resolution). The designer displays the texture image in one layer while painting over it in another. When done the designer saves the painted layer separately as the interaction map. Although this method is simple and allows the designer to use commercial tools, designers may have difficulty determining which section of the texture image maps to the desired part of the 3D object. For example, in the texture map on the far left of Figure 4, the dog's head is fairly easy to locate, but his body and legs are not.

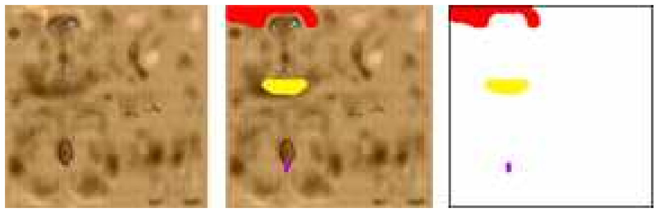

Figure 4: With a 2D paint program the designer paints over the object's texture to create its interaction map. The images show before, during, and after from left to right.

We can instead allow designers to paint the interaction map directly on the 3D model [4][7][12]. The system displays the 3D model with its texture applied so that the texture details are visible. When the designer then paints on the model the system modifies the model's interaction map instead of its texture.

Creating hand-painted textures is generally a difficult, time consuming process. Creating a hand painted interaction map is a much simpler process because we avoid the hard parts of creating a texture: specifying a useful UV mapping (which requires time and experience) and painting in details (which requires artistic talent). The interaction map can use the texture's UV mapping, and the designer traces over details instead of painting them. As a result anyone, regardless of artistic ability, can paint an interaction map.

There are two situations where the designer will not be able to reuse the texture's UV mapping. First, the model may be untextured and lack UV coordinates. Second, sections of a 3D object that are identical (e.g. eyes) or mirror images of each other (e.g. left and right arms) often share the same section of texture. This prevents the designer from creating a different interaction surface for each section. In both situations the designer can create a new UV mapping specifically for the interaction map. The designer can still avoid explicitly specifying the UV mapping by using a tool (e.g. [12]) that dynamically manages the UV mapping while he paints. Automatically generated UV mappings are often non-optimal for complex hand-painted textures, but are sufficient for our purposes.

3.2 Assigning Responses to Surfaces

After painting the dog's interaction map, the designer in our example might want to specify that the dog should wag its tail when an end user clicks on the interaction surface on top of its head. To assign a response to a surface, the designer needs to be able to identify that surface to the system (particularly if he wants to programmatically change a surface's responses at run time). We allow designers to name the interaction surfaces and assign responses to them using their names. We could have used an interaction surface's color to identify it, but we found informally that designers have an easier time working with a distinguished surface by name rather than by color. The designer in our example decides to name the interaction surface on the top of the dog's head

TopOfHead and assigns a WagTail response to it.

We allow designers to assign multiple responses (to the same type of event or to different types) to the same surface. For example, the designer could specify that the dog both wags its tail and spins around when the user left-clicks on TopOfHead, but falls over when another 3D object in the scene collides with TopOfHead.

We also allow interaction surfaces to inherit and provide default responses to and from each other, as well as to and from the model's groups of polygons. This allows us to use our technique in combination with the traditional method of assigning responses to groups of polygons. For example, if the dog's Head is a pre-defined group of polygons the designer might decide that he wants the dog's Chin interaction surface to use the Head's responses to events until he explicitly assigns the Chin a response. We provide this functionality by allowing designers to insert a model's named interaction surfaces into its *response tree*.

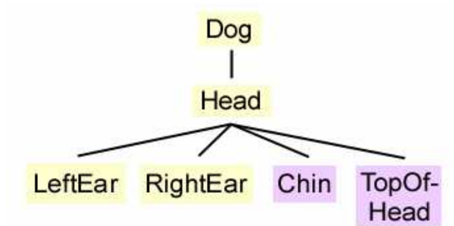

Figure 5: A section of the dog's response tree showing some of the dog's polygon groups (yellow) and interaction surfaces (purple).

A model's response tree arranges its parts into a tree structure that is traditionally identical to the model's geometric structure. A part in the tree inherits responses from its parent, and provides default responses to its children. A part uses its inherited response to a particular event only when the designer has not explicitly assigned it a response to that event. By allowing a model's response tree to contain both groups of polygons and interaction surfaces (e.g. Figure 5) we provide designers with more flexibility in defining how models respond to user actions.

3.3 Implementation

Once the designer has finished painting the interaction map, we create a new interaction surface for each unique color hue in the map. We chose to create interaction surfaces based on hue, rather than by contiguous painted area, because designers can then make geographically distinct sections of the model belong to the same interaction surface, even if those sections are spread across multiple polygon groups. We use hue (and an HLS color encoding) because this allows us to store additional information in a pixel's color saturation and lightness (as we explain in section 3.4).

We could represent interaction surfaces implicitly by just creating a table that maps colors directly to responses. Instead, we represent interaction surfaces explicitly as named instances. When we encounter a new hue in the interaction map we create a new instance and prompt the designer for its name. Each instance stores a dictionary that maps events to a list of responses, and its parent and children in the model's response tree. Representing interaction surfaces as instances rather than as a simple lookup table makes them more flexible: we can insert them into a response tree, we can assign responses to different types of events, and we can assign multiple responses to a single event type. In addition, we can link colors from different interaction maps to the same surface. For example, if there are multiple dogs in a 3D scene we could use the TopOfHead surface instance for all of them to allow us to modify all of their responses simultaneously.

In our implementation we preserve the interaction map and use it to route events to the correct part or surface at run-time. Our implementation allows us to make a 3D model respond to any event (e.g. mouse motion, mouse button clicks, raycasting in immersive worlds, collisions) that yields the polygon and offset within the polygon where the event occurred. We will use left-clicking with the mouse as a sample event to explain our implementation.

When the user positions the mouse cursor over a model and left-clicks, our system performs a pick into the scene to determine the polygon group, the polygon, and the offset within the polygon that the user clicked on. The system retrieves the UV coordinates for the vertices of that polygon, and uses barycentric coordinates [9] to calculate the UV coordinates for the offset position. The system then looks up the color at that position in the interaction map. If the color is white, there is no interaction surface at that point and the system lets the polygon group handle the action. Otherwise the system maps the color's hue to the correct interaction surface and invokes that surface's responses to the event.

The designer can display the model with its original texture to force users to explore the model to locate its interaction surfaces, but if he wants to provide visual feedback about their location he has several options. He can create a new composite texture that blends the model's interaction map with its original texture, for example by overlaying or brightening the parts of the texture that coincide with interaction surfaces, as in the lower right section of Figure 1. A designer could instead compose the interaction map and original texture dynamically using multi-texturing, but this introduces an additional run-time cost. Designers can also provide feedback by making the 3D model respond when the user moves their mouse over an interaction surface. Finally, designers can borrow an idea from 2D applications and change the mouse cursor's shape when it is over an interaction surface.

3.4 Passing Parameters to Surfaces

Because we use only a color's hue in an interaction map to uniquely identify an interaction surface, we can use the color's saturation and lightness to encode a parameter that we pass to a surface's responses when the user interacts with it. We can use this parameter to indicate the strength of the model's response when the user left-clicks on it, the elasticity (hardness/squishiness) of the model when something collides with it, etc. In our system we use the color's saturation to indicate the former. Consider the dog's Chin interaction surface from section 3.2. If this surface indicates a ticklish area, we could change how ticklish the dog is at different points throughout the surface by varying the saturation from 0 (white, no response) to 1 (yellow, strong response). The strength of the dog's response (e.g. how much he wiggles around) when the end user clicks on his chin would then depend on the saturation at the clicked point in the interaction map. The designer can thus specify a fairly complex parameterization in a visually intuitive way.

3.5 Dynamically Modifying Surfaces

With our technique there are multiple ways to dynamically modify the size, shape, location, or response strength of a model's interaction surfaces. The simplest method is to completely switch from one interaction map to another. Consider a character in a 3D fighting game. The character might have two different textures, one showing the character wearing armor and one without (see Figure 6). Because the character's responses will depend on where she is hit and whether she is wearing armor, the interaction designer could create two interaction maps, one for each texture. Note that we can link the red areas from each interaction map to the same interaction surface (VulnerableSpot) and define the responses for the red areas in both interaction map areas simultaneously rather than separately.

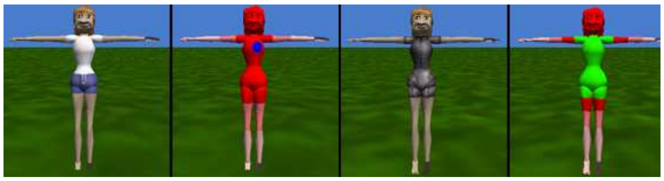

Figure 6: A female character with two different textures and interaction maps. The designer has varied the saturation in the red interaction surface to indicate the character's vulnerability at different points in the surface.

The system can also directly modify the pixels in the interaction map at run-time. For example, in our 3D fighting game the system could write into a character's interaction map when that character is hit to create sore spots. If the character is subsequently hit in a sore spot she could respond differently than if hit in an untouched area. As another example, the system could slowly increase the saturation of the color in a walking character's interaction map that corresponds to the soles of the character's feet. The character's response to contacts with her feet could then depend on how tired and sore her feet are.

4 Limitations and Future Work

Our implementation imposes an additional memory cost because we retain the interaction maps after creating the interaction surfaces. We can lower this cost by using a lower resolution image for the interaction map. For larger savings, we could replace the pixel colors with identifiers for the corresponding interaction surfaces and store the resulting map internally in a more efficient structure, such as a quad tree [15]. The drawback to this approach is that determining which surface an event happens to is more complex, and the system cannot easily display the resulting structure. If the number of interaction surfaces for a given model is sufficiently small, we could also steal the low two or three texture bits and avoid maintaining a separate structure altogether (although this would preclude some of the advanced uses of interaction maps, such as passing parameters to response functions).

Another limitation of our implementation is that designers cannot create overlapping interaction surfaces. We could overcome this limitation by allowing multi-layer interaction maps. Interaction surfaces in different layers could overlap and respond to events independently.

Our technique also only works with geometry-based rendering engines. However, we believe that extending it to work with image-based rendering engines is straightforward. We could modify plenoptic image editing [16] to allow designers to paint interaction map images. We could then determine what interaction surface an end user clicks on using a modified object buffer approach: calculate the (x,y) screen position of the mouse cursor, render the world using the interaction map images into an off screen buffer, and determine the color of the pixel in the buffer at the corresponding (x,y) position.

References

[1] Matthew Conway, et al. Alice: Lessons Learned from Building a 3D System for Novices. CHI 2000, pages 486-493.

[2] Cult 3D. Cycore, www.cult3D.com.

[3] Paul E. Debevec, Camillo J. Taylor and Jitendra Malik. Modeling and Rendering Architecture from Photographs. SIGGRAPH '96, pages 11-20.

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

[5] Oliver Deussen, et al. Realistic Modeling and Rendering of Plant Ecosystems. SIGGRAPH '98, pages 275-286.

[6] Paul Haeberli and Mark Segal. Texture Mapping as a Fundamental Drawing Primitive. Fourth Eurographics Workshop on Rendering, 1993, pages 259-266.

[7] Pat Hanrahan and Paul Haeberli. Direct WYSIWYG Painting and Texturing on 3D Shapes. SIGGRAPH '90, pages 215-223.

[8] Paul Heckbert. Survey of Texture Mapping. IEEE Computer Graphics and Applications, vol. 6 no. 11, 1986, pages 56-57.

[9] John G. Hocking and Gail S. Young. Topology. Addison-Wesley, Reading, MA, 1961.

[10] HTML 4.01 specification, www.w3.org/TR/html4/.

[11] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A Sketching Interface for 3D Freeform Design. SIGGRAPH'99, pages 409-416.

[12] Takeo Igarashi and Dennis Cosgrove. Adaptive Unwrapping for Interactive Texture Painting. 2001 ACM Symposium on Interactive 3D Graphics (to appear).

[13] Kai Martin. Using Bitmaps for Automatic Generation of Large-Scale Terrain Models. Gamasutra, April 27, 2000, www.gamasutra.com/features/20000427/martin_pfv.htm.

[14] QuickTime VR. Apple Computer, www.apple.com/quicktime/qtvr/.

[15] Hanen Samet. The Design and Analysis of Spatial Data Structures, Addison Wesley, 1989.

[16] Steven M. Seitz and Kiriakos N. Kutulakos. Plenoptic Image Editing. International Conference on Computer Vision '98, pages 17-24.

[17]Hank Weghorst, Gary Greenberg, and Don Greenberg. Improved Computational Methods for Ray Tracing. ACM Transactions on Graphics, vol. 3 no. 1, 1984, pages 52 - 69.

[18]WorldUp. Engineering Animation, Inc., www.sense8.com.

[19]Robert C. Zeleznik, Kenneth P. Herndon and John F. Hughes. SKETCH: an interface for sketching 3D scenes. SIGGRAPH ‘95, pages 163-170.