Simulated 3D Painting

Kok-Lim Low

Department of Computer Science
University of North Carolina at Chapel Hill

ABSTRACT

This paper looks at the motivation for simulating painting directly on 3D objects, and investigates the main issues faced by such systems. These issues include the provision of natural user interfaces, the reproduction of realistic brush effects and the surface parameterization for texture mapping so that the results of the painting can be stored on texture maps. The paper further investigates the issues involved in using a haptic interface for simulating 3D painting, and the issues in surface parameterization for texture mapping with application to 3D painting. A survey of some work related to 3D painting, haptic rendering and surface parameterization for texture mapping is presented.

1 INTRODUCTION

1.1 Texture Mapping

In the early days of computer graphics, the quest for realistic imagery has proven polygons and other geometric primitives inadequate in representing the required surface complexity. As detail becomes finer and more intricate, explicit modeling with polygons or other geometric primitives becomes impractical. An elegant solution to this was introduced by Catmull [Catmull74], in which the surface details are represented in images, and the images are then mapped onto the surfaces. This approach is known as texture mapping; each image is called a texture map, and its individual elements are called texels. The rectangular texture map resides in its own texture coordinate space.

Texture mapping has become a successful technique for adding realism to computer-generated imagery. It has become such a common technique that even today's low-end personal computers have hardware-accelerated texture mapping capabilities. Besides surface color, texture maps with other surface properties, such as transparency, normal vector perturbation (bump maps) and specular reflection (environment maps), have been used. Heckbert gives a comprehensive explanation of texture mapping in [Heckbert86], and Haeberli and Segal collected many creative uses of texture mapping in [Haeberli93].

1.2 Why Paint in 3D

Texture maps are typically created by scanning in existing artwork, generated using procedural image models, or painted with a 2D painting program. Before a texture map can be "pasted" on the 3D surfaces, each 3D surface must first be parameterized to establish a mapping from each point on the surface to a point in the texture coordinate space. When the creation of the texture maps and the surface parameterization are separated (which is usually the case), it becomes very hard to get every part of a surface mapped with the desired portion of the texture map. Even with the knowledge of the mapping function, it is still not trivial to accurately create the correct texture maps. Moreover, the user should not be required to know the mathematics of the surface parameterization when creating the texture maps.

A solution to this problem is to allow the user to paint directly on the surface of the 3D model. In this case, the surface parameterization will be used to map the result of the painting on the 3D surface onto the corresponding texture map, which can then be permanently stored. The images may look distorted on the texture maps, but they will appear correct (or as desired) on the 3D surfaces. However, this solution poses many challenges, and a number of issues need to be addressed:

1. The user interface must allow the user to easily paint on the 3D surfaces in a natural style.
2. The nonplanarity of the 3D surfaces and the greater freedom of movement of the brushes enable very interesting brush effects to be simulated. Efficient methods must be used to create these effects and to correctly map the result on the 3D surface onto the texture map.
3. The quality of the surface parameterization of the 3D model for texture mapping should be sufficiently high to allow efficient representation and storage of the painting results. Surface parameterization for texture mapping is not a problem of 3D painting only, but a problem of all texture mapping onto non-trivial surfaces. For the purpose of 3D painting, some additional criteria can be used to judge the quality of parameterization.

These issues are discussed further in the next section.

2 MAIN ISSUES

2.1 User Interface

User interface includes the input and output hardware devices, and the software that is responsible for getting the input data from the input devices, translating them into actions and providing appropriate feedback via the output devices. The main factor that determines the "intuitiveness" of a user interface is the design of the associations that relate the specific physical actions on the input devices to the actual actions to be performed on the computer, and also relate the physical feedback on the output devices to the results of the performed actions.

A general user interface design principle is to use common metaphors. Users can then rely on their everyday experience to infer how a program works by analogy with how everyday things work. Another principle in designing user interfaces is direct manipulation. For example, a mouse can be used to drag graphical representations on the screen. The act of moving the mouse is directly associated with some action to be performed, and feedback is given immediately to reinforce the action. Direct manipulation is able to provide the user a more realistic experience, but it usually places high demand on the capabilities of the hardware devices and the software that controls them.

Taking a 3D painting program as an example: ideally, using the input device to "paint" on a virtual model should be no different from using a real paintbrush (or pencil and other artistic tools) to paint on a real 3D object.

Painting on real 3D objects often involves a high degree of spatial freedom in the movement of the paintbrush and also in the placement of the object. Though 2D input devices have been used for simulated 3D painting, the limited degree of spatial freedom made the user interface awkward and unintuitive [Hanrahan90]. More recently, with the advent of 3D input devices, such as the Spaceball and many trackers, a new dimension in 3D user interfaces was opened up. However, it was the introduction of haptic (force feedback) devices that really created a revolution in 3D user interfaces.

These haptic devices are capable of providing force feedback to the user to recreate the sense of touch, and some are also high degree-of-freedom 3D input devices. Examples of such haptic devices are the PHANToM of SensAble Technologies (Figure 1) and the SARCOS Dextrous Arm. Many of these devices can be fitted with a pen-like probe, which can be held like a paintbrush when used in simulated 3D painting. This creates many possibilities that could significantly narrow the gap between real 3D painting and simulated 3D painting. Not only can the user apply his brush strokes differently each time, he can also "touch" the surface of the virtual object with the brush. Coupled with visual display, this can undoubtedly provide a very natural user interface for 3D painting.

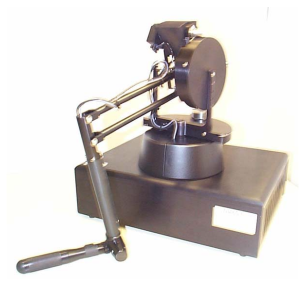
Figure 1: A PHANToM haptic device by SensAble Technologies, Inc. (photograph from SensAble Technologies, Inc.).

However, creating a reasonably realistic sense of touch on the haptic device requires a very high haptic update rate, usually at over 1000 Hz. This high rate is necessary to maintain system stability and to provide a sense of stiff virtual surfaces. Contact determination between the virtual scene and the virtual object controlled by the user is necessary for computing the feedback force. Such high haptic update rate implies that the contact determination must be done very efficiently and accurately in order to generate realistic forces.

Very fast contact determination for real-time haptic interaction has always been a challenge for large complex models. Recently, with very efficient algorithms and more powerful computers, it has become possible to achieve sustained kHz update rates in 6 degree-of-freedom haptic rendering [McNeely99]. Besides contact information, a good and efficient force model is required to produce realistic forces on the haptic device. I have devoted Section 3 and 4 to a further discussion of the issues of haptic rendering.

2.2 Brush Effects and Texture Map Update

Just like in 2D paint programs, many brush effects can be implemented in 3D paint programs. Because of the nonplanarity of the 3D surfaces and the many possible orientations of the brush, many effects can be made more interesting and more realistic. For example, spray-painting on a surface partially occluded by another surface can create disjoint painted regions, which cannot happen in 2D painting. If we use a 3D shape other than a sphere as the brush volume, the brush effect will change with the orientation of the brush. It is also possible to have a brush that is deformable under varying pressure.

For every brush stroke, we need to find out which parts of the 3D surfaces have been affected. Contact determination is needed to compute the intersections between the brush volume and the surfaces. Using the surface parameterization for texture mapping, we can then map the intersection regions onto the texture map and update the affected texels. Several compositing filters can be applied to combine the current paint with the paint previously laid down on the surface.

2.3 Surface Parameterization

Because the result of the painting is recorded on a texture map (or more than one texture map), the surface parameterization must assign every 3D surface primitive a unique region on the texture map, i.e. no two regions overlap.

With 3D painting, we have successfully avoided the distortion of the image features of the texture map when the texture map is mapped onto the 3D surfaces, even though the texture map is still being distorted by the surface parameterization. However, it is still desirable to have a surface parameterization that produces little or no distortion of the texture map, because less distortion implies a more uniform resolution of the texture map on the 3D surfaces. Moreover, with the knowledge that there is little or no distortion, assumptions can be made to simplify and speedup the update of the texture map when paint is applied [Johnson99].

For 3D painting, it is not necessary to have adjacent 3D surface primitives assigned to adjacent regions on the texture map, but there are benefits in keeping those regions adjacent to one another. Firstly, adjacent texture map regions corresponding to adjacent surface primitives can simplify and speedup the update of the texture map. Secondly, texture mapping involves filtering the texture map, and it cannot be done correctly near the boundaries of the surface primitives if adjacent primitives do not map to adjacent texture map regions.

Section 1 of this paper presented the motivation for simulating 3D painting, and introduced the main issues to be addressed by such systems. We further elaborated these main issues in Section 2. Next, we present a survey of some previous work on simulated 3D painting in Section 3. Section 4 discusses the issues of haptic rendering and Section 5 surveys some previous work in this area. In Section 6, we study the issues related to surface parameterization for texture mapping, and look at the criteria for good parameterization with application to 3D painting. Similarly, a brief survey in surface parameterization for texture mapping is presented in Section 7. Section 8 concludes this paper.

3 PREVIOUS WORK ON 3D PAINTING

Though real-time haptic rendering and surface parameterization for texture mapping have been actively researched, there is not much work on 3D painting. The ultimate goal of simulated 3D painting is to provide a realistic experience to the user, both in the user interface and the simulation of realistic paint effects. Below, we review some of the previous work in chronological order, and observe the advancement in the area.

3.1 3D Painting Using 2D Input Devices

Hanrahan and Haeberli [Hanrahan90] developed an interactive paint program that allows the user to directly paint with different types of pigments and materials onto 3D shapes. This is the pioneering work of simulated 3D painting, and its motivation was to overcome the distortion of texture images caused by the surface parameterization.

Although the user can paint directly onto 3D object surfaces, the brush is controlled by a 2D input device, such as a mouse or a tablet, by moving a 2D cursor on the screen. With this, it is impossible for the user to freely position the brush in his intended orientation, so it is necessary for the program to assume the brush is in one of the few predefined orientations. Moreover, the user can only paint on the surfaces that are visible on the screen, and this often abruptly discontinues a brush stroke when the brush reaches the silhouette of the displayed object.

In the paint program, the geometric objects are represented as rectangular meshes. The results of the painting are first internally recorded as vertex attributes, and these attributes are then stored in files as a set of associated texture maps. In this case, each surface parameterization simply maps consecutive rows and columns of mesh vertices to consecutive rows and columns of texels on the texture map. Representing paint information at the mesh vertices has the disadvantage that added paint detail requires an increase in mesh complexity.

When painting, the object remains stationary, and typically many paint strokes are laid down. The whole object needs to be redrawn only when there is a change of view or the object position, and between full redraws, only the polygons affected by the brush need to be redrawn. This method ensures that the rendering can be done in real-time when painting on complex models.

The brush position on the object surfaces is computed using an item buffer, taking advantage of the graphics hardware. Given the position of the cursor on the screen and an item buffer, the brush position can be found by reading the pixel in the item buffer under the cursor. This pixel contains the object identifier of the polygon that the brush is on. Since the object stays stationary for many paint strokes, the same item buffer can be reused many times between redraws.

Having determined the position of the brush, the brush is assigned an orientation with respect to the object surface. The orientation determines how the brush geometry affects the surface. Three types of brushes are considered: (1) parameter-space brushes, (2) screen-space brushes, and (3) tangent-space brushes. With a parameter-space brush, the 2D brush paints directly into the texture maps at the texture coordinates of the brush position on the surface. A screen-space brush has an orientation that is perpendicular to the view plane, and it is capable of producing spray-paint effect on the surface. A tangent-space brush is always oriented perpendicular to the surface at the brush position, and the brush is projected onto the surface in the direction parallel to the surface normal.

3.2 Painting on Scanned Surfaces

In the quest for a more realistic 3D painting experience, Agrawala, Beers and Levoy have extended the above approach to 3D input devices [Agrawala95]. In their work, they used a 6D Polhemus space tracker to track the 3D position and orientation of a stylus. Their system was intended for scanned models. When painting on a scanned model, the registered physical model provides an object to paint against with the stylus. This provides a natural force feedback to the user, without the use of any haptic device.

One problem with this user interface is that while the user is moving the sensor along the physical object, the paint is only applied to the mesh on the monitor. Thus, the user must look at two places at once to see where the paint is being applied.

Since the scanned model is usually a result from multiple scans, the resulting mesh is irregular and difficult to parameterize for texture mapping. Because of this, the painting results are stored at the mesh vertices instead of on texture maps. Thus, this approach shares the same disadvantage as the approach by Hanrahan and Haeberli. To avoid sampling artifacts when displaying the mesh, polygons should be small, and that leads to a large number of polygons in the model.

To speed up the contact determination between the virtual brush and the large model, the model is spatially subdivided into uniform voxels, and each voxel is associated a list of vertices. A hash table is used to index the voxels, and this avoids wasting space storing empty voxels. The tip of the virtual paintbrush is a 3D shape that defines the volume within which paint is applied. During painting, the position and orientation of the brush determine how the brush volume is positioned in the 3D space. The voxels intersected by the brush volume are checked to see which vertices lie within the brush volume. These enclosed vertices are then assigned the selected paint color.

A disadvantage of this 3D painting approach is that only meshes with corresponding physical objects can be painted. Furthermore, the use of a physical object requires it to be accurately registered with the virtual model. The registration process usually takes several minutes and must be done every time the user wants to paint an object. However, registration errors cannot be eliminated and can cause the virtual brush tip to lie some distance away from the mesh even when the stylus is physically touching the object surface.

3.3 Painting NURBS Models with a Haptic Interface

In the areas of CAD modeling and animation, NURBS have become the de facto model representation standard. In their work, Johnson et al. [Johnson99] developed a system to allow the user to paint texture maps directly onto trimmed NURBS models using a 3 degree-of-freedom haptic interface. Using the sense of contact provided by the force-feedback device, the user can draw on portions of the model that are not directly visible.

The contact determination necessary for the haptic rendering is done by directly tracing the surface of the model using a method described in [Thompson97]. This tracing algorithm works only for NURBS. It keeps track of the point on the surface closest to the tip of the probe, and computes the penetration depth of the probe. If the penetration depth is positive, a restoring force proportional to it is then generated on the haptic device to resist penetration into the virtual model.

In their system, each surface parameterization is done by simply applying uniform discretization of the parameter space of each NURBS. Though the method is simple, the texture map can still be highly distorted at regions of high curvature. When painting, the perimeter of the brush in texture space is computed using the surface parameterization, and the enclosed region on the texture map is filled with the brush color using a flood-fill method.

3.4 Painting Polygonal Models with a Haptic Interface

Gregory et al. have developed a system for interactive multiresolution modeling and 3D painting with a 3 degree-of-freedom haptic interface [Gregory2000]. The models are represented as subdivision polygonal meshes and texture maps are used to store the results of the painting. The contact determination for haptic rendering is done by their H-Collide method [Gregory99]. The method uses a hybrid representation of the model, utilizing both spatial partitioning and bounding volume hierarchy. In addition, to achieve the required runtime, the method needs to exploit frame-to-frame coherence.

In [Gregory2000], the authors did not mention how surface parameterization is done. When paint is laid down on the surface, the texture map is updated by doing a standard 2D scan-conversion in the texture space. Each affected 3D triangle is mapped to a 2D triangle in texture space using the surface parameterization. Then, for each texel in the 2D triangle, its corresponding 3D location is computed. This 3D location is used to decide how the texel is affected by the brush function.

4 HAPTIC RENDERING

There are many types of haptic devices, for example, haptic gloves (e.g. Cybergrasp by Virtual Technologies, Inc.) for virtual reality, haptic joysticks and haptic steering wheels for computer games and vehicle simulations. Here, we only consider devices that are similar to the PHAMToM arms of SensAble Technologies and the SARCOS Dextrous Arms. These devices are essentially small robot arms that are used in reverse. Many of these devices can be fitted with a pen-like probe, which can be held like a paintbrush when used in simulated 3D painting. These haptic robot arms also serve as 3D input devices.

During haptic rendering, the position and orientation of the probe are continually sent to a computer. The computer then simulates the contacts between the probe and a virtual environment to compute the appropriate reaction force. Controls are then sent to the haptic device to display the resultant force. These three actions constitute a typical haptic update cycle. In contrast to graphics display update rate, which usually ranges from 20 to 60 frames per second, the haptic update rate must be as high as 1000 updates per second in order to achieve system stability. For example, if the haptic update rate is too low and the user moves too fast, the virtual probe may be “trapped” in the interior of a virtual object. In even worse cases, unstable systems can cause device damage or user injury. Moreover, high haptic update rates are required to produce a sense of stiff surfaces. Also, our skin is sensitive to vibrations of greater than 500 Hz, so changes in force at even relatively high frequencies are perceptible.

There are two main issues concerning realistic haptic display. Firstly, the requirement for very high haptic update rates implies that very efficient contact determination algorithms must be used. Secondly, the force model used to compute the reaction forces must be capable of realistically modeling the different sense of touch in the virtual environment.

Currently, most haptic rendering methods can only produce force feedback in 3 degrees of freedom (3 DOF), i.e. translational force. In these 3-DOF systems, the tip of the probe usually corresponds to a point in the virtual environment. Each haptic update cycle then involves (1) reading in the 3D position of the probe’s tip, (2) computing the contact point between the virtual objects and the path swept out by the virtual point since the last update cycle, (3) computing the translational reaction force according to the force model, and (4) sending controls to the haptic device to actually produce the physical force. If the haptic feedback is coupled with visual display, then the position of the virtual point has to be sampled and sent to the graphics system for display update.

In comparison to 3-DOF haptic rendering, 6-DOF haptic rendering is considerably more difficult. Instead of just a virtual point, the probe can be used to manipulate, by both translations and rotations, a 3D virtual object with arbitrary shape. The resulting force feedback is no longer just translational, but rotational (torque) also. Contact determination must now identify multiple contact points between the manipulated object and the other object in the virtual environment, and a more complex force model is necessary to generate both translational forces and torques.

Ultimately, for realistic simulation of 3D painting, the haptic system must handle deformable 3D virtual objects, in order to simulate the reaction force caused by the change in brush shape under varying pressure and brush motion.

In the next section, we survey three papers on haptic rendering, and see how they address the main issues.

5 PREVIOUS WORK ON HAPTIC RENDERING

5.1 Simple 3-DOF Haptic Rendering

Using a hybrid hierarchical representation for fast and accurate collision detection, Gregory et al. have developed a simple 3-DOF haptic system for polygonal models [Gregory99]. Given a polygonal model, its hybrid hierarchical representation is computed by first spatially decomposing it into uniform grids or cells. Polygons within each cell are then bounded by a tight-fitting oriented bounding box tree (OBB-tree) [Gottschalk96].

Spatial decomposition and hierarchical bounding volumes, such as OBB-trees, are very commonly used to speed up collision detection. Spatial decomposition allows direct constant-time access to each individual cell. Because a cell may contain many polygons, potentially all polygons in an intersected cell must be tested for actual intersections during collision detection. We can use smaller cells, but that can increase the space requirement dramatically. As an alternative data structure, the hierarchical bounding volumes can offer logarithmic-time search down to each individual polygon during collision detection. However, sometimes a bounding-volume tree can be badly skewed, and have leaves at great depths. By combining spatial decomposition and hierarchical bounding volumes, Gregory et al. hope to reduce these undesirable long traversals.

In their 3-DOF haptic system, the tip of the probe corresponds to a point in the virtual environment. The contact point between the tip of the probe and the virtual environment is computed as the intersection between a polygon and the straight-line path swept out by the tip of the probe between two successive time steps. For this, a specialized overlap test between a line segment and an OBB (oriented bounding box) is used. Once the surface contact point is found, a force proportional to the distance between the current probe position and the contact point is generated. It has a direction from the current probe position to the contact point.

Typically, if the haptic update rate is always maintained above 1 kHz, there is little movement in the probe position between successive time steps. Their collision detection algorithm utilizes this frame-to-frame coherence by caching the contact information for use in the next time step.

Their system is able to achieve sustained kHz haptic update rates because very simple object interaction and haptic models are used. More complex haptic simulation requires more complex object interaction and haptic models. However, complex interaction/haptic models are more computationally expensive and make achieving the desired update rates harder. Thererfore, additional measures should be used to reduce instability in the system and improve its robustness when the update rate is not high enough. Moreover, when the tip of the probe corresponds to a zero-sized point, it can fall through gaps between polygons caused by numerical errors.

5.2 Towards Realistic 3-DOF Haptic Rendering

In contrast, Ruspini et al. place their emphasis on realistic and robust haptic rendering more than on collision detection [Ruspini97]. They suggest that a haptic system should be capable of representing the surfaces and objects that are commonly found in graphic environments. These environments are usually composed of many zero-width polygons, lines and points. The haptic system should also make use of the additional graphical information such as surface normals and texture maps. In addition, the haptic controller should be robust and degrade gracefully as the limits of its performance are reached.

In their system, a virtual "proxy" is used to substitute for the physical probe in the virtual environment. Figure 2 illustrates the motion of a virtual proxy as the probe's position is altered. The virtual proxy always attempts to move towards its goal, which is the tip of the probe. When unobstructed, the proxy moves directly towards the goal. When obstructed, direct motion is not possible, but the proxy can move along the surfaces to reduce its distance from the goal, and when the distance cannot be reduced, it stops at the local minimum configuration. With this interaction model, the force is generated on the haptic device by physically moving the goal to the location of the proxy.

The use of the virtual proxy enables their system to be stable, since the proxy can "remember" where the exterior of the object is. Moreover, the use of a finite-size proxy prevents it from slipping through the tiny numerical gaps found in most polygonal meshes. With these benefits, however, the price is the more costly computation to update the proxy position in every time step.

Polygonal models for graphic display usually have surface normals at their vertices. Ruspini et al. utilize these normals to produce "smoother" proxy motion by interpolating the vertex normals of the polygon with which the proxy comes into contact. This is called force shading, and is similar to that done in Phong shading to produce smoother-looking surfaces. Force shading provides more accurate haptic modeling of the object's surfaces. In addition, their system even allows the use of texture maps to modulate any of the surface parameters—friction, viscosity or stiffness—to create different haptic effects.

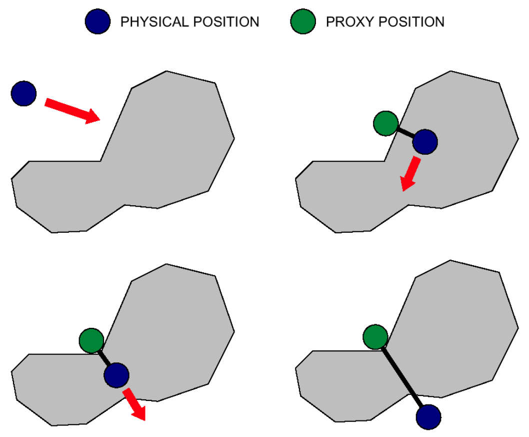

Figure 2: Virtual proxy example (figure from [Ruspini97])

For contact determination between the proxy's path and the surfaces, a simple hierarchical bounding sphere representation is used. Because the bounding-volume tree is constructed in a bottom-up manner, they are able to balance the tree. However, simple bounding spheres are usually very space-inefficient bounding volumes for most polygons, since most polygons cannot be tightly bound. This can cause the proxy's path to frequently intersect many spheres, and result in relatively longer collision detection time.

To further improve stability and robustness, the low-level haptic device control loop is separated from the contact/proxy update loop, so that the haptic device is always controlled at a high fixed rate. As the complexity of the virtual environment increases, the system is able to degrade gracefully instead of becoming unstable and behaving dangerously.

5.3 6-DOF Haptic Rendering

As metioned, 6-DOF haptic rendering is a much more difficult problem than 3-DOF haptic rendering. In 6-DOF haptic rendering, the probe can be used to freely manipulate a 3D virtual object of arbitrary shape. One of the two main challenges is that the manipulated object can simultaneously have multiple contacts with the other objects in the virtual environment. Therefore, an extremely time-efficient contact determination techniques must be used. The second main challenge is the generation of realistic forces and torques on the haptic device.

To achieve the required speed to detect the multiple contact points at each time step, McNeely et al. have sacrificed contact accuracy in their 6-DOF haptic system [McNeely99]. In the system, the virtual environment consists of many static rigid objects, and one dynamic rigid object that can be manipulated by the user using a haptic handle on the haptic device. In off-line preprocessing, the static objects are approximated using uniform-sized voxels, and the dynamic object is approximated using point samples of its surfaces. The set of voxels form a volume occupancy map, or voxmap, and the set of surface point samples, plus associated inward pointing surface normals, form a point shell. Figure 3 shows an example of a point shell colliding with a voxmap.

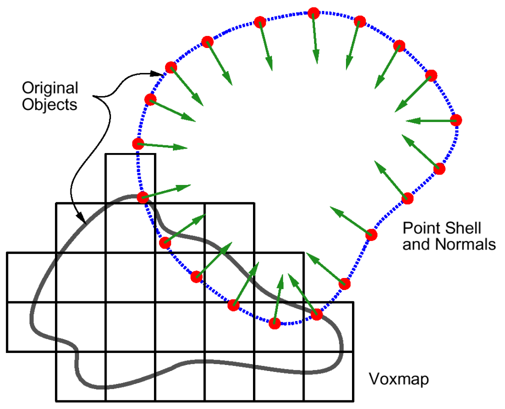

Figure 3: A point shell colliding with a voxmap (figure from [McNeely99]).

They use a tangent-plane force model to compute the reaction force and torque when the point shell comes in contact with some occupied voxels. When a point is inside an occupied voxel, a tangent plane perpendicular to the point's normal is constructed. This plane is made to pass through the center of the voxel. If the point has penetrated below this plane, then the force at this point is proportional to its distance from the tangent plane. This effectively creates a half-voxel-thick force field around the static objects. The net force and torque acting on the dynamic object is obtained as the sum of all force/torque contributions from such point-voxel intersections. We can see that this simple model has discontinuities in force magnitude when a point crosses a voxel boundary.

To improve stability, the occurrence of interpenetration between the exact surfaces of the objects is reduced by offsetting the force field outward away from the surface by two voxel layers. In addition, in the outer proximity voxel layer of the force field, pre-contact braking force is generated to reduce the point's velocity, thereby further reduce the chance of exact-surface interpenetration.

The force discontinuities mentioned earlier are mitigated by a dynamic simulation based on virtual coupling, which connects the user's haptic motions with the motions of the dynamic object through a virtual spring and damper.

With the above, the authors claim that their system is able to produce stable and convincing force feedback.

6 SURFACE PARAMETERIZATION FOR TEXTURE MAPPING

Since the result of the painting is recorded on a texture maps (or more than one texture map), each surface on the 3D object must be assigned a unique area on the texture map. This criterion, which will be called the *unique-region criterion*, alone can be easily satisfied by mapping each surface primitive separately onto an unoccupied location on a large-enough texture map, without considering its connectivity to other primitives. The obvious problem with this approach is that large gaps appear between texture regions of mapped primitives. This waste of texture space results in many texture maps or very large texture map, and can become inefficient for interactive rendering. Optimization algorithms can be used to minimize the waste, however, this problem is inherently NP-hard, and approximation algorithms still cannot produce satisfactory solutions. Moreover, some additional criteria discussed below cannot be satisfied by this separate-mapping approach.

We would like to map adjacent 3D surface primitives to adjacent regions on the texture map. We will refer to this as the *adjacency criterion*. When texture mapping is done during rendering of the 3D object, filtering of the texture map is usually required for antialiasing. Because the filter kernel is normally larger than a texel, pixels near the boundaries of the surface primitive get contributions from texels outside the texture region assigned to the primitive. If the adjacency criterion is not met, the filtered results near the boundaries of the surface primitive can be incorrect. Another advantage of satisfying the adjacency criterion is that painting results can be more easily updated on the texture map. In this case, the texture area to be updated is usually in one contiguous area.

For the third criterion, as mentioned in Section 2.3, we want to have a surface parameterization that minimizes texture map distortion. This problem is still being actively studied by many researchers, but the traditional goal is to map a texture map that already has a non-pre-distorted texture pattern onto 3D surfaces with minimal distortion to the texture pattern. This traditional goal also requires that adjacent regions on the texture map are mapped to adjacent 3D surface primitives.

When an object's surface curvature is high or its topology is non-trivial, finding a satisfactory parameterization becomes very difficult. Sometimes, trade-offs must be made and some criteria may not be fully satisfied. Regardless of that, for 3D painting, the unique-region criterion must always be upheld.

In the following section, we briefly survey two papers on surface parameterization for texture mapping, targeted at the traditional goal.

7 PREVIOUS WORK ON SURFACE PARAMETERIZATION

7.1 Two-Part Texture Mappings

Early attempts to minimize texture map distortions were made by Bier and Sloan [Bier86]. In their method, texture mapping is separated into two steps. In the first step, the texture map is applied to a simple 3D intermediate surface such as a cylinder or a box. This intermediate surface can be any shape that can be made by cutting, folding, and curling paper (i.e. without stretching), and thus has no distortion at all. Texture mapping onto this intermediate surface is simple. In the second step, the intermediate surface is projected onto the target object to derive the final mapping.

The choice of the intermediate surface, its orientation, and the projection method can dramatically affect the results, and a lot of user interaction is therefore required.

Their method is only suitable for objects that are geometrically very similar to the intermediate surfaces. For objects with high surface complexity or non-trivial topology, it is possible that all three criteria are not met.

7.2 Global Minimization of Texture Distortion

More recently, some researchers have used optimization techniques to derive surface parameterizations that minimize texture distortions. In the work by Maillot et al. [Maillot93], a global energy function is minimized. Treating each surface as a deformed elastic plane, the energy function measures the total energy needed to perform the deformation. However, the original energy function is too complex, and the optimization process could be very slow. To obtain a simple expression, Maillot et al. approximate the original energy function by considering only the lengths of the edges of each surface triangle in the 3D object space, and their lengths in the texture space.

This global optimization still cannot handle objects that have high surface complexity, and excessive distortions appear on the mapped texture. Maillot et al. solve this problem by splitting the object surface into several independent regions, and apply the optimization to each of them separately. The splitting of the object is automated, using the curvature information on the surface. Their system also provides an interactive tool for the user to further control the splitting of the object.

Because of the splitting of the object into regions, the adjacency criterion can be considered only partially met. Since most of these regions are relatively large, it is not a very big problem.

8 CONCLUSION

We have presented the motivation for simulating painting directly on 3D objects, and investigated the main issues faced by such systems. For realistic simulation, a haptic interface can be used to provide the user a more natural painting experience. We have looked at some of its issues and surveyed some of the work in haptic rendering. We have also discussed the issues in surface parameterization for texture mapping, and the criteria it needs to meet in order to be useful for 3D painting.

Although non-distorted texture mapping of models with complex shapes still remains an open problem, current surface parameterization methods are sufficient for simulated 3D painting. However, fast and accurate 6 DOF haptic rendering still remains a major challenge.

Another major challenge in simulating 3D painting is the simulation of realistic brush effects. Baxter et al. have implemented an interactive haptic painting system in which the brush heads are deformable, and they produce realistic brush strokes on a flat rectangular virtual canvas [Baxter2001]. Their haptic interface enables 6 DOF of input but only 3 DOF of output. It remains a challenge to extend their method to arbitrary 3D surfaces.

ACKNOWLEDGEMENTS

I would like to thank Anselmo Lastra and Greg Welch for their helpful suggestions.

BIBLIOGRAPHY

[Agrawala95] Maneesh Agrawala, Andrew C. Beers and Marc Levoy. *3D Painting on Scanned Surfaces*. In Proceedings of 1995 ACM Symposium on Interactive 3D Graphics, pp.145–150, April 1995.

[Baxter2001] Bill Baxter, Vincent Scheib, Ming C. Lin and Dinesh Manocha. *DAB: Interactive Haptic Painting with 3D Virtual Brushes*. To appear in Proceedings of ACM SIGGRAPH 2001.

[Bier86] Eric A. Bier and Kenneth R. Sloan, Jr. *Two-Part Texture Mappings*. IEEE Computer Graphics and Applications, pp. 40–53, September 1986.

[Catmull74] Ed Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. Ph.D. thesis, Department of Computer Science, University of Utah, December 1974.

[Gottschalk96] S. Gottschalk, M. C. Lin and D. Manocha. *OBB-Tree: A Hierarchical Structure for Rapid Interference Detection*. In Proceedings of ACM SIGGRAPH 96, August 1996.

[Gregory99] Arthur D. Gregory, Ming C. Lin, Stefan Gottschalk and Russel Taylor. *H-Collide: A Framework for Fast and Accurate Collision Detection for Haptic Interaction*. In Proceedings of IEEE Virtual Reality Conference 1999, pp. 38–45, 1999.

[Gregory2000] Arthur D. Gregory, Stephen A. Ehmann and Ming C. Lin. *inTouch: Interactive Multiresolution Modeling and 3D Painting with a Haptic Interface*. In Proceedings of IEEE Virtual Reality Conference 2000.

[Haebarli93] Paul Haeberli and Mark Segal. *Texture Mapping as a Fundamental Drawing Primitive*. Grafica Obscura—Collected Computer Graphics Hacks, http://www.sgi.com/grafica/texmap/, June 1993.

[Hanrahan90] Pat Hanrahan and Paul Haeberli. *Direct WYSIWYG Painting and Texturing on 3D Shapes*. In Proceedings of ACM SIGGRAPH 90, pp. 215–223, August 1990.

[Heckbert86] Paul S. Heckbert. *Survey of Texture Mapping*. IEEE Computer Graphics and Applications, pp.56–67, November 1986.

[Johnson99] David Johnson, Thomas V Thompson II, Matthew Kaplan, Donald Nelson and Elaine Cohen. *Painting Textures with a Haptic Interface*. In Proceedings of IEEE Virtual Reality Conference 1999.

[Lévy98] Bruno Lévy and Jean-Laurent Mallet. *Non-Distorted Texture Mapping for Sheared Triangulated Meshes*. In Proceedings of ACM SIGGRAPH 98, August 1998.

[Maillot93] Jerôme Maillot, Hussein Yahia and Anne Verroust. *Interactive Texture Mapping*. In Proceedings of ACM SIGGRAPH 93, August 1993.

[McNeely99] William A. McNeely, Kevin D. Puterbaugh and James J. Troy. *Six Degree-of-Freedom Haptic Rendering Using Voxel Sampling*. In Proceedings of ACM SIGGRAPH 99, August 1999.

[Ruspini97] Diego C. Ruspini, Krasimir Kolarov and Oussama Khatib. *The Haptic Display of Complex Graphical Environments*. In Proceedings of ACM SIGGRAPH 97, August 1997.

[Thompson97] Thomas V Thompson II, David E. Johnson and Elaine Cohen. *Direct Haptic Rendering of Sculptured Models*. In Proceedings of 1997 ACM Symposium on Interactive 3D Graphics, pp. 167–176, April 1997.