Procedural Haptic Texture

Jeremy Shopf
University of Maryland
Baltimore County

Marc Olano
University of Maryland
Baltimore County

ABSTRACT

We present the Haptic Shading Framework (HSF), a framework for procedurally defining haptic texture. HSF haptic texture shaders are short procedures allowing an application-programmer to easily define interesting haptic surface interaction and the parameters that control the surface properties. These shaders provide the illusion of surface characteristics by altering previously calculated forces from object collision in the haptic pipeline.

HSF can be used in an existing haptic application with few modifications. The framework consists of user-programmable modules that are dynamically loaded. This framework and all user-defined procedures are written in C++, with a provided library of useful math and geometry functions. These functions are meant to mimic RenderMan functionality, creating a familiar shading environment. As we demonstrate, many procedural shading methods and algorithms can be directly adopted for haptic shading.

ACM Classification: H.5.2 [Information Interfaces and Presentation]: User Interfaces–Haptic I/O I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism – Virtual reality.

General terms: Algorithms, Languages

Keywords: Haptics, Haptic Texture, Haptic Interaction, Texturing, Interface, Virtual Reality, Shading

INTRODUCTION

Haptic devices, such as the SensAble PHANTOM [14], provide force feedback interaction, with the goal of allowing the user to touch, feel and manipulate virtual objects. Haptic interaction has been used in surgical simulation, with particular success for laparoscopic surgery [2, 3, 27]. It has been used to aid in identifying bonding sites for proteins [23]. In general, it can provide a more convincing interaction with any virtual object, combining the senses of touch and vision in an overall experience. Most haptic devices permit interaction using a single Haptic Interaction Point (HIP), or end-effector. McNeely et al. demonstrated more complex haptic tool interaction through line segments, groups of points, and polygons using voxel sampling [16]. The HIP serves as a probe, with which the user can poke, prod or rub a surface.

UIST'06, October 15–18, 2006, Montreux, Switzerland..

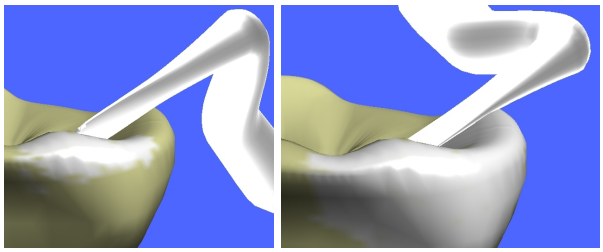

Figure 1: Haptic plaque removal application

The haptic interface can be divided into three types of forces. *Collision response* forces provide "poking" interactions, pushing back normal to the surface when the user pokes into an object. This simulates the solid nature of objects, and works to push the haptic device out of any virtual object. Collision response forces change for hard and soft objects. *Friction forces* simulate the friction felt when rubbing across a surface. Friction forces are tangential to the surface and oppose the direction of motion. Finally, *texture forces* are also tangential to the surface, and can simulate various other texture features.

Unlike visual rendering, where convincing interaction can be achieved at 30Hz or less, realistic haptic response requires a refresh rate of 1000Hz or better [25]. Thus, haptic response computations are quite time critical. Though some researchers have borrowed ideas from procedural surface shading to create more interesting haptic textures, they are typically "baked" into precomputed textures, saving runtime computation but removing the possibility of runtime changes to the interaction.

This paper proposes a procedurally-based dynamic haptic surface. A dynamic haptic surface can change its characteristics based on interaction in the virtual environment, passing of time, or user keyboard/mouse interaction. We show that such a haptic model can be efficient and meet the computational limits imposed by haptic interaction. Since the model is computed, it allows simple procedural definition. It is also computed at runtime, allowing changes with time or user interaction.

PREVIOUS WORK

We present prior work in four areas: general haptic interaction, haptic interaction for haptic texturing, visual procedural shading, and procedural haptic texturing.

Haptic Interaction

Haptic collision and haptic response are the two traditional parts of haptic interaction. Haptic collision entails determining when the Haptic Interaction Point has intersected the virtual object. This is relatively simple for geometric primitives such as spheres and cones, but becomes more difficult with more complicated objects such as polygonal models.

Often, algorithmic interference detection techniques such as oriented bounding box (OBB) trees, [7, 11], are required to speed intersection detection by the haptic device. Ray-polygon collision is needed to detect the intersection between the moving interaction point and object polygons. OBBs narrow the number of ray-polygon collision tests [1].

For a point interaction system such as the PHANTOM used in our work, collision response is the calculation of a reactionary force to be applied to the haptic device once collision is detected. Typically, this force is applied to return the interaction point to the corresponding surface position, preventing the user from penetrating the object. This force is calculated as a spring returning the HIP to the surface (using Hooke's Law):

$$\vec{F} = k\Delta\vec{x} \tag{1}$$

Where k is the stiffness and Δx is the depth of penetration into the object. Spring stiffness controls the hard or soft feel of the object. Since it is still possible for the HIP to penetrate the virtual surface, either due to device force limits or motion between haptic updates, total haptic experience is improved by showing a visual proxy effector that does not penetrate the rendered surface [28].

Haptic Texturing

By applying additional forces to the haptic interaction device when in contact with the surface, it is possible to simulate surface texture and friction. The total response can be calculated as:

$$\vec{F}_{result} = \vec{F}_c + \vec{F}_t + \vec{F}_f \tag{2}$$

Fritz and Barner described this equation, where F_c is the contact force described above, F_t is the force determined by the haptic texture and F_f is the force due to surface friction [6].

A lateral force applied in the opposite direction of device movement provides dynamic and static surface friction [21]. Perturbing the surface normal or displacing the surface according to some function provides a sensation of roughness. Fritz and Barner determined that altering the surface position is significantly more effective than perturbing of the surface normal for haptic textures [6], though Robles-De-La-Torre determined perturbing the surface normal by the addition of surface tangential forces provides a perceptually accurate virtual haptic surface [20].

Minsky and colleagues provided the first virtual textures via 2 degree-of-freedom haptic joysticks in what was dubbed the Sandpaper system [17]. Minsky used a tangential force-gradient algorithm for 2D texture rendering, where the displayed force is in the plane of the textured surface and proportional to the gradient of the surface-height profile. Hayward and Dingrong later extended this method to devices with three or more degrees of freedom [9].

Human perception of haptic texture has been studied extensively. In one study described by McGee et al., it was determined that participants had no difficulty in distinguishing that two textures were different, but when presented with several textures, they were unable to determine which of the textures was the roughest [15]. Sachtler et al. determined through a series of trials that a haptic device user's perception of a surface can be reliably altered by adding or subtracting tangential surface forces [22].

Procedural Shading

Procedural shading has been a mainstay of computer graphics for over two decades. Cook's Shade Trees [5] showed that a procedurally based description of a surface could result in increased realism and provide the programmer with a new level of control over the appearance of an object.

Shade trees replaced fixed surface shading with expressions parsed into a tree structure. Appearance parameters such as surface normal or diffuse shading coefficient form the leaves of the tree, combined by operators and functions to produce a final color at the root. One important aspect of Shade Trees was its use of precompiled libraries containing useful functions, such as *mix* and *specular*. This made shaders easier to write and allowed them to execute more efficiently.

Perlin extended shading from expressions to a full language, and also introduced the *noise* function for adding controlled randomness to shaders [19]. Many of these ideas were combined into the RenderMan shading language [8], which remains the most commonly used shading language for offline production rendering.

Shaders for all of these are written from a single surface sample's point of view, answering the question "How does a single sample on the surface compute its color?". This sample-based organization is key to the efficiency of HSF. Recently, it has become possible to perform some visual shading in graphics hardware. This is primarily made possible through independent execution of shading samples in parallel on graphics hardware [12, 13, 18].

Procedural Haptic Texturing

It is not hard to envision how procedural shading could be extended into the realm of haptics. Just as procedural shading adds realism and flexibility to the appear-

ance of an object, a procedural texture can provide haptic interaction with an interesting and dynamic surface.

Stochastic methods, which enjoy widespread use in procedural shading, have already been extensively applied in the realm of haptics. Siira and Pai were among the first to apply stochastic methods, using Gaussian white noise to provide a rough surface [24]. Fritz and Barner extend the use of stochastic methods to Fractional Brownian Motion (fBm) [6]. Methods of cellular texture, reaction-diffusion and spot noise have also been applied to haptic textures [10].

Though researchers have used procedural methods to generate interesting static surfaces, their focus has been on constructing or extending surface textures over a new novel surface in a preprocess.

HAPTIC SHADING FRAMEWORK

One key observation makes the Haptic Shading Framework feasible. Visual shaders run at least once per image pixel, with potentially millions of shader executions per frame. Some of that computation can be shared across samples, but visual shading requires huge computational resources. In contrast, a haptic shader need only be run once, at the haptic interaction point. Even a visual update rate of only 10 frames per second at 1280x1024, leads to approximately 13 million visual shader executions per second. A haptic update rate of 1000 Hz is just 1000 haptic shader executions per second.

Given this relatively modest computational requirement, we do not need parallel hardware to achieve real-time haptic shading. In order to achieve the 1kHz update rate, the HSF shader should execute in well under 1ms, leaving time for the necessary collision detection and force computation. Using high resolution cycle counting timing methods, we determined over a large sample of executions that the wood shader (our most intensive shader) executes in less than 79 μs on a Pentium 4 3.6GHz system.

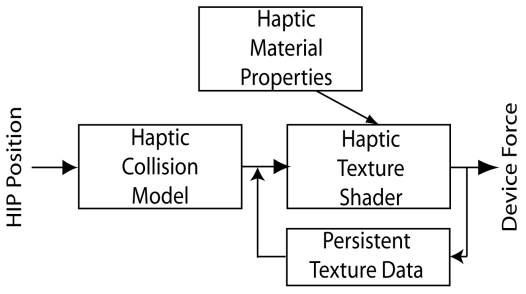

Figure 2: Data flow into and out of the haptic shader

The shader can be viewed as a black box through which information is passed after a haptic collision. Like the interface for visual shaders, we specify the haptic shader by its inputs and outputs. The inputs are values the system promises to have available if necessary. The outputs are values the haptic shader computes and delivers back to the system.

Inputs to the haptic shader come from two sources, as indicated in Figure 2. Inputs from the haptic collision model vary depending on the collision point, and are analogous to varying parameters in a visual shader. Haptic material property inputs are fixed across the object, and are analogous to uniform parameters in a visual shader.

The haptic collision model provides surface normal, surface position, surface parameters (u, v), and direction of tool tip movement. Direction of tool tip movement allows the programmer to generate direction-specific haptic forces.

Because haptic shaders execute after haptic collision, they can only be used to model small-scale surface displacements. Large displacements will cause inconsistent collision results and will disrupt the impression of the shaded surface structure. Each object will have its own base haptic material properties from which the haptic shader will calculate its results. These include static and dynamic friction, stiffness and damping.

Haptic texture inputs	Haptic texture outputs
Surface position	Displaced surface position
Normal	Perturbed surface normal
Binormal	
Tangent	
Collision force	Collision + texture force
Static friction	Static friction
Dynamic friction	Dynamic friction
Persistent Texture Data	Persistent Texture Data
Stiffness	Stiffness
Damping	Damping
Proxy direction	
Proxy acceleration	
Proxy velocity	
Surface parameters	

Figure 3: Haptic shader inputs and outputs

Static and dynamic friction are important in defining the "stick-slip" properties of the surface. Stiffness defines how hard or soft a surface feels through the coefficient of the spring force calculated between the tool tip and the corresponding surface proxy.

Persistent texture data is a two dimensional array of floating point values used to maintain information about the surface. Use of the texture requires that the model have surface parameters available. The persistent texture is dynamically allocated, allowing for a user-defined size. This texture is required to store information resulting from user interaction.

The output of the function is the resulting haptic force vector calculated from the inputs. Alternately, the shader can compute static and dynamic friction, stiffness and damping. The function may also return a displaced surface position and/or a perturbed surface normal.

```
surface wood
   float ringscale = 10;
   color lightwood = color (0.3, 0.12, 0.03),
   darkwood = color (0.05, 0.01, 0.005);
   float Ka = 0.2,
   Kd = 0.4,
   Ks = 0.6,
   roughness = 0.1)
{
   point NN, V;
   point PP;
   float y, z, r;

   NN = faceforward(normalize (N), I);
   V = -normalize(I);

   PP = transform("shader", P);
   PP += noise(PP);

   y = ycomp (PP);
   z = zcomp (PP);
   r = sqrt (y*y + z*z);

   /* map radial distance r into ring position [0, 1] */
   r *= ringscale;
   r += abs (noise(r));
   r -= floor (r); /* == mod (r, 1) */

   /* use ring position r to select wood color */
   Ci = mix(lightwood, darkwood, r);
   /* shade using r to vary shininess */
   Oi = Os;
   Ci = Oi * Ci * (Ka * ambient()
                + Kd * diffuse(NN))
               + (0.3 * r + 0.7)
               * Ks * specular(NN, V, roughness);
}
```

Figure 4: Visual wood shader (compare to Figure 5)

```
HapticTextureOut GetHapticTexture(HapticTextureIn before,
                                  DLLparams params)
{
   float ringscale = params.GetValue("ringscale", 5.0);
   float lightwood_staticF = params.GetValue("lightwood_staticF", 0.02);
   float lightwood_dynF = params.GetValue("lightwood_dynF", 0.02);
   float darkwood_staticF = params.GetValue("darkwood_staticF", 0.5);
   float darkwood_dynF = params.GetValue("darkwood_dynF", 0.5);

   HapticTextureOut after;
   vector3 PP;
   float y, z, r;



   /* ----- Shader Body ------ */
   PP = before.p;
   PP += noise(before.p);

   y = ycomp(PP);
   z = zcomp(PP);
   r = sqrt(y*y + z*z);

   /* map radial distance r into ring position [0, 1] */
   r *= ringscale;
   r += abs(noise1(r,r,r));
   r -= floor (r); /* == mod (r, 1) */

   /* use ring position r to select wood friction */
   after.staticF = mix(lightwood_staticF, darkwood_staticF, r);
   after.dynamicF = mix(lightwood_dynF, darkwood_dynF, r);
   after.f = before.f;
   /* ----- End Shader Body ------ */

   return after;

}
```

Figure 5: Haptic wood shader (compare to Figure 4)

The dynamic haptic texture allows the programmer to define additional parameters to the texture function that can be adjusted by the user during execution. This functionality is much like RenderMan, in that the number and names of parameters are variable, defined by the shader.

As with all visual shading languages, a library of useful functions is provided to make the shader-writing task easier. The list of functions should be familiar to anyone who has written visual shaders, including such functions as *mix, step, smoothstep, transform, min, max, clamp, distance, normalize, reflect* and *noise*.

IMPLEMENTATION

We have created a Haptic Shading Framework implementation on top of the SensAble PHANTOM 3D Touch SDK with OpenHaptics version 1.0. Haptic shaders are written in C++ and compiled into a DLL that can be loaded at run-time into the application. This run-time loading allows shader changes without requiring a recompile of the full application. This saves time, and in production use could allow haptic shaders to be changed without requiring users to have the application source at all. The common shading functions are provided through a C++ library.

In order to achieve seamless plug-in replacement of shaders, each haptic shader DLL must have one function named GetHapticTexture that is the external entry point. This function has two arguments, one structure containing all collision model parameters and one class object containing all haptic material property parameters. The function returns a class object containing the requisite haptic texture outputs.

Since the haptic material properties include user-defined parameters that the application may or may not have set, we access them through a GetValue method that returns the value (if set), or a default value (as demonstrated in Figure 5).

The Haptic Shading Framework is designed to fit on top of an existing haptic interaction environment. In this respect, the haptic texturing is limited by the information that the existing model can provide.

A comparison of RenderMan wood and HSF wood shaders

One goal of HSF was to provide an interface similar to the RenderMan shading language. Figure 4 is the RenderMan wood shader found in the RenderMan Companion [26]. Figure 5 is a corresponding haptic wood shader providing varying surface friction based on the wood grain pattern.

Obviously there are some differences in the details of the computation, since the haptic shader isn't concerned with calculating lighting. However, the similarity of structure between the two shaders is quite evident.

The **ycomp** and **zcomp** functions in the haptic wood shader are equivalent to **PP[1]** and **PP[2]**. Their use is purely aesthetic and furthers the RenderMan analogy.

Example Shaders

Five shaders that have been written with the initial demonstration application are discussed in this section.

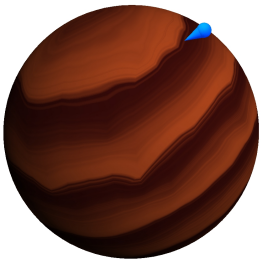

Figure 6: The virtual stylus interfacing a model using HSF wood shader. The object is rendered with a standard visual shading model, and also a wood haptic shader.

For each, we show the main portion of the shader body (as marked in Figure 5).

```
freq = 1.0;
for (int i = 0; i < 3; i++)
{
   u += noise1(before.p * domain * freq) * range/freq;
   freq *= 2;
}
after.f = before.f * (1 + u);
```

Figure 7: Segment of SimpleNoise shader

SimpleNoise provides a rough uneven surface determined by summing values from several octaves of Perlin noise (Figure 7). This function is similar to the granite displacement shader found in the RenderMan Companion [26]. Friction values in this shader are not altered, isolating the effects of scaling the magnitude of the resultant force vector.

```
float dot_result = DotProduct(textureDir, before.toolDir);
dot_result = (dot_result /2.0 )+ .5;
anisoStaticF = mix(minStaticF, maxStaticF, dot_result);
after.staticF = anisoStaticF;
```

Figure 8: Segment of Anisotropic shader

The anisotropic shader (Figure 8) illustrates the use of the tool-tip movement direction in a shader. The static friction is at a maximum when the tool-tip is being moved in the direction specified by textureDir, a user-defined vector.

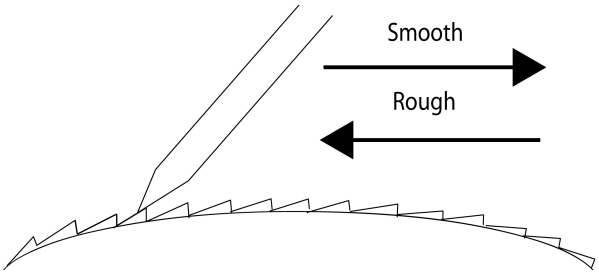

Figure 9: Surface interaction created by anisotropic shader

In the anisotropic shader, the output force is not altered, only the static friction values. These values are converted to appropriate lateral forces by the haptic toolkit outside of the texture function. Figure 9 illustrates a surface that may generate the described forces.

The Circles HSF shader in Figure 10 simply creates a metal inlay effect by using a varied friction on the surface of the rings. The vectors c1-c3 are the user-defined positions of the center of each circle. The innerRadius and outerRadius control the radius of circle and the radius of the center hole. If the collision point is between the innerRadius and outerRadius, a raised surface with no friction is felt. Otherwise, the object feels very rough.

```
float d1 = Distance(c1, projectedP),
      d2 = Distance(c2, projectedP),
      d3 = Distance(c3, projectedP);

if ((d1 < outerRadius && d1 > innerRadius) ||
    (d2 < outerRadius && d2 > innerRadius) ||
    (d3 < outerRadius && d3 > innerRadius))
{
   after.staticF = 0.02;
   after.f = before.f * 1.4;
} else
   after.staticF = 0.25;
```

Figure 10: Segment of Circles shader

Dentistry Application

As an application domain demonstration of the flexibility of the procedural haptic texturing method, we have implemented a haptic plaque removal program. We have chosen plaque removal because of the dynamic nature of the texture of the tooth. The initial tooth surface is covered in plaque. The goal is for the user to scrape the surface of the tooth to remove the plaque.

As plaque is removed, the tooth surface is made smooth by reducing the static and dynamic friction properties. This is achieved by supplying a plaque coverage texture through the texture array input of the haptic shader. This texture is initialized with a value of 1.0, indicating completely covered. As force is applied to regions on the surface, plaque is removed proportional to the magnitude of the force. The same texture can be used by a surface shader to provide the visual correspondence, as done in Figure 1.

Dynamic alteration of haptic texture due to tool interaction is a feature that has not been explored by previous haptic texuring methods. A procedural definition of a surface with the HSF inherently permits this.

An area that could benefit greatly from haptic shading is surgical simulation. There are many surfaces that change properties during surgery. For example, tissue becomes more slick if bleeding is occuring (or more sticky if blood is coagulating) and bone becomes rough if encountered with a scalpel or saw. Outside of the tool-interaction paradigm, another application domain is driving simulation. Road surfaces change as the driver moves along the road, causing changes in the vibration generated at the wheel and seat, and HSF would be more than capable of modeling this.

VISUAL/HAPTIC CORRESPONDENCE

It was quickly apparent to the authors that the exact intent of some haptic textures is not immediately apparent without some visual cues. This observation was noted by the authors and did not warrant a user-study.

We attribute this absence of understanding to a lack of intermodal integration. Feeling a surface without any visual correspondence may not provide presence and therefore is not convincing to the haptic device user [4]. This was especially true of the wood shader. We believe this is because it contains more subtle, non-uniform features (changes in surface friction) than the other shaders. The wood shader requires more consideration of the surrounding haptic texture to decipher the general layout of the texture.

Real-time shaders are typically written in a language like GLSL, HLSL or Cg. To align visual and haptic features, the shaders will have very similar code, as seen in Figures 4 and 5. Maintaining the one-to-one relationship requires extra care, especially when using stochastic methods such as noise functions, which must be identical, not just statistically similar, to maintain consistency.

For future work, we are interested in compiling a unified language for both visual and haptic shading. Using standard compiler dependency analysis techniques, it should be possible to automatically separate the code and inputs from a single unified source into a traditional visual shader and a haptic-only shader as described here.

As an example of shared input, the normal, binormal and tangent could be used as input to both the visual and haptic shader. Both shaders would use these vectors for transforming vectors (such as the light vector or the vector describing the movement of HIP) to surface tangent space. As another example, the same texture could be use to perturb the surface normal for lighting calculation (bump mapping) and to calculate the perturbed force vector for haptic interaction. This unified shading language would provide for a greater ease of use in authoring virtual environments.

CONCLUSION

We have presented a framework for procedurally defining haptic textures. Procedural haptic textures can be quickly written and easily modified during development. Previously, haptic texture was a static property. With our framework, objects rendered in a haptic environment can be dynamically altered according to user-defined parameters and interaction. This work allows significant improvement in user interaction and material flexibility in a haptic scene.

ACKKNOWLEDGMENTS

We would like to thank Dr. Alan Liu, Alark Joshi, Dana Wortman, and Kishalay Kundu for their constructive comments and insights for this paper. We would also like to thank Jason Bevins for libnoise (http://libnoise.sourceforge.net), which we used for our noise generation.

REFERENCES

1. Badouel, D. An efficient ray-polygon intersection. *Graphics Gems* (1996).
2. Basdogan, C., De, S., Kim, J., Muniyandi, M., Kim, H., and Srinivasan, M. A. Haptics in minimally invasive surgical simulation and training. *IEEE Computer Graphics and Applications 24:2* (2004), 56–64.
3. Basdogan, C., Ho, C. H., and Srinivasan, M. A. Virtual environments in medical training: Graphical and haptic simulation of laparoscopic common bile duct exploration. *IEEE/ASME Transactions on Mechatronics* (2001).
4. Biocca, F., Inque, Y., Polinsky, H., Lee, A., and Tang, A. Visual cues and virtual touch: Role of visual stimuli and intersensory integration in cross-modal haptic illusions and the sense of presence. In *Proceedings of Presence 2002* (2002).
5. Cook, R. L. Shade trees. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques* (New York, NY, 1984), ACM Press, pp. 223–231.
6. Fritz, J., and Barner, K. Stochastic models for haptic texture. In *SPIE Intl. Symposium on Intelligent Systems and Advanced Manufacturing – Telemanipulator and Telepresence Technologies III* (1996).
7. Gottschalk, S., Lin, M. C., and Manocha, D. OBBTree: a hierarchical structure for rapid interference detection. In *In Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques SIGGRAPH 96* (1996), ACM Press, pp. 171–180.
8. Hanrahan, P., and Lawson, J. A language for shading and lighting calculations. *SIGGRAPH Comput. Graph. 24*, 4 (1990), 289–298.
9. Hayward, D., and Dingrong, Y. Change of height: An approach to the haptic display of shape and texture without surface normal. In *Experimental Robotics VIII, Springer Tracts in Advanced Robotics* (2003), pp. 570–579.
10. Ho, C., Basdogan, C., and Srinivasan, M. Efficient point-based rendering techniques for haptic display of virtual objects. *Presence 8*, 5 (1999), 477–491.
11. Konig, H., and Strotthotte, J. Fast collision detection for haptic displays using polygonal models. In *Simulation and Visualisierung 2002* (2002), T. Schulze, S. Schlechtweb, and V. Hinz, Eds., pp. 289–300.
12. Lindholm, E., Kligard, M. J., and Moreton, H. A user-programmable vertex engine. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (New York, NY, 2001), ACM Press, pp. 149–158.
13. Mark, W., Glanville, R., Akeley, K., and Kilgard, M. Cg: A system for programming graphics hardware in a C-like language. *ACM Trans. Graph. 22*, 3, 896–907.
14. Massie, T. H., and Salisbury, J. K. The PHANTOM haptic interace: a device for probing virtual objects. *ASME Winter Annual Meeting, Symposium on haptic interfaces for virtual environments and teleoperator systems* (1994), 295–299.

15. McGee, M., Gray, P., and Brewster, S. Haptic perception of virtual roughness. In *CHI'01 Extended Abstracts on Human Factors in Computing Systems* (2001), ACM Press, pp. 155–156.

16. McNeely W. A., Puterbaugh K. D., T. J. J. Six degree of freedom haptic rendering using voxel sampling. In *Proc. ACM SIGGRAPH Int. Conf. on Computer Graphics and Interactive Techniques* (1999), pp. 401–408.

17. Minsky, M. *Computational Haptics: The Sandpaper System for Synthesizing Texture with a Force-Feedback Haptic Display*. PhD thesis, MIT, 1995.

18. Olano, M., and Lastra, A. A shading language on graphics hardware: the PixelFlow shading system. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (New York, NY, 1998), ACM Press, pp. 159–168.

19. Perlin, K. An image synthesizer. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques* (New York, NY, 1985), ACM Press, pp. 287–296.

20. Robles-De-La-Torre, G., and Hayward, V. Force can overcome object geometry in the perception of shape through active shape. *Nature 412*, 6845 (2001), 445–448.

21. Ruspini, D. C., Kolarov, K., and Khatib, O. Haptic interaction in virtual environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems:IROS'97*.

22. Sachtler, W. L., Pendexter, M. R., Biggs, J., and Srinivasan, M. A. Haptically perceived orientation of a planar surface is altered by tangential forces. In *Proceedings of Phantom User's Group 2000* (2000).

23. Sankaranarayanan, G., Weghorst, S., Sanner, M., Gillet, A., and Olson, A. Role of haptics in teaching structural molecular biology. In *Proceedings of Phantom User's Group 2003* (2003).

24. Siira, J., and Pai, D. Haptic texturing: A stochastic approach. In *International Conference on Robotics and Automation, IEEE* (1996), pp. 557–562.

25. Srinivasan, M. A., and Basdogan, C. Haptics in virtual environments: Taxonomy, research status, and challenges. *Computer and Graphics 21* (1997), 393–404.

26. Upstill, S. *The RenderMan Companion*. Addison-Wesley, 1992.

27. Webster, R., Haluck, R., Zoppetti, G., Benson, A., Boyd, J., Charles, N., Reeser, J., and Sampson, S. A haptic surgical simulator for laparoscopic cholecystectomy using real-time deformable organs. In *Proceedings of IASTED International Conference* (2003).

28. Zilles, C., and Salisbury, J. A constraint based god-object method for haptic display. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, Human Robot Interaction, and Cooperative Robots, Vol 3* (1995), pp. 146–151.