Human Skin Simulation by Generating Wrinkles along Vector Fields

by

Yosuke Bando

A Senior Thesis

Submitted to
the Department of Information Science
on February 13, 2001
in partial fulfillment of the requirements
for the Degree of Bachelor of Science

Thesis Supervisor: Tomoyuki Nishita
Title: Professor of Information Science

Abstract

In computer graphics, one of the important factors for rendering realistic human bodies is quality of skin. To improve this, it is necessary to represent small wrinkles on skin surfaces, which are locally oriented to certain directions. However, no method is developed to generate wrinkles over body parts like hands or arms, taking into account local directions. Thus, this paper proposes a method to generate wrinkles along the vector field assigned over the mesh which represents a body part. A vector field is obtained by interpolating the direction vectors which the user specified at some vertices of the mesh. A wrinkle pattern is generated by drawing along the vector field a number of line segments as wrinkle furrows, and by changing depth and width of the furrows depending on the magnitude of the vector field. The obtained wrinkle pattern is displayed using the bump mapping technique. An interface was implemented so that the user can easily edit a vector field, and it was shown that using the vector field whose assignment required relatively simple operations through this interface, the generated wrinkle patterns simulated the actual ones well.

Acknowledgments

I would like to thank Professor T. Nishita for his helpful suggestions to this work. I also thank all the members of Nishita Lab. for their useful discussions. And I acknowledge the BRDF data provided by the Cornell University Program of Computer Graphics.

Contents

1 Introduction 4

2 Topography of skin surface 6

3 Skin surface modeling 7

3.1 Vector field assignment 8
3.2 Mapping to texture space 10
3.3 Height field generation 12

4 Skin surface rendering 15

4.1 Normal map generation.................... 15
4.2 Color computation....................19

5 Implementation 22

5.1 User interface 22
5.2 Height field generation 26
5.3 Color computation 27

6 Results 29

7 Conclusion 39

References 40

1 Introduction

Recently, computer generated human models often appear in various fields such as movies, video games, commercial films, surgical operation simulations and so on. In many cases, they should be realistic enough to make one believe that they really exist. Thus, in computer graphics, it is getting more and more important to enhance their realism. Many approaches have been already taken, but there are only a few studies on human skin rendering, which has much room to improve.

There are two factors to display realistic human skin. First, colors of skin should imitate that of real human beings. To compute such colors, it is necessary to take into account the *bidirectional reflectance distribution functions (BRDF)* of the skin materials. The BRDF of a material is a function which describes how light is scattered at its surface. Hanrahan and Krueger regarded skin as layered surfaces and simulated its BRDF [1]. Marschner *et al.* measured the BRDF of skin by their device [2]. Second, small unevenness of skin surfaces should be well represented. This factor is what this paper mainly deals with.

The main elements which cause unevenness of human skin surfaces and strongly affect its visual appearance are wrinkles. Distinct wrinkles appear on specific body parts (*e.g.* lines on the palm or expressive wrinkles on the face) and have a different structure depending on the part, showing its characteristics. On the other hand, small wrinkles exist everywhere on the skin and have a similar structure, which partially determine the properties of skin material. There are many studies on the former wrinkles (*e.g.* [3-6]), but there are only a few on the latter. Ishii *et al.* used a hierarchical *Voronoi division* for creating polygonal patterns of small wrinkles, and made a curved surface on each polygon to represent a wrinkle ridge [7]. Wu *et al.* used a hierarchical *Delaunay triangulation* instead [6]. However, both did not sufficiently deal with local directions of wrinkles. The former partially took wrinkle directions into account, but it did not refer to a method to assign directions to the body part. The latter proposed a method to do this, but it was only for distinct wrinkles and could not be employed efficiently for small wrinkles.

Thus, this paper proposes a method to assign local directions to the body part and to generate wrinkles along them. A triangle mesh is used as the geometric model of the body part, and local directions are represented by a vector field the user assigned over the mesh. For assigning a vector field, an interface was implemented so that the user can easily edit a vector field. The user has only to specify direction vectors at some vertices of the mesh, and a vector field is obtained by automatically interpolating the specified vectors. The mesh and the vector field are mapped to a two-dimensional texture space using planar or cylindrical projection, and there a wrinkle pattern is generated by drawing along the projected vector field a number of line segments as wrinkle furrows. Besides, depth and width of the furrows are changed depending on the magnitude of the vector field. The obtained wrinkle pattern is displayed using the *bump mapping* technique. In addition, the BRDF of skin measured in [2] is optionally taken into account when displaying.

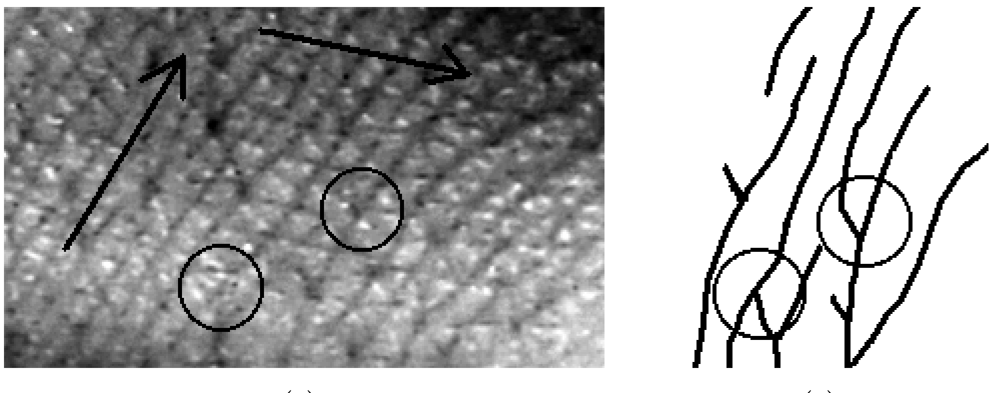
The system was implemented using OpenGL, widely spread graphics library for C language. Using this library efficiently so that the system can benefit from graphics hardware acceleration, enabled fast generation and rendering of wrinkle patterns. As a display result, it was shown that the generated wrinkle patterns simulated the actual ones well. And it was also shown that the system could control several properties of skin material and therefore could express some variety such as smooth skin and rough skin.

The rest of this paper consists of five sections as follows. Section 2 describes the topography of the skin surface on which the proposed method is based. Section 3 describes how to model the small variation on the skin surface caused by wrinkle furrows. Section 4 describes how to render the skin surface modeled in Section 3. Section 5 describes how the system is implemented. Section 6 shows the results. Section 7 concludes. References are given at the end.

2 Topography of skin surface

According to [8] and [9], the skin consists of, from top to bottom, the *epidermis*, the *dermis*, and the *hypodermis*. The epidermis is therefore the surface of the skin, and is a continually renewing, layered sheet which is mainly populated by the cells called *keratinocytes*. The thickness of epidermis is 0.1*mm* on the average, but it varies from 0.04*mm* on the eyelids to 1.6*mm* on the palms. Though it is the thinnest component, the epidermis functions as a barrier, protecting the body from the outer environment and helping maintain the inner environment.

The skin surface patterns begin to form during the fourth and fifth month of fetal life, and become increasingly clear during childhood. But once acquired, they remain stable throughout life. Patterned **intersecting lines** cover the entire skin surface except palms and soles, forming small polygons. Most of these polygons are roughly **diamond-shaped**, although this is not always the case (Figure 2.1). Thus, there seems to be **two** directions along which these lines run. Figure 2.1 also shows that an intersection point of two lines along the same direction is the endpoint of either of them. The developmental processes that determine the directions of these lines are not known for sure, although some factors are thought to affect them. In this paper, these lines are called *wrinkle furrows*.



(a) (b)

Figure 2.1: (a) Skin surface of a hand. Lines run along two directions (indicated by arrows), forming roughly diamond-shaped polygons. An intersection point of two lines along the same direction is the endpoint of either of them (indicated by circles). (b) A sketch of (a) showing the circled intersection points.

3 Skin surface modeling

Based on the description in Section 2, the proposed method draws wrinkle furrows along local directions rather than subdivide the surface into polygons as [6] and [7] did, for those polygons are made as a result of intersection of wrinkle furrows. And because these lines are likely to form a diamond-shape, the proposed method assumes that there are two local directions along which furrows run. Thus, two different and independent vector fields are assigned over a certain body part. A triangle mesh is used to represent the body part. Figure 3.1 shows an example of such a mesh.

There are three steps to generate and display wrinkle patterns on the mesh. First, assign two vector fields on the mesh so that they indicate two local directions of a wrinkle furrow at each point. Next, map the mesh and the vector fields to a two-dimensional texture space. Finally, generate a height field so that it represents the undulation of the skin surface, making wrinkle furrows along the vector fields.

The *xyz* coordinate system is used, with *x*-axis toward right, *y*-axis toward up, and *z*-axis toward the user (Figure 3.2).

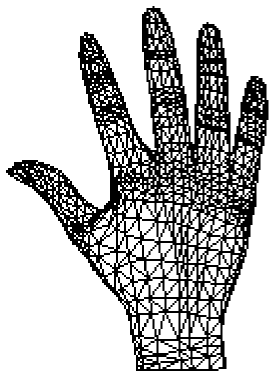


Figure 3.1: A triangle mesh representing a hand.

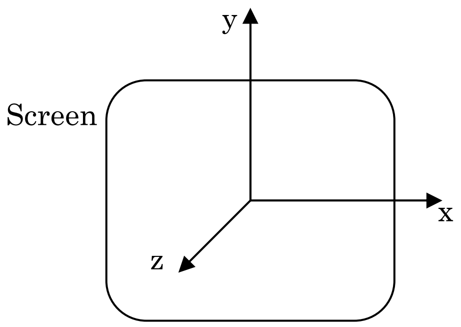


Figure 3.2: The coordinate system used.

3.1 Vector field assignment

Vector fields are assigned to the mesh and indicate the local directions of wrinkle furrows. Strictly speaking, a vector field must be such that direction vectors are defined at every point on the mesh. But in this step, they are defined only at the vertices. This will do because a direction vector at any point on a triangle is linearly interpolated in the texture space from the three vertices of the triangle. As mentioned in Section 2, two vector fields are needed, so two direction vectors must be specified at every vertex. Two direction vectors at vertex $\mathbf{v}$ are denoted by $\mathbf{d}^{(1)}(\mathbf{v})$ and $\mathbf{d}^{(2)}(\mathbf{v})$. The method in this subsection refers to [10].

Initially, $\mathbf{d}^{(1)}(\mathbf{v})$ is set to the projected vector of $(0,1,0)$ onto the tangent plane (*i.e.* the plane perpendicular to normal vector $\mathbf{n}(\mathbf{v})$ at the vertex). And $\mathbf{d}^{(2)}(\mathbf{v})$ is set to $\mathbf{d}^{(1)}(\mathbf{v})\times\mathbf{n}(\mathbf{v})$, where $\times$ denotes the cross product of two vectors (Figure 3.3). The user changes direction and magnitude of the vectors at some vertices, and vectors at the rest of the vertices are interpolated. Let N vertices at which the user edited the direction vectors be denoted by $\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_N$, and the interpolated direction vector at the vertex $\mathbf{v}$ is obtained using the following equations. The user can also specify λ_i (initially set to 1) appearing in the second equation.

$$\mathbf{d}^{(j)}(\mathbf{v})=\frac{\sum_{i=1}^{N} w_i \mathbf{d}^{(j)}(\mathbf{v}_i)}{\sum_{i=1}^{N} w_i}, \quad w_i=\frac{\lambda_i}{\| \mathbf{p}(\mathbf{v})-\mathbf{p}(\mathbf{v}_i) \|}, \tag{3.1}$$

where j is the index of direction vector, which is 1 or 2, and $\mathbf{p}(\mathbf{v})$ is the position of vertex $\mathbf{v}$. The reason why this equation is adopted is predictability. It is simply the weighted average where the weight w_i is inversely proportional to the distance and proportional to λ_i, thus the result of interpolation is predictable, which makes it relatively easy for the user to decide which vectors to edit. Figure 3.4 (a) shows the edited vectors and (b) shows the vector fields obtained by interpolation.

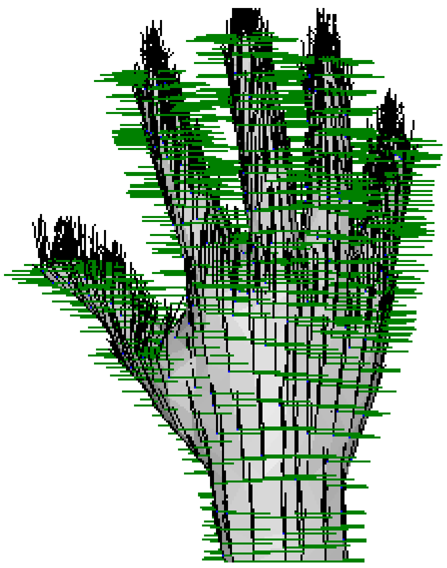
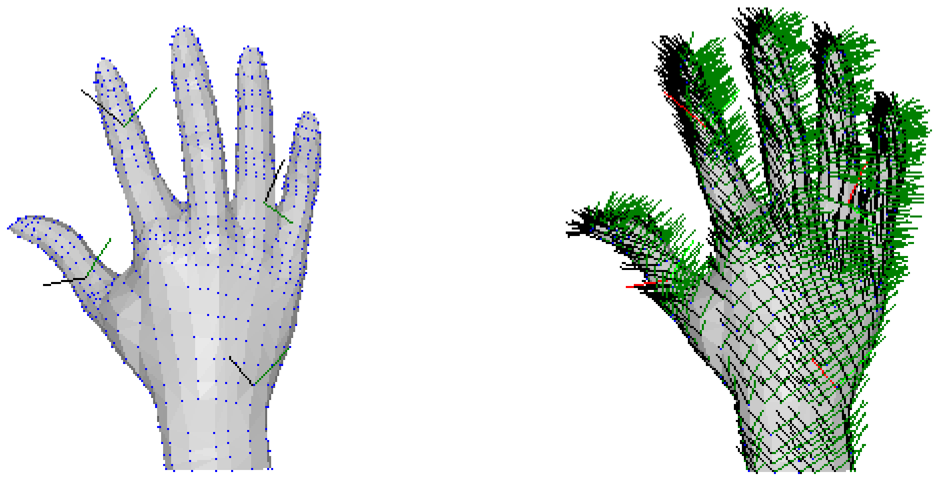


Figure 3.3: Initial vector fields assigned over the mesh. Lines represent direction vectors at each vertex.



(a)

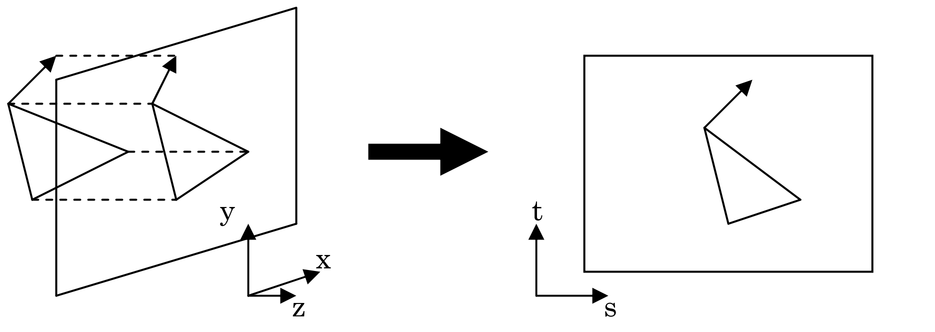
(b)

Figure 3.4: (a) Two direction vectors specified at four vertices. (b) The obtained vector fields by interpolating from specified direction vectors in (a).

3.2 Mapping to texture space

To make the wrinkle generation easier and faster, the mesh and the vector fields are mapped to a two-dimensional texture space. Wrinkles are drawn in this space, and the generated image is mapped back to the mesh when displayed. Planar projection and cylindrical projection are used, for most parts of the body can be mapped with small distortion by selecting appropriate projection of the two. For example, the hand and the foot can be mapped using planar projection, the arm and the leg cylindrical projection. The orthogonal coordinate system is used for the texture space, consisting of *s*-axis and *t*-axis.

Planar projection is very simple. Vertex (p_x, p_y, p_z) of the mesh is mapped to point (p_x, p_y). Figure 3.5 shows this mapping. The image in the texture space is just the same as the user sees in the screen (of course when the orthogonal projection is used for displaying the mesh). Direction vector (d_x, d_y, d_z) specified at the vertex is projected in the same way, and therefore the projected vector is (d_x, d_y).



Three-dimensional space Two-dimensional texture space

Figure 3.5: Mapping a triangle and a vector using planar projection.

Cylindrical projection is a little bit complicated. When the cylinder is placed vertically, centered at the origin, its surface is expressed by radius r, angle s from *z*-axis, and height t (Figure 3.6). Therefore, any point $\mathbf{P}$ is cylindrically projected if the values of s and t are known (this is projection, so the value of r is discarded). Because $\mathbf{P} = (p_x, p_y, p_z)$ is expressed as $(r\sin s, t, r\cos s)$, the following equations are obtained.

$$r = \sqrt{p_x^2 + p_z^2}\ ,\quad s = \begin{cases} \cos^{-1}\left(\dfrac{p_z}{r}\right) & (p_x > 0) \\ 2\pi - \cos^{-1}\left(\dfrac{p_z}{r}\right) & (\text{otherwise}) \end{cases}\ ,\quad t = p_y\ . \qquad (3.2)$$

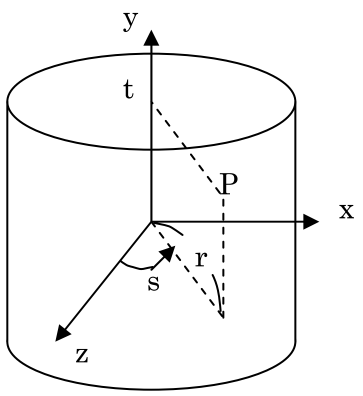
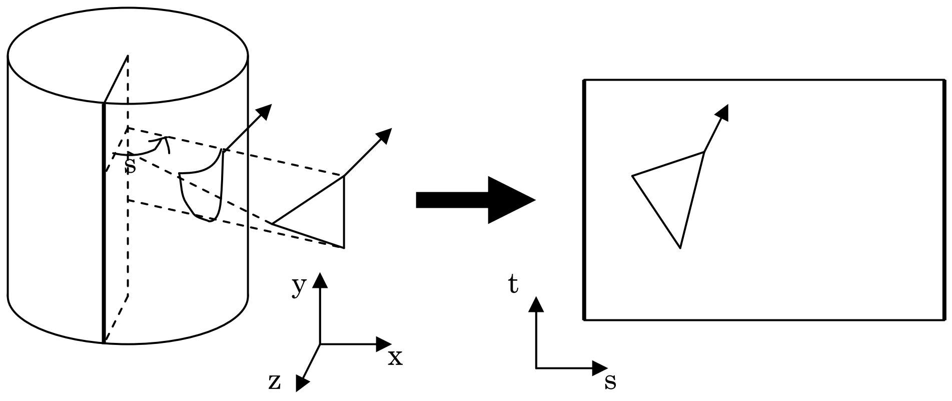


Figure 3.6: Parameterization of a cylinder.

Thus, vertex (p_x, p_y, p_z) is mapped to point (s,t). Figure 3.7 shows this mapping. A triangle is projected onto the cylinder, and the cylinder is cut and unfolded into a plane. Because of this unfolding, projection of a vector involves rotation. A vector is projected onto the cylinder and the projected vector is rotated by angle $-s$ around y-axis so that it is included in the plane. But in this case the projection and the rotation is exchangeable, so it is no problem to rotate a vector first and to project the rotated vector onto the plane (just by omitting the z component) next. Therefore, direction vector (d_x, d_y, d_z) is mapped to $(d_z \sin(-s) + d_x \cos(-s), d_y)$.



Three-dimensional space   Two-dimensional texture space

Figure 3.7: Mapping a triangle and a vector using cylindrical projection.

3.3 Height field generation

The proposed method uses a height field to represent the undulation of the skin surface. Height field $h(s,t)$ is a function which represents the height at point (s,t) in the texture space. This method regards it as an image, a rectangular color buffer. Grayscale color is used and it is represented by one value between 0 (black) and 1 (white). This value corresponds to the height. At first, the image is cleared in white, and here gray line segments are drawn so that they represent the furrow of wrinkles. In addition, boolean buffer B is used for marking points which locate near the already drawn furrows. This marking forbids drawing a new furrow from here and thus prevents furrows from becoming too crowded. Because a height field is generated in the texture space, the **projected** mesh and the **projected** vector field are used in the algorithms below.

When point P is given as a starting point, the algorithm for drawing one wrinkle furrow from here is as follows. N_{limit} is initialized to infinity.

1. Obtain direction vector $\mathbf{d}$ at P by linearly interpolating from three vertices of the triangle to which P belongs. The interpolation is done in the similar way to Gouraud's smooth shading [11].
2. Rotate $\mathbf{d}$ by angle $f_{angle}(\|\mathbf{d}\|)$.
3. Move from P by length $f_{length}(\|\mathbf{d}\|)$ in the direction obtained in step 2. If there are some other furrows or the border of the mesh in the way, set Q to the intersection point nearest to P. Otherwise set Q to the reached point (Figure 3.8).
4. In buffer B, set the points covered with segment PQ whose width is $f_{density}(\|\mathbf{d}\|)$, to TRUE.
5. Draw segment PQ with width $f_{width}(\|\mathbf{d}\|)$ and in color $f_{depth}(\|\mathbf{d}\|)$.
6. If $N_{limit} = 0$ or Q is the intersection point in step 3, stop. Otherwise return to step 1, setting $N_{limit} \leftarrow \min\{N_{limit}, f_{limit}(\|\mathbf{d}\|)\} - 1$, $P \leftarrow Q$.

N_{limit} holds the number of additional iteration. In step 6, N_{limit} decreases by 1 when returning to step 1 unless $f_{limit}(\|\mathbf{d}\|)$ (this is an integer value) is smaller than N_{limit}. If $f_{limit}(\|\mathbf{d}\|)$ is smaller, N_{limit} decreases more. The functions such as f_{angle} and f_{length} control the characteristics of wrinkle furrows as shown in Table

3.1. What kind of functions the implemented system uses is described in Section 5. The reason why Q, in step 3, is set to the intersection point of another furrow if any, is that such point becomes the endpoint of the furrow as mentioned in Section 2.

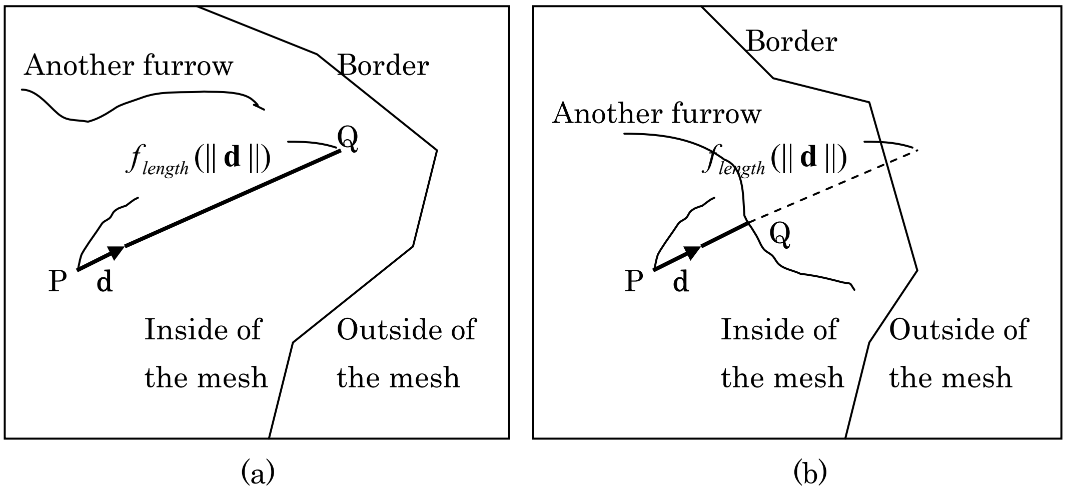


Figure 3.8: (a) There are no other furrows nor the border of the mesh in the way, thus Q is set to the reached point. (b) There is another furrow and the border, thus Q is set to the nearest intersection point to P.

Table 3.1: Functions and the characteristics of wrinkle furrows they control.

Function	**Characteristic**
$f_{angle}(\| \mathbf{d} \|)$	Small variation in direction of furrows
$f_{length}(\| \mathbf{d} \|)$	Length of one segment forming furrows
$f_{density}(\| \mathbf{d} \|)$	How crowded furrows are
$f_{width}(\| \mathbf{d} \|)$	Width of furrows
$f_{depth}(\| \mathbf{d} \|)$	Depth of furrows
$f_{limit}(\| \mathbf{d} \|)$	Total length of furrows

The following is the algorithm for generating height field $h(s,t)$ by using the algorithm written above. Recall that the algorithm above sets some points to TRUE in buffer B and forbids drawing furrows from those points. Two vector fields are denoted by $\mathbf{F}_1$ and $\mathbf{F}_2$ here.

1. Clear the image in white, and initialize B to FALSE.
2. For each triangle of the mesh, set point P to its center (of gravity) and draw one furrow from here along $\mathbf{F}_1$ and draw another furrow from here along the opposite direction of $\mathbf{F}_1$ if P is FALSE in B.
3. For each pixel of the image, set P to its center and draw one furrow from here along $\mathbf{F}_1$ and draw another furrow from here along the opposite direction of $\mathbf{F}_1$ if P is FALSE in B.
4. Store the image into $h(s,t)$.
5. Clear the image in white, and initialize B to FALSE.
6. For each triangle of the mesh, set P to its center (of gravity) and draw one furrow from here along $\mathbf{F}_2$ and draw another furrow from here along the opposite direction of $\mathbf{F}_2$ if P is FALSE in B.
7. For each pixel of the image, set P to its center and draw one furrow from here along $\mathbf{F}_2$ and draw another furrow from here along the opposite direction of $\mathbf{F}_2$ if P is FALSE in B.
8. For each point (s,t), if the value of pixel (s,t) is smaller than $h(s,t)$, set $h(s,t)$ to that value.
9. For each point (s,t), $h(s,t) \leftarrow h(s,t) - rk_{noise}$ where r is a random number between 0 and 1.

Obtained $h(s,t)$ is the generated height field. The opposite direction of the vector field is obtained simply by replacing direction vector $\mathbf{d}$ with $-\mathbf{d}$. By drawing another furrow along the opposite direction, starting point P becomes a midpoint of the furrow. First, in step 2 or 6, the center of gravity of each triangle is selected as a starting point, and the entire mesh is sparsely covered with furrows. Next, in step 3 or 7, the center of each pixel is selected, and the still uncovered regions are stuffed with furrows. Furrows along $\mathbf{F}_1$ and those along $\mathbf{F}_2$ are drawn independently, and two generated height fields are merged in step 8 by taking the minimum value at each point (s,t). In step 9, some noise is added to the height field to make the surface rougher. The user can control how much noise to add by setting the value of k_{noise}.

4 Skin surface rendering

There are two steps to display wrinkle patterns on the mesh. First, generate a normal map, which stores a unit normal vector at each point, from the height field obtained in Section 3. These normal vectors indicate the orientation of the wrinkled surface. Next, compute a color at each point in the texture space by applying some lighting equation to the normal vector, and map the generated image back to the mesh.

4.1 Normal map generation

Normal map $\mathbf{n}(s,t)$ stores the unit normal vector at the point on the mesh which is mapped to point (s,t) in the texture space. These normal vectors are made to represent the orientation of the wrinkled surface, and used in the lighting equation. First, the normal vector at each vertex of the mesh is normalized, and mapped to the corresponding vertex of the projected mesh. This is done in the way similar to the mapping of direction vectors except the z component is not omitted. Next, the normal vector at each point (s,t) is obtained by linearly interpolating from three vertices of the projected triangle to which point (s,t) belongs. This normal vector is normalized and stored in the normal map. Finally, the normal vectors are perturbed so that they represent unevenness of the surface given by the height field. The way of perturbing the normal vectors is based on the bump mapping technique.

The bump mapping technique, invented by Blinn [12], is a normal vector perturbation technique for simulating lighting effects caused by patterned irregularities on otherwise locally smooth surfaces. The classic formulation of bump mapping computes perturbed normal vectors for a parametric surface as if a height field displaced the surface in the direction of the unperturbed normal vector (Figure 4.1). Assuming a surface is parameterized by u and v, and expressed as $\mathbf{P}(u,v)$, the normal vector at $\mathbf{P}(u,v)$ is defined as

$$\mathbf{N}(u,v) = \frac{\partial \mathbf{P}}{\partial u} \times \frac{\partial \mathbf{P}}{\partial v}. \tag{4.1}$$

The following equation shows how this normal vector is perturbed using height field $h(u,v)$.

$$\mathbf{N}' = \mathbf{N} + \frac{\partial h}{\partial u}\left(\frac{\mathbf{N}}{\|\mathbf{N}\|} \times \frac{\partial \mathbf{P}}{\partial v}\right) + \frac{\partial h}{\partial v}\left(\frac{\partial \mathbf{P}}{\partial u} \times \frac{\mathbf{N}}{\|\mathbf{N}\|}\right). \tag{4.2}$$

The derivation of this equation is available in [12] and [13]. Figure 4.2 shows the relation described in Equation (4.2).

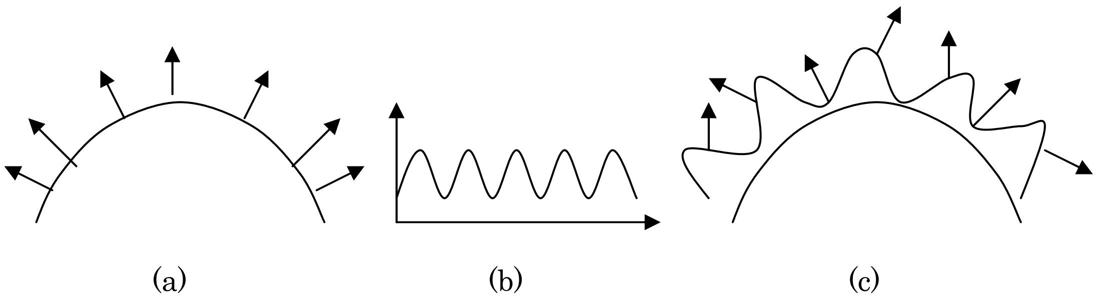


Figure 4.1: (a) A surface and its normal vectors. (b) A height field. (c) The normal vectors are perturbed as the height field displaces the surface.

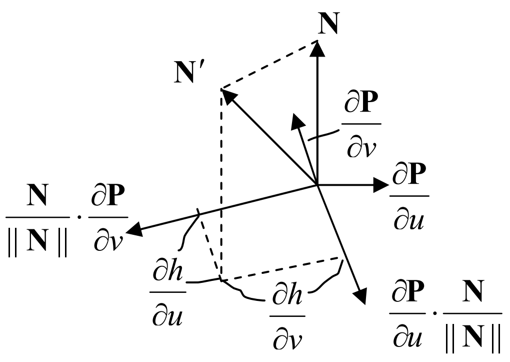

$\mathbf{N}$
$\mathbf{N}'$
$\frac{\partial \mathbf{P}}{\partial v}$
$\frac{\partial \mathbf{P}}{\partial u}$
$\frac{\mathbf{N}}{\|\mathbf{N}\|} \cdot \frac{\partial \mathbf{P}}{\partial v}$
$\frac{\partial h}{\partial u}$
$\frac{\partial h}{\partial v}$
$\frac{\partial \mathbf{P}}{\partial u} \cdot \frac{\mathbf{N}}{\|\mathbf{N}\|}$

Figure 4.2: Perturbing normal vector $\mathbf{N}$ of parameterized surface $\mathbf{P}(u,v)$ using information of height field $h(u,v)$.

In this method, h is defined at each point (s,t) in the texture space, so the surface must be also expressed as $\mathbf{P}(s,t)$. In this case, however, the surface represented by a polygonal mesh cannot be regarded as parameterized by the texture coordinate. The reason is that the surface is not smooth. The partial derivatives of $\mathbf{P}$ are not available and also cannot be approximated by the finite differences, for these finite differences suddenly change between polygons and this prevents smooth perturbation of the normal vectors.

Therefore, this method uses the surface defined by unperturbed normal map $\mathbf{n}(s,t)$. This surface is denoted by $\mathbf{P}'(s,t)$, and is defined so that it satisfies the followings.

1. $\mathbf{P}'(s,t)$ is expressed as $(s, t, f(s,t))$ using some function f.
2. The surface normal vector is $\mathbf{n}(s,t)$.

This surface is smooth because the normal vector at each point (s,t) is obtained by linear interpolation. Thus the partial derivatives of $\mathbf{P}'$ (and f) are available and expressed as

$$\frac{\partial \mathbf{P}'}{\partial s} = \left(1, 0, \frac{\partial f}{\partial s}\right), \quad \frac{\partial \mathbf{P}'}{\partial t} = \left(0, 1, \frac{\partial f}{\partial t}\right), \tag{4.3}$$

and satisfy the following equation.

$$\mathbf{n}(s,t) \cdot \frac{\partial \mathbf{P}'}{\partial s} = \mathbf{n}(s,t) \cdot \frac{\partial \mathbf{P}'}{\partial t} = 0\,. \tag{4.4}$$

Therefore, expressing $\mathbf{n}(s,t)$ as (n_x, n_y, n_z) leads to

$$\frac{\partial \mathbf{P}'}{\partial s} = \left(1, 0, -\frac{n_x}{n_z}\right), \quad \frac{\partial \mathbf{P}'}{\partial t} = \left(0, 1, -\frac{n_y}{n_z}\right). \tag{4.5}$$

Thus, Equation (4.2) evaluates to the following, replacing (u,v) with (s,t), and $\mathbf{P}$ with $\mathbf{P}'$.

$$\mathbf{N}' = \left(\frac{n_x}{n_z}, \frac{n_y}{n_z}, 1\right) + \frac{\partial h}{\partial s}\left\{\mathbf{n} \times \left(0, 1, -\frac{n_y}{n_z}\right)\right\} + \frac{\partial h}{\partial t}\left\{\left(1, 0, -\frac{n_x}{n_z}\right) \times \mathbf{n}\right\}. \tag{4.6}$$

Note that the normal map stores the unit normal vectors, thus

$$\mathbf{n}(s,t) = \frac{\mathbf{N}(s,t)}{\| \mathbf{N}(s,t) \|}. \tag{4.7}$$

Also note that

$$\mathbf{N}(s,t) = \frac{\partial \mathbf{P}'}{\partial s} \times \frac{\partial \mathbf{P}'}{\partial t} = \left(\frac{n_x}{n_z}, \frac{n_y}{n_z}, 1 \right). \tag{4.8}$$

Because scaling a vector does not change its direction, the perturbed normal vector $\mathbf{N}'' = n_z \mathbf{N}'$ is obtained using the following equation.

$$\mathbf{N}'' = \mathbf{n} + \frac{\partial h}{\partial s} \left\{ \mathbf{n} \times \left(0, n_z, -n_y\right) \right\} + \frac{\partial h}{\partial t} \left\{ \left(n_z, 0, -n_x\right) \times \mathbf{n} \right\}. \tag{4.9}$$

Here the scaling factor k_{bump} which the user can specify to control the influence of the height field is introduced, which leads to

$$\mathbf{N}'' = \mathbf{n} + k_{bump} \frac{\partial h}{\partial s} \left\{ \mathbf{n} \times \left(0, n_z, -n_y\right) \right\} + k_{bump} \frac{\partial h}{\partial t} \left\{ \left(n_z, 0, -n_x\right) \times \mathbf{n} \right\}. \tag{4.10}$$

The partial derivatives of h are not available, but they can be approximated using finite differences.

$$\frac{\partial h(s,t)}{\partial s} \approx \frac{h(s+\Delta s, t) - h(s,t)}{\Delta s}, \quad \frac{\partial h(s,t)}{\partial t} \approx \frac{h(s, t+\Delta t) - h(s,t)}{\Delta t}, \tag{4.11}$$

where Δs and Δt are the size of a pixel in the texture space. Finally, $\mathbf{N}''$ is normalized and stored in the normal map, and in this way the perturbed normal map is obtained from the unperturbed one.

4.2 Color computation

A color at each point (s,t) in the texture space is computed by applying some lighting equation to normal vector $\mathbf{n}(s,t)$ stored in the normal map. Because these normal vectors represent the orientation of the wrinkled surface, the generated image also appears to be wrinkled. Either the equation based on Blinn's lighting model [14] or that on the BRDF data is applied (the user can select which model to use). The generated image is mapped back to the mesh, and consequently the mesh with wrinkle furrows on its surface is displayed.

Blinn's lighting model is an empirical but widely used model because of its simplicity and computational efficiency. Color c_j (j is the color component, which is red, green or blue) is computed using the following equation.

$$c_j = I_{ambient}\, r_{ambient,j} + I_{light}\, r_{diffuse,j} \cos\theta + I_{light}\, r_{specular} \cos^n \gamma, \qquad (4.12)$$

where $I_{ambient}$ is the intensity of light coming from the environment,
I_{light} is the intensity of light coming from the light source,
$r_{ambient,j}$, $r_{diffuse,j}$ and $r_{specular}$ are the percentages of ambient, diffuse and specular component of reflected light,
n is the shininess of the material,
and θ and γ are the angles defined in the figure below.

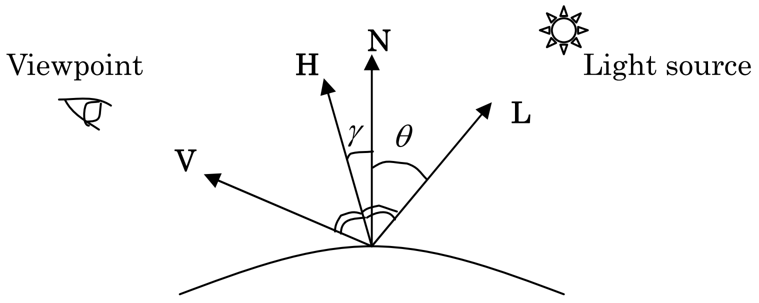


Figure 4.3: Relation between vectors N, L, V and H and angles θ and γ.

$\mathbf{L}$ is the unit vector oriented to the light source, $\mathbf{V}$ is the unit vector oriented to the viewpoint, and $\mathbf{N}$ is the unit normal vector of the surface. $\mathbf{H}$ is the unit vector oriented to the middle direction between $\mathbf{L}$ and $\mathbf{V}$.

$$\mathbf{H} = \frac{\mathbf{L}+\mathbf{V}}{\|\mathbf{L}+\mathbf{V}\|}. \qquad (4.13)$$

These vectors are used to compute $\cos\theta$ and $\cos\gamma$.

$$\cos\theta = \mathbf{L}\cdot\mathbf{N}, \quad \cos\gamma = \mathbf{H}\cdot\mathbf{N}. \qquad (4.14)$$

The optical properties of a material are determined by setting $r_{ambient,j}$, $r_{diffuse,j}$, $r_{specular}$ and n to the appropriate values. The values the implemented system uses for skin colors are described in Section 5.

In most cases, Blinn's model is used because it is fast and sufficiently accurate, but optionally, the lighting equation based on the BRDF data of skin can be used. The data is measured by Marschner *et al.* [2], and because the data is huge and cannot be applied directly to the lighting equation, they approximated the BRDF with the representation proposed by Lafortune *et al.* [15]. Their representation is a generalization of the *cosine lobe model* that is based on Phong's lighting model [16].

BRDF $f_r(\mathbf{u},\mathbf{v})$ is a function of incident light direction $\mathbf{u}$ and reflected light direction $\mathbf{v}$, which describes the intensity of the reflected light as compared with that of the incident light. Here $\mathbf{u}$ and $\mathbf{v}$ are assumed to be normalized and defined in the local coordinate system of the surface as illustrated in Figure 4.4, with the surface normal vector as *z*-axis. Because the BRDF of skin is isotropic, direction of *x*-axis does not matter. The following is the formulation.

$$f_r(\mathbf{u},\mathbf{v}) = D + \sum_i \max\left(0, C_{x,i}\,u_x v_x + C_{y,i}\,u_y v_y + C_{z,i}\,u_z v_z\right)^{n_i}, \qquad (4.15)$$

where $\mathbf{u} = (u_x, u_y, u_z)$ and $\mathbf{v} = (v_x, v_y, v_z)$. The first term, D, concerns with non-directional diffuse, the second with directional diffuse and specular. In the case of isotropic reflection, $C_{x,i} = C_{y,i}$. The values of these coefficients for approximating the measured BRDF are provided by the Cornell University Program of Computer Graphics.

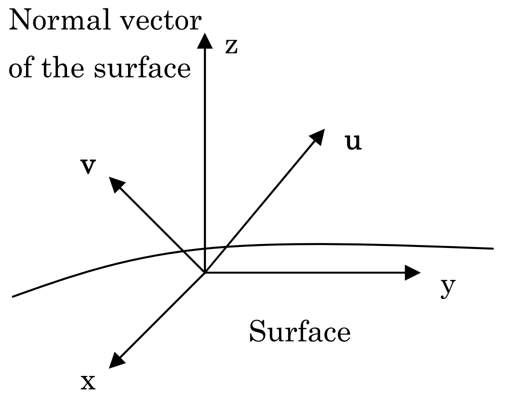


Figure 4.4: The local coordinate system is aligned to the normal vector of the surface.

Because BRDFs include both diffuse and specular terms, color c_j is computed using the following equation.

$$c_j = I_{ambient} r_{ambient,j} + I_{light} f_{r,j}(\mathbf{u}, \mathbf{v}) \cos\theta \,, \tag{4.16}$$

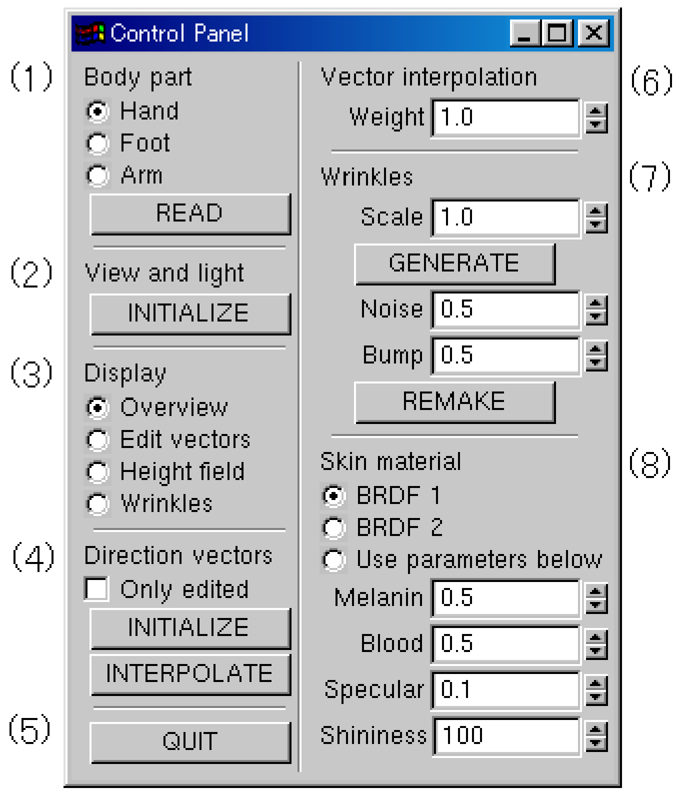
where $I_{ambient}$, I_{light}, $r_{ambient,j}$ and θ is the same as those in Equation (4.13).

5 Implementation

The system was implemented using OpenGL. GLUI, the user interface library based on OpenGL Utility Toolkit (GLUT) is also used for building the interface. This section describes how the implemented interface is easy to handle, and how the system benefit from graphics hardware acceleration when generating a height field and computing colors. The functions used for drawing wrinkle furrows and the values of the parameters in the lighting equations are also described (see Section 3.3 and Section 4.2).

5.1 User interface

Figure 5.1 shows the user interface implemented using GLUI. The following is its usage.



Body part: [✓ Hand] Foot Arm

View and light

Display: [✓ Overview] Edit vectors Height field Wrinkles

Direction vectors: ☐ Only edited

Vector interpolation: Weight 1.0

Wrinkles: Scale 1.0; Noise 0.5; Bump 0.5

Skin material: [✓ BRDF 1] BRDF 2 Use parameters below

Melanin 0.5

Blood 0.5

Specular 0.1

Shininess 100

Figure 5.1: Implemented user interface.

(1) Body part

The user can select the body parts listed here. When the “READ” button is pushed, the mesh which represents the selected part is read from the file and displayed.

(2) View and light

The user can set the direction of the light and view to the initial value, which are both $(0,0,1)$. Parallel light source is used.

(3) Display

“Overview” mode displays the smooth shaded mesh using skin material specified in (8). “Edit vectors” mode displays the flat shaded mesh in white, vertices of the mesh in blue, one vector field in red and the other in green. In this mode, the user can edit the vector fields. “Height field” mode displays the mesh with a generated height field. “Wrinkles” mode displays the mesh with a shaded wrinkle pattern using skin material specified in (8). Orthogonal projection is used for displaying the mesh. (Figure 5.2)

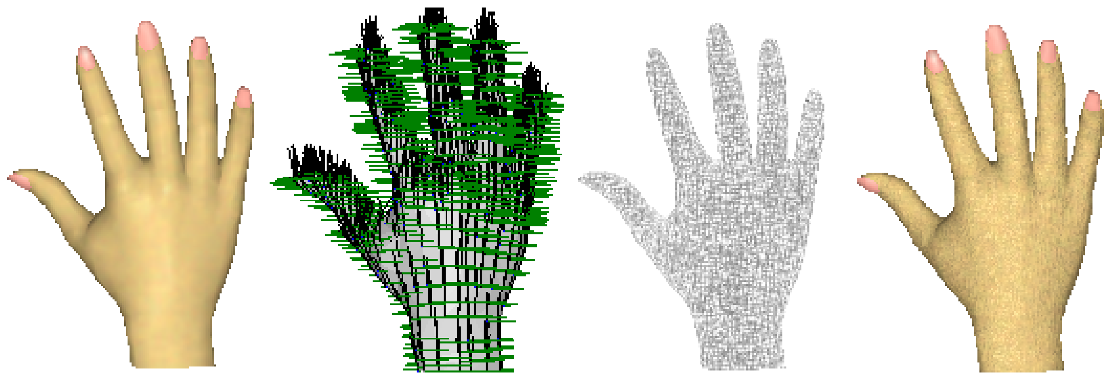
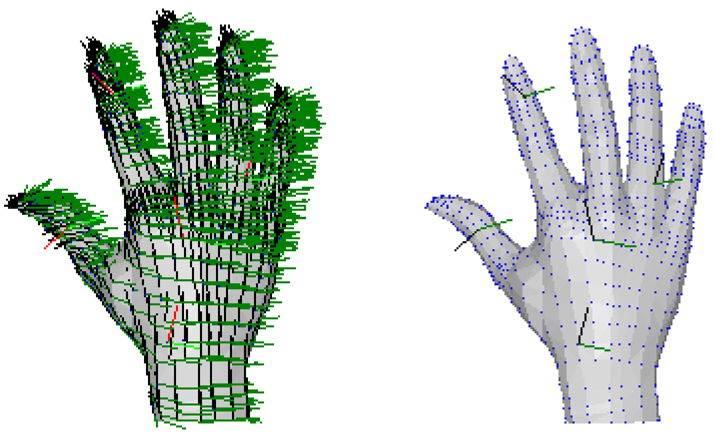

(a) (b) (c) (d)

Figure 5.2: Four display modes. (a) “Overview” mode. (b) “Edit vectors” mode. (c) “Height field” mode. (d) “Wrinkles” mode.

(4) Direction vectors

In “Edit vectors” mode, when “Only edited” is checked, only the edited direction vectors are displayed (Figure 5.3). “INITIALIZE” button initializes and “INTERPOLATE” button interpolates direction vectors as described in Section 3.1.



(a) (b)

Figure 5.3: (a) All direction vectors are displayed.
(b) Only the edited direction vectors are displayed.

(5) Quit

“QUIT” button terminates the system.

(6) Vector interpolation

In “Edit vectors” mode, “Weight” shows weight λ of the selected vertex. This is used in interpolating direction vectors (see Section 3.1). The user can change this value, which varies from 0 to 10.

(7) Wrinkles

“Scale” shows the scaling factor k_{scale} used in generating a height field (see Section 5.2). The user can change this value, which varies from 0.5 to 2. “GENERATE” button generates a height field and a normal map. “Noise” shows the value of k_{noise}, how much noise is added to the height field (see Section 3.3). “Bump” shows the value of k_{bump}, how much the generated height field influences the surface (see Section 4.1). The user can change these values, which vary from 0 to 1. “REMAKE” button remakes the height field and the normal map if only k_{noise} and k_{bump} are changed. This is faster than “GENERATE” because the height field is obtained by only adding noise to the already generated height field to which noise is not added yet.

(8) Skin material

"BRDF1" and "BRDF2" modes use the lighting model based on the BRDF data. There are two BRDFs. These are only for display mode "Wrinkles". "Use parameters below" mode uses Blinn's lighting model, and skin material is determined by four parameters, "Melanin", "Blood", "Specular" and "Shininess". This is for display mode "Overview" and "Wrinkles". "Melanin" shows the amount of melanin in the skin, and "Blood" shows the amount of blood. Both vary from 0 to 1. There is no physical or biological meaning for these two parameters. "Specular" shows the intensity of highlight, and "Shininess" shows how much highlight is concentrated. "Specular" varies from 0 to 1, while "Shininess" from 0 to 128. The user can change these four values. How these parameters are applied to Blinn's model is described in Section 5.3.

Besides the graphical user interface mentioned above, the user can also control the system by a mouse. In any display mode, the user can click and drag the mouse to rotate the mesh by the left button, and to change the direction of light by the right button. Further, in "Edit vectors" mode, the user can click and select a vertex, and drag the vector at the selected vertex to change its direction and magnitude (Figure 5.4).

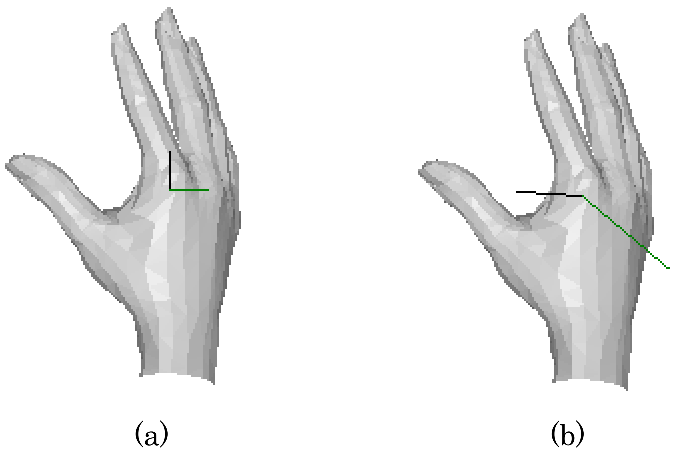


Figure 5.4: Editing direction vectors. (a) A vertex is selected by clicking it, and two initialized direction vectors of the selected vertex are displayed. (b) The direction vectors are modified by dragging them.

5.2 Height field generation

As described in Section 3.3, a height field is regarded as an image. The system displays the mesh using double buffering, so two color buffers are available for generating a height field. The system uses the front buffer for drawing lines which represent wrinkle furrows, and the back buffer for marking the points which locate near the already drawn furrows (Buffer B). Lines are drawn by calling OpenGL functions, glBegin(GL_LINES) and glVertex2f(), thus benefiting from hardware acceleration.

The followings are the functions which the system uses when drawing furrows.

$$f_{angle}(x) = 1 - 0.4x\ . \tag{5.1}$$

$$f_{length}(x) = k_{scale}(4 + x)\ . \tag{5.2}$$

$$f_{density}(x) = k_{scale}(1 + 6x)\ . \tag{5.3}$$

$$f_{width}(x) = k_{scale}x\ . \tag{5.4}$$

$$f_{depth}(x) = 0.25x^2\ . \tag{5.5}$$

$$f_{li\,mit}(x) = \lfloor 30 - 10x \rfloor . \tag{5.6}$$

The unit of angle is radian, and the unit of length is pixel. The larger the magnitude of the direction vector is, the straighter, longer, sparser, wider and deeper the furrow becomes. These functions except f_{depth} are linear to x so that the user can predict the result relatively easily. f_{depth} is proportional to the square of x, so that deep furrows stand out. The magnitude of each direction vector is assumed to vary from 0 to 2. f_{length}, $f_{density}$ and f_{width} concern with length, and length depends on the conditions such as the scale of the mesh and the texture resolution. Thus, these functions have a scale factor k_{scale}, which is set to the appropriate value by the user.

5.3 Color computation

Since OpenGL uses Blinn's lighting model, colors are computed fully by hardware. The normal map is used for lighting calculation, so the stored normal vectors must be rotated together as the mesh rotates. This rotation is simply done by referring to OpenGL *modelveiw* matrix, which transforms the vertices of the mesh when displaying. The system calls glGetFloatfv(GL_MODELVIEW_MATRIX, M) and gets this matrix $\mathbf{M}$, and rotates the normal vectors with it. The rotated vector is $\mathbf{Mn}$, where $\mathbf{n}$ is the original normal vector. The color is computed by calling glNormal3fv(), giving the rotated vector to its argument. Each point in the texture space is painted in the computed color by calling glBegin(GL_POINTS) and glVertex2i().

The parameters used in Blinn's lighting equation (see Equation (4.12)) are determined as follows. $I_{ambient}$ and I_{light} are set to 0.3 and 0.7, respectively. $r_{specular}$ and n are set to the values specified in "Specular" and "Shininess" of the user interface. Setting the value of $r_{diffuse,j}$ is based on [17], which describes that the locus of all normal skin colors forms a simple surface patch within RGB color space bound by two axes corresponding to the amounts of melanin and blood. For simplicity, the surface patch is assumed to be plane, and $r_{diffuse,j}$ is calculated using the following equation.

$$r_{diffuse,j} = r_{base,j} - k_{melanin}\, a_{melanin,j} - k_{blood}\, a_{blood,j}\ , \qquad (5.7)$$

where j is the color component, which is red, green or blue,
$r_{base,j}$ is the color of skin with no melanin and blood,
$k_{melanin}$ is the amount of melanin specified in "Melanin",
k_{blood} is the amount of blood specified in "Blood",
$a_{melanin,j}$ is the percentage of light which melanin absorbs,
and $a_{blood,j}$ is the percentage of light which blood absorbs.

Table 5.1 shows the values used for $r_{base,j}$, $a_{melanin,j}$ and $a_{blood,j}$. There is no physical or biological meaning for these values. $r_{ambient,j}$ is set to the same value as $r_{diffuse,j}$.

Table 5.1: The values used for $r_{base,j}$, $a_{melanin,j}$ and $a_{blood,j}$.

	red	**green**	**blue**
$r_{base,j}$	0.97	0.87	0.67
$a_{melanin,j}$	0.69	0.71	0.90
$a_{blood,j}$	0.00	0.24	0.24

For the lighting model based on the BRDF data, the color computation is done in software. The parameters used in the lighting equation (see Equation (4.16)) are determined as follows. $I_{ambient}$ is set to 0.3, and I_{light} 0.7. $r_{ambient,j}$ is set to the same value as D_j, the non-directional diffuse term for the color component j in the BRDF (see Equation (4.15)).

6 Results

This section shows some display results created by the implemented system. The calculation time for generating and rendering wrinkle patterns are also shown. Image size is 512×512.

Figure 6.1, Figure 6.2 and Figure 6.3 are examples of the hand, the foot and the arm. The hand and the foot are mapped with planar projection, while the arm is mapped with cylindrical projection. For each figure, (a) shows the triangle mesh which represents the body part. (b) shows the edited direction vectors, and (c) shows the vector fields obtained by interpolating the direction vectors shown in (b). (d) shows the generated height field from the vector fields. (e) and (f) are the final images of the body part with generated wrinkle pattern on its surface. The vector fields of these examples are all specified within 10 minutes. Table 6.1 shows the calculation time for these examples. The time for generating a height field is expressed in *second (s)*, and the time for rendering the mesh with the wrinkle pattern using each lighting model is expressed in *frame per second (fps)*. The machine used for measuring time has Athlon 1GHz for CPU and GeForce for graphics accelerator.

Table 6.1: Calculation time for generating a height field and for rendering the mesh with the wrinkle pattern using Blinn's lighting model and two BRDFs. The number of polygons of the mesh is also shown.

Figure	**Body part**	**Polygon**	**Generation**	**Rendering**		
				Blinn's	BRDF1	BRDF2
6.1	Hand	2807	8 *s*	19.0 *fps*	14.8 *fps*	14.3 *fps*
6.2	Foot	2332	8 *s*	18.2 *fps*	18.2 *fps*	16.0 *fps*
6.3	Arm	1373	10 *s*	10.5 *fps*	6.2 *fps*	5.3 *fps*

The limitation of the mapping with planar projection is seen in Figure 6.1(f) and Figure 6.2(f). The side of the mesh is mapped with strong distortion, thus the shading is not properly performed.

Figure 6.4 shows how the generated wrinkle pattern simulates the actual one. First, the wrinkle furrows run along some directions. Second, the intersecting furrows form polygons, and most of them are roughly diamond-shaped. Third, the intersection point of two furrows along the same direction is the endpoint of either of them.

Figure 6.5, Figure 6.6, Figure 6.7 and Figure 6.8 show that the system can control several properties of the skin material. Figure 6.5 shows the variation in distinctness of wrinkles which k_{bump} (see Section 4.1) controls. The wrinkles are more distinct in (b) than in (a). Figure 6.6 shows the variation in roughness of surface which k_{noise} (see Section 3.3) controls. The surface is rougher in (b) than in (a). Figure 6.7 and Figure 6.8 show the various skin colors. BRDF1 is used in Figure 6.7(a), BRDF2 Figure 6.7(b) and Blinn's model Figure 6.8. Figure 6.8 shows the difference depending on $k_{melanin}$, k_{blood} , $r_{specular}$ and n (see Section 4.2 and 5.3). All these parameters can be specified using the interface which the system provides (see Section 5.1). The values of the parameters used in each figure in this section are listed in Table 6.2. k_{scale} is set to the appropriate value depending on the actual scale of the body parts (see Section 5.2).

Table 6.2: The values of the parameters used for the examples.
All these parameters are specified through the user interface.

Figure	k_{noise}	k_{bump}	**Lighting model**	$k_{melanin}$	k_{blood}	$r_{specular}$	n	k_{scale}
6.1	0.0	0.5	Blinn's	0.2	0.5	0.2	100	1.1
6.2	0.0	0.5	Blinn's	0.2	0.5	0.2	100	1.0
6.3	0.0	0.5	Blinn's	0.2	0.5	0.2	100	0.8
6.4	0.0	1.0	Blinn's	0.2	0.5	0.2	100	1.1
6.5(a)	0.0	0.3	Blinn's	0.2	0.5	0.2	100	1.1
6.5(b)	0.0	0.8	Blinn's	0.2	0.5	0.2	100	1.1
6.6(a)	0.0	0.3	Blinn's	0.2	0.5	0.3	100	1.0
6.6(b)	0.3	0.3	Blinn's	0.2	0.5	0.3	100	1.0
6.7(a)	0.0	0.5	BRDF1					0.8
6.7(b)	0.0	0.5	BRDF2					0.8
6.7(c)	0.0	0.5	Blinn's	0.05	0.2	0.1	120	0.8
6.7(d)	0.0	0.5	Blinn's	0.5	0.6	0.4	70	0.8

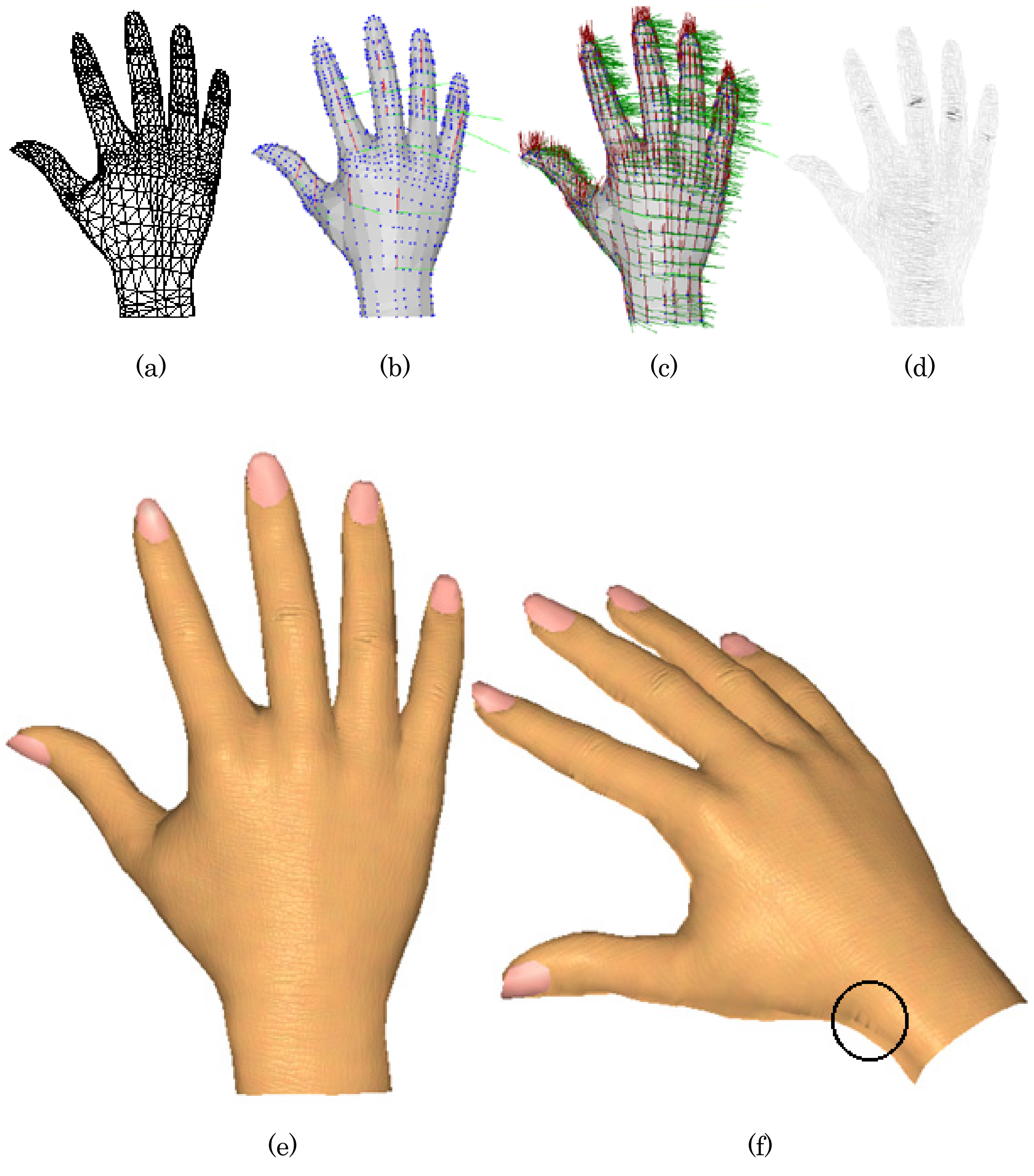


Figure 6.1: Display results of the hand. (a) A triangle mesh representing the hand. (b) Edited direction vectors at twenty-one vertices of the mesh. (c) Interpolated vector fields. (d) Generated height field. (e) and (f) The mesh with generated wrinkle pattern on its surface. Limitation of the mapping with planar projection is seen in (f) (indicated by a circle).

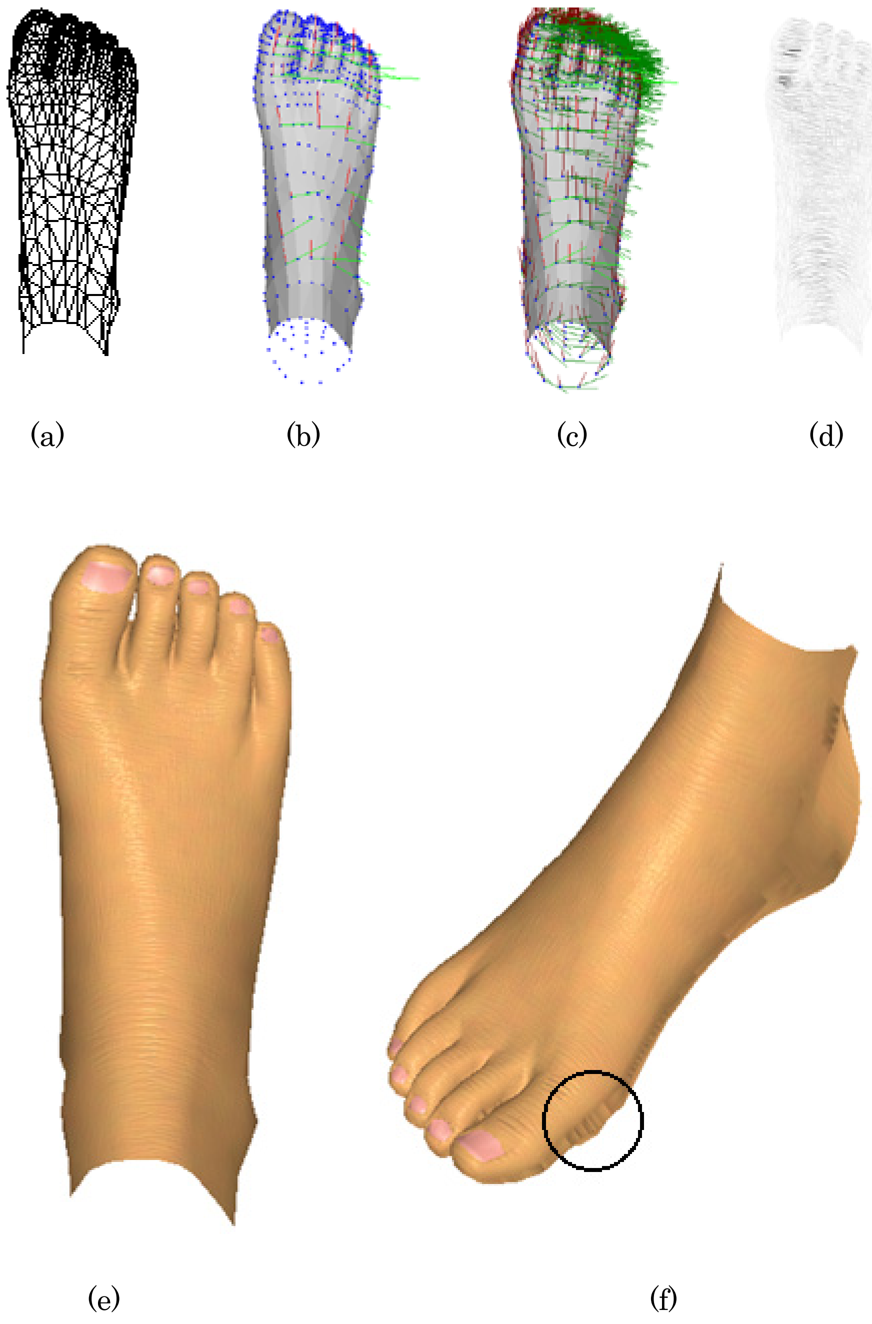


Figure 6.2: Display results of the foot. (a) A triangle mesh representing the foot. (b) Edited direction vectors at twenty-nine vertices of the mesh. (c) Interpolated vector fields. (d) Generated height field. (e) and (f) The mesh with generated wrinkle pattern on its surface. Limitation of the mapping with planar projection is seen in (f) (indicated by a circle).

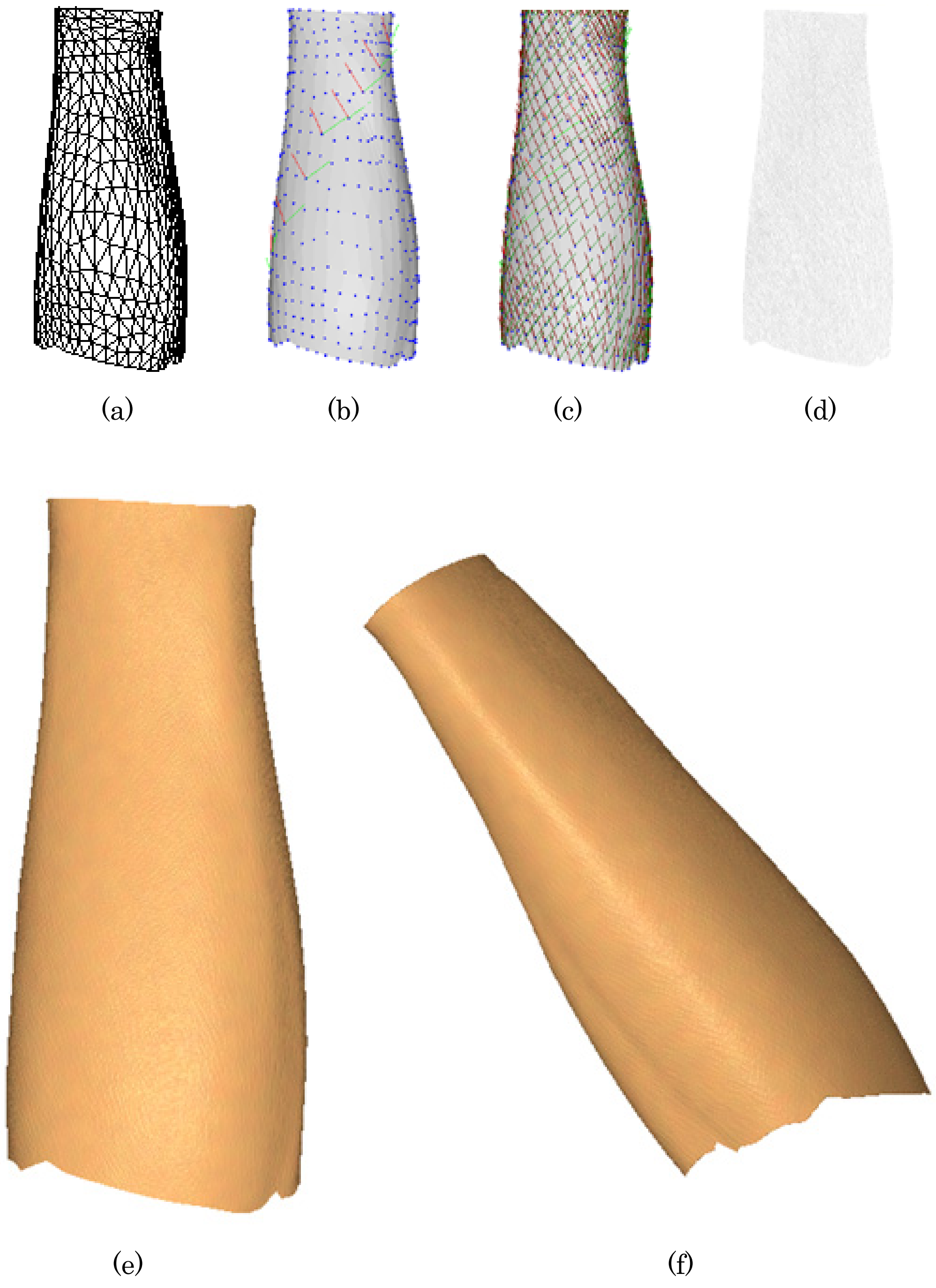


Figure 6.3: Display results of the arm. (a) A triangle mesh representing the arm. (b) Edited direction vectors at nine vertices of the mesh. (c) Interpolated vector fields. (d) Generated height field. (e) and (f) The mesh with generated wrinkle pattern on its surface.

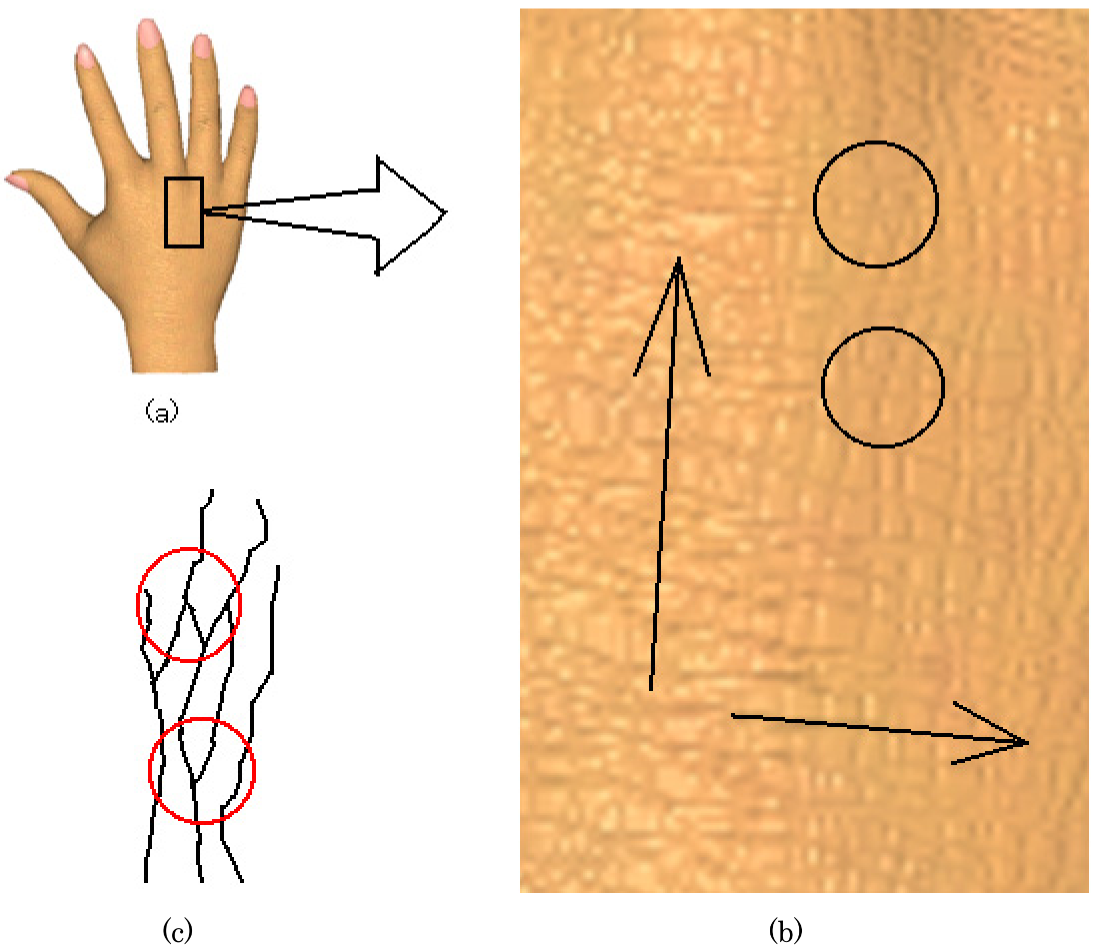
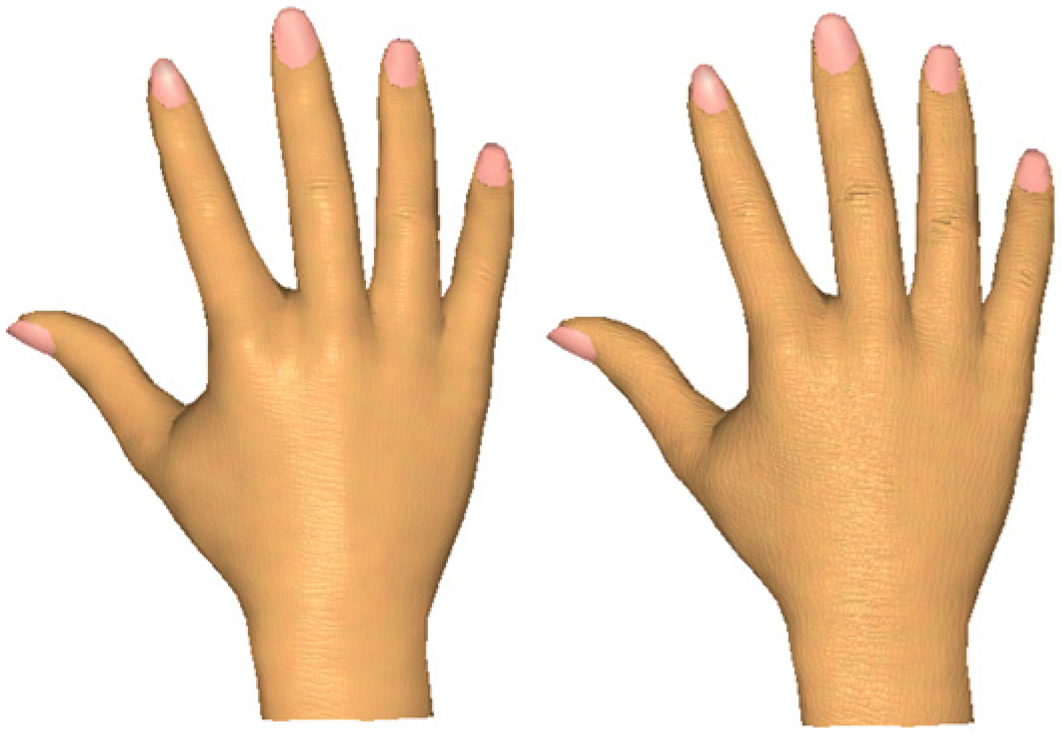
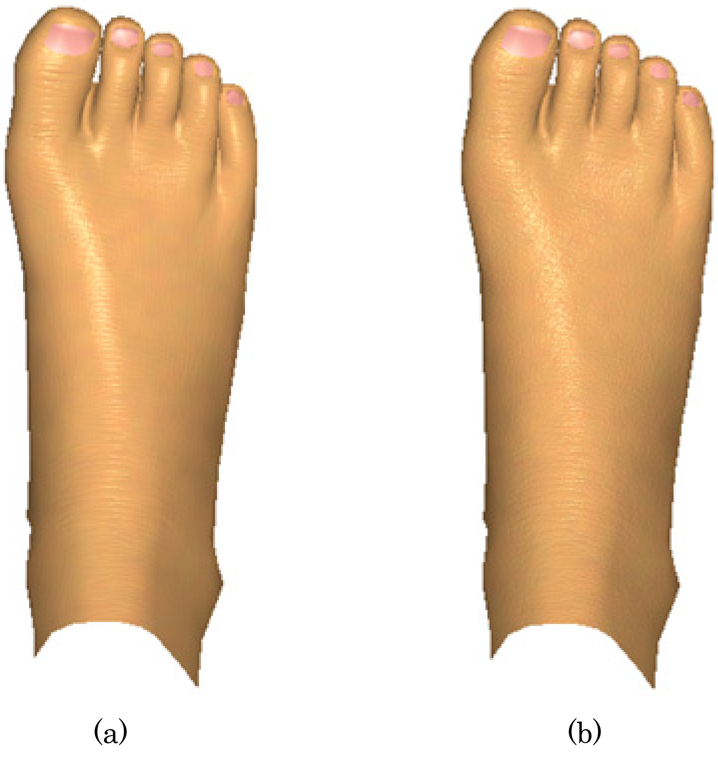


Figure 6.4: (a) Generated wrinkle pattern of the hand. (b) Magnified image of one area indicated by a rectangle in (a). The furrows run along two directions (indicated by arrows), forming roughly diamond-shaped polygons. An intersection point of two furrows is the endpoint of either of them (indicated by circles). (c) A sketch of (b) showing the circled intersection points.



(a) (b)

Figure 6.5: Variation in distinctness of wrinkles. The wrinkles are more distinct in (b) than in (a).



(a) (b)

Figure 6.6: Variation in roughness of surface. The surface is rougher in (b) than in (a).

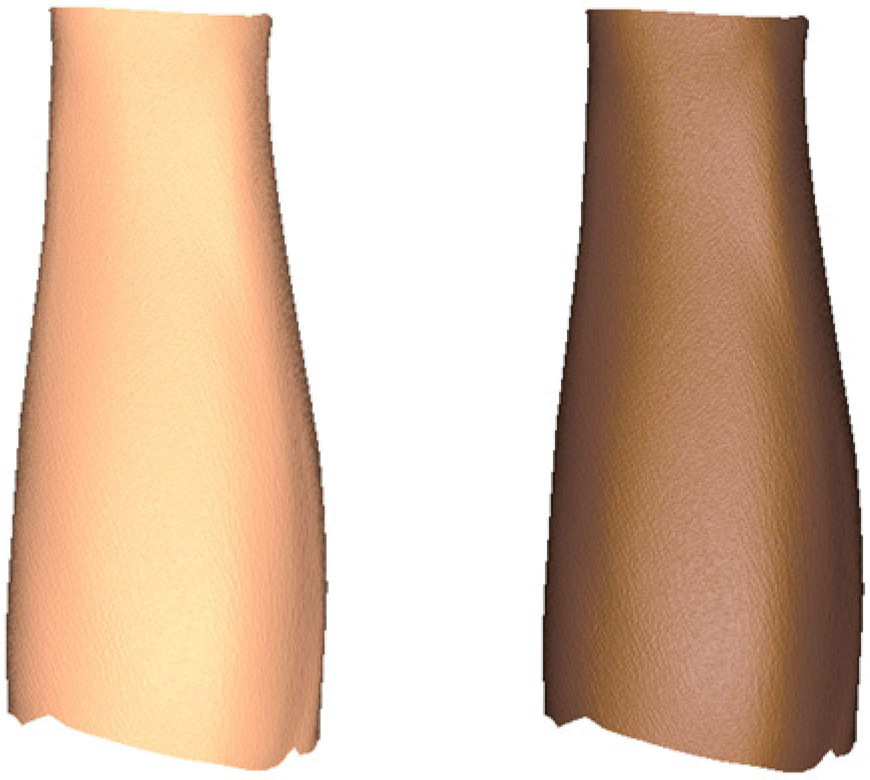
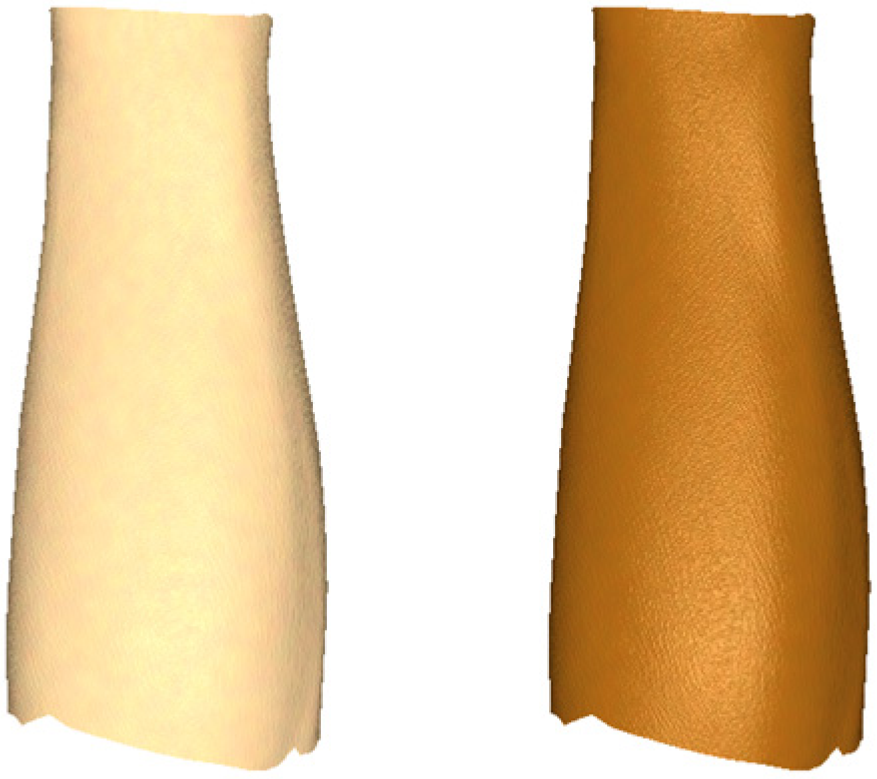


Figure 6.7: Display results using the lighting model based on the BRDF data of two persons' skin.



(a) (b)

Figure 6.8: Two examples of skin color using Blinn's model. Various skin colors can be produced by specifying $k_{melanin}$, k_{blood}, $r_{specular}$ and n.

7 Conclusion

This paper has proposed the method to simulate the skin surface by drawing wrinkle furrows along the vector fields. The generated wrinkle patterns simulated the actual ones well in the following points.

- The wrinkle furrows run along some local directions.
- The intersecting furrows form polygons, and most of them are roughly diamond-shaped.
- The intersection point of two furrows along the same direction becomes the endpoint of either of them.

Besides, the implemented system achieved the followings.

- It provides the user interface which is easy to handle, especially for assigning vector fields.
- It enables fast generation and rendering of wrinkle patterns by benefiting from graphics hardware acceleration.
- It can control several properties of skin by changing some parameters.
- It can use the lighting model based on the measured BRDF data.

This method can be applied to many kinds of body parts because the entire skin surface except palms and soles is covered with the similarly structured wrinkle furrows. However, this method cannot deal with distinct and large wrinkles because it does not consider the shape of a furrow and its bottom is therefore flat. Mapping with planar projection has some limitations because the side of the mesh is mapped with strong distortion and the wrinkle pattern is not properly generated nor displayed, although such areas are not so large. These problems will be dealt with in the future. In addition, the method to animate the body parts with the generated wrinkle patterns will be also studied in the future.

References

[1] P. Hanrahan and W. Krueger. Reflection from Layered Surfaces Due to Subsurface Scattering. *Computer Graphics (Proceedings of SIGGRAPH '93)*, pp. 165-174, 1993.

[2] S.R. Marschner, S.H. Westin, E.P.F. Lafortune, K.E. Torrance and D.P. Greenberg. Image-Based BRDF Measurement Including Human Skin. *Proceedings of 10th Eurographics Workshop on Rendering*, pp. 139-152, 1999.

[3] M. Viaud and H. Yahia. Facial Animation with Wrinkles. *3rd Workshop of Animation, Eurographics '92*, Cambridge, 1992.

[4] D. Terzopoulos and K. Waters. Physically-Based Facial Modeling, Analysis, and Animation. *Journal of Visualization and Computer Animation*, Vol. 1, No. 2, pp. 73-80, 1990.

[5] Y. Wu, N.M. Thalmann and D. Thalmann. A Plastic-Visco-Elastic Model for Wrinkles in Facial Animation and Skin Aging. *Proceedings of Pacific Conference '94*, pp. 201-213, 1994.

[6] Y. Wu, P. Kalra and N.M. Thalmann. Simulation of Static and Dynamic Wrinkles of Skin. *Proceedings of Computer Animation '96*, pp. 90-97, 1996.

[7] T. Ishii, T. Yasuda, S. Yokoi and J. Toriwaki. A Generation Model for Human Skin Texture. *Proceedings of CG International '93*, pp. 139-150, 1993.

[8] S.L. Moschella and H.J. Hurley. Dermatology (third edition), Vol. 1, pp. 3-14, W.B. Saunders Company, 1992.

[9] W. Montagna, A.M. Kligman and K.S. Carlisle. Atlas of Normal Human Skin, pp. 3-15, Springer-Verlag, 1992.

[10] E. Praun, A. Finkelstein and H. Hoppe. Lapped Textures. *Computer Graphics (Proceedings of SIGGRAPH 2000)*, pp. 465-470, 2000.

[11] H. Gouraud. Continuous Shading of Curved Surfaces. *IEEE Transactions on Computers*, Vol. C-20, No. 6, pp. 623-629, 1971.

[12] J. Blinn. Simulation of Wrinkled Surface. *Computer Graphics (Proceedings of SIGGRAPH '78)*, pp. 286-292, 1978.

[13] M.J. Kilgard. A Practical and Robust Bump-mapping Technique for Today's GPUs. *GDC (Game Developers Conference) 2000*, nVidia Corporation, 2000.

[14] J. Blinn. Models of Light Reflection for Computer Synthesized Pictures. *Computer Graphics (Proceedings of SIGGRAPH '77)*, pp. 192-198, 1977.

[15] E.P.F. Lafortune, S.C. Foo, K.E.Torrance and D.P. Greenberg. Non-Linear Approximation of Reflectance Functions. *Computer Graphics (Proceedings of SIGGRAPH '97)*, pp. 117-126, 1997.

[16] B.T. Phong. Illumination for Computer Generated Pictures. *Communications of the ACM*, Vol. 18, No. 6, pp. 311-317, 1975.

[17] S. Cotton, E. Claridge and P. Hall. A Skin Imaging Method Based on a Color Formation Model and its Application to the Diagnosis of Pigmented Skin Lesions. *Proceedings of Medical Image Understanding and Analysis '99*, pp. 49-52, 1999.