5 G-buffers for Non Photo realistic Rendering

A new rendering technique is proposed that produces 3D images with enhanced visual comprehensibility. Shape features can be readily understood if certain geometric properties are enhanced. To achieve this, we develop drawing algorithms for discontinuities, edges, contour lines, and curved hatching. All of them are realized with 2D image processing operations instead of line tracking processes, so that they can be efficiently combined with conventional surface rendering algorithms.

Data about the geometric properties of the surfaces are preserved as Geometric Buffers (G-buffers). Each G-buffer contains one geometric property such as the depth or the normal vector of each pixel. By using G-buffers as intermediate results, artificial enhancement processes are separated from geometric processes (projection and hidden surface removal) and physical processes (shading and texture mapping), and performed as post-processes. This permits a user to rapidly examine various combinations of enhancement techniques without excessive re-computation, and easily obtain the most comprehensible image.

Our method can be widely applied for various purposes. Several of these, edge enhancement, line drawing illustrations, topographical maps, medical imaging, and surface analysis, are presented in this chapter.

5.1 Introduction

Techniques for the comprehensible drawing of 3 dimensional shapes are indispensable for various applications such as industrial design or medical imaging. Their importance in computer graphics is not at all inferior to that of photo-realistic rendering techniques. Comprehensibility is mainly created through suitable enhancement lather than by accurately simulating optical phenomena. For shape comprehension, line drawings are effectively used as an addition to or substitute for surface coloring and shading [Kondo88]. For example, profiles and edges can be

enhanced with black or white border lines. Curved surfaces can be made more comprehensible by hatching with curved lines. These techniques are commonly used in hand drawn illustrations. However, they have not been adequately developed for computer graphics compared to photo-realistic rendering techniques.

The major problem for synthesizing a comprehensible image is determining the most suitable combination of enhancement techniques. The reason is that comprehensibility depends on the object, purpose, and sometimes the viewers' preferences, and cannot be expressed with theoretical definitions. Therefore, we must find the best combination by trial and error for each object or application. In order to maintain high productivity, graphics systems must be flexible and interactive to match the users' experimentation. For photo-realistic rendering, there is a lot of excellent research that aims to reduce image re-computation cost by preserving intermediate information [Duff85],[Mammem89],[Nakamae89],[Perlin85],[Porter84], and/or to build a rendering system flexibly by separating it into small procedures which can be combined freely [Cook84],[Crow84],[Nadas87],[Whitted82]. These techniques might appear to be effective for comprehensible rendering. However, enhancement using line drawings and conventional surface rendering are so different that it is difficult to combine them efficiently. This difficulty arises, for example, when eliminating hidden lines and surfaces for the same image [Saito89].

We propose a new enhancement technique for 3-D shapes that conceptualizes geometric properties. We have developed drawing algorithms for the basic enhancement operations, the drawing of discontinuity lines, contour lines, and curved hatching. All operations are realized with 2-D image processing operations, not with line tracking processes, so that they are suitable for interactive surface rendering environments.

The geometric properties are preserved as a set of Geometric buffers (G-buffers). A G-buffer set is obtained by forming projection views and removing hidden surfaces. Each buffer contains one geometric property, such as the depth or the normal vector, of the visible object in each pixel. The basic enhancement operations can be performed using G-buffer contents during post-processing. If geometric factors (i.e. shapes and camera parameters) are fixed, any combination of enhancement can be examined without changing the contents of the G-buffers.

The proposed method is also useful for photo-realistic rendering. It can be

considered an extension of Perlin's Pixel Stream Editor [Perlin85]; it means that Perlin's mapping techniques can be easily performed on a G-buffer set. Since the G-buffer set contains no physical (or optical) properties such as reflectance or colors, photo-realistic techniques can be used in post-processing and performed independently from enhancement operations. Therefore, the proposed method can be considered a very powerful and flexible rendering concept for various purposes.

5.2 Geometric Buffers

In this section, Geometric buffer set contents are introduced. A G-buffer set is the intermediate rendering result, and used as the input data for enhancement operations. Each buffer contains a geometric property of the visible object in each pixel. The following properties are the typical contents of a G-buffer set.

- **id**: object/patch identifier
- **ou**: patch coordinate u
- **ov**: patch coordinate v
- **sz**: screen coordinate z (perspective depth)
- **wx**: world coordinate x
- **wy**: world coordinate y
- **wz**: world coordinate z
- **nx**: normal vector x
- **ny**: normal vector y
- **nz**: nounal vector z

Note that this list is not exclusive nor a requirement; the required G-buffer set depends on the required enhancement techniques.

One of the significant advantages of preserving only geometric information in a G-buffer set is that the rendering processes can be separated into the following three groups.

- *geometric processes*:

  processes based on geometric factors such as object shapes and camera

parameters; (*ex.* perspective projection, hidden surface removal)

- *physical processes*:
  processes based on physical (optical) factors such as reflectance, colors, textures; (*ex.* shading, texture mapping)
- *artificial processes*:
  processes based on psychological or artistic factors; (*ex.* enhancement)

G-buffers are formed during the geometric processes, and are used by the physical and artificial processes. When physical and/or artificial factors are changed, the new image can be recalculated without modifying existing G-buffers if the geometric factors are fixed. Since physical and artificial processes can be applied independently, we can rapidly examine various combinations. Post-processing is performed as a combination of image processing operations. Since they are uniform operations for 2-D arrays, special hardware or vector processors can effectively accelerate the calculations.

In this chapter, intermediate data which contains the scalar value of each pixel is called an '*image*'. A G-buffer is also called an image; one example is the '**sz** *image*'.

5.3 Basic Enhancement Operations

In this section, basic enhancement operations - discontinuities, edges, contour lines, and curved hatching - are described. Though all of them are line drawings, they are realized with 2-D image processing operations instead of line tracking.

5.3.1 Drawing Discontinuities

Discontinuities of an image can be extracted with a first order differential operator. Various operators developed in the image processing field [Rosenfeld82] are available, however, we recommend Sobel's:

$$g = (|A + 2B + C - F - 2G - H| + |C + 2E + H - A - 2D - F|) / 8, \quad (5.1)$$

where *A-H* and *X* are values of the neighboring pixels in Fig.5.1. In *Eq.*(5.1), g is normalized so that it corresponds to the gradient per pixel.

Discontinuities of the first order differentials of an image can be extracted with a second order differential operator. For this calculation, we recommend the following operator:

$$l = (8X - A - B - C - D - E - F - G - H) / 3. \tag{5.2}$$

Discontinuities can be extracted as sequences of peak levels by a differential operator, however, they are not suitable for comprehensible rendering. Using a differential operator only, it is impossible to draw discontinuities as uniform lines because of the following artifacts:

(1) it is hard to distinguish discontinuities from large continuous changes;

(2) darkness of extracted lines depends on the degree of gaps.

Furthermore, second order differential operators have one more artifact:

(3) 0-th order discontinuities are extracted as double (negative and positive) lines.

These undesirable artifacts can be corrected using the minimum and maximum of neighboring differential values. An example of the normalization operator is as follows:

$$p = \begin{cases} \dfrac{(g_{min} - g)}{(g_{max} - g_{min})} & (g_{max} - g_{min} > k_g) \\ \dfrac{(g_{min} - g)}{k_g} & (g_{max} - g_{min} <= k_g), \end{cases} \tag{5.3}$$

where g is the gradient value of a pixel, g_{max} and g_{min} are the maximum and minimum gradient values in the 3×3 neighboring pixels, and p is the normalized value. The constant k_g distinguishes discontinuities from continuous changes; its value depends on

the object. *Equation* (5.3) can almost correct artifacts (1) and (2) of gradient images. For discontinuities of first order differential, *Eq.*(5.3) can be applied to second order differential images with simple modifications. For artifact (3), the following operation can be applied :

$$e = \begin{cases} l & (g_{max} <= k_l) \\ \dfrac{l}{(g_{max}/k_l)^2} & (g_{max} > k_l), \end{cases} \quad .. \quad (5.4)$$

where l is the second order differential value of a pixel, and e is the corrected value. The constant k_l is the limit of gradient for the elimination of 0-th order discontinuities.

A	B	C
D	X	E
F	G	H

Fig.5.1 Neighboring pixels.

5.3.2 Drawing Edges

The most significant application of drawing discontinuity is edge drawing. Here, edge has the two following meanings:

- *profile* - the border line of an object on the screen;
- *internal edge* - a line where two faces meet.

Profiles and edges are the 0-th and first order discontinuities of the depth image (**sz** *image*) respectively, thus the operations described in Subsection **5.3.1** can be simply applied. Edges can be drawn stably with 2-D image processing operators even for complicated free-form surfaces.

When an image is synthesized by perspective projection, the projection must be performed to depth values. In this case, the general relation between the depth in the eye coordinate zv and that in the screen coordinate **zs** (perspective depth) is as follows [Newman79]:

$$z_s = \alpha + \frac{\beta}{z_v}.$$

(5.5)

With *Eq.*(5.5), linearity of depth values on the screen is ensured. However, we recommend the following equation:

$$z_s = \frac{d^2}{w z_v},$$

.. (5.6)

where d is the distance between the view point and the screen, and w is one pixel length on the screen in eye coordinate (Fig.5.2). The advantage of *Eq.*(5.6) is that equalizes the gradient value of depth image with the slope of the surface.

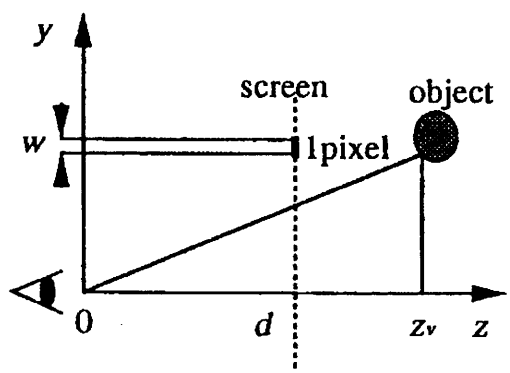
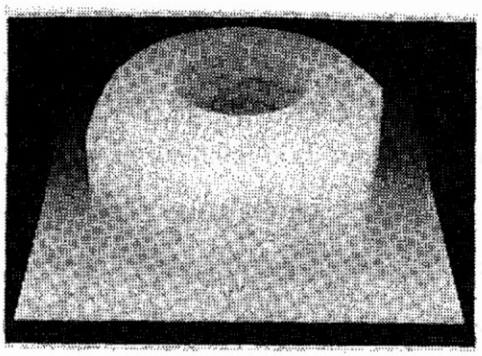


Fig.5.2 Perspective depth.

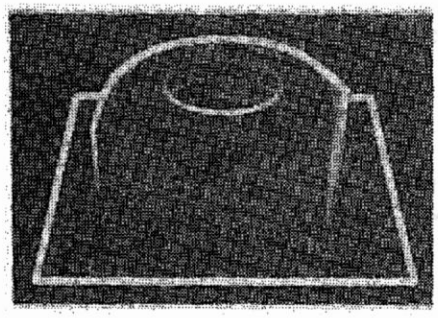
An example is shown in Fig.5.3. The depth image of a machine nut, its first and second order differential images, and corrected profile and internal edge images are presented. The normalization of the profile image was performed by using *Eq.*(5.3) with k_g = 10. The correction of the internal edge was realized by using *Eq.*(5.4) with k_l = 2. However, the artifacts (1) and (2) in Subsection **5.3.1** are not normalized for the internal edge image; the sign of an internal edge indicates its convexity, and the strength corresponds to its sharpness.

Note that edges can also be extracted by using the object/patch identifier (**id** *image*). This method is simple, but not complete for concave curved surfaces. To draw edges

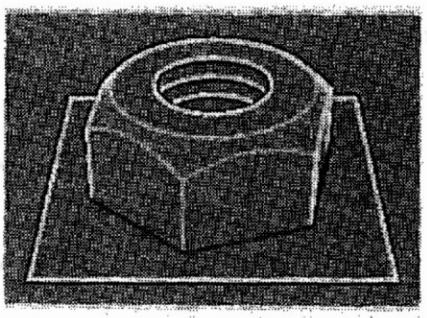
exactly, it is possible to combine the two methods.



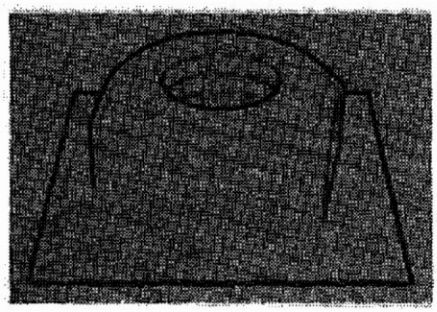
depth image



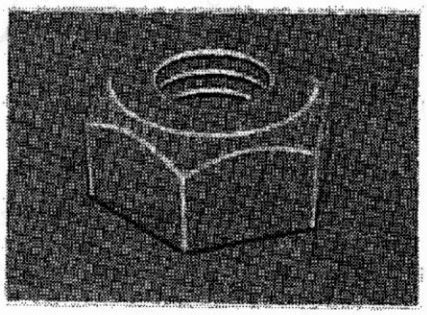
1st order differential



2st order differential



profile image



internal edge image

Fig.5.3 An example of edge drawing.

5.3.3 Drawing Contour Lines

This subsection proposes a new algorithm for drawing contour lines by using image processing techniques. Contour lines are usually drawn by tracking [Beck86], [Dickinson89], however, it is difficult to find starting points for all contours and to reliably trace the contour when the scalar field has singular points. The proposed method generates contour lines as raster data; both input and output data are images and the process consists of homogeneous calculations on each pixel and its neighbors. The algorithm is robust even for irregular or complex scalar fields, and can draw anti-aliased smooth lines easily. Furthermore, the thinning of condensed contour lines is also possible.

Assume that only the contour lines of value p are required. Let s be the value of a pixel of input image, g be the gradient value at the pixel, and d be the contour width in pixels. As the simplest method, each pixel value c of the output contour image can be obtained as follows:

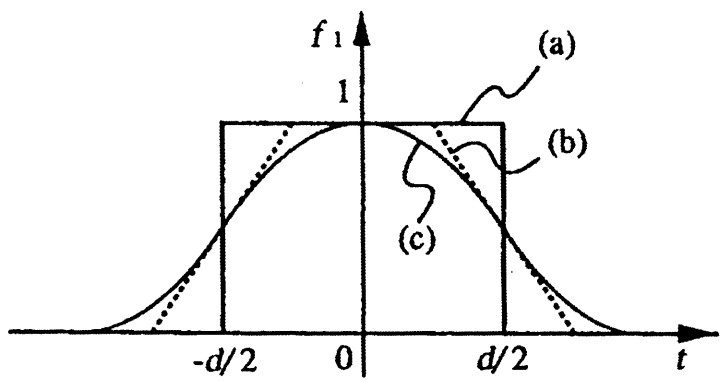
$$c = c_b + f_1\left(\frac{|s-p|}{g}\right)\cdot(c_c - c_b), \qquad .. \quad (5.7)$$

$$c = c_b + f_1\left(\frac{|s-p|}{g}\right)\cdot(c_c - c_b), \qquad .. \quad (5.8)$$

where c_c and c_b are the densities of contour lines and background respectively. The function $f_1(t)$ defines the density change of contours. The gradient value g can be obtained by first order differential operators. The following operator is recommended rather than the Sobel Operator (*Eq.*(5.1)):

$$\begin{aligned} g = (\ & |A-X| + 2|B-X| + |C-X| + 2|D-X| \\ & + 2|E-X| + |F-X| + 2|G-X| + |H-X|) / 8, \end{aligned} \qquad (5.9)$$

where *A-H*, *X* are neighboring pixel values in Fig.5.1. The method using *Eq.*(5.7),(5.8) leads to excessive aliasing. By using a linear (Fig.5.4(b)) or a higher order function (c) for $f_1(t)$ instead of a bilevel function (*Eq.*(5.8), Fig.5.4(a)), aliasing artifacts can be reduced.



(a) bilevel function

(b) linear function

(c) higher order function

Fig.5.4 Color change of a contour line.

It is generally difficult to draw accurate contour lines in flat regions where $p \sim s$ and $g \sim 0$, because the contours in such regions are basically unsteady due to noise if the scalar field is given by measured data. For the proposed algorithms it is necessary to add an exceptional process when $g = 0$. One solution is to let c be close to a constant value (such as c_b) if g is less than a threshold value g_ϵ:

$$c = c_b + f_g\left(f_1\left(\frac{|s-p|}{g}\right), g\right) \cdot c_c, \qquad .. \quad (5.10)$$

$$f_g(c,g) = \begin{cases} \frac{g}{g_\epsilon} c + \left(1 - \frac{g}{g_\epsilon}\right) c_{sparse} & (0 \le g < g_\epsilon) \\ c & (g_\epsilon \le g), \end{cases} \qquad .. \quad (5.11)$$

where the function f_g modifies the contour density when the gradient g is too small. The

constant c_{sparse} corresponds to the contour density of flat regions; if c_{sparse} = 0, the density is the background color c_b. Using this process, the contour lines sometimes disappear locally or look noisy in flat regions. Even in such cases, however, these algorithms never fall into endless loops nor miss important contours completely.

Contour lines with regular intervals can be drawn by applying *Eq.*(5.7) to each nominated value. Assume that the scalar values p_n are nominated at an interval of q:

$$p_n = p_0 + nq,$$

.. (5.12)

then pixel densities are given as follows:

$$c = c_b + f_g\left(f_1\left(\frac{|s - p_k|}{g}\right), g\right) \cdot c_c,$$

.. (5.13)

where

$$k = \left\lfloor \frac{s - p_0}{q} + \frac{1}{2} \right\rfloor .$$

.. (5.14)

If the gradient is large enough, the region is filled with the contour density using *Eq.*(5.13),(5.14). However, it is desirable to change the density of concentrated regions (application examples are shown in Subsection **5.4.3**). By modifying the function f_g into *Eq.*(5.15), any density c_{dense} can be selected:

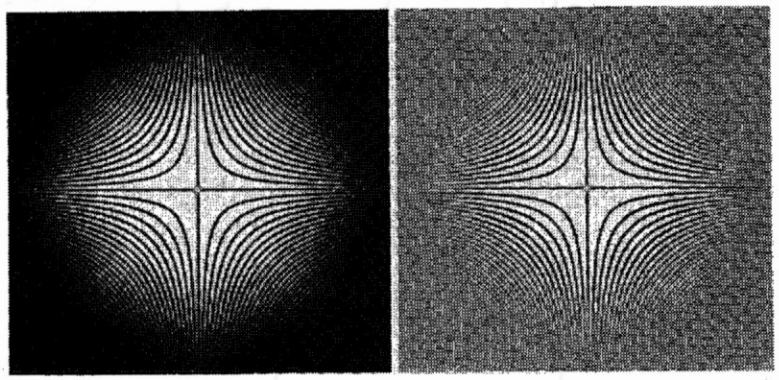
$$f_g(c, g) = \begin{cases} \frac{g}{g_\epsilon} c + \left(1 - \frac{g}{g_\epsilon}\right) c_{sparse} & (0 \le g < g_\epsilon) \\ c & (g_\epsilon \le g < \frac{q}{d}) \\ \frac{q}{dg} c + \left(1 - \frac{q}{dg}\right) c_{dense} & (\frac{q}{d} \le g). \end{cases}$$

.. (5.15)

An example of drawing the contour lines of a scalar field:

$$s(x, y) = xy \tag{5.16}$$

is shown in Fig.5.5. This example uses *Eq.*(5.10) and (5.15). The gradient is zero at the center (0, 0), where the density c_{sparse} is white. For the region of large gradient, the density c_{dense} is black in (a), and gray in (b).



(a) (b)

Fig.5.5 An example of contour lines.

5.3.4 Curved Hatching

In this subsection, we propose a method to express hatching with curved lines that indicate some type of structure lines. Such lines include the latitudes and longitudes of a sphere or a rotated object, intersections by a set of parallel planes, and u-v mesh of a parametric surface. For the above examples, a set of structure lines can be defined as contour lines of a scalar field, so that they can be drawn with the method given in Subsection **5.3.3**. However, contour lines drawn at regular intervals become too dense or sparse depending on the gradient, and are not suitable for hatching. To uniformly hatch a surface, contour lines must be drawn at regular pixel intervals.

Contour lines with uniform density can be drawn by using the binary thinning out technique. When the gradient is large and the contour lines become dense, alternate contours are thinned out. If the contour lines become sparse, new contours are added between existing lines. One example of the binary thinning out algorithm is as follows:

$$c = c_b + f_d\left(\frac{s - kp_n}{g}\right) \cdot (c_c - c_b),$$

.. (5.17)

$$f_d(t) = f_1(t) + \left(\frac{p_n}{d_i g} - 1\right) \cdot f_1\left(\frac{p_n}{2g} - |t|\right),$$

.. (5.18)

where

$$p_n = 2^n p_d, \tag{5.19}$$

$$n = \left\lfloor \log_2 \frac{d_i p}{p_d} + 1 \right\rfloor,$$

.. (5.20)

$$k = \left\lfloor \frac{s}{p_n} + \frac{2}{1} \right\rfloor,$$

.. (5.21)

and p_d is the standard interval in the scalar field. The function f_d has two terms; the first term corresponds to the density of normal contour line, and the second term corresponds to thinned or added contour line between normal lines. With these functions, contour lines are approximately spaced at intervals of d_i on the screen. An example of *Eq.*(5.16) is shown in Fig.5.6.

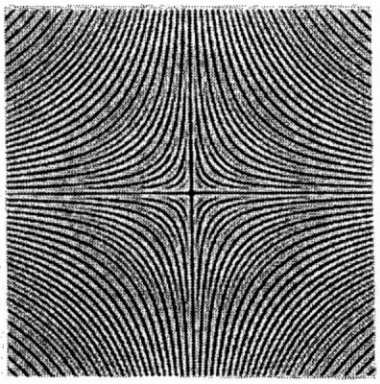


Fig.5.6 An example of curved hatching.

5.4 Examples and Applications

5.4.1 Edge Enhancement

A shaded image of 3-D shapes can be more comprehensible by enhancing edges (profiles and internal edges) with black or white lines. This technique is commonly used in hand drawn illustrations in the field of industrial design [Kondo88]. When the shaded image is generated with conventional computer graphics techniques, the edge drawing method in Subsections **5.3.1** and **5.3.2** is easily applied. This is because the depth image (the **sz** *image*) can be obtained as a by-product of hidden surface elimination. An enhanced image is generated by combining the edge image with the conventional shaded image. This technique is not the original usage of G-buffers; the shaded image and depth image are preserved as intermediate results, and both have geometric and physical factors. However, existing rendering software can be used with little modification, and the enhancement can be rapidly examined.

Two examples are shown in Fig.5.7. The original shaded image is (a), and the depth image is (b). Applying the differential operations on (b) (this process is shown in Fig.5.3), an edge image (c) was obtained. A final enhanced image (d) was generated by composing (b) and (c). In images (c) and (d), convex edges are drawn with white lines, which present the edge highlight effect [Kondo88], [Saito89]. Another example of enhancement is shown in images (c') and (d'); all profiles and edges are enhanced with

black lines. It was generated by taking absolute values of the internal edge image.

Edges in reflected or refracted objects can be also enhanced with the proposed method. For this purpose, a ray length image is used instead of a depth image. The ray length image contains the ray length form the eye to the last reflected (or refracted) object in each pixel, which can be obtained by ray-tracing. An example is shown in Fig.5.8. Note that this method is simple but not complete; a complete method is discussed in Subsection **5.5.1**.

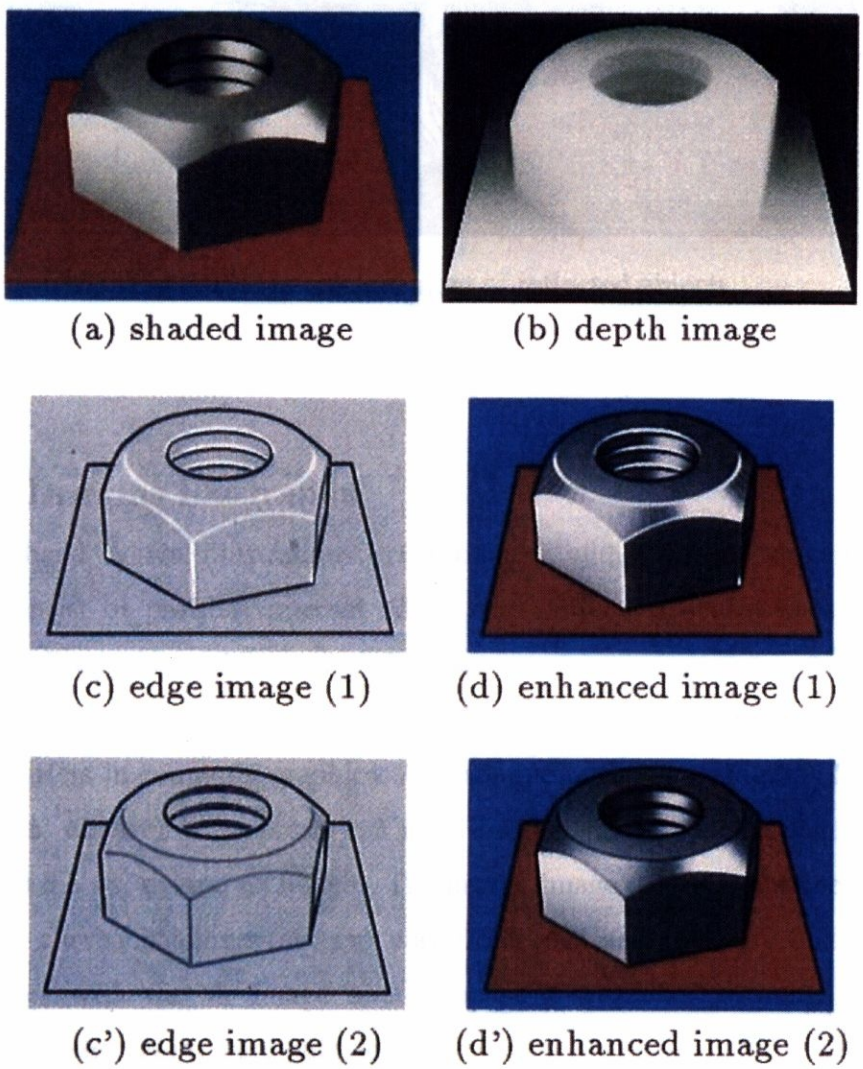


Fig.5.7 Two examples of edge enhancement.

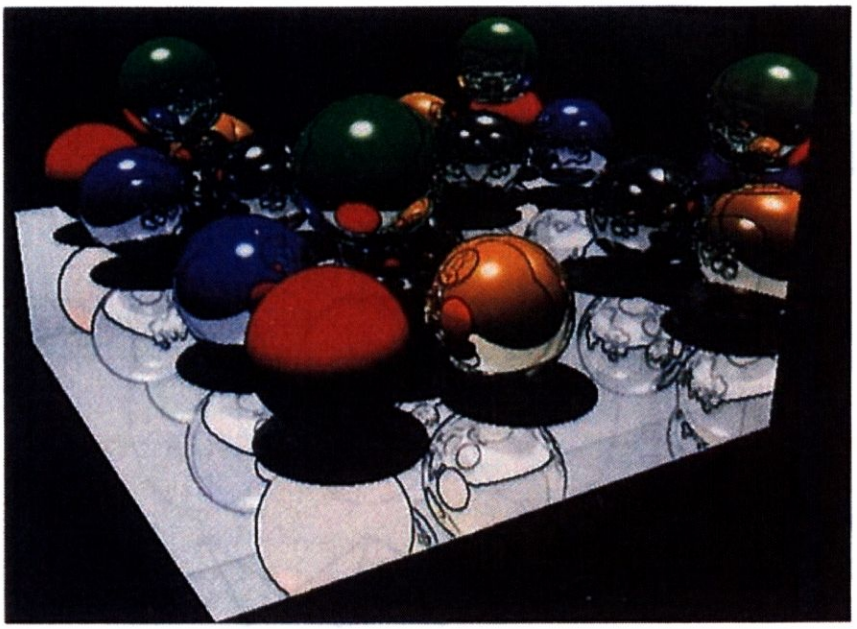


Fig.5.8 Edge enhancement of reflected objects.

5.4.2 Line Drawing Illustration

A lot of hand drawn illustrations are produced with just line drawings. Such illustrations consist of profiles, internal edges, and surface structure lines. Hatching techniques are effectively used instead of shading.

With the proposed method, these basic techniques can be examined quickly through the use of G-buffers in computer graphics. An example is shown in Fig.5.9. Six images (**nx**, **ny**, **nz**, **sz**, **ou**, **ov**) were preserved as G-buffers. The shaded image (**sh**) was calculated from the **nx**, **ny** and **nz** images. The profile image (**pr**) was obtained from the **sz** image. The curved hatching (**cu**, **cv**) was from (**ou**) and (**ov**). By enhancing the hatching images with the **sh** image and composing with the **pr** image, the final illustrations (**shu**, **shv**, **shuv**) were obtained.

Line drawing illustrations are easy to print or copy. No special treatment for gray scale is required, and even an inexpensive copy machine maintains the image quality. The quality of the copied **sh** image is completely poor, however, the copied **cv** image still has almost the same quality as the original.

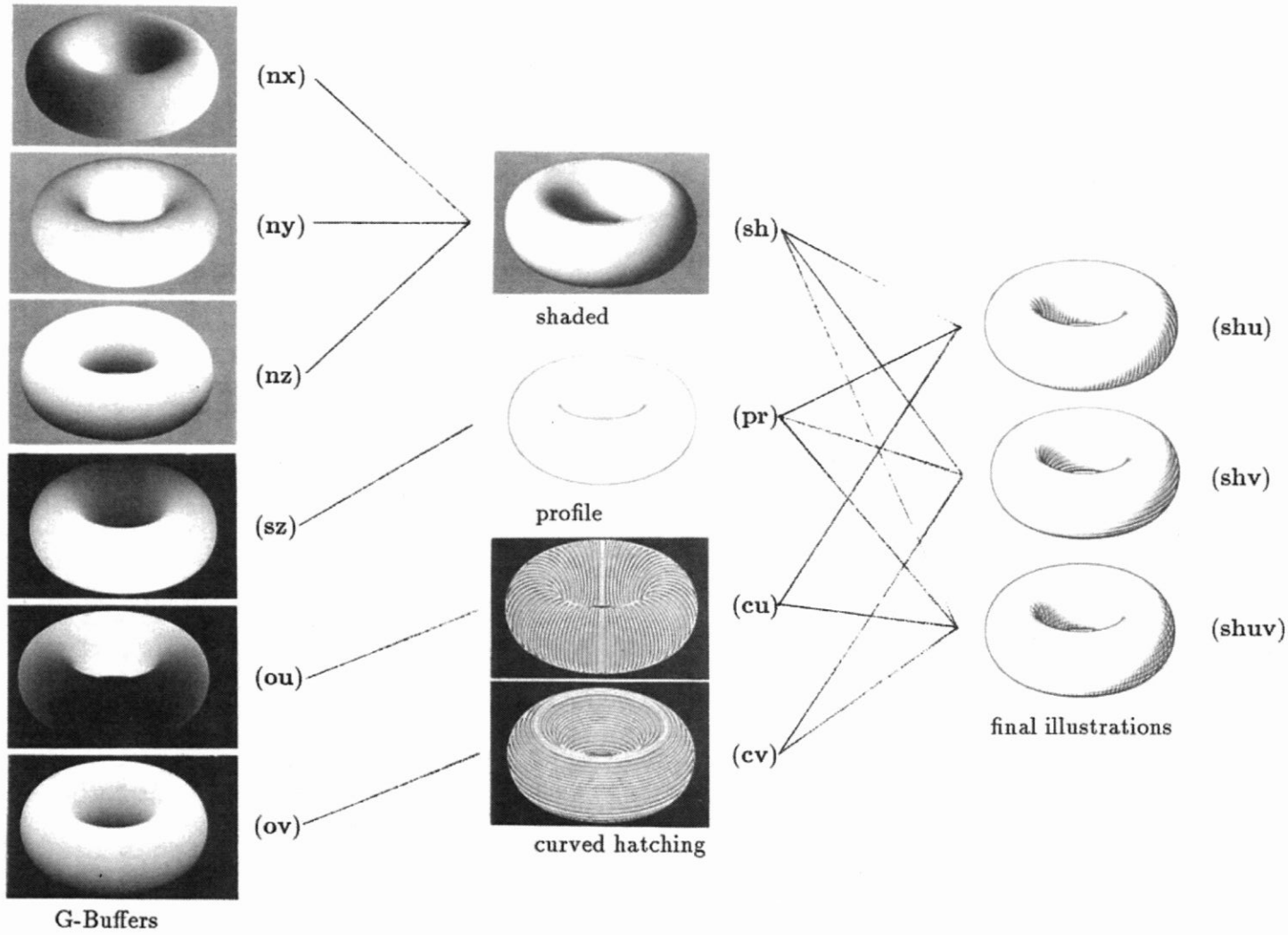


Fig.5.9 Process of drawing illustrations.

5.4.3 Topographical Maps

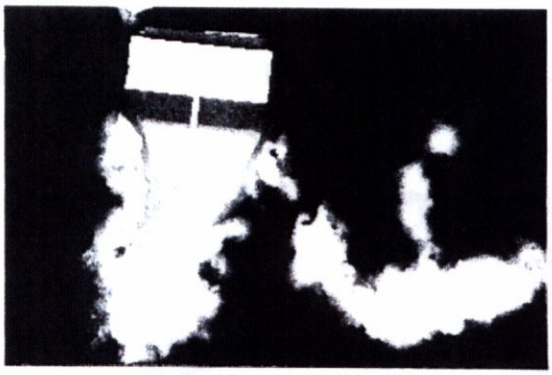
A topographical map can be drawn with a combination of multiple techniques in order to effectively visualize the height data. For example, the following basic techniques are very familiar:

- *contour lines* - Usually they are drawn at regular intervals. Often two contour thicknesses are used: thick contours for large intervals such as $100m$ and thin contours for small intervals such as $20m$.
- *color bands* - To present absolute height, several discrete color bands are used. Continuous color change is also available.
- *relief* - To visualize the direction of slopes, shading is applied.

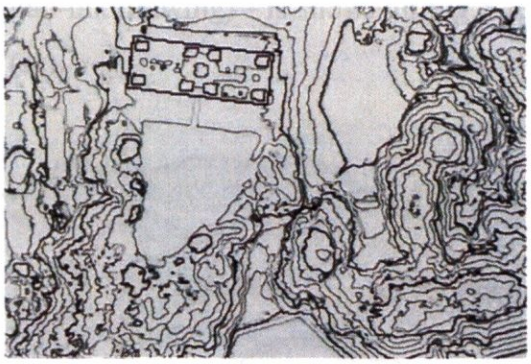
These techniques can be easily simulated with the proposed method if the height value is prepared for each pixel.

Figure 5.10 shows the process of making an enhancement map of the region around NTT's Yokosuka R&D Center. Image (**wz**) is the original height data. In the contour line image (**cn**), two contour thicknesses were used for $20m$ and $5m$ intervals. The constant c_{dense} of *Eq.*(5.15) was set to the line color (black) for $20m$ contours. On the other hand, it was set to background color (white) for $5m$ contours. With this technique, the combined contour image (**cn**) can present both large and small gradient regions. When the gradient becomes large and 5m contours become too dense, they are thinned out and only $20m$ contours are displayed. Image (**sh**) is the relief (shaded image), which was obtained from the gradient images of the (**wz**) image. Image (**mx3**) is the combination of (**cn**), (**sh**), and color bands.

The above processes for normal (cartesian) maps can be also applied to draw bird's-eye maps, which is but one advantage of our method. Example bird's-eye maps are shown in Fig.5.11, which shows the same region as in Fig.5.10. Four G-buffers (**sz**, **wx**, **wy**, **wz**) were generated from the original height data. The contour image (**cn**) and the relief image (**sh**) were obtained from (**wz**) and (**wx**, **wy**, **wz**) respectively. For bird's-eye maps, the profile image (**pr**) is also effective; it shows the shape of mountains clearly. In Fig.5.11, four images (**cn**, **pr**, **mx2**, **mx4**) are presented, where (**mx2**) is the combination of (**pr**) and (**sh**), and (**mx4**) is the combination of (**pr**), (**cn**), (**sh**), and color bands.

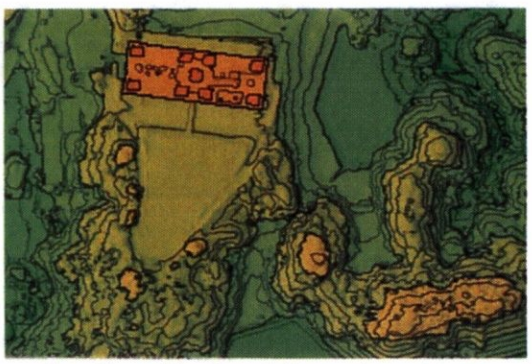


(wz) original height data



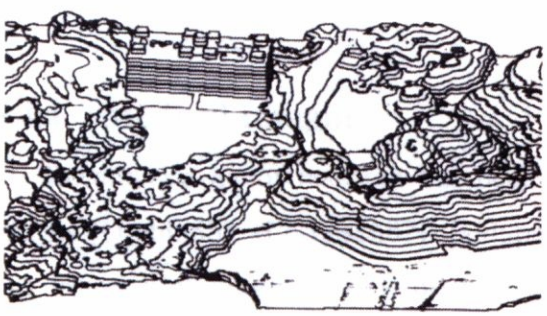
(cn) contour image

(sh) shaded image

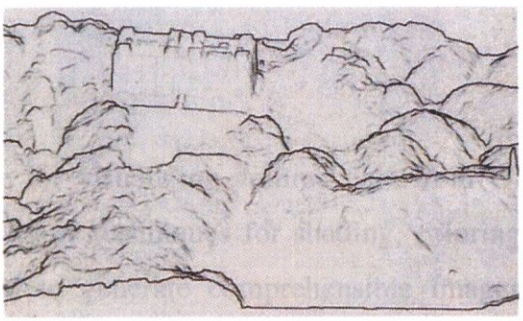


(mx3) combination of three enhanced images

Fig.5.10 Process of making a topographical map.

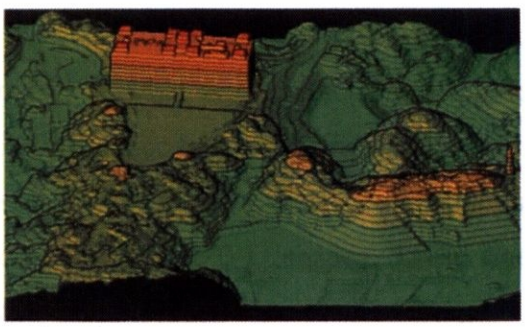


(cn) contour image



(pr) profile image

(mx2) combination of (pr) and (sh)



(mx4) combination of four enhanced images

Fig.5.11 Bird's eye maps.

An artistic example - Japanese sumi-e (Indian-ink drawing) - is shown in Fig.5.13. This picture was easily obtained from (**cn**) and (**sz**).

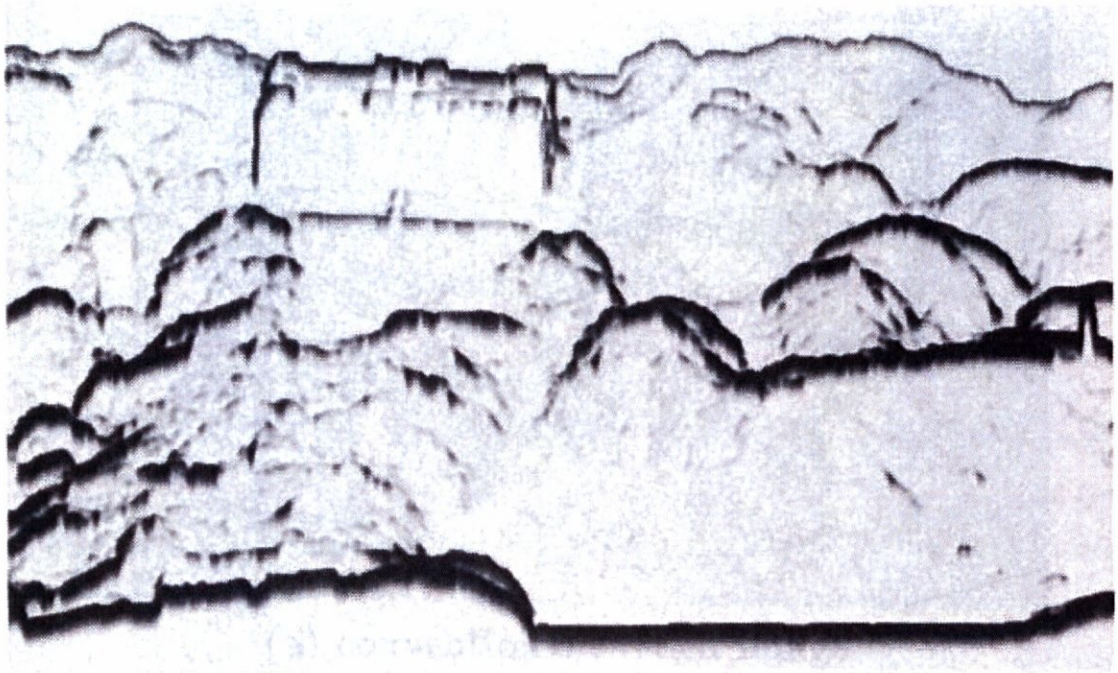
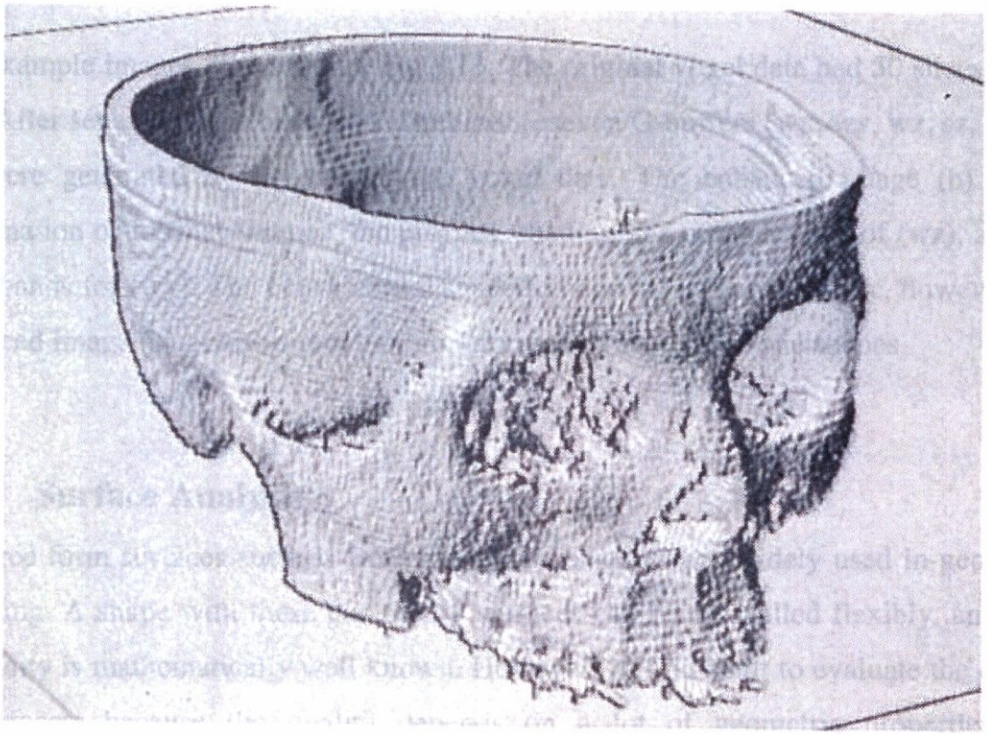


Fig.5.12 A sumi-e.

5.4.4 Medical Imaging

Recently, a lot of research has been done for visualizing volume data from CT images [Drebin88], [Lorensen87], [Udupa89]. Many techniques for shading, coloring, and transparent drawing have been developed to generate comprehensible images. These techniques are effective to present an overview of the data. However, medical doctors usually require the information about a more specific part; how the shape of the object has been deformed by the disease or injury, or what is the exact place of the diseased part. For this requirement, we can make the image more comprehensible with G-buffers and 2-D image processing techniques. It is easy to draw profiles and contour lines that show us some useful geometric information of the 3-D shapes. These line drawings can be combined interactively with a conventional shaded or colored image.



(a) conventional shaded image

(b) enhanced image using G-buffer method

Fig.5.13 Medical imaging from CT data.

Data courtesy of Dr. Jin Tamai, Department of Radiology, Nippon Medical School.

Example images are shown in Fig.5.13. The original voxel data had 50 slices of CT data. After separating the bone part [Drebin88], seven G-buffers (**wx**, **wy**, **wz**, **sz**, **nx**, **ny**, **nz**) were generated by ray-tracing the voxel data. The enhanced image (b) is the combination of four techniques: the profiles, shading, the contour lines of (**wz**), and the color bands for (**wy**). The conventional shaded image (a) is more realistic, however, the enhanced image (b) gives us much more information about the bone shapes.

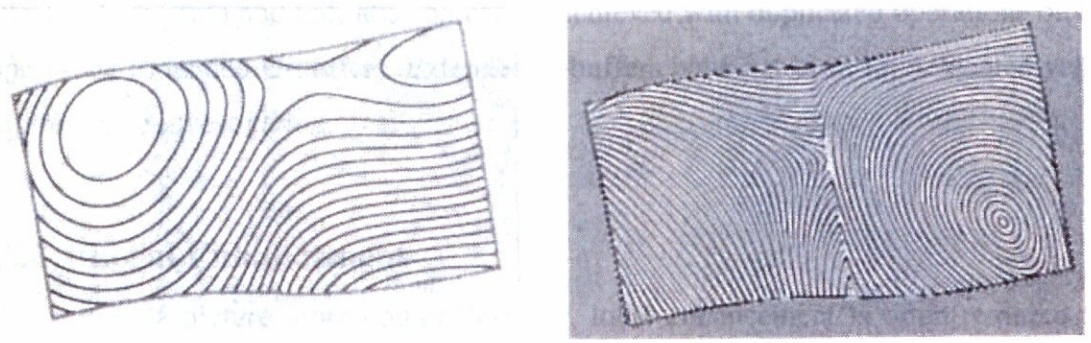
5.4.5 Surface Analysis

Free form surfaces such as Bezier or spline surfaces are widely used in geometric modeling. A shape with these parametric surfaces can be controlled flexibly, and their continuity is mathematically well known. However, it is difficult to evaluate the quality of surfaces; because the quality depends on a lot of geometric properties, and photo-realistic rendering is insufficient. For this purpose, it is important to visualize and analyze the geometric properties. For example, contour lines, pseudo-highlight patterns, and curvature maps are effective to describe the features of a curved surface [Beck86], [Dickinson89], [Higashi83], [Higashi90]. In conventional methods, line drawings are calculated by tracking, which requires a lot of consideration about numerical analysis.

Some of the surface analysis techniques are easily realized with our method. For example, a pseudo-highlight pattern can be obtained as follows. A pseudo-highlight pattern is the reflected image on a curved surface of parallel lines that are assumed to lie at an infinite distance [Higashi90]. Assume the cylindrical coordinate whose z-axis is parallel to the parallel lines. Then, each line has a constant e value. For each pixel, the θ value in the reflected image on the visible surface is easily calculated from the normal vector and the position of the surface, and the eye position. By drawing contour lines or curved hatching for the image of θ value, the pseudo-highlight pattern is generated.

In Fig.5.14, contour lines (a), and a pseudo-highlight pattern (b) of a curved surface are presented. The curved surface consists of two bicubic patches that connect with C^1 (not C^2) continuity. The overview of the shape is comprehensibly presented with the

contour lines. The discontinuity is clearly shown in the pseudo-highlight pattern.



(a) contour lines (b) pseudo-highlight pattern

Fig.5.14 An example of surface analysis.
Two bicubic patches are connected with C^1 contmurty

5.5 Discussions

5.5.1 Anti-aliasing and Reflective/Transparent Objects

A G-buffer contains the property of only one surface per pixel. This restriction leads the following problems:

- aliasing artifacts occur on surface borders;
- reflected or transparent images cannot be enhanced.

Some simple solutions are possible. For example, edges can be anti-aliased by calculating the **sz** image as the average depth value in each pixel. Edges in reflected images can be drawn with the method in Subsection **5.4.1**. However, these are not fundamental solutions.

These problems can be solved by preserving the properties of all surfaces visible in each pixel. This can be realized with Extended G-Buffers. In Extended G-buffers, each G-buffer has an extra memory area; the main area has the property of the primary visible surface at each pixel, and the extended area has the property of the other visible surfaces. A couple of additional G-buffers are preserved to retain pixel coverage information and the pointer to the next area for each pixel. This method can be

considered as an extension of the A-buffer method [Carpenter84], and reflection/refraction and anti-aliasing can be achieved with duplicated operations on the appropriate Extended G-buffers. Extended G-buffers have not been implemented yet; it requires more investigation.

5.5.2 Local Enhancement

To draw a picture more comprehensibly, local enhancement is often required for some specific regions. This can be provided with conventional 2-D paint systems. Various interactive paint systems have been developed and effectively used. A system with useful enhancement tools for technical illustration has also been developed [Kondo88]. Using such a paint system to enhance computer generated images, a designer can draw an image with any enhancement as he likes. However, it requires a great deal of effort to apply the same enhancement technique globally, *i.e.* apply it to a whole image, or a set of similar images such as an animation sequence. Furthermore, it is difficult to apply the enhancement uniformly.

Our method is mainly for global enhancement. However, it is also possible to realize local enhancement by applying operations only where some condition is satisfied. The object/patch identifier (the **id** image) can be effectively used for this condition.

5.5.3 Errors and Artifacts

To implement G-buffers, the data type for each property should be carefully considered. In our experimentations shown in Section **5.4**, all images including G-buffers are preserved as floating point data in order to avoid digitization errors. However, it is rather inefficient in both execution time and memory space. Though it is difficult to generally discuss the required precision of images, the following expectation is usually true. The required precision of an image depends on the subsequent operations. If the image is just used for linear operations, 1 byte integers are usually sufficient. Normal vectors (the **nx**, **ny**, **nz** images) are an example if they are used for the calculation of diffuse reflection only. On the other hand, if the image is used for a differential operation, 2 or 4 byte integers or floating point numbers are needed. Since

the process of drawing edges has differential operations, the **sz** image must have higher precision.

It is also necessary to maintain the precision in the geometric processes. If a G-buffer is generated with an approximation and is used for a differential operation, undesirable artifacts sometimes occur. Such artifacts are shown in Fig.5.3; thin lines shown on smooth curved surfaces are the border of tessellated polygon patches. Linear interpolation of normal vectors can make the shaded image smooth, however, the interpolation of depth values leads artifacts in the internal edge image.

5.6 Conclusion

We have proposed a new technique for rendering 3-D shapes comprehensibly. Enhancement techniques - drawing discontinuities, contour lines, and curved hatching - are developed with 2-D image processing operations, so that these line drawing algorithms can be easily combined with conventional surface rendering algorithms. By preserving geometric properties in G-buffers and visualizing the properties in post-processes, various combinations of enhancement techniques can be rapidly examined and a user can efficiently select the best enhancement technique. Furthermore, G-buffers are also useful for photo-realistic rendering. Example images of edge enhancement, line drawing illustrations, topographical maps, medical imaging, and surface analysis confirm that our method can be flexibly and efficiently applied in various fields.