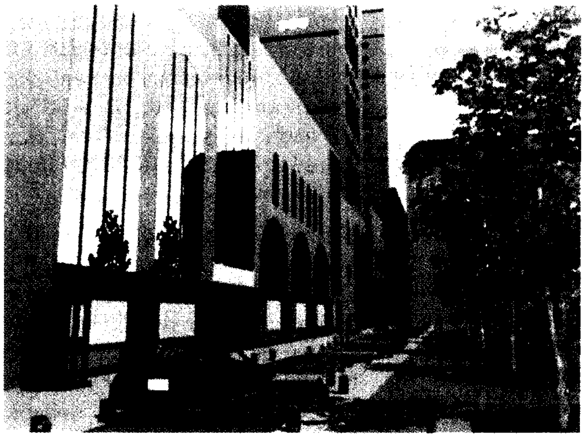
Compositing 3D Images with Antialiasing and Various Shading Effects



Eihachiro Nakamae
Hiroshima University

Takao Ishizaki
Daikin Industries, Ltd.

Tomoyuki Nishita
Fukuyama University

Shinichi Takita
Kagawa University

To make complex images look realistic, various types of geometric models and shading effects are needed. Programs capable of dealing with these are usually large and complex, and they are expensive to develop. This article proposes a method for compositing 3D images produced by different programs taking depth order into consideration. The method can add the following effects to composited images:

1. Antialiased images with scaling are displayed by a simple algorithm.
2. The algorithm can add shading effects due to various types of light, such as area sources and skylight.
3. Such shading effects as transparency and refraction are usually accomplished by ray tracing, but at the expense of enormous computation time. Our method allows ray tracing to be performed in localized regions, producing realistic results without the computational expense of ray tracing the whole image.

In addition to the above processes, such shading effects as fog and texture mapping can be processed with conventional methods. Thus it becomes possible to display complex scenes with various shading effects, using relatively small computers.

Advances in 3D rendering techniques have given rise to a variety of geometric models and shading models: polygonal objects, curved surfaces, and fractal faces, reflection, refraction, transparency, texture mapping, bump-mapping, fog effect, and various types of light sources. To process all these models in a single program is usually difficult. That is, the program becomes large and complex, it requires tremendous time to develop, and as the amount of data becomes large, it becomes troublesome to check. Subdividing the processing so that it is performed by a collection of separate programs can lead to a more flexible and more easily maintained system.

To address this problem, Whitted and Weimer[1] proposed a flexible scan-conversion processor that can simultaneously display several different surface types on

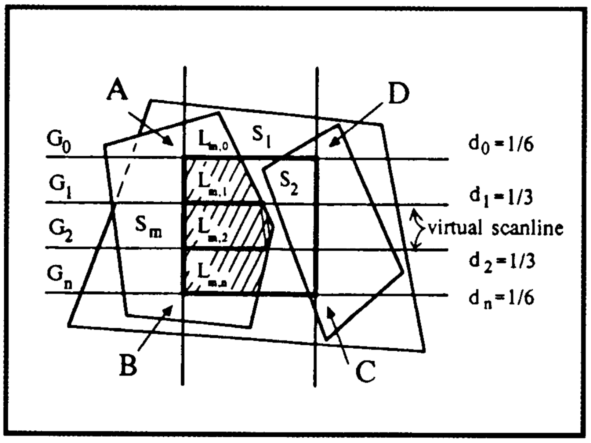


Figure 1. Density calculation for pixel ABDC, using the multiscanning method.

an image. This is designed to be a "test-bed" that can easily incorporate a variety of user-defined techniques. Data passed from the scan converter to the shader can be routed through a "span buffer." In the span buffer, all segment information (e.g., intersections between polygons and scanlines, and depths at each intersection) on every scanline is stored. Visible-surface processing and various shading effects are carried out on the data in the span buffer. This method can display high-quality images.

Crow[2] divided the display process into scene analysis and object rendering. A supervisory process determines the priority of objects, and the image is produced by overwriting objects into a frame buffer. This system can also merge different types of objects into an image.

Porter and Duff [3] proposed a compositing method adding a mixing factor (called an alpha-channel) to the color channels. This method can easily produce composite images with pixel-by-pixel antialiasing. However, it requires the priority of images to be manually entered. To overcome this drawback, Duff[4] added z-values to the four channels. However, the z-values apply only to the corner of each pixel. In addition, these two methods do not apply when the edges of multiple polygons project onto a single pixel.

For various types of texture mapping, Perlin developed the Pixel Stream Editor.[5] In this system the stored image contains the surface number, 3D coordinates, and surface normal at each pixel. In this way many types of texture mapping are easily processed by the PSE.

These developments indicate that the notion of separate modules is an excellent way to create complex scenes and handle many types of objects—composite images with antialiasing, map textures, fog effect, etc. A general-purpose processor should have the following capabilities in addition to those available in previously developed systems:

1. *Antialiased images with scaling:* Zooming is sometimes used in animation. In CAD systems, scaling of a generated image is useful for displaying details of shapes. For shape design, independent stretching in the vertical and horizontal directions is useful. Therefore, it is desirable not only to composite images but to be able to scale the composited images. Whitted's method displays high-quality images by using an antialiasing tiler such as Catmull's,[6] but the antialiasing process is complicated. Methods using alpha-channel are simple because of the pixel-by-pixel nature of the processing. However, none of these methods can composite images with scaling. In this article we perform antialiasing based on the multiscanning method,[7] with visible faces stored for each sub-scanline. That is, sufficient data is stored to display antialiased images with arbitrary scaling factors.
2. *Variety of light sources:* For lighting design, it is necessary to display many types of light sources in an image. The shading effect due to skylight is useful for realistic images in building design. In animation systems, light sources are changed or moved frequently. Our method can add these various shading effects to composited images.
3. *Local processing:* The ray-tracing algorithm is useful for treating transparency and refraction, but it is computationally very expensive. For composite images, our method can perform ray tracing on restricted local areas of the image, tremendously reducing computation time.

Compositing method with antialiasing

The method we propose can composite any number of images with hidden-surface removal, and display the composited images with antialiasing.

The images can also be scaled up or down with antialiasing. In this section we describe an antialiasing method and a data structure for processing antialiasing.

Antialiasing using a multiscanning method

There are useful antialiasing methods, such as increasing sampling density[8] and the area sampling methods accounting for the contribution of the visible portion of each face.[6] We use here the multiscanning method we developed earlier.[7] Below we briefly describe this area-sampling method.

For antialiasing, the intensity of each pixel is determined by the area occupied and intensity of image polygons projected onto the pixel. As shown in Figure 1, the area of a pixel occupied by a polygon (i.e., S_m) is approximated by trapezoidal integration as follows: We assume that a scanline is divided into n virtual scanlines, called sub-scanlines. When the length of the intersection segment between sub-scanline G_j and face S_m is $L_{m,j}$ ($j=0,1,2,\ldots,n$), the area of face S_m (the hatched area in Figure 1), A_m is approximated by

$$A_m = \frac{(L_{m,0} + L_{m,n})}{2n} + \frac{1}{n}\sum_{j=1}^{n-1} L_{m,j} \tag{1}$$

The intensity of the pixel is determined by the product of this area and the intensity of face S_m. In many cases multiple polygons will be projected onto a pixel. The intensity of pixel i, C_i, is obtained by

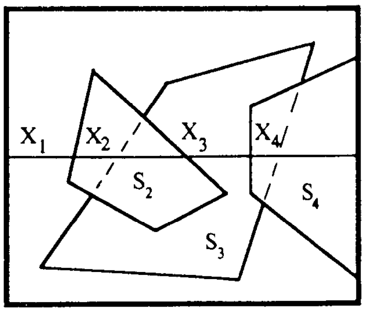
$$C_i = \sum_{j=0}^{n} d_j \cdot C'_{i,j} \tag{2}$$

where

$$C'_{i,j} = \sum_{f=1}^{m} L_{f,j} \cdot I_f$$

$$d_j = \begin{cases} \frac{1}{2n} & (j=0,n) \\ \frac{1}{n} & (0<j<n) \end{cases}$$

where d_j is a weighting factor (see Figure 1) and I_f is the intensity of face S_f ($f=1,2,\ldots,m$: m=number of faces). This equation indicates that the intensity of a pixel is achieved by summing the intensities of the sub-scanlines with weighting factors. Some sub-scanlines may contribute to two adjacent actual scanlines.



a visible segments on a virtual scanline

intersection point	face No.	intensity	depth	slope of depth
X_1	0	0	0	0
X_2	S_2	I_2	Z_2	ΔZ_2
X_3	S_3	I_3	Z_3	ΔZ_3
X_4	S_4	I_4	Z_4	ΔZ_4

b span buffer

Figure 2. Image data structure.

Image data structure

The visible segments on each scanline are stored so you can apply the antialiasing method discussed above. The depth of each visible segment is also stored for use in compositing images (see the later section on compositing 3D images). After hidden-surface removal, the following are stored for each sub-scanline:

1. The intersections between visible faces and sub-scanlines.
2. The face number and intensity at each intersection.
3. The depth at each intersection and its slope (increments per pixel).

Also, for the entire image

4. A color table for faces.
5. The coordinates of vertices of faces.

The intensity stored (see item 2 above) is due to a parallel light source, because this method produces a uniform intensity on each face and we can avoid storing data for each pixel. At the intersections between shadows and sub-scanlines the intensities change, so these intersections are also stored. For other types of light sources, the shading effects are calculated when the images are composited (see the section on shading effects due to various types of light sources), using the 3D vertices of faces. In this article curved surfaces are represented by triangle patches, and the normal of the surface at each vertex of each triangle patch is also stored.

Figure 2 illustrates the method used to store visible segments. The memory requirement is proportional to the number of intersections between faces and scanlines; i.e., it depends on image complexity. This method is equivalent to compressing data by run-length code, and it is much more efficient in use of memory than storing data for each pixel.

Regeneration of images

In this section we discuss compositing of images. We assume that the process is done from top to bottom. First, the sub-scanline number j is set to 0, C_i and C'_i

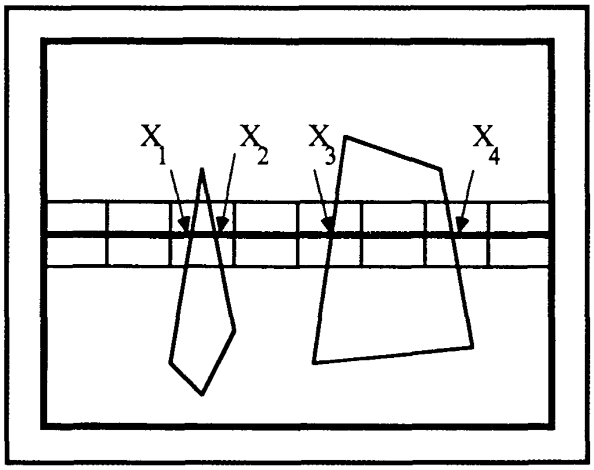


Figure 3. Intensity calculation using antialiasing.

Definition of symbols

x_k:	coordinate of the kth intersection on a sub-scanline ($k=1,2,\ldots,m$: m = number of intersections)
j:	sub-scanline number ($j=0,1,\ldots,n$: n = number of virtual scanlines per pixel)
C_i:	intensity of the ith pixel on a scanline ($i=1,2,\ldots,N$: N = the number of pixels on a scanline)
C'_i:	intensity of the ith pixel on a sub-scanline (see Equation 2)
d_j:	weighting factor for the jth sub-scanline
C_k:	intensity of the face containing x_k

($i=1,2,\ldots,N$) are initialized, and the following process is done for each scanline (the symbol "[ ]" indicates truncation):

1. Read intersections x_k ($k=1,2,\ldots,m$).
2. Execute the following process for each span (x_k,x_{k+1}).

```
if $[x_k]=[x_{k+1}]$ then
    $C'_i=C'_i + (x_{k+1} - x_k)C_k$        $(i=[x_k])$                 (3)
  else
$C'_i=C'_i + ([x_k]+1 - x_k)C_k$           $(i=[x_k])$                 (4)
$C'_i=C'_i + C_k$                          $([x_k] < i < [x_{k+1}])$   (5)
$C'_i=C'_i + (x_{k+1} - [x_{k+1}])C_k$     $(i=[x_{k+1}])$             (6)
    (if $[x_k]=[x_{k+1}]$ -1 then equation 5 is unnecessary)
```

3. $C_i=C_i+d_jC'_i$ $\qquad$ ($i=1,2,\ldots,N$).
4. If $j<n$ then $j=j+1$, return to step 1.
5. Output C_i to CRT.
6. $C_i=d_jC'_i, C'_i=0$ ($i=1,2,\ldots,N$), $j=1$ and return to step 1.

Step 2 can be explained as follows: If x_k and x_{k+1} lie in the same pixel (e.g., x_1,x_2 in Figure 3), $(x_{k+1}-x_k)$ is the weighting factor for intensity calculation. If not, for the pixel including x_k (e.g., x_3 in Figure 3) the weighting factor is $([x_k]+1-x_k)$, and for the pixel including x_{k+1} (e.g., x_4 in Figure 3) the weighting factor is $(x_{k+1}-[x_{k+1}])$. For other pixels the weighting factor is 1.

Even when multiple polygons project onto a pixel, the image can easily be reproduced using this algorithm.

Regeneration of images with scaling

Scaling is often used in animation and CAD systems. The scaling proposed here allows magnifying or reducing with antialiasing.

Scaling in the x direction is easily done by computing the product of T_x (the x direction scale) and the coordinates of intersections. However, for scaling in the y direction, the following computation is required: If an image is stored using n_0 sub-scanlines per pixel, it can be displayed at vertical scale T_y with n_0/T_y sub-scanlines per pixel. However, n, the number of sub-scanlines per pixel in the scaled image, must be an integer. We set

$$n = \left[\frac{n_0 I_y}{T_y}\right] - \left[\frac{n_0 I_y}{T_y} - \frac{n_0}{T_y}\right] \tag{7}$$

where I_y is the scanline number. For example, for $n_0=3$ and $T_y=1.2$, the numbers of sub-scanlines for each scanline are 2, 3, 2, 3,.... For $T_y=0.8$ they are 3, 4, 4, 3, 4, 4,.... For $n_0=3$, we can enlarge as far as triple size, with good quality until double size. We can select any large scaling factor when we use large enough n_0.

This method performs zooming and panning easily, without repeating hidden-surface removal, which makes it very useful for animation.

Compositing 3D images

When there are multiple stored images, these images can be combined with hidden-surface removal by using the z-values of each image. Image data (i.e., visible segments) are stored sub-scanline by sub-scanline, and hidden-surface removal can be done by a scanline algorithm such as Watkins's algorithm, which allows polygons to intersect (see Figure 4).

Consider a composite image c formed by two images a and b. We prepare span buffers A, B, and C for images a, b, and c, respectively. The process is carried out sub-scanline by sub-scanline, reading A and B, and writing C. An antialiased image is obtained by applying the process described in the section on antialiasing to the buffer C. If there is a third image, we can get the composite image by setting C to A and the third image to B, and repeating the above process. By using these three buffers cyclically, multiple images can be easily composited.

Figure 4 shows two images, a and b, and the composite image c. In Figure 5 a, b, and c show a representative scanline with perspective depth for each face for the same images. (In this article, perspective depth decreases with distance from the viewpoint). We assume that the background (for spans where there is no visible segment) consists of a plane whose z is a large negative value. Using this assumption, each of the images a and b have exactly one segment in each sample span. Thus, combinations of these segments are limited to three cases as shown in Figure 6, allowing easy hidden-surface removal. In this example, the bold lines in Figure 6 are visible segments.

Because our hidden-surface removal method has time complexity proportional to the square of the number of faces, it is more efficient to operate on clusters of object data and composite the results.

Shading effects due to various types of light sources

In our system, many types of light sources—such as point sources, linear sources,[9] area sources, polyhedral sources,[10] and skylight[11]—are available in the shading processor.

The system allows the light sources (including lighting directions, types of light sources, luminous intensity, and colors) to be freely changed at the display stage, while the viewpoint is fixed. Intensities due to a parallel light source are stored and may be immediately displayed. When the lighting conditions are changed, intensities must be recalculated, but not hidden surfaces. In many cases a broad range of shading effects, such as linear or area light sources, are required, particularly for lighting design.

In the proposed method, the composited image obtained as described in the section headed "Compositing method with antialiasing" contains information about visible faces and their depths. Therefore, various shading effects can easily be added. Fog effect can be calculated using only depths. For other shading effects 3D coordinates at each pixel and information about all polyhedrons are required, but again these can easily be

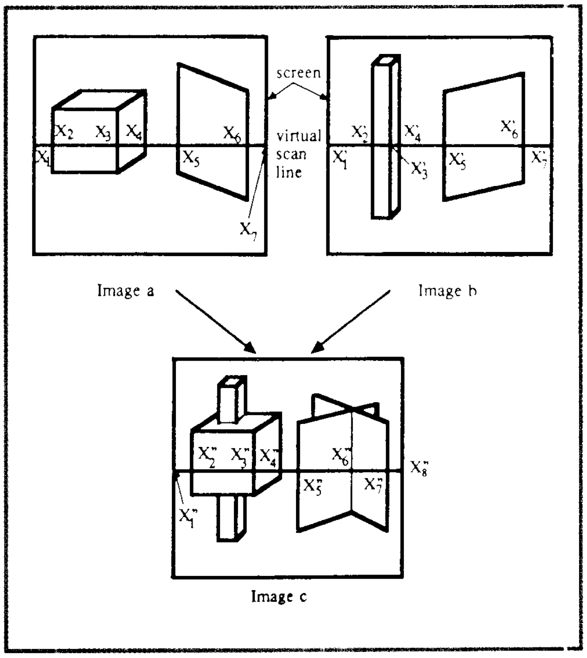

screen
virtual scan line
Image a
Image b
Image c
X_1 X_2 X_3 X_4 X_5 X_6 X_7
X'_1 X'_2 X'_3 X'_4 X'_5 X'_6 X'_7
X''_1 X''_2 X''_3 X''_4 X''_5 X''_6 X''_7 X''_8

Figure 4. Composition of two images.

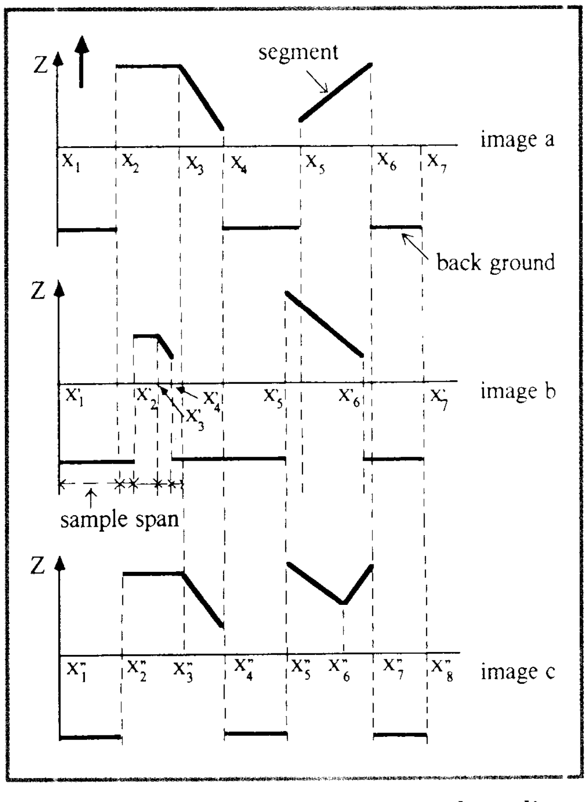

segment
image a
back ground
image b
sample span
image c
Z
X_1 X_2 X_3 X_4 X_5 X_6 X_7
X'_1 X'_2 X'_3 X'_4 X'_5 X'_6 X'_7
X''_1 X''_2 X''_3 X''_4 X''_5 X''_6 X''_7 X''_8

Figure 5. Depths of segments on a virtual scanline.

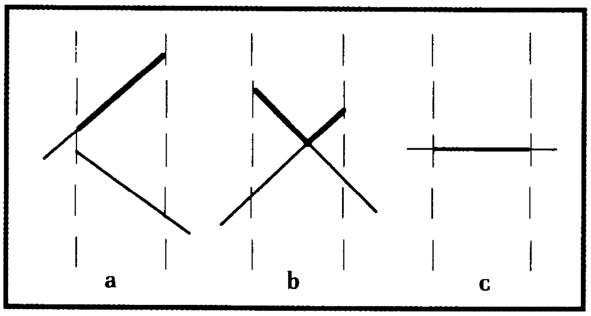


Figure 6. Visibility test in sample span, showing (a) not crossing, (b) crossing, and (c) overlap.

applied using the depth of visible segments and screen coordinates.

Illuminance calculation

Consider a scene that contains many types of light sources. The calculation for each type of light source is usually different. If a single program includes all types of light sources, it becomes unmanageably large. Intensity at each pixel can be achieved by summing the intensity of each light source. Using the same composited image data repeatedly, we can get shading effects by independently calculating each light source. In this method, the visible faces are already known, so the shading effects due to multiple light sources are achieved with only one execution of hidden-surface removal.

By subdividing the program into hidden-surface remover and shader, various shading effects can easily be added.

Shadow processing

In the previous compositing methods only Whitted and Duff calculated shadows.[1,4] They used Williams's method,[12] a z-buffer algorithm with the viewpoint at the light source. In this method, light sources are limited to a parallel source or a point source, and the position of the point source must be outside of the field of view.

For arbitrary light-source position or various types of light sources, using shadow volumes is helpful, and adding precalculation of shadow boundaries on visible faces is more useful. In these processes, however, the memory requirement for shadows becomes enormous when the number of objects is large. To overcome this problem, objects are divided into clusters, and the process is done cluster by cluster. For each cluster, shadows can be detected by using only those shadow volumes casting shadows on the visible faces of this cluster. For example, consider shadow processing after composition of images a and b. The visible faces for each image are obtained using the method described in the section on compositing 3D images. Then shadow detection is executed first for the visible faces of a only, then for those of b, resulting in reduced memory requirements and increased speed.

Local processing

Because the visible faces are available, the following refinements can be calculated for selected faces. After displaying a rough image on a CRT, the image is improved by performing these processes in restricted regions. Antialiasing against a background can easily be done by blending pixels from an image displayed on the CRT and the colors arrived at by the method described in the section headed "Regeneration of images."

Local ray tracing

Ray tracing is commonly used to represent transparency and refraction, but this method spends tremendous time detecting intersections between the ray and objects. If only a few objects need to be processed by ray tracing, ray tracing the entire screen is inefficient. To avoid this problem, a method using a z-buffer algorithm as a visible-surface preprocessor has been developed.[13] However, using this method, it is difficult to memorize areas of polygons within a pixel. Our method stores all visible faces within all pixels, so it can be used as a preprocessor for ray tracing. In this article we use a hidden-surface removal method combining scanline processing and ray tracing, which we developed earlier.[14] A reserved face number is used to indicate that a face is to be ray traced. When the scan converter encounters such a face, it invokes the local ray tracer for each pixel covered by the face. For efficient calculation the following are considered:

1. The shadow process is done by precalculation of shadow boundaries on faces, not pixel by pixel.
2. The environment is subdivided hierarchically; bounding boxes for each level of objects are used and a hierarchical description is also used for shadow volumes.
3. Curved surfaces are subdivided into triangle patches to apply the scanline algorithm.

Texture mapping

Various types of textures—such as color maps, bump maps, reflection maps, and transparency maps—can be added to the composited images. Texture mapping generally requires the entire texture map to be present in memory simultaneously. By appropriately decomposing into separate images, each of which requires only one or a small number of textures, and compositing later, the number of texture maps required to be simultaneously present in memory can be reduced, thus reducing the total memory requirement.

Furthermore, the proposed method can add such complex objects as trees and grass to the composited image with hidden-surface removal. In Cook's shade trees,[15] these objects are generated by another program and displayed by creating a texture mapped onto a transparent polygon (in this method, compositing is performed by the alpha-channel method[3]). However, modeling these objects is time consuming. To overcome this problem, we use an image of a tree extracted from a photograph and map it onto a transparent polygon. For presenting each tree, the transparent polygon is placed standing vertically with its normal facing the viewpoint. When these trees overlap on the screen, hidden-surface removal is done from the nearest transparent polygon by ray tracing.

Shadows and reflections of trees can also be processed. For shadow calculation, the transparent polygons are rotated to face the light source, and for reflection mapping, they are rotated to face the reflecting surface. These calculations are possible because the intermediate representation includes 3D information about visible faces and depths.

Background processing

Because the area where the background is visible is known, a background picture can be inserted with antialiasing. Similarly, clouds generated by a separate program can be inserted in the sky.

Example and conclusion

Figure 7 shows some examples which demonstrate the proposed system. Picture (a) shows an example of scaling (upper left is the original). We can manipulate height and width for shape design, executing visible surface processing only once.

Pictures (b) and (c) are examples of the use of image compositing as a tool in urban planning (Osaka, Japan). In these images, buildings, trees, and cars were produced separately, then composited. The reflections in windows are processed by local ray tracing. For moving objects, such as cars, the compositing is done by recalculating only moving elements.

In picture (b), the leaves of the artificial trees are expressed by a set of triangles, while in picture (c), textures extracted from an actual photograph are used, yielding a more realistic image. Note that the shadows and reflections of the textures of the real trees are also very realistic. This method can be applied in urban planning to simulate views including various types of real trees.

Pictures (d) and (e) are examples of estimation of environmental impact due to a new building. In this example, the building and the city background are composited. Part of the building was produced by a program handling curved surfaces, and other parts were produced by a program dealing only with polyhedrons.

In picture (d), fog effect is added by using stored depths. Picture (e) shows the same scene as (d) at night. The building is illuminated by five floodlamps at each corner where the actual luminous intensity characteristics of the planned lamps were used in the calculation.

Picture (f) is an example of fleecy clouds. In this picture the building and cars are composited. After composition, we can add texture mapping and shading effects. The shading effects under fleecy clouds are due to a combination of clear skylight (i.e., a hemispherical light source with large radius[11]) and weak direct sunlight (i.e., a parallel source). Note that penumbrae due to skylight are displayed.

Pictures (c) and (f) are stills from an animated film entitled "CG Town" (SIGGRAPH 87 film and video show). Frames of the animated film are not antialiased to save time in producing animation, so pictures (c) and (f) exhibit aliasing. In the film, the intensity of skylight varies from overcast sky to clear sky, cars move, and colors of leaves change. All these effects were calculated using the methods described in this article.

Picture (g) is an example of lighting design. The room is illuminated by two types of light sources, a polyhedral light source and the light from the window, an area source.

All examples were calculated using supersampling of three virtual scanlines per scanline. Computing statistics for (c), the most complex image among the examples, are as follows:

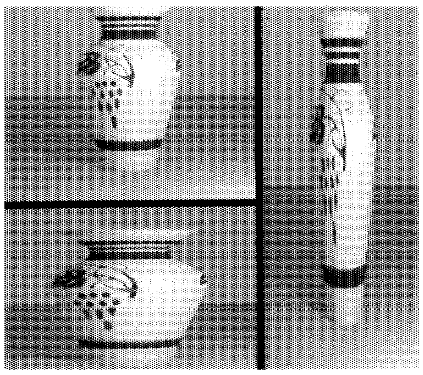
Computer:	Tosbac Data System 600/80 (2.8 MWIPS)
Image Complexity:	870 polygons
	2359 visible faces
Image calculated in four clusters:	buildings, roads, trees, cars
Size of Intermediate Representation:	353 kb per cluster, average
Compute 1 cluster:	2.8 mins
Final Composite:	26.8 mins (including ray tracing)

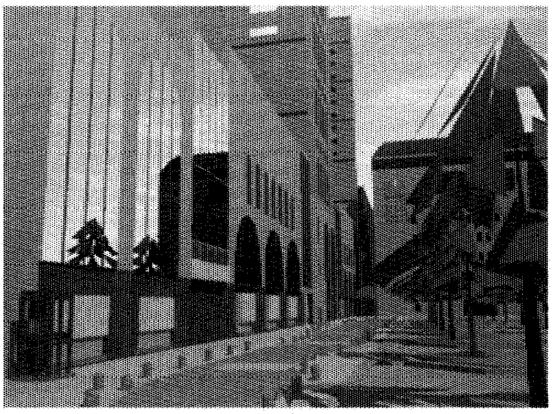
As shown in these examples, the ability to add various shading effects leads to heightened realism, as well as a more convenient framework for development . ■

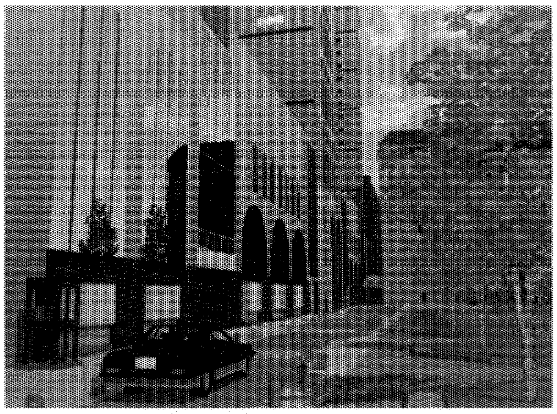
Acknowledgments

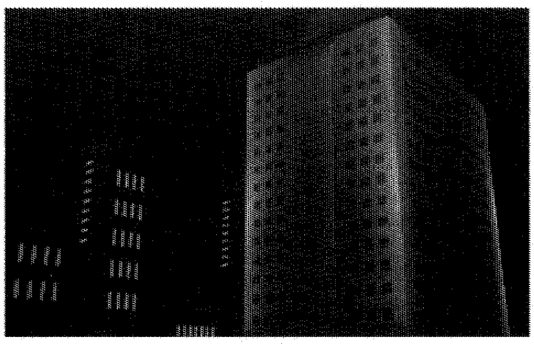
We thank Bonnie Sullivan for developing the program for extraction of trees from photographs, and for carefully reading and commenting on this article.

Figure 7. Examples.

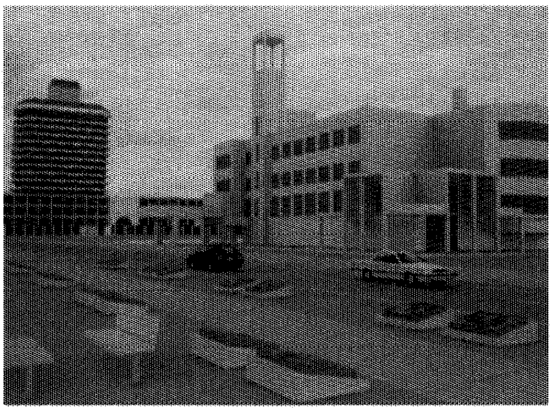

a

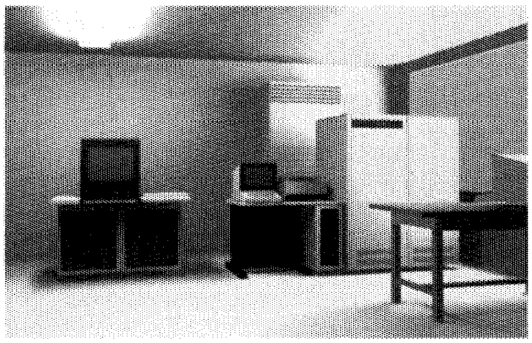

b


c


d

e


f


g

References

1. T. Whitted and D.M. Weimer, "A Software Test-Bed for the Development of 3D Raster Graphics Systems," *Computer Graphics* (Proc. SIGGRAPH), Vol. 15, No. 3, Aug. 1981, pp. 271-277.
2. F.C. Crow, "A More Flexible Image Generation Environment," *Computer Graphics* (Proc. SIGGRAPH), Vol. 16, No. 3, July 1982, pp. 9-18.
3. T. Porter and T. Duff, "Compositing Digital Images," *Computer Graphics* (Proc. SIGGRAPH), Vol. 18, No. 3, July 1984, pp. 253-259.
4. T. Duff, "Compositing 3D Rendered Images," *Computer Graphics* (Proc. SIGGRAPH), Vol. 19, No. 3, July 1985, pp. 41-44.
5. K. Perlin, "An Image Synthesizer," *Computer Graphics* (Proc. SIGGRAPH), Vol. 19, No. 3, July 1985, pp. 287-296.
6. E. Catmull, "A Hidden-Surface Algorithm with Antialiasing," *Computer Graphics* (Proc. SIGGRAPH), Vol. 12, No. 3, Aug. 1978, pp. 6-11.
7. T. Nishita and E. Nakamae, "Half-Tone Representation of 3D Objects with Smooth Edge by Using a Multi-Scanning Method," *J. Information Processing* (in Japanese), Vol. 25, No. 5, Sept. 1984, pp. 703-711.
8. F.C. Crow, "A Comparison of Antialiasing Techniques," *CG&A*, Vol. 1, No. 1, Jan. 1981, pp. 40-48.
9. T. Nishita, T. Okamura, and E. Nakamae, "Shading Models for Point and Linear Sources," *ACM Trans. Graphics*, Vol. 4, No. 2, Apr. 1985, pp. 124-146.
10. T. Nishita and E. Nakamae, "Half-Tone Representation of 3D Objects Illuminated by Area Sources or Polyhedron Sources," *IEEE Proc. COMPSAC 83*, (microfiche only), CS Press, Los Alamitos, Calif., 1983, pp. 237-242.
11. T. Nishita and E. Nakamae, "Continuous Tone Representation of Three-Dimensional Objects Illuminated by Skylight," *Computer Graphics* (Proc. SIGGRAPH), Vol. 20, No. 4, Aug. 1986, pp. 125-132.
12. L. Williams, "Casting Curved Shadows on Curved Surfaces," *Computer Graphics* (Proc. SIGGRAPH), Vol. 12, No. 3, Aug. 1978, pp. 270-274.
13. H. Weghorst, G. Hooper, and D.P. Greenberg, "Improved Computational Methods for Ray Tracing," *ACM Trans. Graphics*, Vol. 3, No. 1, Jan. 1984, pp. 52-69.
14. E. Nakamae, K. Tadamura, and T. Nishita, "Half-Tone Representation Using Local Ray Tracing," *J. Information Processing* (in Japanese), Vol. 27, No. 11, Nov. 1986, pp. 1077-1085.
15. R. Cook, "Shade Trees," *Computer Graphics* (Proc. SIGGRAPH), Vol. 18, No. 3, July 1984, pp. 223-231.

Eihachiro Nakamae is a professor at Hiroshima University where he was appointed as research associate in 1956 and professor in 1968. He was an associate researcher at Clarkson College of Technology, Potsdam, N.Y., from 1973-1974. His research interests include computer graphics and electric machinery.

Nakamae received the BE, ME, and DE degrees in 1954, 1956, and 1967 from Waseda University. He is a member of IEEE, IEEE Computer Society, IEE of Japan, IPS of Japan, and IECE of Japan.

Nakamae can be contacted at the Faculty of Engineering, Hiroshima University, Saijo-cho, Higashi-Hiroshima, 724 Japan.

Tomoyuki Nishita is an associate professor in the Department of Electronic and Electrical Engineering at Fukuyama University, Japan. He was on the research staff at Mazda from 1973 to 1979 and worked on design and development of computer-controlled vehicle systems. He joined Fukuyama University in 1979. Since April 1988 he has been a visiting professor and research associate in the Engineering Computer Graphics Laboratory at Brigham Young University. His research interests involve computer graphics, including lighting models, hidden-surface removal, and antialiasing.

Nishita received his BE, ME, and PhD in engineering in 1971, 1973, and 1985, respectively, from Hiroshima University. He is a member of ACM, IPS of Japan, and IEE of Japan.

Nishita can be contacted at the ECGL, Brigham Young University, 368 CB, Provo, Utah 84602, through the end of March 1989, and at the Faculty of Engineering, Fukuyama University, Sanzo Higashimura-cho, Fukuyama, 729-02 Japan from April on.

Takao Ishizaki is a system engineer in the Engineering and Development Department of the Electronics Division, Daikin Industries, Ltd. His research interests include computer graphics and its applications.

Ishizaki received the BS in electronics engineering and the MS in system engineering from Hiroshima University.

Ishizaki can be contacted at the Engineering and Development Department, Electronics Division, Daikin Industries, Ltd., Box 37, Shinjuku-Sumitomo-Bldg., 2-6-1, Nishi-Shinjuku, Shinjuku-ku, Tokyo, 163 Japan.

Shinichi Takita is a professor in the Department of Education at Kagawa University, Japan. His research interests include computer graphics and CAI.

Takita received his BE and ME degrees in electrical engineering from Hiroshima University in 1964 and 1966, respectively. He is a member of the IEE of Japan and the Japan Society of Industrial and Technical Education.

Takita can be contacted at the Faculty of Education, Kagawa University, 1-1, Saiwai-cho Takamatsu, 760 Japan.