Data visualization using a general-purpose renderer

by A. Doi
M. Aono
N. Urano
K. Sugimoto

This paper describes a general approach to data visualization, based on the Rendering Subroutine Package (RSP). RSP is a general-purpose polygon-based renderer, and is IBM's first rendering application programming interface (API) for users who wish to develop their own applications. We present an overview of the system, details of the image synthesis tools, and several examples of the application of RSP to architectural CAD, molecular graphics, and computer tomography.

Introduction

In recent years, CAD/CAM, computer graphics animation (CF, CM), and presentation graphics have become big business as a result of the progress of computer graphics technology. Data visualization of large numerical simulations has also become important for the understanding of simulation results. As a result, the user needs a variety of representations, flexible interactive operation, and high-quality and realistic image display capability in a wide range of application fields. In order to satisfy this requirement, we have developed a general-purpose polygon-based renderer, the Rendering Subroutine Package (RSP).

It is easy to imagine the ideal rendering system that most users would like to use for visualization. It would provide good-quality pictures quickly. Although many rendering methods, for example [1–15], have been proposed, it is very difficult to satisfy the requirements for all applications. This is not surprising, because there is always a trade-off between the quality of the generated picture and the time taken to generate the picture. Thus, one of the best answers is to provide several different types of rendering software.

We therefore provide users with three rendering methods (a list-priority method, a scan-line method, and a ray-tracing method) that cover most of their needs. **Figure 1** shows the relationship among the methods in terms of image quality and processing time. The user can select any of them, taking account of the trade-off between rendering time and quality. Both scan-line and ray-tracing support a new texture-mapping technique, which we call "attribute mapping." It can generate more realistic images and runs more efficiently than ordinary texture-mapping techniques for surfaces with complex textures. In this paper, we give a system overview of RSP, details of the functions, and several examples of the application of RSP to architectural CAD, molecular graphics, and computer tomography.

System overview of RSP

Figure 2 shows an overview of the functions of the RSP system. RSP uses polygonal models as geometry models, and can treat convex and concave polygons with holes and surface anomalies. The polygon data are defined by the RSP data format, which has two file formats: a

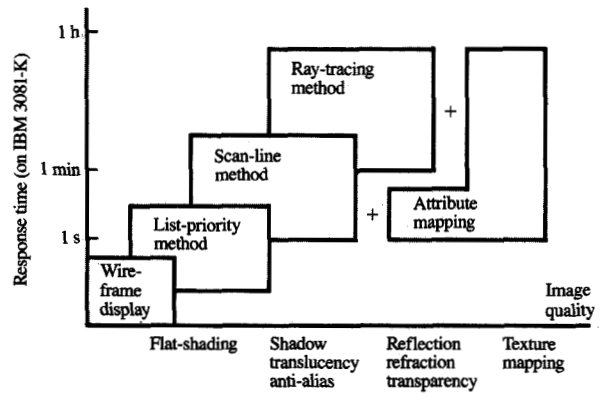

Figure 1

Image quality and processing time (image size 1024 × 1024, polygon count more than 3000).

character file format and a binary file format. The character file format is used to edit values in the files on a terminal. The binary file format is used to read and write rapidly from large polygonal data files. The use of a polygonal model greatly simplifies shaded-image synthesis, interactive operation on objects, and data conversion from other data models such as boundary representation (B-Rep), constructive solid geometry (CSG), and free-surface models. Consequently, the user can easily generate polygon data or write a program to convert data from other systems. In fact, RSP has reinforced existing modelers used by IBM, such as CADAM,[1] CATIA,[2] and CAEDS.[3]

The editing functions of RSP (inquiry, setting, memory allocation, and memory deletion) allow users to access the RSP data model, which can be used for interactive operations. The user can build a suitable system or user interface in the field by means of the editing functions.

As mentioned before, three rendering methods have been used to control the quality of the image and the purpose, and are executed on the same data model. A list-priority method gives an application the capability to display shaded graphics in near real time. The idea of the method is to draw (i.e., to generate components of the image in a controlled sequence) from the furthest polygons to the nearest polygons to remove hidden surfaces; a drawn polygon "paints over" some parts of the polygons behind it. The method is most suited to a set of data of about a thousand polygons. Hence, it is good for editing a small part of an object interactively. On the other hand, it is not suitable for displaying an object with a very large volume of data. A scan-line method is better for a large volume of polygon data. Our scan-line method gives an application the capability to generate shaded color images with shadowing. The user can obtain pictures within a reasonable response time. Ray-tracing is a rendering method that can treat reflection, refraction, and translucence. Our ray-tracing method is accelerated by a three-dimensional digital analyzer (3DDA) on the voxel data structure [11]. The details of each rendering method are described later.

In addition to the rendering functions, RSP supports color image quantization and dither functions for display devices with a color lookup table (CLUT) and image composition functions.

RSP is written in the C language, and is now running on VM/CMS, MVS/SP,[4] MVS/XA,[4] and AIX[5] operating systems.

Data structure

- *Geometry data structure*

In this section, the geometric data of the system are discussed. Although rendering of 3D objects is the major goal of the system, the design of the geometric data structure is very important, because it is one of the most essential parts of the interface between an application and the system. The rendering system is often used with another system that yields 3D geometric data to be visualized, such as a modeling system. The following were considered as the minimum criteria for the geometric structure of the system:

- Cost of converting geometric data for the rendering system.
- Richness of representability of 3D geometric data.
- Manageability of the 3D geometric data by the user.

Some geometric data in the user's model must be converted, because it is impossible to have exactly the same geometric structure for every application. The first criterion is that the conversion should be minimized. The second criterion is that it should be possible to represent even complicated 3D geometric data efficiently. The third criterion is that the user should be able to edit the 3D geometric data locally, so that there is no need to convert all the data again when there is only a small change. In view of the above criteria, questions must be answered

[1] CADAM is a trademark of CADAM, Inc.

[2] CATIA is a trademark of Dassault Systems Corporation.

[3] CAEDS is a registered trademark of International Business Machines Corporation.

[4] MVS/SP and MVS/XA are trademarks of International Business Machines Corporation.

[5] AIX is a registered trademark of International Business Machines Corporation.

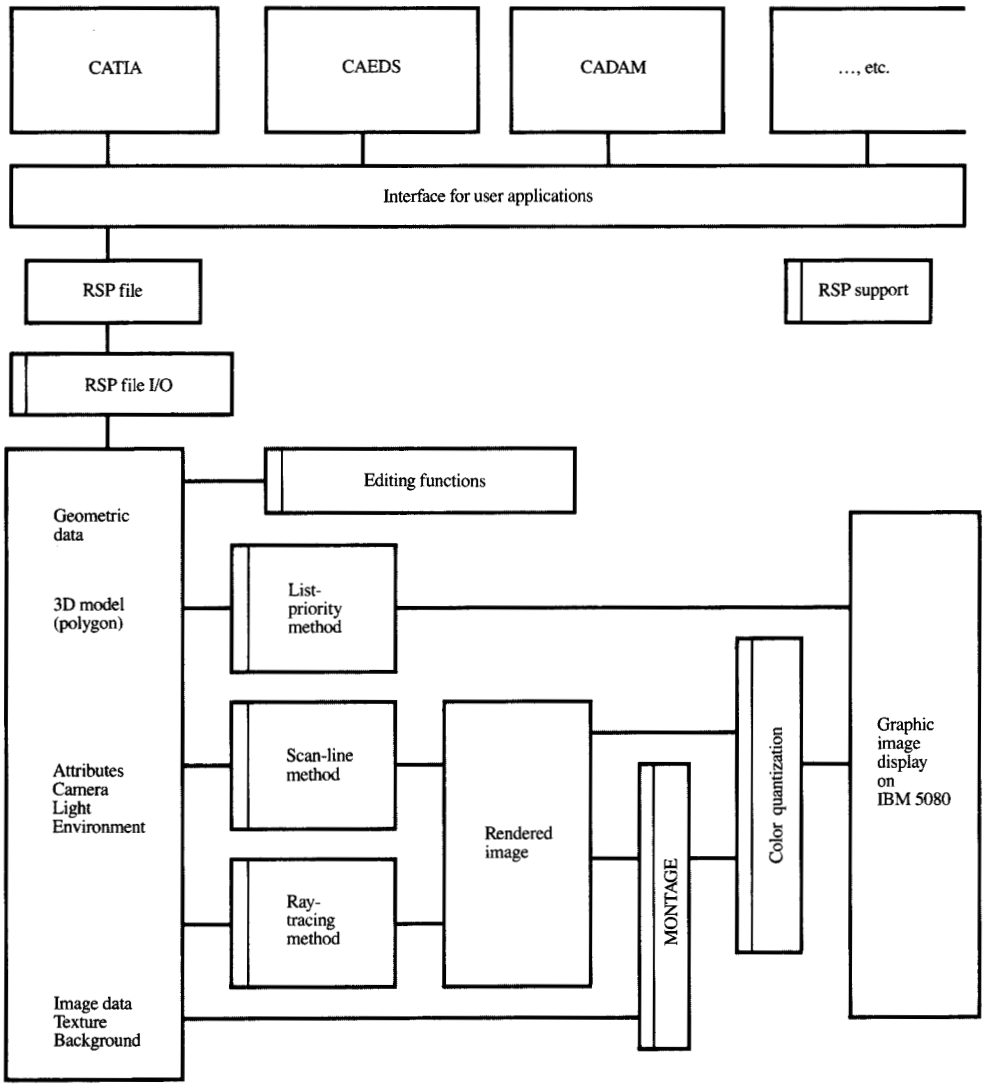

Figure 2

System overview using RSP.

about the design of the geometric data structure for a visualization system. One question is what kind of geometric primitives should be supported, and another is how the relationship among them should be represented.

In our system, polygon-based data were chosen as the geometric primitives, because it is usually easy to convert any user's models to polygon data. This does not cost too much and it satisfies the first criterion. However, simple polygon data alone cannot represent a common object efficiently. It is more natural for the user to represent a face (surface) with a hole by one primitive rather than multiple primitives. Thus the idea of faces, seals, and holes, which is described later, is introduced in this

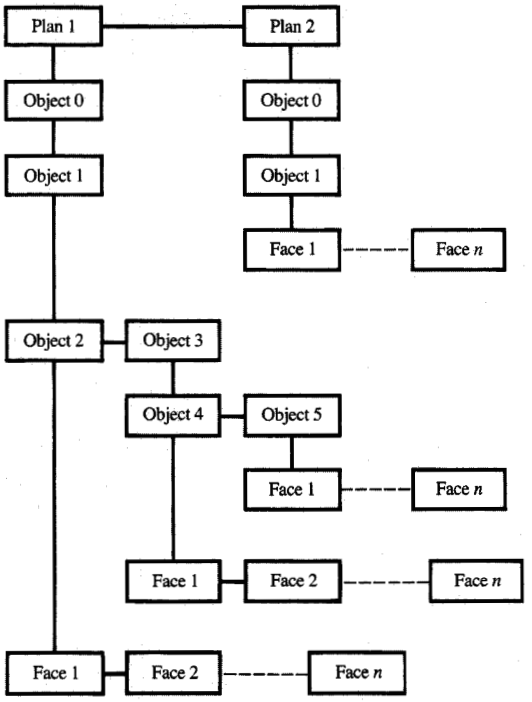

Figure 3

Example of the structure of plan, object, and face.

system. These generalities allow a user to convert data without any trouble. Thus, these polygon-based primitives satisfy in most cases the first and second criteria described above.

To satisfy the third criterion, a hierarchy for representing the relations among geometric data was introduced into the structure. The user can dynamically edit this hierarchy, using the functions provided by the system. Five entities are introduced to represent the geometric data of the rendering system: *plan*, *object*, *face*, *seal*, and *hole*. The plan is the largest entity and consists of several objects. The object consists of several faces. The face is a polygon that includes some seals and holes.

Plan

The plan, which is the largest entity, is normally used as a project unit from the user's viewpoint. A user can have several plans in the system, where they are stored as a list structure (**Figure 3**).

Object

The object is a unit under a plan. There are hierarchical relations among objects. A parent object can have several children. Each child can be a parent of other objects. A parent object may not have any faces, but otherwise an object consists of a group of faces (Figure 3).

Each object has its own coordinate system. The coordinate system is represented by a transformation matrix with respect to the parent's coordinate system.

Each object has its own display flag. By setting the value of the flag, a user can control the visibility of the object. The flag controls three states: displaying faces and seals, displaying faces only, and displaying neither.

The system provides a user area for each object. The user area can serve as the user's own attribute for the object. It may be a physical property of the object.

A root object is automatically generated by the system when the user generates a plan and its object. This root object is employed when the user wants to manipulate all the objects under the plan. For example, the user sets the transformation matrix to move and rotate everything under the plan.

Face, seal, and hole

The face data represent the object's geometry and have a list structure. A face element consists of the polygon of the face, the seals on the face, and the holes inside the face (**Figure 4**). The face is an area-defining geometric primitive which is defined by the face polygon. The polygon can have any number of edges as long as they do not intersect. The seal is an area inside the face and can have its own attributes. The seal has higher visibility than the face. The hole is an area inside the face that is always transparent. In other words, the seal and the hole are those areas whose attributes are different from those of the face. Multiple seals and holes are also allowed for the face. The seal data and the hole data are also polygonal lists. The polygons of the face, the seal, and the hole are represented by a list of vertices (**Figure 5**).

The face has an edge attribute and a face attribute. The edge attribute controls the appearance of the outer lines of the face. The face attribute controls the appearance of the face inside the edges. Since it is useful to distinguish the front and the back of the face, the face can have a different attribute for each side. The seal has attributes similar to those of the face, and different seals can have different attributes. The hole is not given any attributes, and the appearance of the edges of the hole depends on the attributes of the face to which the hole belongs.

• *Attribute data structure*

Attribute data in RSP are basically separate from geometric data. A few attribute data, however, are closely related to geometry, and are called "geometry-dependent

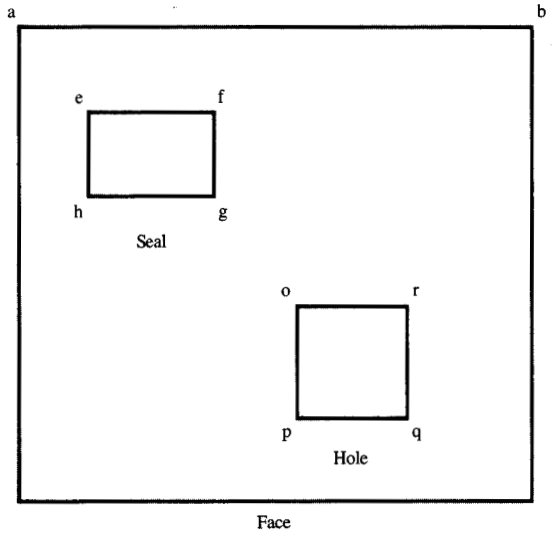

Figure 4

Example of a face with a seal and a hole.

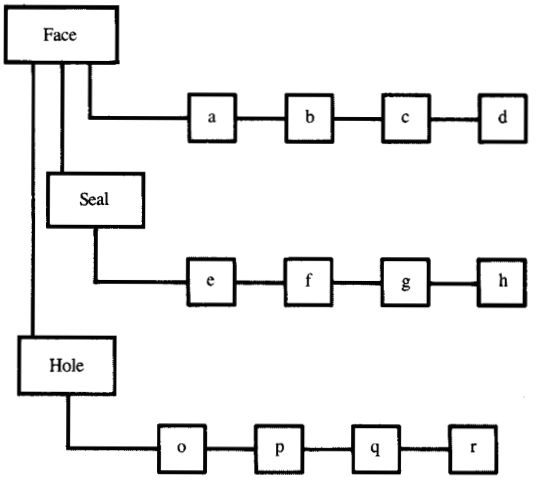

Figure 5

Example of the structure of face, seal, and hole.

attribute data." Mapping attribute data and smooth-shading attribute data are categorized as geometry-dependent attribute data. All other attribute data are called "geometry-independent attribute data" (**Figure 6**). Before the rendering procedure can be used, both geometry-dependent and geometry-independent attribute data must be linked to geometry data.

Geometry-dependent attribute data

Careful attention must be paid to the assignment of geometry-dependent attribute data, because they are allowed to be linked only to pertinent geometric data. Specifically, smooth-shading attribute data can be linked only to triangles. They consist of normal vectors corresponding to the three vertices of each triangle. Mapping attribute data consist of a mapping origin, a mapping direction, an actual length along each mapping direction, and a mapping offset. The geometric primitives to be mapped are basically restricted to rectangles, but it is possible to map image data to a triangle, for example, as long as the user specifies a virtual rectangular mapping canvas including the mapping origin, mapping direction, and so on.

Geometry-independent attribute data

Geometry-independent attribute data are divided into two types: "attribute domain definition data" and "rendering attribute data." The role of attribute domain

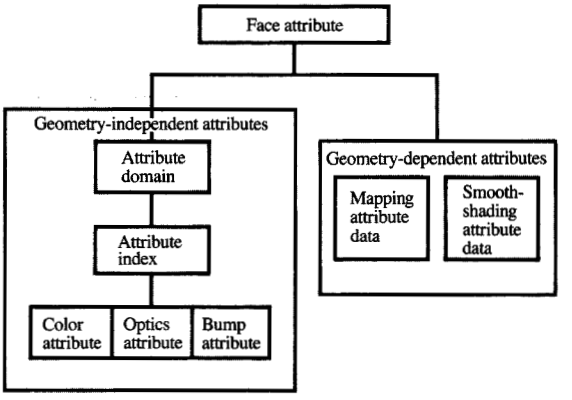

Figure 6

Attribute data structure.

definition data is to define an "attribute domain," which represents a region with the same rendering attributes to be assigned as a face attribute. One of the unique features of RSP originates in the attribute domain structure. The concept of the attribute domain is discussed in more

detail in the section on attribute mapping. An attribute domain is coupled via an "attribute index" with rendering attribute data. Rendering attribute data consist of color, optic, and bump attribute data. Usually they are referenced by a pointer corresponding to an attribute domain. Color attribute data represent either a RGB color value or a pointer to a color image. Optical attribute data represent a set of optical coefficients to be used in shading. Examples of optic data are ambient, diffusive, reflective, and refractive coefficients. Bump attribute data represent a pointer either to a functional bump generator or to a bump image. A functional bump generator generates a two-dimensional wavy bump pattern. Both a wavy bump pattern and a bump image are used to perturb surface normal vectors, and perturbed normals are then used in shading calculation.

• *Rendering condition data structure*

RSP has four rendering conditions, which are classified as the camera, light, environment, and special conditions of each rendering method.

Camera data are defined by the viewing parameters familiar to users of GDDM/graPHIGS[6] [16]. Therefore, a user can easily combine RSP and graPHIGS environments on a workstation. RSP, however, employs a normalized projection into a logical workstation space called the "normalized projection coordinates" (NPC). The NPC of RSP are defined as a cubic space from zero to one along each axis. RSP maintains a camera data list, and one of the cameras is selected for the rendering process.

RSP supports three types of light source: a point light source, a spot light source, and a light source parallel to the image. RSP maintains a light data list, and the lights in this list can be turned on or off. Each light has several parameters, such as light color, position, direction, spotlight angle, and so on.

Environmental information consists of the ambient light term, background information, the fog-effect ratio, sky-effect parameters, and a global shadow calculation switch.

The special conditions of each rendering method involve parameters such as the maximum ray reflection number of the ray-tracing method and the anti-aliasing level of the scan-line and ray-tracing methods.

Attribute mapping

Attribute mapping [17, 18] is a generalization of texture mapping. While texture mapping maps attribute values directly, attribute mapping maps "indices" that correspond to a bundle of various rendering attributes including colors, bumps, optics, and shading models. These indices are called "attribute indices," and each is defined for a closed region called an "attribute domain" within a rectangular mapping canvas. Rendering attributes within an attribute domain are assumed to be coherent. The core concept underlying attribute mapping is the attribute domain definition function (ADDF), which defines attribute domains and the associated attribute indices. By virtue of the ADDF, the integration of image analysis and image synthesis is easily attained, which in turn makes it possible to produce very realistic images.

• *Attribute index and bundle table*

On the basis of the available shading models implemented in RSP, once parameters associated with a shading model are determined, we can determine the intensity of an object at an arbitrary point. The attribute values necessary for attribute mapping are simply a set of these parameters in what we call a "bundle." Let us conventionally divide attribute values into three categories: 1) color attributes, 2) bump attributes, and 3) optic attributes. Color and bump attributes are independent of any shading model, but optic attributes are not. Color, bump, and optic attributes together are referred to as "rendering attributes." Each parameter of a rendering attribute is specified in one of three ways: a unique, fixed value (Type 1), a functional value (Type 2), or a value looked up in a two-dimensional table (Type 3).

The value is independent of the position in the first method, but not in the other two methods. The last method is usually employed in texture mapping. When we access a rendering attribute, it is necessary to specify a "Type" parameter, as defined above, and the associated pointers. These data are kept in a table called an "attribute bundle table." To be more specific, the contents of an attribute bundle table are a specification type (Type 1, 2, or 3) and three pointers to rendering attributes: a pointer to the color attribute, a pointer to the bump attribute, and a pointer to the optic attribute. The attribute bundle table itself can be accessed by using an index called an "attribute index." Texture mapping maps attribute values directly. In contrast, attribute mapping maps attribute indices, which in turn are replaced by a specification type and pointers to rendering attributes.

• *Attribute domain*

Compound objects typically seen in minerals and rocks have different attributes intermixed, and distinct coherent regions are distributed over the surface of the object. An example of an artificial compound object is one with metallic seals pasted onto a package that is not metallic by itself. An example of a natural object of this kind is a granite that naturally contains different minerals such as quartz, mica, and feldspar, each of which has different

[6] GDDM and graPHIGS are trademarks of International Business Machines Corporation.

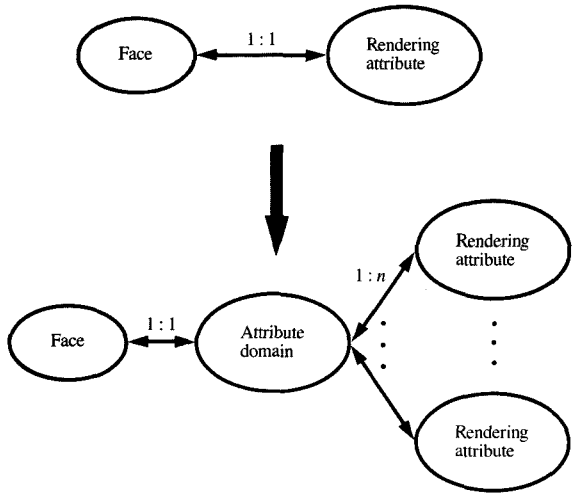

Figure 7

Texture mapping and attribute mapping.

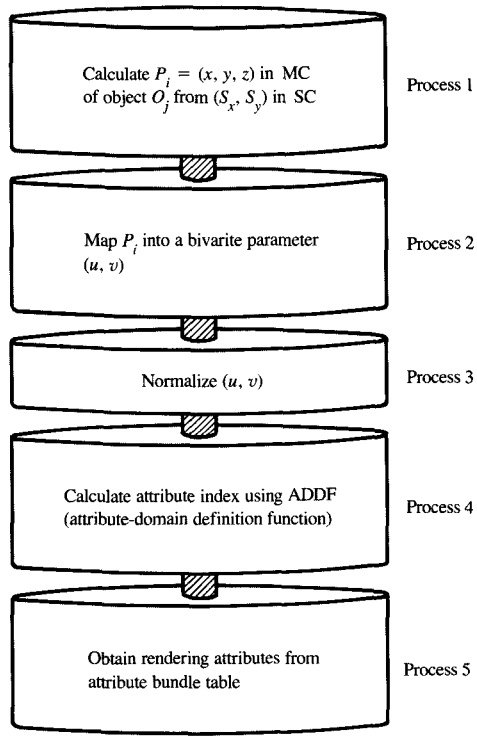

Figure 8

Mapping pipeline of attribute mapping.

rendering attributes. These compound objects are very difficult to represent realistically. To solve such difficulties, we introduce what we call "attribute domains." By using attribute domains, compound objects can be simulated very realistically. Another purpose of defining attribute domains is to reduce the amount of memory required, taking advantage of area coherence.

As shown in **Figure 7**, there is a one-to-one correspondence between a surface and a rendering attribute in ordinary texture mapping. In contrast, if we define attribute domains between a surface and a rendering attribute, the relation between a surface and the rendering attributes in general becomes one-to-many.

With each "attribute domain," we associate a function called an "attribute-domain definition function" (ADDF). An ADDF is defined either functionally or by assigning a black-and-white image called a "trigger pattern." Examples of functionally defined ADDFs include a tiled attribute-domain generation function and a bricked attribute-domain generation function. A trigger pattern is a rectangular pixel pattern whose content is used as a pointer to an attribute bundle table. In other words, the contents are used as attribute indices.

- *Rendering pipeline in RSP*

Figure 8 shows a so-called "rendering pipeline" or "mapping pipeline." This mapping pipeline includes attribute mapping and other functions, starting with a process for obtaining an object's position and ending with a process for obtaining the values of its rendering attributes.

Process 1

Starting with a screen coordinate value (S_x, S_y), this process obtains a modeling coordinate value corresponding to a point on the object of concern. It generally depends on the rendering method used. For example, in ray-tracing we can work in the world coordinate (WC) system directly. In other words, it is simply necessary to convert data from world coordinates (WCs) to modeling coordinates (MCs). The scan-line method usually begins with screen coordinates (SCs) because their shading calculation often depends on the number of scan lines in the screen. In these methods, SCs are first converted to view coordinates (VCs), then to WCs, and finally to MCs. The conversion among multiple coordinate systems can be expressed in a homogeneous four-by-four matrix. The output of this process is a point whose value is expressed in MCs.

Attribute index	Color attribute		Bump attribute		Optic attribute		Comments
	type	id	type	id	type	id	
#1	3	#1	1	#1	1	#1	quartz
#2	3	#1	1	#1	1	#2	mica
#3	1	#1	1	#1	1	#3	feldspar
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.

Figure 9

Attribute bundle table.

Process 2

This process depends on the geometry of the object and the definition of a mapping function from 3D to 2D geometry. Even if the geometry of the object does not change, an almost infinite variety of definitions of mapping functions may exist. For instance, let P be a point on a parametrically defined bivariate bicubic patch, $P = (s, t)$. A simple mapping function from P to 2D coordinates (u, v) is established by setting $s \rightarrow u$, $t \rightarrow v$. Another mapping function is obtained by first converting the value of P into (x, y, z) according to the bicubic patch definition and then setting $y \rightarrow u$, $z \rightarrow v$, which is equivalent to projecting the point P to the plane $X = C$ (constant). The polar coordinate system or cylindrical coordinate system can also be used for defining a mapping function.

Process 3

This process converts (u, v) in a 2D real space obtained in Process 2 into (u', v') in a rectangular normalized UV-space. Mapping parameters such as mapping direction, mapping scale factor, and wraparound control flag are included here. The flexibility of the mapping process is greatly enhanced by a suitable choice of parameters.

Process 4

The kernel of attribute mapping lies in this process. Here the values (u', v'), together with the trigger pattern, if any, are given to an ADDF defined on a surface of an object, and an attribute index is produced as output.

Process 5

In this process, the attribute bundle table is accessed by an attribute index obtained in the previous process. By using the contents of the table entry corresponding to the attribute index, each rendering attribute is obtained. A typical example of an attribute bundle table is shown in **Figure 9**.

Image synthesis tool

- *Near-real-time display*

BSP tree

A simple way to display an object is to sort a group of polygons by using the relationship between the view and the normal of a polygon. However, it is still computationally expensive, because a slight change in view makes it necessary to sort all the polygons again. Since many more applications need view changes than need world-model changes, we decided to use a binary space partitioning (BSP) tree, which was proposed by Fuchs et al. [6].

A BSP tree is a structure in which processed polygon data are stored. Traversing the BSP tree with a given eye location performs hidden-surface removal in near real time. The procedure for generating a BSP tree can be summarized as follows:

1. Pick an arbitrary polygon from the polygon list, and assign it as the root polygon.
2. Process the rest of the polygons to make a front polygon list and a back polygon list. If all the vertices of a polygon are in front of the root polygon above, the polygon is added to the front polygon list. If all the vertices of the polygon are behind the root polygon, the polygon is added to the back polygon list. Otherwise, the polygon is divided into two polygons by the plane on which the root polygon is located. Then the front part of the polygon is added to the front polygon list, and the back part of the polygon is added to the back polygon list.
3. Repeat this process recursively for the front polygon list and the back polygon list until no more polygons are left.

Traversing the BSP tree gives the display order of the polygons. At each node of the BSP tree, the dot product of the given viewing vector and the normal of the partitioning polygon are calculated. If the dot product is negative, the eye is in front of the polygon, and the polygons behind the node polygon are traversed before those in front of it. If the dot product is positive, the polygons in front of the node polygon are processed first. If the node is a leaf of the BSP tree, the polygon is drawn and then the process goes back to traverse the rest of the BSP tree. This process is continued until all the polygons are drawn. Details of the algorithm for the BSP tree can be found in [6].

BSP tree for interactive use

One of the most suitable applications of the BSP tree is for interactive editing of 3D objects. Here, interactive

editing means the editing of both the shape and the attributes of the 3D objects. Editing the shape, that is, changing the topology and geometry of the 3D objects, can be done through a wire-frame picture. But editing the attributes of the object is difficult to do in the environment of a wire-frame picture. Of course, it is possible to edit the attribute by picking an appropriate edge of the object's face and then entering the appropriate values from a keyboard. However, this is no longer interactive. In an interactive environment, changes in the objects' attributes are reflected in the picture displayed. When the color is changed, the color of the shaded image should be changed in real time. Editing the individual color of polygons is a typical application.

To take advantage of the interactive capability, the system provides four stages in the rendering, as shown in **Figure 10**.

First, a user must generate the BSP trees for the objects to be displayed. The user picks an arbitrary node of the body, and then the system automatically makes a BSP tree for the polygon data under the body node.

After generating the BSP trees, the user must prepare to set several attributes. At this stage, appropriate attributes are associated with the BSP tree generated in the previous stage. The user chooses arbitrary camera, light, environment, and global attributes, and can change these attributes independently of one another.

To increase the flexibility with which 3D objects can be edited interactively, we provide the following interface associated with a polygon's attributes, which affects the appearance of faces, seals, and edges when they are displayed. The color lookup table architecture is assumed in the system. We decided to calculate a set of colors according to the attributes set by the user, and load the colors to the color lookup table before the display stage, so that the advantages of the BSP tree and the lookup table architecture are fully utilized. The dot product of the viewing vector and the face normal gives the intensity of luminescence. This value is converted to an appropriate color table index, depending on the number of color lookup table entries to which the color is assigned.

Thus the system is designed to give the user an opportunity to assign an arbitrary number of color lookup table entries to arbitrary face or seal attributes. If the user would like to see a subtle change in the color due to the lighting, a large portion of the color lookup table should be assigned to the color. If the user does not care much about it, a few lookup table entries are enough for the purpose. The above color lookup allocation can be done through a system's function call. The user can specify which part of the lookup table, the starting entry or the number of the entries, is used for a particular face, seal, or edge attribute.

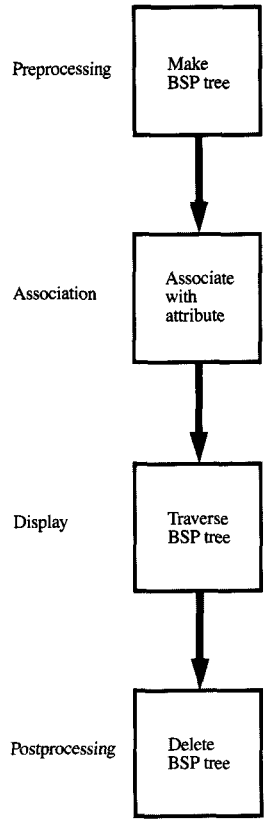

Figure 10

Process flow of the list-priority method.

After setting up the attribute, the user specifies the designated workstation in the display stage. The user associates the BSP tree with the workstation where a picture of the object represented by the BSP tree is supposed to be displayed, and can associate several BSP trees with the workstation if he wants to display more than one object simultaneously.

Finally, when the user does not need to display the object of the BSP tree any more, the tree should be deleted to save memory space.

• *Shading display with shadow*

Our shaded display with shadow is processed by the Bouknight scan-line algorithm [1, 2] with the shadow volume algorithm [3, 8]. Basically, the Bouknight approach consists of a y-bucket sort on the edges. The first step is to create an edge table for all nonhorizontal

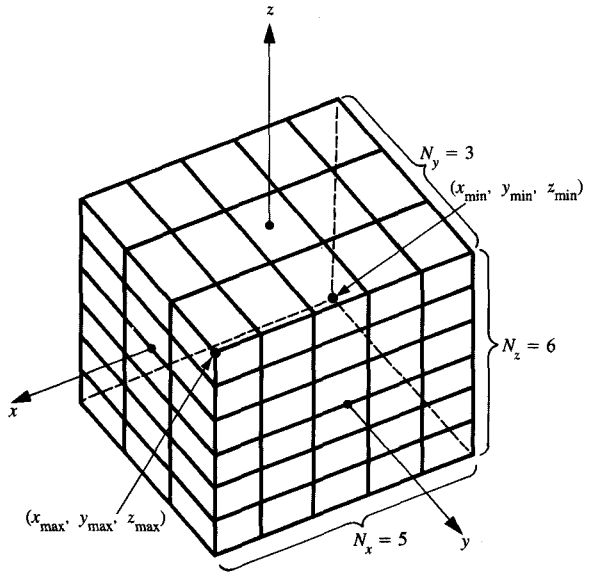

Figure 11

Voxel window and voxel size.

edges of all polygons. Each edge is stored in the table according to its smaller y-coordinate. Next, the active edge list is created from the edges that intersect the current scan-line. For each scan-line, an x-coordinate sort is run on the active edge list, which is based on scan-line coherence. The scan-line depth buffer of active polygons shows the visible one.

The addition of shadows vastly complicates the image synthesis process, though it contributes considerably to the realism of a scene and increases the perception of depth. In 1977, F. Crow presented a technique for shadow casting that was based on shadow volumes [3]. A shadow volume is defined by the shadow polygon given by planes defined by 1) contour edges and 2) light source position. The contour edges are those edges owned by both front-facing polygons and back-facing polygons for each light source. The endpoints of the edges from the light source positions are the bounds of the field of view. The shadow polygons are added to the ordinary geometrical data, and do not influence visibility. However, the depth order of shadow surfaces and visible surfaces determines the shadowing. If a visible point lies within the shadow volume, the point is in shadow. The shadow-volume method allows concave polygons with holes and any number of light sources anywhere in 3D space in the scan-line process. This unconstrained environment is the most efficient feature in other shadow algorithms based on a scan-line method or z-buffer methods [1, 2, 5, 7].

The most efficient feature of our scan-line method is its shadow-polygon data reduction techniques [19]. A shadow-volume algorithm is expensive for image synthesis in a complex environment, because it generates a lot of shadow polygons. In order to relax the restriction, we have proposed shadow-polygon reduction techniques. These consist of the use of a coplanar surface (seal) data structure, determination of shadowing polygons by using six-space subdivision at a point light source, and techniques for extraction of contour edges. Use of these techniques makes the shadow-volume algorithm effective in a complex environment.

In order to overcome this aliasing problem, we apply color-blending techniques at each scan-line process in the scan-line method. The color at each pixel is approximated by the product of the area sums of visible polygons. In order to calculate the precise area of visible polygon at each pixel, we applied a subscan-line division technique, which divides a scan-line into N subscan-lines:

$$Color_{pixel} = \sum_{i=1}^{N} (L_i \times C_i)/N. \quad (1)$$

Here, $Color_{pixel}$ is the color of a pixel and L_i is the length of the face on the subscan-line; C_i is the color of the face on the subscan-line, and N is the number of subscan-lines in a scan-line.

- *High-quality image display*

A ray-tracing method gives RSP the function of high-quality image display. It enables transparency, translucency, refraction, reflection, and shadowing to be used as standard functions.

In ray-tracing methods, a ray is traced from the eye through each pixel into the polygonal data environment. At each polygon struck by the ray, a reflected and/or a refracted ray can be generated. The rays are traced recursively to establish what polygons they intersect, and an intersection tree is constructed for each pixel. The final pixel intensity is determined by traversing the tree and computing the intensity contribution of each node according to the shading model.

There is usually a trade-off between high-quality image display and high performance. Whitted [9] shows that 75 percent of the total time is spent on calculating intersections between rays and objects for simple scenes. To reduce this burden, we employ a "voxel"-based method [11] with user-controllable parameters such as the number of voxels along each coordinate axis and the voxel size (**Figure 11**). A voxel is an orthogonal cuboidal cell, which can be thought of as a 3D extension of a

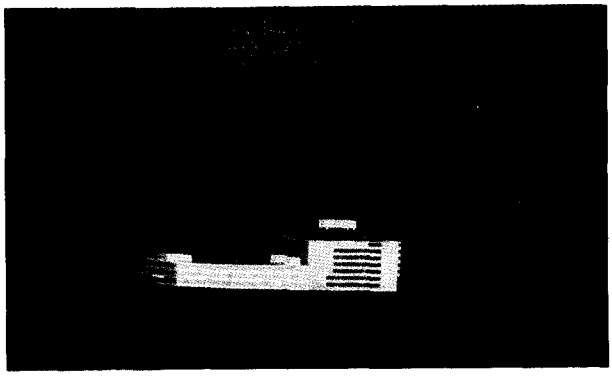

Figure 12

Simulation of a landscape by the list-priority method. Face number 1135; CPU (BSP-generation) time 0.29 s; CPU (traverse-and-display) time 0.05 s.

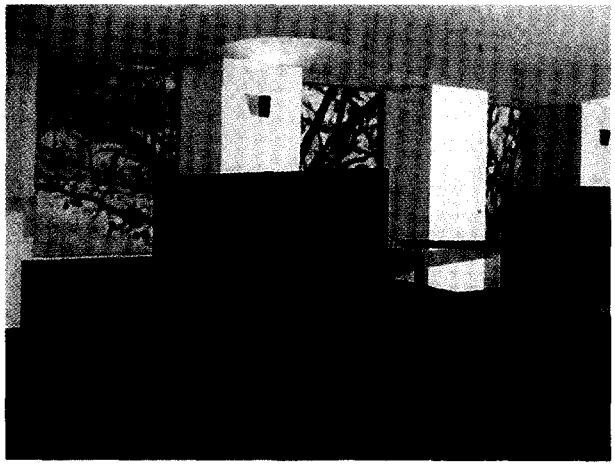

Figure 13

Visual effect of the interior room I. Face number 1641; CPU time 4 min 22.20 s.

raster grid, with pixels becoming voxels. The voxel data structure is constructed before the intersections between rays and objects are calculated. Each cell has information on which polygons are involved in the cell. The 3DDDA (three-dimensional digital differential analyzer [11]), which is like a 3D line generator, is a basic tool for traversing voxel data structures. It is applied in the direction of the ray, and continues pursuing the ray in the same direction until some object is intersected or until it leaves the voxel domain. The calculation of motions from one cell to another is achieved by incremental logic, without any multiplication or division.

A salient feature of our ray-tracing method is that it incorporates attribute mapping, using a full set of attribute data assigned to each face. Attribute mapping coupled with ray-tracing has the capability of producing extremely high-quality pictures. Supersampling coupled with ray-tracing also enhances the image quality. The user can specify the degree of supersampling with parameters provided by rendering condition data.

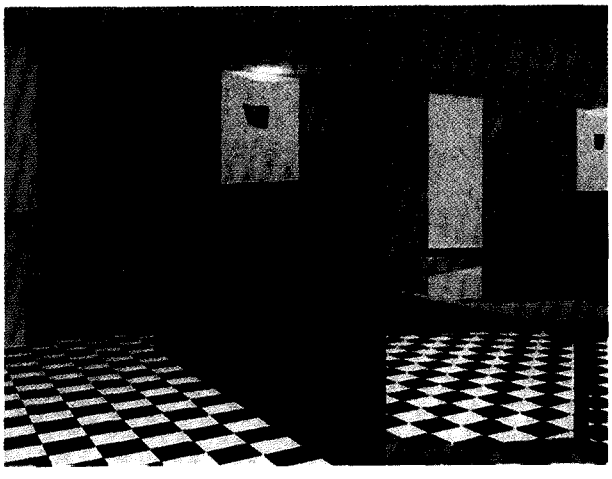

Figure 14

Visual effect of the interior room II. Face number 1641; rendering time 36 min 29.22 s.

Applications

In this section, we describe some applications of RSP.

Visual simulation has become popular in architectual CAD. It is very important to understand the potential appearance of new buildings, or the view, lightness, and shadows of the interior rooms. **Figure 12** shows a simulation of a landscape by the list-priority method. The user can rotate the view, pick a visible polygon, and change the attributes on the screen interactively. **Figures 13** and **14** show the visual effect created by changing from carpet to polished tile patterns on the floor. They are respectively rendered by the scan-line method and the ray-tracing method. **Figure 15** shows an example of image composition functions. It combines a computer-generated image and a photographed picture. The photographed foreground picture is extracted by image segmentation by thresholds, and overlaid on the computer-generated picture. **Figure 16** shows an example of attribute mapping. Attribute mapping is used at the transoms near the ceiling. Each transom consists of a polygon on which transparency and color image information is mapped.

The spatial shape of molecular orbital functions plays an important role in allowing researchers to understand the nature of chemical reactions, as the frontier orbital

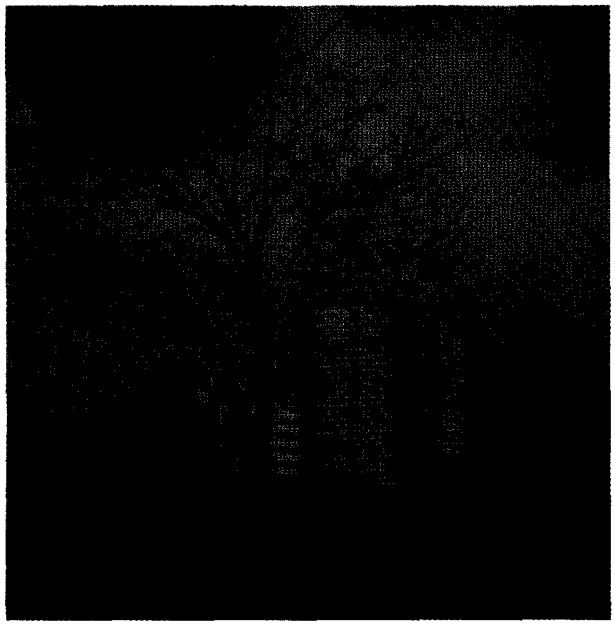

Figure 15

Example of image composition. CPU time 28.51 s.

Figure 16

Example of attribute mapping. Face number 10981; CPU time 5 min 8.71 s.

Figure 17

Example of smoothly shaded image display. Face number 7174; CPU time 2 min 19.61 s.

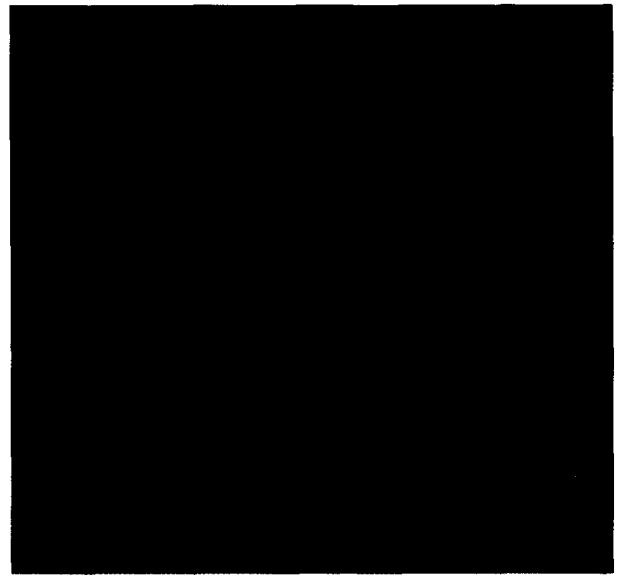

Figure 18

Constant-density surface by computer tomography. Face number 29134; CPU time 6 min 3.41 s.

theory [20, 21] and Woodward–Hoffmann theorem [22, 23] have shown. If polyhedral approximation is done in molecular orbital functions, the user can rotate, zoom, shift, and overlay equivalued surfaces. The overlay of molecular orbital functions of two molecules is useful for recognizing reaction sites. **Figure 17** shows an equivalued surface of the electron density of C_2H_5OH. The electron density values are 0.1 and 0.125 $bohr^{-3}$.

Three-dimensional visualization has become important in helping physicians to understand the complex anatomy of the human body. If the triangulation of a constant-density surface by 3D computer tomography is achieved, the resulting models can easily be displayed with RSP. **Figure 18** shows some results of computer tomography, with the soft-tissue surfaces displayed in transparent form.

The polyhedral data of Figures 17 and 18 are generated by the tetrahedral grid method [24, 25]. Figures 13, 14, 15, and 16 were rendered with a resolution of 800 by 800 pixels, and the resolution of Figures 17 and 18 is 512 by 512 pixels. Figures 13 and 15 are generated by two subscan-lines, and Figure 16 by one subscan-line. The former is more anti-aliased than the latter.

Conclusions

We have described data visualization techniques for use with a general-purpose renderer. The polygonal model approach was proposed for easy and economical implementation of flexible interactive manipulation, real-time animation display, and high-quality image display. When the number of polygons becomes large, there may be a memory problem in an application program. But we believe that in this case the problem can be processed completely in a large virtual-memory environment. In particular, the enhancement of program addressability [26] will overcome the problem.

In order to satisfy the requirements of all applications, we provide three rendering methods. The user can select the most suitable of them for the image generation process. This allows stepwise refinement of the image quality. Texture mapping is a powerful tool for realistic image synthesis in computer graphics. As a generalization of texture mapping, we have proposed "attribute mapping." By virtue of ADDF, it can generate more realistic images and run more efficiently than ordinary texture mapping for surfaces with complex textures.

Currently, we are continuing to improve RSP, and are developing an interactive graphics environment. A user-friendly interactive graphics environment will help novice users and nonprogrammers to generate pictures easily. We also aim to develop higher-quality rendering methods, which will be able to support mutual interreflection of light, color bleeding, penumbra, and realistic texture generation.

Acknowledgments

We are extremely grateful to Akio Koide, Tatsuo Miyazawa, and Hideo Suzuki for their suggestions and helpful discussions when we were at the stage of generating the pictures. The models of Figure 12 and Figure 15 were originally created by Ohbayashi Corporation, and the model of Figure 16 was originally created by Fujita Corporation. We also thank the Department of Medicine (Section III) of Sapporo Medical College for providing us with CT image data.

References

1. W. J. Bouknight and K. Kelly, "An Algorithm for Producing Half-Tone Computer Graphics Presentation with Shadows and Movable Light Sources," *AFIPS Conf. Proc.* **36,** 1–10 (1970).
2. W. J. Bouknight, "A Procedure for Generation of Three-Dimensional Half-Toned Computer Graphics Presentations," *Commun. ACM* **13,** No. 9, 527–536 (1970).
3. F. C. Crow, "Shadow Algorithm for Computer Graphics," *ACM Proc. SIGGRAPH* **11,** No. 2, 242–248 (1977).
4. P. Atherton, K. Weiler, and D. Greenberg, "Polygon Shadow Generation," *ACM Proc. SIGGRAPH* **12,** No. 3, 275–281 (1978).
5. L. Williams, "Casting Curved Shadows on Curved Surfaces," *ACM Proc. SIGGRAPH* **12,** No. 3, 270–274 (1978).
6. H. Fuchs, G. D. Abram, and E. D. Grant, "Near Real-Time Shaded Display on Rigid Objects," *ACM Proc. SIGGRAPH* **17,** 65–72 (1983).
7. L. S. Brotman and N. I. Badler, "Generating Soft Shadows with a Depth Buffer Algorithm," *IEEE Computer Graph. & Appl.* **4,** 5–12 (1984).
8. P. Bergeron, "A General Version of Crow's Shadow Volumes," *IEEE Computer Graph & Appl.* **6,** 17–28 (1986).
9. T. Whitted, "An Improved Illumination Model for Shaded Display," *Commun. ACM* **23,** 270–274 (1980).
10. A. S. Glassner, "Space Subdivision for Fast Ray Tracing," *IEEE Computer Graph. & Appl.* **4,** 15–22 (1984).
11. A. Fujimoto, T. Tanaka, and K. Iwata, "ARTS: Accelerated Ray-Tracing System," *IEEE Computer Graph. & Appl.* **6,** 16–26 (1986).
12. P. Sabella, "A Rendering Algorithm for Visualizing 3D Scalar Fields," *ACM Proc. SIGGRAPH* **22,** No. 4, 51–58 (1988).
13. C. Upson and M. Keeler, "V-BUFFER: Visible Volume Rendering," *ACM Proc. SIGGRAPH* **22,** No. 4, 59–64 (1988).
14. R. A. Drebin, L. Carpenter, and P. Hanrahan, "Volume Rendering," *ACM Proc. SIGGRAPH* **22,** No. 4, 65–74 (1988).
15. M. F. Cohen, S. E. Chen, J. R. Wallace, and D. P. Greenberg, "A Progressive Refinement Approach to Fast Radiosity Image Generation," *ACM Proc. SIGGRAPH* **22,** No. 4, 75–84 (1988).
16. *Understanding graPHIGS*, Order No. SC33-8102, 1989; available through IBM branch offices.
17. M. Aono and T. L. Kunii, "Attribute Mapping," *Research Report TR-87/1010*, IBM Tokyo Research Laboratory, 1986.
18. M. Aono, "Methodology of Attribute Mapping and Its Applications," *JIPS Proc. Graph. & CAD*, pp. 31–38 (1988).
19. A. Doi et al., "Some Technigues for F. Crow's Volume Algorithm," *IPSJ Proc. 38th Annual Convention* (1989).
20. K. Fukui, "An MO-Theoretical Illumination for the Principle of Stereoselection," *Bull. Chem. Soc. Jpn.* **39,** 498–503 (1966).
21. K. Fukui, "Recognition of Stereochemical Paths by Orbital Interaction," *Acc. Chem. Res.* **4,** 57–64 (1971).
22. R. B. Woodward and R. Hoffmann, "Stereochemistry of Electrocyclic Reaction," *J. Amer. Chem. Soc.* **87,** 395–397 (1965).
23. R. B. Woodward and R. Hoffmann, "Selection Rules for Sigmatropic Reaction," *J. Amer. Chem. Soc.* **87,** 2511–2513 (1965).
24. A. Koide, A. Doi, and K. Kajioka, "Polyhedral Approximation Approach to Molecular Orbital Graphics," *J. Molec. Graph.* **4,** 149–156, 160 (1986).
25. A. Doi and A. Koide, "An Efficient Method of Triangulating Equi-Valued Surfaces by Using Tetrahedral Cells," *J. IEICE Trans.* **E-74,** 214–224 (1991).
26. B. R. Aken, Jr., "Large Systems and Enterprise Systems Architecture," *IBM Syst. J.* **28,** No. 1, 4–14 (1989).

Received October 31, 1989; accepted for publication September 5, 1990

Akio Doi *CIM System Development, IBM Yamato Laboratory, IBM Japan Ltd., Shimotsuruma 1623-14, Yamato-shi, Japan.* Mr. Doi received his B.S. and M.S. degrees in civil engineering from Kobe University in 1980 and 1982. He joined the Tokyo Scientific Center in 1982. In 1986, with Dr. Akio Koide, he developed a scientific visualization tool using tetrahedral elements, and has been improving it. Mr. Doi has contributed to the development of the Rendering Subroutine Package, in particular a scan-line method with shadow polygon generation algorithms. His research interests include computer graphics, scientific visualization, and application programming interfaces. He is a member of the IEEE Computer Society and the Computer Graphics Society.

Masaki Aono *Tokyo Scientific Center/Tokyo Research Laboratory, IBM Japan Ltd., 5-19 Sanban-cho, Chiyoda-ku, Tokyo 102, Japan.* Mr. Aono received his B.S. and M.S. degrees in information science from the University of Tokyo in 1981 and 1984. He published a paper entitled "Botanical Tree Image Generation" in the *IEEE Computer Graphics and Applications* journal with Prof. Kunii of the University of Tokyo in 1984. Since joining the Tokyo Research Laboratory in 1984, Mr. Aono has worked on computer graphics projects. He developed a new texture mapping called "attribute mapping" and has contributed to the development of the Rendering Subroutine Package. His research interests include computer graphics, computational geometry, and physically and biologically based modeling. Mr. Aono is a member of the IEEE Computer Society and the ACM.

Naoki Urano *Tokyo Scientific Center/Tokyo Research Laboratory, IBM Japan Ltd., 5-19 Sanban-cho, Chiyoda-ku, Tokyo 102, Japan.* Mr. Urano received his B.S. and M.S. degrees in computer science from Rensselaer Polytechnic Institute in 1981 and 1984. Since joining the Tokyo Research Laboratory in 1984, he has worked on computer graphics projects. He developed a rule-based animation system called ROMA in 1986, and has contributed to the development of the Rendering Subroutine Package. Mr. Urano's interests include 3D interactive graphics, geometric modeling, and computer graphics standardization. He is a member of the IEEE Computer Society and the ACM.

Kazutoshi Sugimoto *Tokyo Scientific Center/Tokyo Research Laboratory, IBM Japan Ltd., 5-19 Sanban-cho, Chiyoda-ku, Tokyo 102, Japan.* Mr. Sugimoto is a manager of Applied Graphics at the Tokyo Research Laboratory. He received his B.S. and M.S. degrees in applied physics from Waseda University in 1976 and 1978, joining the Tokyo Scientific Center in 1978. From 1979 to 1982, he participated in a partnership program on a data processing system for land use, in which he developed a regional information system which has been installed by local and national governments. In 1984, as a CIP of IBM Japan, Mr. Sugimoto developed a raster-to-vector conversion software (GIFTS) and a PC version of GIFTS (MicroGIFTS) that became one of the main functions of the IBM PC CAD in 1989. His research interests include computer graphics, computer vision, image processing, and multimedia applications. He is a member of the ACM.