Preprint of the Invited Paper: T. L. Kunii, Y. Saito and M. Shiine, "A Graphics Compiler for 3-Dimensional Captured Image Database and Captured Image Reusability", *Proceedings of IFIP Workshop on Modelling and Motion Capture Techniques for Virtual Environments (CAPTECH98)*, November 26-27, 1998, Geneva, Switzerland, Lecture Notes in Computer Science (LNCS), pp.128-139, Springer-Verlag,

A GRAPHICS COMPILER FOR A 3-DIMENSIONAL CAPTURED IMAGE DATABASE AND CAPTURED IMAGE REUSABILITY

Tosiyasu L. Kunii*, **, *Fellow, IEEE*, Yoshifuru Saito*, *Member, IEEE*, and Motoyoshi Shiine*

***Abstract*—We present a new research direction named a graphics compiler for capturing and dynamic recovering of the shapes of 3-dimensional graphics targets. / The graphics compiler is based on graphics parsing precedence rules. / It interactively parses a 3-dimensional graphics shape data into a set of the basic geometrical and topological constructs of cellular spatial structures. / It enables us to recover the target object shapes as seen from any viewpoints.**

An object-centered motion capture method is developed, as opposed to a camera-centered motion capture method. / After capturing a graphics target images dynamically, a set of captured image data is compiled interactively and then the complied objects are stored in a graphics database. / A sequence of images of the target objects captured at different time instances makes it possible to produce a set of compiled shape information of the objects containing a complete 3-dimensional geometrical shape information that inherits topological properties. / Thus, the captured shapes are turned into reusable components to build new graphics contents on demand by assembling them.

***Index terms*– graphics compiler, graphics database, cellular spatial structure, graphics parsing precedence rule-based capturing technique, object-centered motion capture method, topological information inheritance**

I. INTRODUCTION

Most of the conventional image capturing is oriented to immediately store the snap shots of graphics image data as they are. / Thereby, conventional image capturing methodology greatly depends on the development of hardware. / With the development of the software and hardware technologies, the demand of high quality and high speed image capturing is explosively increasing for various industrial applications, e.g. product inspecting processes, security systems and so on. / However, the simple increasing of the number of captured images as well as pixels ends up with an enormous data storage capacity. /

In the present paper, we propose a new methodology named the graphics compiler. / Fundamental difference between the conventional and proposed methodologies is in the conceptual aspects of image capturing techniques as well as applications. / Graphics compiler translates the original image data into a set of basic geometrical objects. / Each of the captured data is compiled into the basic geometrical objects deleting the duplicated object information. / The compilation proceeds very often interactively as usual due to visual occlusion, and also the ambiguity and incompleteness of captured data. /

As a matter of fact, the compilation consists of two separate phases. / The first is the conversion of the image to a graph theoretical representation that is the only phase often requires human interaction. The second phase is

* Department of Electronics and Electrical Engineering, Graduate School of Engineering, Hosei University, 3-7-2 Kajino-cho, Koganei City, Tokyo 184-8584 Japan; e-mails: kunii@k.hosei.ac.jp; ysaitoh@ysaitoh.k.hosei.ac.jp; shiine@ysaitoh.k.hosei.ac.jp. ** Monolith Co., Ltd., 1-7-3 Azabu-Juban, Minato-ku, Tokyo 106-0045 Japan; e-mail: tlk@mbp.com. Institute of Digital Art and Science (IDAS), 1-25-21-602 Hongo, Bunkyo-ku, Tokyo 113-0033 Japan. e-mail: tosi@kunii.com.

for the repeated reuse of such compiled data obtained in the first phase and stored in the graphics database. This phase is completely automated and that is the phase to generate object images in the *scope of interest.* The scope of interest is given as a 3-dimensional range, in our case is a boxel that contains the graphics object of our interest. Since the graphics object in the graphics database is already compiled in the first phase into a graph theoretical representation, we can automate the compilation based on the graphics parsing precedence rules, as explained in this paper.

After obtaining a set of complete geometrical objects by the graphics compiler, the target image in the scope of interest can be recovered in a 3-dimensional manner and projected onto a 2-dimensional screen. This makes it possible to show the target image in a static and dynamic manner. Also it serves to save the storage capacity by making all the objects and their components in the images reusable. The compiled reusable resources are stored in a database. Thus, the *graphics compiler* is an innovative methodology. It translates the captured image data into a set of basic geometrical objects containing the complete graphics information. Hence, original target object can be visualized in whatever ways we like to present it.

The kernel of the graphics compiler is based on the hierarchical space indexing method [1] and cellular spatial structures [8-10]. The hierarchical space indexing method is one of the methodologies for indexing a 3-dimensional space. When we employ a regular decomposition of the 3-dimensional space, this leads us to a tree structure and to store the geometrical objects in table forms. Thereby, quick access to an interest point can be carried out by means of a leaf node as an index.

This paper outlines the design strategy of graphics compiler and basic properties of the hierarchical space indexing method, and presents a prototype implementation example.

II. A GRAPHICS COMPILER

A. Basic Structure

The graphics compiler is composed of four major units. One is an image-capturing unit. This unit is similar to those of conventional one. The second is a compiling unit that translates the captured image data into hierarchical space indexed table objects. The third is a database unit, and the last is an image-recovering unit that displays a target image onto a computer screen according to the instructions given by an observer. **Figure 1** shows a schematic diagram of the graphics compiler.

B. Space Indexing

In order to work out the graphics compiler, the space-indexing method is of paramount importance. This is because the graphics compiler translates the captured image data into a set of primitive geometrical objects. One of the ways to carry out this translation is to employ a *space indexing method*, which constitutes a tree structure.

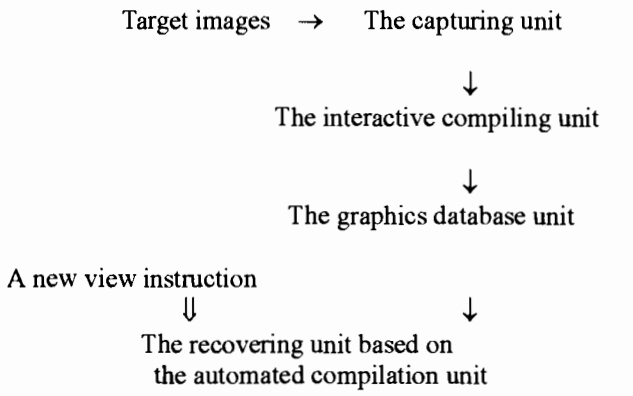


Fig. 1 A schematic diagram of the graphics compiler.

Indexing methods generally have been studied for the context searching problems. In such cases, quick access to the data to be selected from a large amount of data is of extremely importance. In database oriented management, efficient access and retrieval of stored data have been widely studied. In dealing with a large amount of data, it is effective to classify the search spaces into subspaces, recursively. This makes it possible to skip the data not containing interest items. A binary tree data structure is a typical one [2].

The key idea of the *hierarchical space indexing method* is to apply the indexing strategy to a 3-dimensional object for capturing and processing image data. The strategy is based on the precedence rules of parsing the topology of objects represented in a cellular spatial structure. Graphics objects are algebraic topologically modeled very simply as cellular design. We take *cellular spatial structures* as the foundation of the modeling. As a cell, we take an n-dimensional topological ball called an n-cell in terms of algebraic topology. For example, a 0-cell is a vertex, a 1-cell is an edge, and a 2-cell is a surface. When a surface is closed, it becomes a 3-cell that is a topological ball. Each hole is a 2-cell. Any cells are attached through a *cell-attaching map.* In a cellular structured space, the space thus obtained by attaching cells are called an *attaching space*, an *adjunction space* or an *adjoining space* [8-10].

The object shapes topologically defined have to be visualized and made into physically realizable forms. A three level hierarchical architecture for shape modeling is an excellent architecture: 1 the topological layer, 2 the geometrical layer and 3 the visualization layer. Defining shapes actually is carried out interactively through this architecture. It starts from the topological layer to define topological shapes, and visualizes the shapes in the visualization layer after giving temporary geometrical shapes to the objects in the geometrical layer. It goes back to the topological layer to change the topological definition as needed. In the geometrical layer, a 1-cell, that is an edge in the topological layer, can be a line or a curve; a 2-cell can be a plane or a curved surface, and a 3-cell a curved body.

A limited case of cellular spatial structure representations is a class of *boundary representations* usually abbreviated as *B-reps*. Generally, any object can be identified in terms of four geometrical entities in B-reps: a vertex, an edge, a surface and a body [3]. A body is the part inside an object enclosed by surfaces. A surface is a plane enclosed by its boundaries. An edge is composed of two vertices as its two end points. We have to impose a restriction that no object should be self-intersecting and inconsistency should be detected. Further, we have to hold the following conditions:

1) *Any two surfaces are either disjoint, meet at a common vertex or along with a common edge.*

2) *Any two edges are either disjoint or meet at one of their end vertices.*

3) *Any edge and surface are either disjoint, meet at a common vertex, or the edge is part of the surface boundary.*

In order to introduce a tree structure for indexing, a cube called the *entire universe* U should be defined. An arbitrary object to be represented must be in this entire universe U corresponding to the root of the tree. The entire universe U is subdivided into eight subspaces called *octants* corresponding to eight children in the tree. Each of the octants is recursively subdivided into eight octants. This means that every non-terminal node has eight children. Thus, an octal tree structure is usually referred to an *oct-tree* [3-7]. As opposed to pure oct-tree where subdivision recurs until either an octant becomes completely inside the object or completely outside the object within available resolution, we halt the subdivision when an octant becomes simple enough. The following definitions concerning with the index tree give the *precedence rules of parsing the topology of a graph-theoretically represented object shape*:

TYPE 1 Boundary -- An octant includes the boundary of the object. This type is denoted as P (partial).

TYPE 2 Non-boundary -- An octant does not include the boundary. This type is either completely inside the object denoted as 1 (full), or completely outside the object denoted as 0 (empty). This type never requires any more subdivision.

Each boundary octant P is further classified into two sub-types:

TYPE 1-1 Dividing -- The space in the octant is still dividable and requires further subdivision. The octant is denoted as T representing an oct-tuple (o, o, o, o, o, o, o, o), where the order o refers to the position of the octant.

TYPE 1-2 Non-dividing -- The space is simple enough to represent the cubical subset of the world and no more subdivision is necessary. This octant is denoted as L.

Three kinds of the non-dividing spaces are:

(1) An octant includes one vertex. This octant neither includes the edge not having the vertex as its endpoint nor the face not having the vertex as its boundary point. Such an octant is called a *vertex space* and denoted as V. Corresponding node to this space in the tree is called a *vertex node*.

(2) An octant does not include the vertex but one edge fraction, which is a part of an edge. This octant does not include the face not having the edge as its boundary. Such an octant is called an *edge space* and denoted as E. The corresponding node to this space in the tree is called an *edge node*.

(3) An octant does not include any vertex as well as edge but includes one edge fraction, which is a part of an edge. Also, this octant does not include the face not having the edge as its boundary. Such an octant is called a *surface space* and denoted as S. The corresponding node to this space in the tree is called a *surface node* and denoted as S. **Figure 2** shows the typical examples of the above-defined spaces.

Each geometrical object consists of a node that is a point (a 0-cell e^0), an edge that is a line (a 1-cell e^1), a surface (a 2-cell e^2) or a body (a 3-cell e^3). All cells are open. Further, they are hierarchically organized. As explained earlier, this hierarchically organized structure serves exactly as the basis of the *hierarchical space indexing*

method. In a cellular spatial structure,

$$X^0 \subseteq X^1 \subseteq X^2 \subseteq X^3,$$

where X^i (i = 1, 2, 3) is a discrete space whose element is e^i. X^i is constructed from X^{i-1} by attaching a disjoint union of e^i to X^{i-1} via a continuous and surjective map called *an attaching map*

$$f_i: \partial e^i \rightarrow X^{i-1}$$

such that

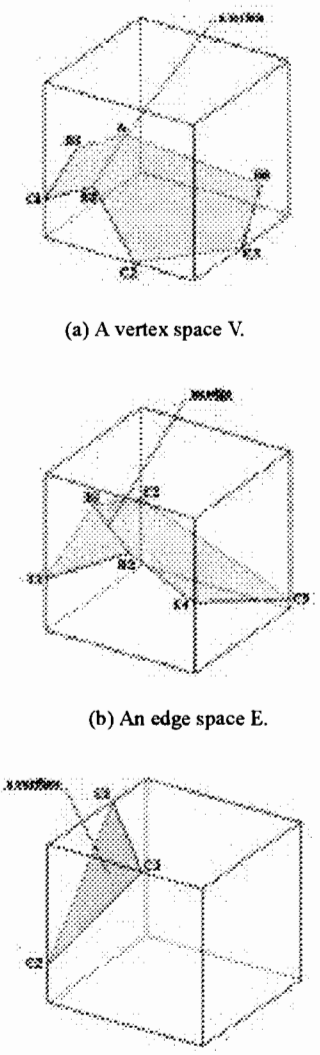
$$X^i = X^{i-1} \sqcup \cup_k e^i_k / \sim = X^{i-1} \cup_{fi} \cup_k e^i_k / (x \sim f_i(x) \mid x \in \partial e^i),$$

where ∂e^i is the topological boundary of e^i, and k is a natural number [10]. This method of constructing a cellular structured space is generally called *cell composition*. The reverse is *cell decomposition*. Basically, the graphics compiler performs cell decomposition. The type of graphics compiler presented in this paper is a special case based on the space index method. It produces a *intermediate meta-code set* that consists of 0-, 1- and P- octants. Each P-octant is a T- or L- octant, and the T-octant is further decomposed until the result become L-octants. Each L-octant is a V-, E- or S-space. $\sqcup$ denotes a disjoint union and often a + symbol is used instead (sometimes it is called "exclusive or"). $\sim$ is an equivalence relation. An *equivalence relation* is simply a relation that is reflexive, symmetric and transitive. It can be a set theoretical equivalence relation, a homotopic equivalence relation, a topological equivalence relation or a geometric equivalence relation. The transitivity divides the space into a disjoint union of subspaces called *equivalence classes*.

C. *Implementation*

Practical implementation requires a series of following steps. At first, each of the captured graphics data is compiled into the object cells by the hierarchical space indexing method. The data captured at the different viewpoints of the same target are similarly compiled and continued until the object cells do not grow up. This means that the stored object cells contain a set of complete 3-dimensional information of the target object. Using the stored object cell data, the 3-dimensional target object can be projected onto the 2-dimensional screen from any arbitrary viewpoints. Thus, application of the graphics compiler to the captured graphics data makes it possible to identify the 3-dimensional object in the indexing table but also makes it possible to reproduce the target object as seen from any viewpoints. When the hardware chips realize this graphics compiler, it leads us to a new graphics technology.

III. AN EXAMPLE



(a) A vertex space V.

(b) An edge space E.

(c) A surface space S.

Fig. 2 The octant spaces V, E and S as the intermediate meta-codes.

In this example, we employ an object-centered motion capturing method. It differs from the traditional camera-centered motion capturing method.

The first is the introduction of our image-capturing concept. Most of the computer image data capturing methodology is based on the idea such that captured images are memorized in a computer data storage system and presents the processed images onto a computer screen according to human instructions. That is basically the digitized photograph concept. However, when we look into the way a human captures images, a human captures the images conceptually as objects not as images *per se*.

Let us consider a simple example: a set of cups on a table. A human recognizes this image as a combination of a cup and a table. Further, the cup is identified by its characteristic shape, and the table is identified by its four leg structure as the shape characteristics. Furthermore, the cup and the table are regarded as a set of geometrical objects. Thus, human image capturing ability converts the image data into these object data and classifies them in a hierarchical manner. By means of the memorized object data and experience information, human is able to imagine the backside of the table shape and the other details. It is difficult to realize this human capturing ability by computers exactly. However, when we can derive 3-dimensional object data from captured static and dynamic target images, it is possible to recover the target objects viewed from any points by computers. This is our basic image capturing objective.

Secondly, we outline a simple and practical implementation of our image capturing technique. The image projected onto each of the human eyes is 2-dimensional. A human recognizes the target 3-dimensionally. This is because a human is equipped with the two eyes with the characteristic fixed distance. This obviously means that two cameras placed at a given distance can recognize 3-dimensional targets. However, computers initially lack memorized object data, experience information and the standard measure for estimating the relative target sizes. Thereby, in order to capture complete 3-dimensional object data, computers require 3 frames of images viewed from 3 distinct positions.

Let an arbitrary vertex point in an *i-th* 2-dimensional image data be denoted by $(p_i\Delta x, p_i\Delta y)$, then we project a unit vector (1, 1, 1) in a 3-dimensional space onto the 2-dimensional *i*-th plane coordinate, where the origin (0, 0, 0) in the 3-dimensional coordinate should coincide with the 2-dimensional origin (0, 0). Considering a relationship between the 3- and 2- dimensional coordinate systems in **Figure 3**, we have a relationship:

$$(1, 0, 0) \rightarrow (U_x x_i, U_x y_i),$$

where $U_x x_i$ and $U_x y_i$ are the referred unit lengths on the X_i- and Y_i- 2-dimensional coordinate axes, respectively. Similarly, for the vertices (0,1,0) and (0,0,1) in 3-dimensions, we have the following relationships:

$$(0, 1, 0) \rightarrow (U_y x_i, U_y y_i)$$

and

$$(0, 0, 1) \rightarrow (U_z x_i, U_z y_i).$$

In these relationships, $U_y x_i$, $U_y y_i$, $U_z x_i$ and $U_z y_i$ are the referred unit lengths on the X_i- and Y_i- 2-dimensional coordinate axes, respectively. Further, the subscripts x, y and z denote the 3-dimensional x-, y- and z- axes, respectively. By means of these relationships, it is possible to derive a set of system equations for the 3 distinct images.

Thus, for the 3 distinct images, a following set of system equations holds:

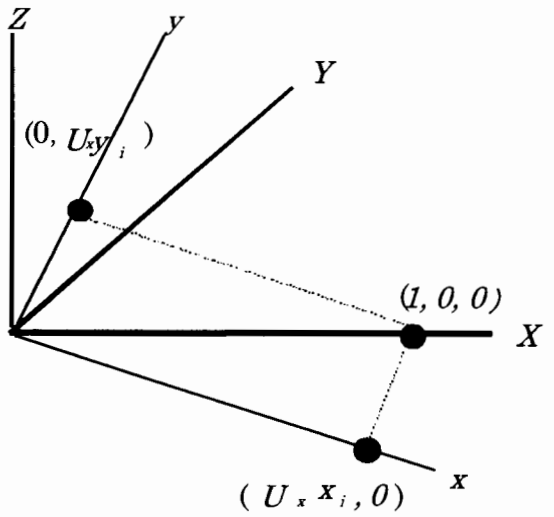


Fig. 3 A relationship between the 2- and 3- dimensional coordinate systems.

$$\begin{bmatrix} p_1\Delta x \\ p_1\Delta y \\ p_2\Delta x \\ p_2\Delta y \\ p_3\Delta x \\ p_3\Delta y \end{bmatrix} \begin{bmatrix} U_x x_1 & U_y x_1 & U_z x_1 \\ U_x y_1 & U_y y_1 & U_z y_1 \\ U_x x_2 & U_y x_2 & U_z x_2 \\ U_x y_2 & U_y y_2 & U_z y_2 \\ U_x x_3 & U_y x_3 & U_z x_3 \\ U_x y_3 & U_y y_3 & U_z y_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \qquad (1)$$

or

$$\mathbf{P} \quad C\mathbf{Q}, \qquad (2)$$

where the elements X, Y and Z are the exact 3-dimensional coordinate values to be evaluated. In order to obtain the unique X, Y and Z from Eq.(1), we have to impose the following constraint:

$$1 \quad \sqrt{(U_x x_i)^2 + (U_x y_i)^2},$$

$$1 \quad \sqrt{(U_y x_i)^2 + (U_y y_i)^2}, \text{ and}$$

$$1 \quad \sqrt{(U_z x_i)^2 + (U_z y_i)^2}, \qquad (3)$$

where i 1, 2, 3.

Using the constraint (3), it is possible to evaluate the vertex coordinate values X, Y and Z in the 3-dimensional coordinate. Thus, we can obtain the most fundamental indexing term of the target.

Figure 4 shows the examples of a series of the captured images. The target object in this example is a simple electronic calculator and our purpose is to get a set of compiled object data of this calculator in terms of vertices, edges, surfaces and bodies. The small dots on the captured images in **Figure 4** refer to the selected vertices in order to obtain the compiled object data. Also, a set of orthogonal lines denotes an attached 3-dimensional coordinate system. These 3-dimensional rectangular coordinate axes are fixed to the particular target position. Also, an A4 size curved paper sheet is included in the **Figures 4(a)** and **4(c)** as an extra rough reference of the target physical dimensions.

Using these captured images and the relationships (1)-(3), we calculated and compiled the graphics data. After obtaining the graphic object data, we recovered the electronic calculator image. **Figure 5** shows one of the recovered calculator images. As shown in **Figure 5**, the shape of target electronic calculator is roughly reproduced. This roughness reflects the fuzzy nature of human pattern recognition and the ambiguity in the course of interactive generation of vertex points via graphics compilation. We can refine it by utilizing the selected object-centered information such as object shape information and image analysis information such as wavelet analysis information in generating the vertex points. By semi-transparently overlaying the reproduced images on the original images, they are further interactively refined to any extent as required. *Refinement by interactive semi-transparent overlay* is being pursued as another research theme.

Thus, we have worked out a prototype graphics compiler and have made a basic verification of our methodology proposed here. The selected target object is relatively simple and is only to illustrate our image capturing and reproducing method at the most primitive level up to 3-cells as we have discussed so far.

IV. CONCLUSIONS

We have developed a new methodology named a graphics compiler. An engine of the graphics compiler based on the space indexing approach has been described. After obtaining a set of complete geometrical object cells by the graphics compiler, the target object shapes are recovered in a 3-dimensional manner and projected onto a 2-dimensional screen.

The graphics compiler is an innovative methodology that translates the captured image data into a set of reusable basic geometrical objects containing the complete 3-dimensional graphics information. Hence, the original target object can be visualized on demand as seen from any sides.

To be more specific, after capturing a graphics target images dynamically, a series of captured image data is compiled and then the complied objects are stored in a graphics database. A sequence of images of the target objects captured at different time instances makes it possible to produce a set of compiled shape information of the objects containing a complete 3-dimensional geometrical shape information that also inherits topological properties in cellular structured spaces. Thus, the captured shapes are turned into reusable components to build new graphics contents freely by assembling them.

When a hardware chip realizes this graphics compiler as a system-on-chip, it leads us to a new graphics technology.

This paper has outlined the designing strategy of the graphics compiler and the basic properties of the hierarchical space indexing method. A simple example has shown our new methodology for image data capturing. Another example to illustrate the reusability of the compiled object data is under preparation. Further research is in progress for wide applications including sporting motion capturing and dynamic scene capturing.

V. REFERENCES

[1] K. Fujimura and T. L. Kunii, "A hierarchical space indexing method," *Proceedings of Computer Graphics Tokyo '85,* edited by T. L. Kunii (Springer-Verlag 1985) pp.21-33.

[2] J. L. Bentley, "Multi-dimensional binary search tree used for associative searching," *CACM* Vol.18, No.9 (1975) pp.509-517.

[3] K. Yamaguchi, T. L. Kunii, K. Fujimura and H. Toriya, "Octree related data structure and algorithms," *IEEE CG & A,* Vol.3 (1983).

[4] K. Fujimura, H. Toriya, K. Yamaguchi and T. L. Kunii, "An enhanced Oct-tree data structure and operations for solid modeling," *Proceedings of NASA Computer-Aided Geometry Modeling,* Hampton, Virginia, April 20-22 (1983) pp.279-287.

[5] L .J. Doctor and J. D. Torborg, "Display techniques for octtree encoded object," *IEEE CG & A,* Vol.1, No.1 (1981) pp.29-38.

[6] C. L. Jacking and S. L. Tanimoto, "Oct-trees and their use in representing 3-dimensional objects", *Computer Graphics and Image Processing,* Vol.14, No.3 (1980) pp. 249-270.

[7] D. Meagher, "Geometric modeling using oct-tree encoding", *Computer Graphics and Image Processing,* Vol.19, No.2 (1982) pp.129-147.

[8] T. L. Kunii, "The 3rd Industrial Revolution through integrated intelligent processing systems", *Proceedings of IEEE First International Conference on Intelligent Processing Systems,* October 28-31, 1997, Beijing, China (ICIPS '97), pp. 1-6, The Institute of Electrical and Electronics Engineers, New York, NY, U.S. A.

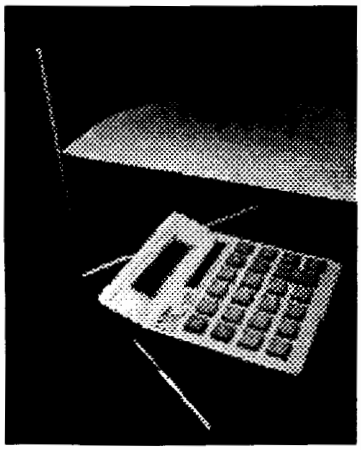
[9] T. L Kunii and T. Wachi,, "Topological dress making as fashion media modeling", *Proceedings of MultiMedia Modeling Conference*

(MMM'98), October 12-15, 1998, Lausanne, Switzerland, in press, IEEE Computer Society Press, Los Alamitos, California, U.S. A.

[10] T. L. Kunii, "Graphics with shape property inheritance", *Proceedings of Pacific Graphics '98*, (October 26-29, 1998, Singapore), in press, IEEE Computer Society Press, Los Alamitos, California, U.S. A.

(a) The 1st captured image.

(b) The 2nd captured image.



(c) The 3rd captured image.

Fig. 4 An example of a series of captured images.

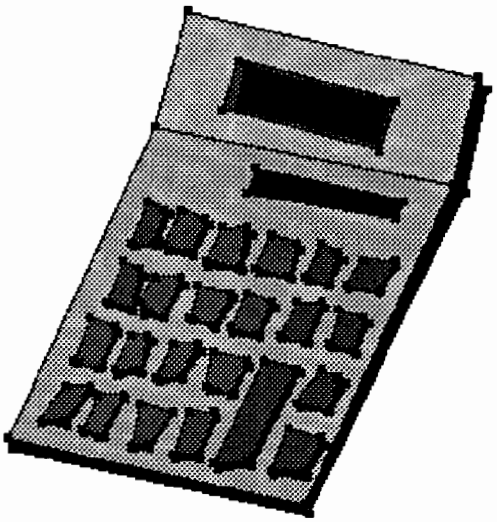


Fig. 5 An examples of a reproduced image.