The Visual Computer

Laying out objects with geometric and physical constraints

Mikio Shinya and Marie-Claire Forgue

NTT Human Interface Laboratories,
3-9-11 Midori-cho, Musashino-shi, Tokyo 180, Japan
email: shinya@nttarm.ntt.jp

Modeling scenes involves two tasks: *object modeling* and *object layout.* This paper focuses on object layout and proposes a constraint-based approach which yields a powerful object layout environment. The approach uses collision detection and physical simulation to ensure geometric and physical consistency of the resulting scenes, such as no interpenetration, and physical stability of the objects. A prototype system is developed, providing six basic operations; PUT, PUSH/PULL, TURN/TILT, PICK-UP, TRANSLATE, and ROTATE. The system: ensures geometric and physical consistency; provides easy-to-use operations analogous to object placement in real life; allows two-dimensional control easily specified by mouse. Interactive speed is achieved on graphics workstations by using rasterized collision detection and simple quasi-static motion simulation. The system is interfaced to modeling/rendering/animation systems, and realizes an integrated environment for object modeling, object layout, rendering, and animation. We describe several scenes that have been modeled using the system and argue that these experiments confirm that the scene modeling task is greatly simplified by our constraint-based approach.

Key words: Object layout – Geometric and physical constraints – Collision detection

Correspondence to: M. Shinya

1 Introduction

One of the main tasks in computer graphics is to realistically model complex scenes. Modeling scenes involves two different subtasks: *object modeling* and *object layout.* In object modeling, the shapes and other attributes of individual objects are defined, and in object layout, the defined objects are placed so as to construct the desired scenes.

Although many proposals have been made to facilitate object modeling, very little work has addressed object layout. Obviously, the functions desirable for object layout differ significantly from those needed for object modeling; for instance, in object layout, geometric and physical consistency must be maintained (e.g., interpenetration among objects should be avoided), while in object modeling, free transformation and deformation is desired (e.g., penetration should be allowed when building shapes). Despite these differences, objects are usually laid out in object modeling environments by manually adjusting their positions with careful mouse manipulation, or by numerically specifying transformations from a keyboard. This is a time-consuming and error-prone process, which imposes limitations on the complexity and realism possible in computer-generated scenes. For example, computer-generated scenes of laboratories are always clean and well-organized, unlike most real graphics labs.

In this paper, we propose an object layout system which yields a powerful environment for modeling complex scenes. This system applies geometric and physical constraints to object manipulation. Its advantages are as follows:

- It ensures consistency in the scene models produced.
- It supports easy-to-use operations analogous to real life manipulation, e.g., push/pull, put pick up, etc.
- It reduces the degrees of freedom on transformations (typically to two dimensions) to allow easier control by mouse.

A difficulty with this approach is the need for collision detection, a potentially costly process. We solve this problem by adopting a fast collision detection algorithm based on rasterization (Shinya and Forgue 1991), enabling objects to be manipulated as interactive speeds on graphics workstations.

2 Related work

Geometric and physical consistency have been mainly investigated in physically based approaches. Most of these have been aimed at generating realistic motion in computer animation, but some of the results and methods are also applicable to object modeling.

Witkin et al. (1987) and Barzel and Barr (1988) have developed constraint-based methods for object deformation and assembly, as well as animation. In these methods, users specify their desired final states using energy constraints or dynamic constraint forces, and objects are deformed or transformed according to dynamic equations. Terzopoulos et al. (1987, 1988), and Platt and Barr (1988) have modeled deformable objects using physical energy or user-specified energy.

There have been many studies on rigid body dynamics. Among them, Moore and Wilhelms (1988) and Hahn (1988) simulated the impact dynamics of rigid bodies using collision detection. Baraff analyzed the physics of rigid bodies in resting contact (Baraff 1989), including curved objects (Baraff 1990) and objects with friction (Baraff 1991), and then extended the methods to nonpenetrating flexible bodies (Baraff and Witkin 1992).

3 Interactive object layout

This section characterizes the concept of interactive object layout. The user's task is to model complex but consistent scenes by laying out objects defined during object modeling. The objective of the layout system is to provide an interactive, easy-to-use environment where geometric and physical consistency is automatically ensured. Here, geometric consistency means that objects do not penetrate each other, and physical consistency means, for example, that objects placed at some position remain at rest.

To achieve such an environment, we propose to restrict object transformations (rotation/translation) in ways that preserve consistency. For example, when an object (object 1) is in area-contact with another object (object 2), as shown in Fig. 1, translation with a $-\vec{N}$ component is prohibited and rotation along $\vec{N}$ is only allowed such that penetration is prevented. In this paper, such restrictions are called *constraints.*[1]

To further simplify the manipulation process, we also propose easy-to-use operations analogous with real life manipulation, such as push/pull, turn/tilt, put, pickup, and so on. During these operations, collisions and interferences are checked to avoid penetrations and, when collision is detected, geometric constraints are calculated to determine the next motion. In some operations, physical simulation is also applied to ensure that objects achieve static equilibrium.

There are many similarities between our work and physically based approaches for computer animation, but there are also significant differences. Firstly, only the final state of objects is important in layout, while the process (movement) itself is important in animation. This feature considerably simplifies the problem with dynamics, as shown later. Secondly, interactivity is a critical requirement in layout operations, unlike in animation. The most time-consuming part of the operations is collision detection. We use the fast collision detection algorithm based on rasterization described in an earlier paper (Shinya and Forgue 1991) to achieve interactive execution speed.

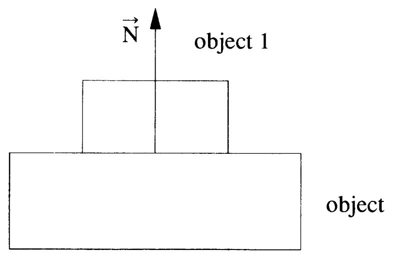
4 Basic operations and data structures

4.1 Basic operations

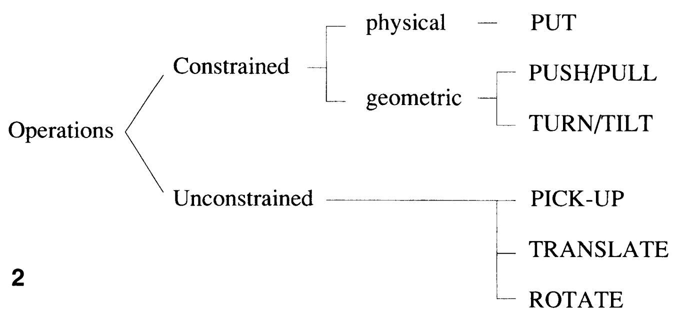
Operations can be classified according to how they use constraints, as shown in Fig. 2. We have implemented the six basic operations listed in the figure.

PUT: This operation simulates putting one object onto another object. Until it collides with other objects, the object under operation translates in the direction of gravity. When collision is detected, physical simulation is applied to calculate the object's motion until it reaches

[1] Our notion of constraint is slightly different from the "constraints methods" of physically based approaches, where constraints are explicitly specified by users as their goals. In layout, constraints are rather passive, and are automatically obtained from geometric calculation and physical simulation.



1



2

Fig. 1. Constraints

Fig. 2. Basic operations

equilibrium. An example is given in Fig. 3a and the procedure is outlined in Fig. 3b.

PUSH/PULL: This operation translates the object so as to trace (slide on) the surface with which it is in contact. In this operation, the object is first translated in the tangent direction of the contacting surface, and then interference among objects is checked (Fig. 4a). If the moving object is still in contact, the operation is continued. If the object interferes with other objects, the system decides whether to continue tracing the surface or give up. Currently, this decision is made according to the interference distance in the z-direction (d_z). For example, in the situation shown in Fig. 4b, the distance d_z is too large and the object is pulled back to the collision point. In the situation shown in Fig. 4c on the other hand, the object is pushed up to the surface to continue PUSH/PULL. The decision is made by comparison with a specified threshold value. If no contact is detected, the system decides whether to drop the object. In the example shown in Fig. 4d, the distance is small and the object is translated in the $-z$-direction until contact is made, while the object is pulled back in the example shown in Fig. 4e. The procedure for PUSH/PULL is outlined in Fig. 4f.

TURN/TILT: This operation rotates the object while maintaining user-specified constraints. Users interactively specify contact points to be held, and the system computes possible rotations. In Example 1 of Fig. 5, the area contact e_1-e_2-e_3-e_4 is specified, and thus the direction of the rotation axis is calculated as the normal $\vec{N}$. The center of rotation (a point on the rotation axis) can be either user-specified or be the default position (the center of the contacts). In Example 2, the line contact, e_1, is selected, and the rotation axis becomes e_1. In Example 3, the point contact, p_1, is specified as the constraint. In this case, any axis passing through point p_1 is acceptable.

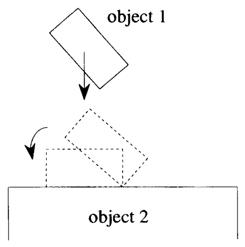
PICK-UP: This operation translates the object in the direction of the normal vector of the contact surface. It also deletes the logical connection with the objects previously in contact.

ROTATE and **TRANSLATE:** These operations freely rotate and translate the object just as in conventional object modeling systems.

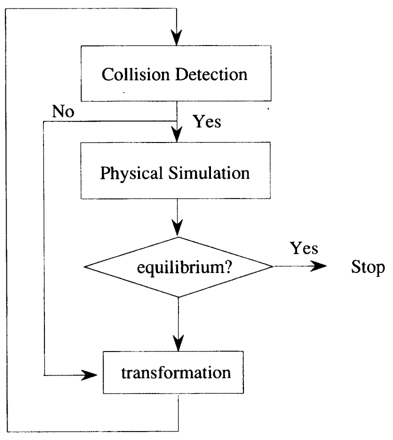
4.2 Data structures

The main differences between our system and conventional object modeling systems, as far as data structures are concerned, lie in the way contacts are represented. When an object is moved, it is expected that everything that is in contact with the object also moves. To support this, the relationship between contacting objects must be represented.

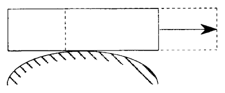
Object is the smallest unit that can be operated on. In the current implementation, only polygonal objects are available, and curved objects are polygonized. To make contact analysis easier,



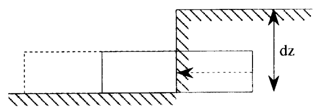
3a



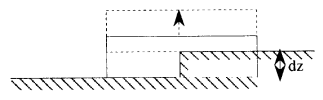
3b



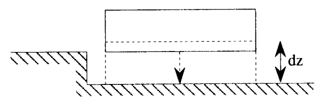
4a



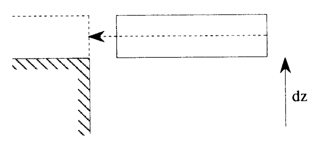
4b



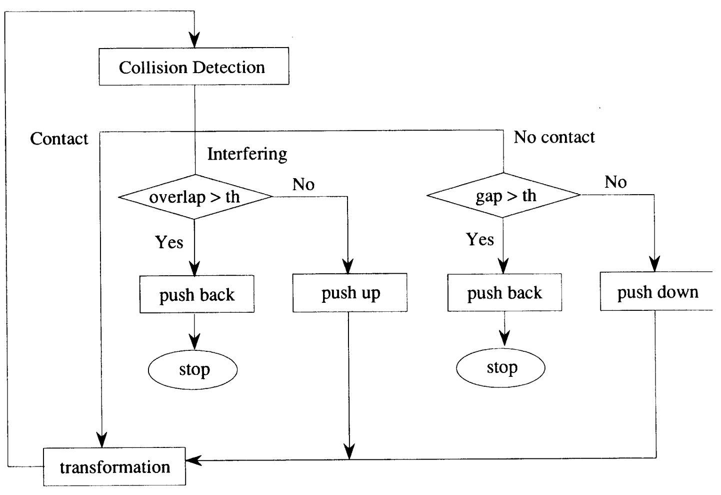
4c



4d



4e



4f

Fig. 3a, b. The PUT operation

Fig. 4a–f. The PUSH/PULL operations

a winged-edge-type structure (Baumgart 1974) is adopted, allowing bidirectional access among objects, faces, edges, and vertices. *Object* also has a pointer to *Contacts* to refer to contact information.

Load-support list is a list structure representing contacting objects. Typically, one object (the load object) is supported by other objects (support objects). In our system, the object which is PUSHed, PUT, TURNed, etc., is assumed to be the load object. To describe the load-support relationship, the list contains a field pointing to the support object and the load object. The convex-hull of the contact points is also stored in the list, which can be used in stability checking. The load-support list for the objects in Fig. 6a is illustrated in Fig. 6b. All load objects of a support object can be identified using this list structure. This allows them to be operated on together.

Contacts represents information on contacts necessary to calculate constraints. It contains the convex-hull of contact points and the topologies of contact points, edges, and faces. It also contains the contact type, i.e., area-contact, line contact point-contact, etc., to simplify constraint calculation.

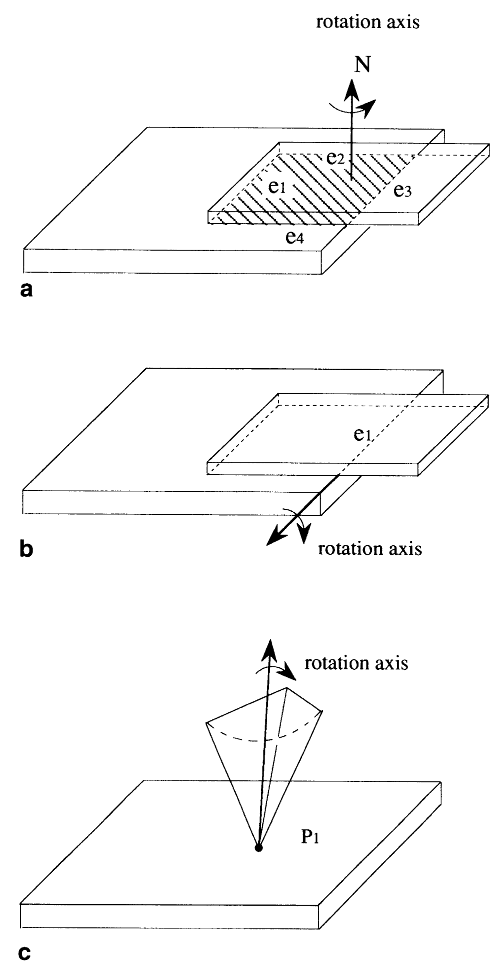


Fig. 5a–c. The TURN/TILT operations

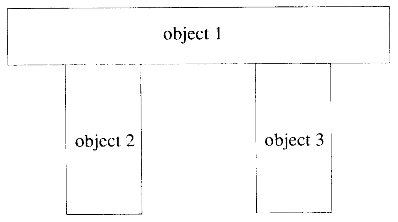
5 Algorithms

5.1 Interference detection

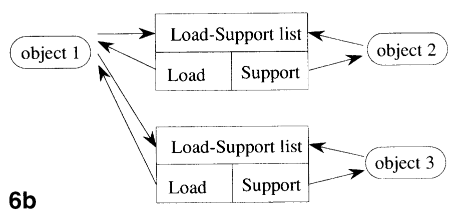
Detecting collisions and intersections is the most time-consuming function, and thus, the most critical process. Geometric algorithms like that of Boyse (1979) work fine for simple objects, but are too slow for complex objects. To achieve interactive speed for complex objects, we use the z-list interference detection algorithm (Shinya and Forgue 1991) The algorithm first rasterizes the projection and calculates the z-values, just like the z-buffer visible surface algorithm (Fig. 7a). Here, the projection can be in any desired direction. For interference detection, z-values and pointers to the corresponding objects/faces are saved in a z-list for each pixel (Fig. 7b). Sorting the z-value in the z-lists allows the detection of overlapping objects, and thus, interference (Fig. 7c). In the example shown in Fig. 7, interference is detected at pixels x_1 and x_2. The advantages of this algorithm are:

- Fast execution comparable to the display speed
- Time complexity linear in the number of faces
- Robustness due to limited use of topological information
- Easy implementation

Hardware z-buffering generally does not provide full z-list information. To take advantage of graphics hardware, we implemented the algorithm in three steps (Fig. 8a). First, a bounding box check is done between the moving object and other objects. If overlapping bounding boxes are detected, a simplified z-list interference detection is applied to the moving object and the objects that fail the bounding box test. In this second step, only the maximum and minimum z-values of the objects are stored at each pixel by hardware z-buffering. In the example in Fig. 8b, only z_1, z_3, z_4, and z_6 are saved at the pixel x_3, and the possibility of interference is detected. Note that the simplified z-list test provides exact results in two important cases: first, for convex objects, second for a translating object when projected in the direction of movement. If the second step detects the possibility of interference, complete z-lists are made by software z-buffering. In the example, intersection is ruled out at the third step.
Selection of the projection direction and projection volume (clipping planes) is important in practice. We adopt the bounding box of the moving object as the projection volume in the second step. In the third step, we reduce the projection area according to the pixels for which interference is detected in the second step. The projection direction is selected so as to be close to the direction of movement. For example, the direction of gravity is selected in the PUT operation, while in the PUSH/PULL operation, the translation direction is selected.



6a



6b

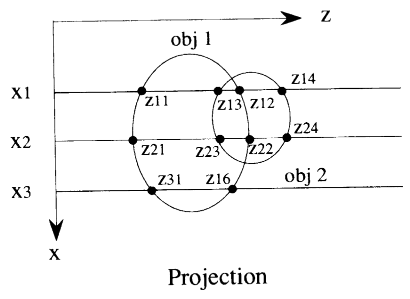
$x_1 \rightarrow (z_{11},1) \rightarrow (z_{12},1) \rightarrow (z_{13},2) \rightarrow (z_{14},2)$

$x_2 \rightarrow (z_{21},1) \rightarrow (z_{22},1) \rightarrow (z_{23},2) \rightarrow (z_{24},2)$

$x_3 \rightarrow (z_{31},1) \rightarrow (z_{32},1)$

z-lists (z-value,object-id)

7b



7a

$x_1 \rightarrow (z_{11},1) \rightarrow (z_{13},2) \rightarrow (z_{12},1) \rightarrow (z_{14},2)$

$x_2 \rightarrow (z_{21},1) \rightarrow (z_{23},2) \rightarrow (z_{22},1) \rightarrow (z_{24},2)$

$x_3 \rightarrow (z_{31},1) \rightarrow (z_{32},1)$

sorted z-lists

7c

Fig. 6a, b. The load-support list

Fig. 7a–c. The *z*-list

5.2 Collision detection and contact calculation

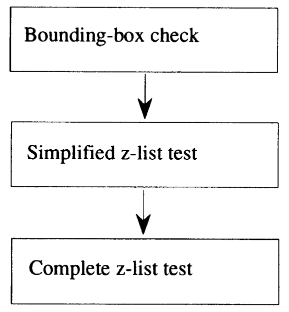
The system performs collision detection by looking for interference in each frame. This may miss slight collisions in the intervals between frames, but we can ignore this because it rarely affects the consistency of the final state. When interference is detected by the *z*-list procedure, the collision time is calculated by intersection calculation (Boyse 1979). This calculation is done only for the interfering faces to save time. According to the collision time, the moving object is translated and/or rotated to the assumed contact position. Another *z*-list interference test is made to ensure that the objects are in contact. When new interference is detected, the collision time is recalculated for the detected interfering faces. When the objects are in contact, the topology of the contacts (point contacts, line contacts, and area contacts) is calculated through a geometric calculation. A diagram of the procedure is shown in Fig. 9.

The fast interference detection algorithm described in Sect. 5.1 allows fast execution of collision detection. Since the contact time calculation become expensive when a large number of faces are processed, we apply the process only to the detected interfering faces. The number of interfering faces is usually small, and thus the computation cost is kept reasonable even for complex scenes, as shown in Sect. 7.

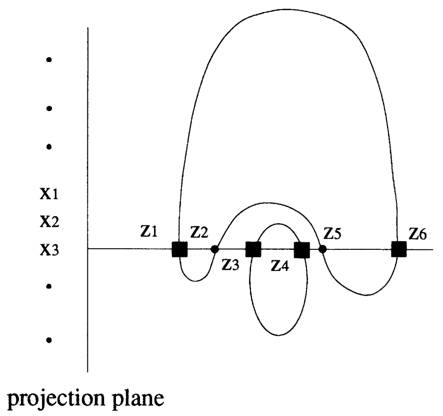
5.3 Physics

Computing exact solutions to rigid-body dynamics is a very difficult task, in fact, it is NP-hard in the worst case (Baraff 1989). Fortunately, layout tasks do not necessarily require exact physical simulation because simpler operations can produce similar results. Thus, we simplified the problem in the following ways:

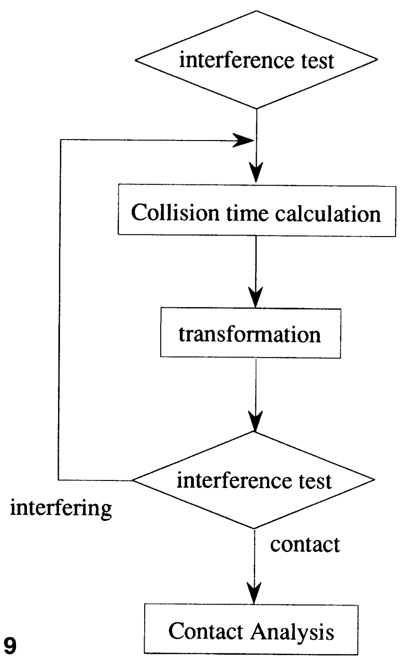
Localization. In most layout tasks, interest is focused on the object currently being moved.



8a



8b



9

Fig. 8a, b. Interference detection

Fig. 9. Collision detection

Consequently, we only consider the physical behavior of the moving object, and the other objects are assumed to be fixed. This allows us to bypass the NP-hardness problem.

Quasi-static motion. In animation, full dynamic simulation is desired to generate realistic motion, such as object bouncing. In layout, however, we do not need realistic bouncing, and are more interested in the simulation of smooth motion. Thus, we assume that the velocity of the object is zero in each frame, and that the object moves according to the direction of the calculated linear and angular acceleration. Such motion is almost static and is called *quasi-static motion.*

Perfect friction. Since inclined frictionless surfaces cannot realistically support objects, we must assume that friction forces are present. If objects slide on surfaces, the problem becomes ill-conditioned under the quasi-static condition because the direction of friction force is the direction of movement, which is always zero in this case. To avoid this situation, we do not allow objects to slide; perfect friction is assumed.

With these simplifications, the physical simulation for the PUT operation can be achieved by the following simple algorithm:

Case 1 – *Point contact*: When contacting at a point P (Fig. 10), the object rotates along the line l, which passes through P and is perpendicular to both PG and the z-axis, where G is the center of gravity and the z-axis is the opposite direction to gravity.

Case 2 – *Line contact*: When contacting at two points P_1 and P_2, or on an edge P_1P_2, there are two cases depending on the relative position of the center of gravity. If the foot of the perpendicular of G to P_1P_2 lies between P_1 and P_2, the object rotates along P_1P_2 (Fig. 11a). Otherwise, one of the vertices closer to G becomes the only contact point, and the object follows the rule for point contact. In the example of Fig. 11b, P_2 becomes the contact point.

Case 3 – *Area contact*: When contacting more than three points which are not on the same line, the convex-hull of the contact points is calculated. If the projection of G in the $-z$-direction is inside the convex-hull, the object is stable (Fig. 12a).

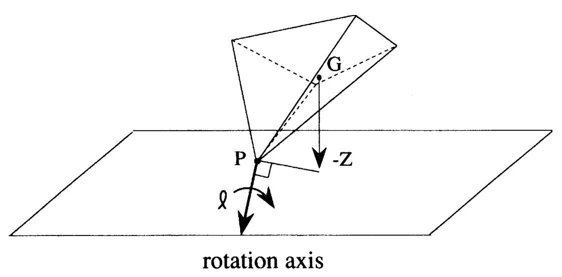


Fig. 10. Point contact

Otherwise, if the closest point to G on the convex-hull is one of the object's vertices, it becomes the only contact point and the rule for point contact is followed (Fig. 12b). If the closest point is on one of the edges, this edge becomes the rotation axis (Fig. 12c).

6 Implementation

We implemented the object layout system on an IRIS 4D-series workstation. The main features of the implementation are described in this section.

6.1 System configuration

The system consists of four main parts: Window/Command Manager, Data Manager, Operation Manager, and I/O Manager, as shown in Fig. 13. The flow of control is simply determined by events: the Window/Command Manager detects an event and calls other Managers according to the current command. After the completion of the command, control goes back to the Window/Command Manager who handles the next event.

Window/Command Manager: This module deals with window display, menu management, and other events like mouse action[2], keyboard input, etc. It also analyzes commands and calls other Managers to perform appropriate commands.

[2] Interaction with buttons, sliders, etc., is managed using *forms*, software developed by Mark H. Overmars.

Data Manager: This module maintains object data and the load-support lists. Its main tasks are:

- To update object data based on results of operations, including the locations of objects, load-support lists, contacts, etc.
- To delete object data when so ordered
- To append object data when new objects are read from files

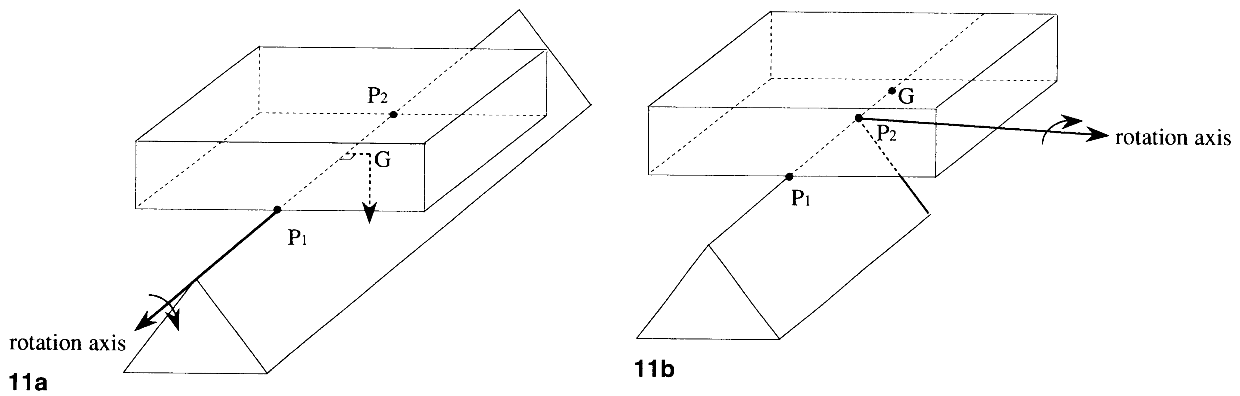
Operation Manager: This module executes basic operations like PUT, PUSH/PULL, etc. This includes collision/interference detection, quasi-static motion simulation, and contact analyses. The resulting contacts are returned to the Data Manager for updating.

I/O Manager: This module manages file I/O, including data conversion from/to object modelers and renderers. Its main tasks are:

- File management
- Data conversion from/to other systems, (currently, the Alias Modeler and Renderer, and some in-house software)
- Polygonization of NURBS surfaces

6.2 Window configuration

The system display includes four windows, as shown in Fig. 14. The upper right window is called the *perspective window*; this is where the user can interactively change the perspective view. The lower right window is called the *constraint window*; the viewing direction is automatically selected according to the active constraints. In the example shown in the figure, the currently moving object (a bottle) is constrained by the table, and thus, the normal direction of the table (the z-direction) is chosen as the viewing direction. The information provided in this window helps users control the constrained operations. The lower left window is called the *three-view window*; this is where user specifies one of the three available views (TOP, SIDE, FRONT). The top-left window is called the *miscellaneous window*; it displays the control panels for file I/O, motion control, etc.
Figure 15 contains another example of the display of a moving object — a banana. The bounding



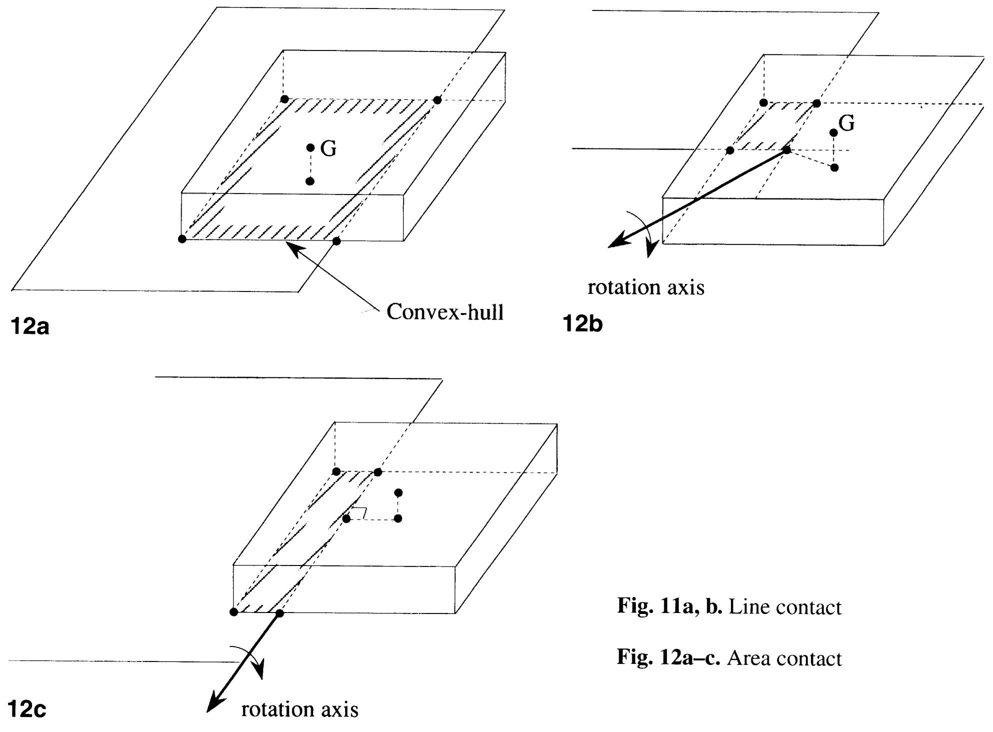


Fig. 11a, b. Line contact

Fig. 12a–c. Area contact

box (i.e., the projection volume for the z-list interference detection) and the gravity line (the straight line starting at the center of gravity in the $-z$-direction) are drawn in the perspective window (Fig. 15a). To facilitate constrained operations, the convex-hull of the contact (red line) is drawn in the constraint window (Fig. 15b).

6.3 Examples of basic operations

Figure 16 shows a sampled sequence during the PUT operation. In Fig. 16a, the object (a spoon) starts moving in the $-z$-direction, and in Fig. 16b, it collides with the teacup. In Fig. 16c, the object rotates according to the quasi-static

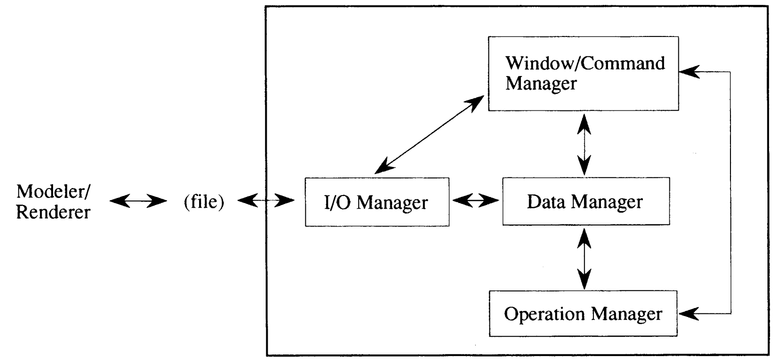


Fig. 13. System configuration

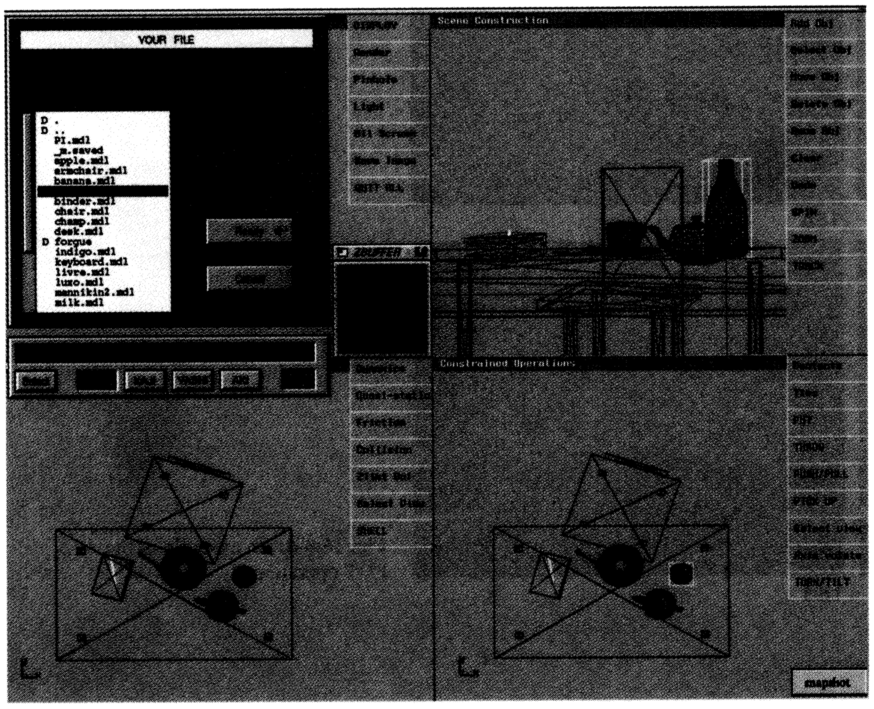


Fig. 14. Four windows

analysis, and after several collisions, finally settles into a stable position in Fig. 16d. Note that no user intervention is required during the operation. Figure 17 involves an example of the PUSH/PULL operation. The user specifies the direction to PUSH in the constraint window with the mouse, and the object (a spoon) moves in that direction, while remaining in contact with the curved surface (a saucer). Note that the direction is specified in two dimensions, while the movement is in three dimensions.

Figure 18 contains an example of the TURN/TILT operation. In the constraint window, the user has specified a contact edge as constraint, as shown in Fig. 18a. The viewpoint of the window is automatically switched to the direction of the specified edge (Fig. 18b), and the user specifies the amount of rotation along the edge

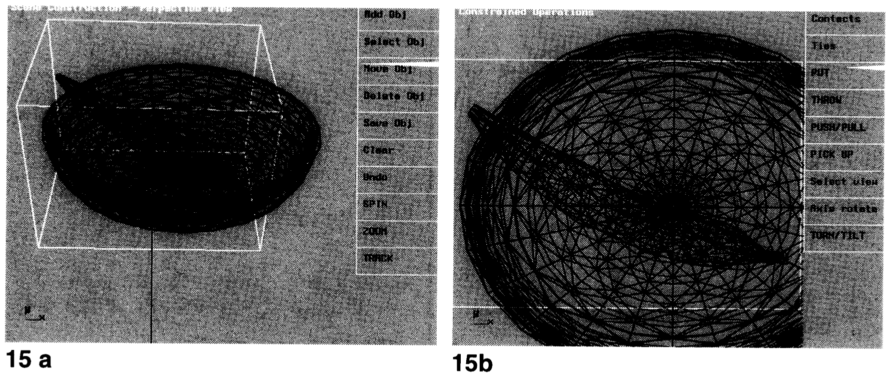


Fig. 15a, b. An example of object display

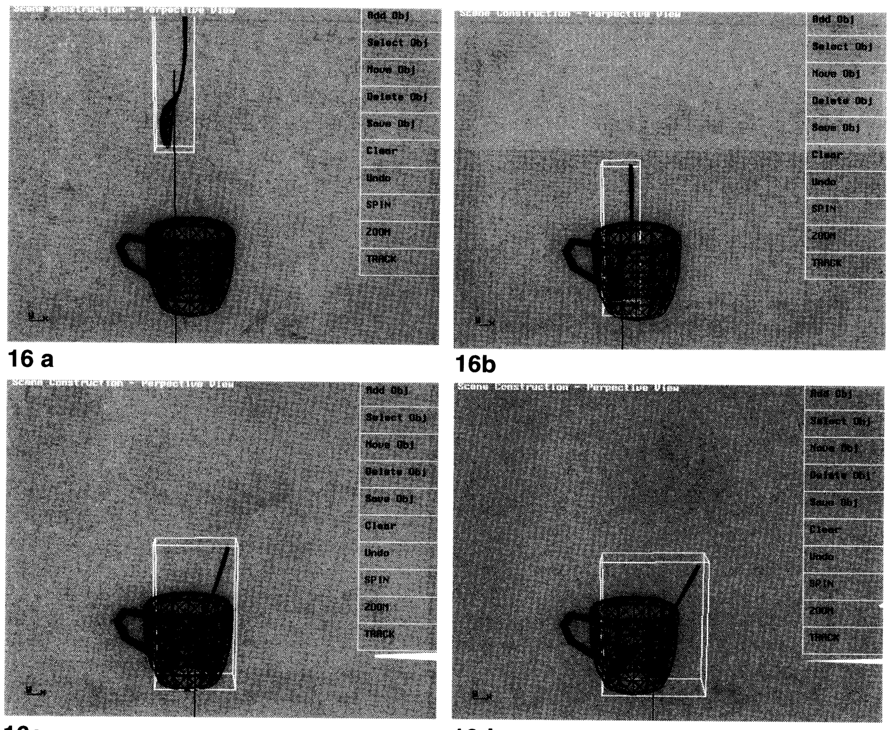


Fig. 16a–d. An example of the PUT operation

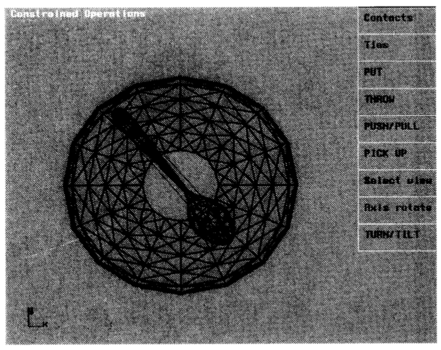
(Fig. 18c); again, this requires only two-dimensional values.

7 Results and discussion

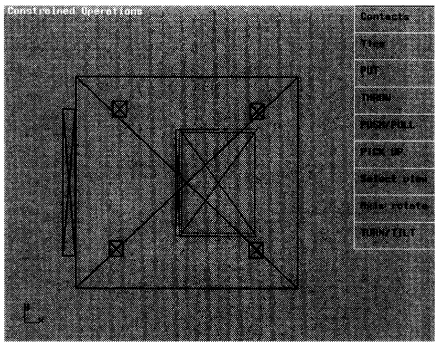
Several scenes were constructed on the system, and the execution time for various operations was measured on an IRIS 4D/310 VGX. Since response time is the most critical factor in interactive systems, we first measured the execution time of the basic operations. In a simple scene, SPOON-SAUCER (Fig. 19), a comfortable operation speed was achieved.

As shown in Table 1, the PUT operation takes 0.41 s/movement, and the PUSH/PULL operation takes 1.1 s/movement. To check the influence of scene complexity, we also constructed a complex scene BREAKFAST[3] (Fig. 20). As shown in Table 1, acceptable speed is achieved even in this

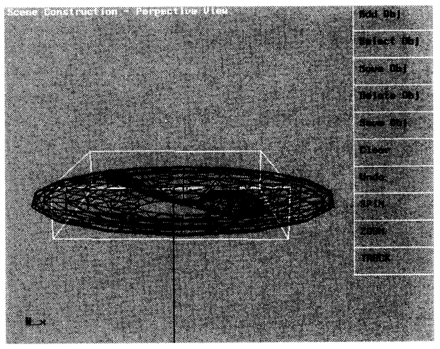
[3] The scene actually consumed all of the available 32MB main memory.



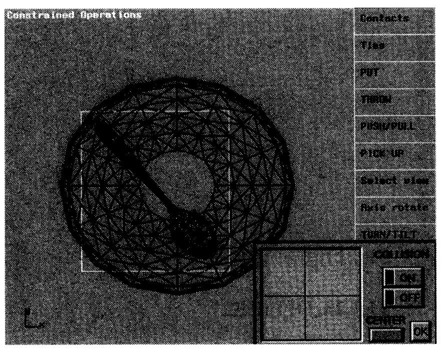
17 a



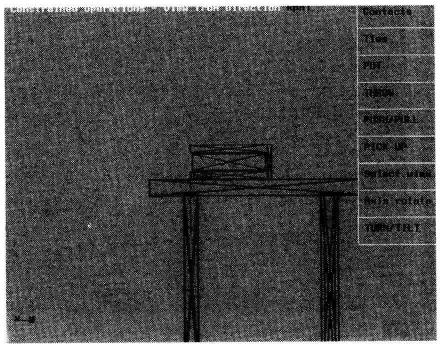
18 a



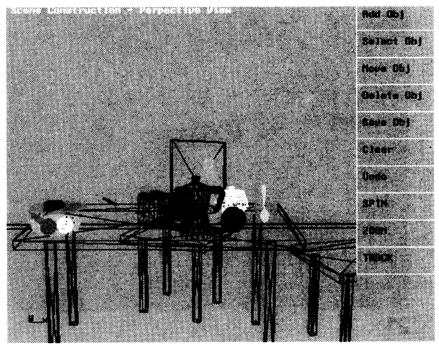
19



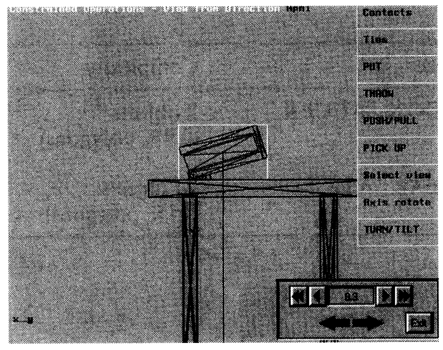
17b



18b



20



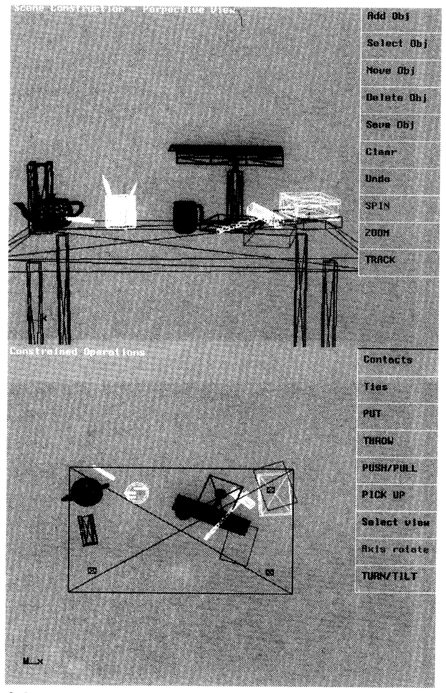
18c

Fig. 17a, b. An example of the PUSH/PULL operation
Fig. 18a–c. An example of TURN/TILT
Fig. 19. SPOON-SAUCER
Fig. 20. BREAKFAST

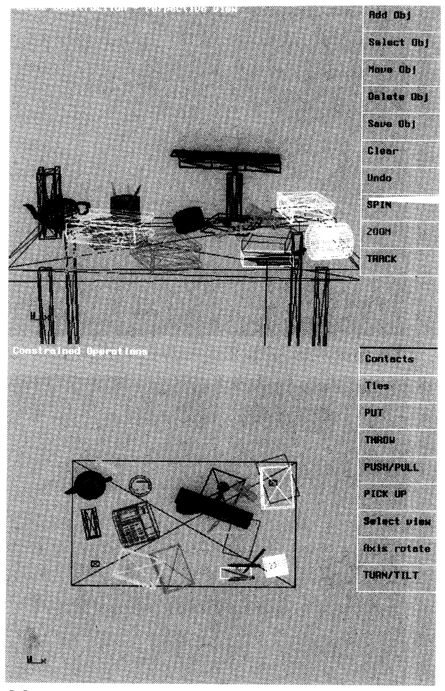
situation. Considering that the display speed of this scene is 0.4 s/frame, the additional computation cost for the constrained operations is reasonable.

Two scenes were modeled on the system for evaluation. The scene OFFICE1 (Fig. 21) consists of 16 objects, and it took only 20 min to complete the layout. The scene OFFICE2 (Fig. 22) is a modified version of OFFICE1, with 9 additional objects. This modification also took 20 min. Throughout both layout jobs, we used only the six basic operations described in Sect. 4. The operations are easy to use and considerably simplify the layout task. In particular, PUT and PUSH/PULL are very powerful tools for laying out curved (tessellated) objects (e.g., the fruits in BREAKFAST) and to model disorganized scenes like OFFICE2.

The system was interfaced to the Alias Modeler/Renderer, which was used to render the BREAKFAST scene. The rendered image is shown in Fig. 23. The total number of triangles is about 57 K, and the rendering time was 1 h 10 min on an IRIS Indigo R3000.



21



22

Fig. 21. OFFICE1

Fig. 22. OFFICE2

Table 1. Execution speed of basic operations

Scene	Complexity	PUT	PUSH/PULL
SPOON-SAUCER	2 objects (0.9 K polygons)	0.41 s/movement	1.1 s/movement
BREAKFAST	24 objects (15 K polygons)	2.0 s/movement	4.0 s/movement

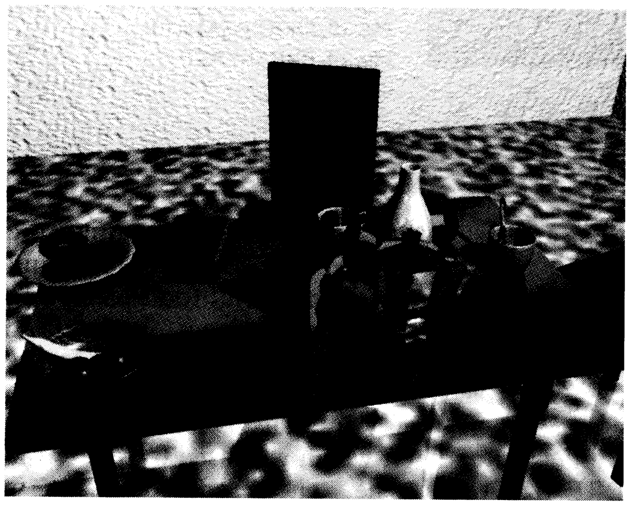


Fig. 23. Rendered image of BREAKFAST. The image was rendered by Alias Ray Tracer, and shading/lighting was designed by Shoukou Nomura, Sumisho Electric, Ltd.

8 Conclusion

We have proposed an object layout system which uses geometric and physical constraints to facilitate the construction of complex scene models. The advantages of the system are:

- Geometric and physical consistency is ensured.
- Easy-to-use operations analogous with object placement in real life are provided, (e.g. put, push/pull, turn/tilt, etc.)
- Manipulation can be performed in two dimensions by taking advantage of constraints.

Interactive operation speeds are achieved by using rasterized collision detection and simple quasi-static motion simulation.

We have implemented the system on a Silicon Graphics IRIS 4D series workstation. This system is interfaced to an Alias Modeler and Renderer, and realizes an integrated environment for object modeling, object layout, rendering, and animation. Several scenes were laid out using the system. These experiments confirmed that the constraint-based operations are very powerful tools for object layout.

In future work, we would like to address the following issues:

- Flexible objects, like cables, cloth and paper, are very common in indoor scenes, and extension of the system to flexible models would allow scenes that include such objects to be achieved by applying deformable models (1986).
- Further physics such as a global stability check would enhance the physical consistency of scene models.
- Computer-aided layout design for offices, shops, labs, etc., would be an attractive application of this technology.

Acknowledgements. We would like to thank Yves Lespérance who provided comments and corrections on this paper. We also thank Atsushi Kajiyama and Hiroki Kobayashi for their technical help, Yoshitaka Nomura for his advice on Alias software, and Hohiji Watanabe and Toshimitsu Tanaka for helpful discussion. Finally, thanks go to Rikuo Takano and Masashi Okudaira for providing a supportive research environment.

References

Barzel R, Barr AH (1988) A modeling system based on constraints. Comput Graph 22:179–188

Baraff D (1989) Analytical methods for dynamic simulation of non-penetrating rigid bodies. Comput Graph 23:223–232

Baraff D (1990) Curved surfaces and coherence for non-penetrating rigid body simulation. Comput Graph 24:19–28

Baraff D (1991) Coping with friction for non-penetrating rigid body simulation. Comput Graph 25:31–40

Baraff D, Witkin A (1992) Dynamic simulation of non-penetrating flexible bodies. Comput Graph 26:303–308

Baumgart BG (1974) Geometric modeling for computer vision, Technical report AIM 247, Standard Artificial Intelligence Laboratory

Boyse JW (1979) Interference detection among solids and surfaces. Commun ACM 22:3–9

Hahn JK (1988) Realistic animation of rigid bodies. Comput Graph 22:299–308

Moore M, Wilhelms J (1988) Collision detection and response for computer animation. Comput Graph 22:289–298

Platt JC, Barr AH (1988) Constraint methods for flexible models. Comput Graph 22:279–288

Shinya M, Forgue M-C (1991) Interference detection through rasterization. Comput Animation Visualization 2:132–134

Terzopoulos D, Platt JC, Barr A, Fleischer K (1987) Elastically deformable models. Comput Graph 21:205–214

Terzopoulos D, Fleischer K (1988) Modeling inelastic deformation: viscoelasticity, plasticity, fracture. Comput Graph 22 (4):269–278

Weil J (1986) The synthesis of cloth objects. Comput Graph 20:49–54

Witkin A, Fleischer K, Barr A (1987) Energy constraints on parameterized models. Comput Graph 21:225–232

MIKIO SHINYA is currently a Senior Scientist at NTT Human Interface Labs, Japan. He received a BSc in 1979, an MS in 1981, and a PhD in 1990 from Waseda University. He was a visiting scientist at the University of Toronto in 1989. His research interests include computer graphics, computer vision, and visual science.

MARIE CLAIRE FORGUE is a native of southern France, born in 1959 in Avignon. She holds a Ph.D. in Computer Science — on ray-tracing parallelization — from the University of Nice and INRIA (1988). After a year as a Postdoctoral Fellow at the Dynamic Graphics Lab at the University of Toronto (Canada), she worked in NTT's Graphics Lab (Japan) for 2 years. Her research interests were focused on illumination algorithm parallelization and scene modeling. She spent another year in Canada (1992) where she studied film-making at the Vancouver Film School. Since then, she has directed several short films and documentaries. Back in France, she is now interested in interactive multimedia.