Proceedings of the 1991 Winter Simulation Conference
Barry L. Nelson, W. David Kelton, Gordon M. Clark (eds.)

MODELLING PHYSICAL OBJECTS FOR SIMULATION

Paula Sweeney
Alan Norton
Robert Bacon
David Haumann

IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Greg Turk

Department of Computer Science
University of North Carolina
Chapel Hill, NC 27514

ABSTRACT

We present a framework for modelling and simulating objects with physical attributes. Rather than attaching physical properties to geometric shapes, we directly design and manipulate objects with intrinsic dynamic properties such as stiffness, mass, angular momentum, Flexible and rigid bodies can interact in the same simulated physical environment, responding to collisions, fluid velocity fields, friction and gravity.

A modeller and simulator organized along these principles has been implemented and used in computer animation. The results obtained with these systems are presented.

1 INTRODUCTION

Physically based animation is a new area in the field of computer graphics. Using physics to determine the motion of objects adds realistic effects to animations. Animations are automatically created from an initial set of physical properties and models. Physical laws are used to produce animations that are visually appealing and communicate information about how different models interact in a simulated environment.

Introducing physics as a way to generate motion has produced a variety of effects. Realistic motion of rigid bodies has been used by Hahn (1988) and Barzel and Barr (1988). The motion of flexible objects has been used by Terzopoulos et al. (1987), Miller (1988), and Platt and Barr (1988). The motion of rigid and flexible objects can also be simulated together as in Terzopoulos and Witkin (1988). Fracture of objects has been studied by Terzopoulos and Fleischer (1988) and Norton et al. (1990). Fluid flows have been studied using particles systems as in Reeves (1983) and Sims (1990). Wind field represented by fluid flow techniques is used by Wejchert and Haumann (1991).

Most of the referenced methods above use a specific mathematical model to produce a specific physical effect. Our goal is to create an animation system for experimenting with different physically based animation techniques. We currently have a system that has allowed us to model a breaking teapot, a flexible swing affected by wind, and leaves which float or move in wind velocity fields (Haumann et al, 1991).

In our system objects are represented in a topological hierarchy associated with points, edges, faces and cells. This structure permits simulation of objects with 0, 1, 2, and 3 dimensional components. Operators are defined that permit topological modification of the structures; e.g. aggregate objects are built by physically attaching subcomponents, and solid objects are disintegrated during fracture. Mass and velocity are associated with points, spring forces are transmitted through edges, and wind forces are applied to faces. Cells are required to represent the integrity of solid structures and for computing boundary operators.

2 ANIMATION SYSTEM

Many components are needed in a system for computer animation. At a minimum we need a component to create or model the objects to be animated, a component to animate those objects and a component to create an image or render a frame in the animation. The modelling and animating components will differ among animation systems depending on the algorithms used for animating objects. We choose to animate our models using forward simulation of physical environments. Refer to figure 1 for a layout of the components of our system and for the data flow between components.

Before the components of this system are discussed it is important to talk about the models that are to be created and acted upon. We refer to three types of models: a *geometric model*, a *physical model* and a *surface model*. A *geometric model* is simply a mathematical description of an object. The mathematical

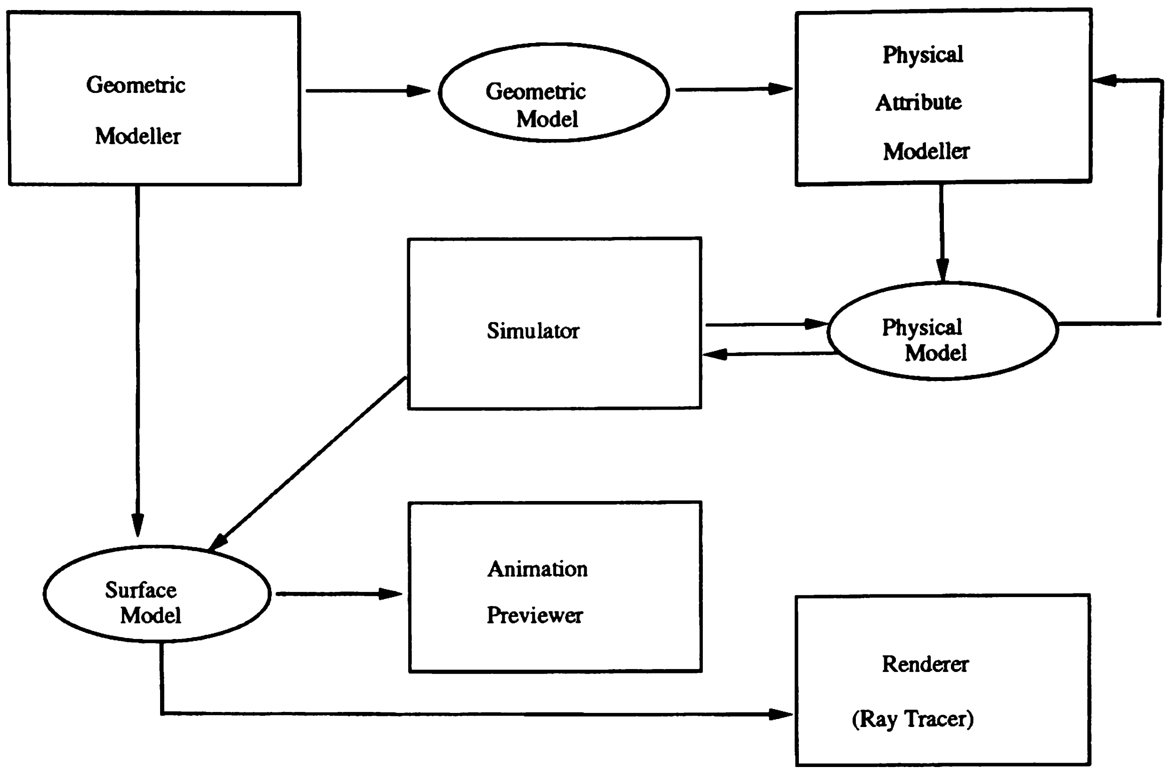

Figure 1: An Animation System for physically-based modelling

description could be surface definitions, networks of vertices and edges, sets of volumetric primitives as in the case of CSG, or any other geometric entity. A ***physical model*** is the combination of a geometric model with physical properties or attributes of an object at an instant of time. Each physical attribute is usually coupled with a specific geometric primitive. Physical models are input to a simulated environment where they react with forces and other physical entities. A *surface model* is a subset of the *geometric model.* It is a representation of the external surfaces of an object. In our system this is represented as a list of polygons. The *surface model* is important for visualizing the objects that are being simulated.

The first component needed is the modeller. The modeller is separated into two components: a geometric modeller and a physical attribute modeller. The geometric modeller uses mathematical equations to define the geometry which creates a topological model of an object. For three dimensional objects topology should describe the internal and external structure of an object. The physical attribute modeller takes a geometric model and adds physical properties or attributes to geometric primitives to create a physical model.

The next component in the system is the simulator. The simulator is a program that models an environment for physical objects to interact. Forward simulations create snapshots of the environment and the objects inside the environment at an instant of time. The collection of these snapshots results in an animation sequence based on physical events.

The environment has forces such as gravity, friction and wind fields. Physical models interact with each other and collision forces are applied to keep objects from passing through each other. The objects in this environment range from flexible objects which can bounce to stiffer objects which can break.

A different physical environment results from each run of the simulator. A physical model from one instant of time in one environment can be introduced at any time instant in another environment. Physical models can be put together in the same environment or simulation so that they are allowed to interact with each other, or they can be simulated in separate environments and be combined later for rendering.

The last component mentioned for an animation system was a renderer. High quality rendering is very time consuming. We use an animation previewer which performs quick rendering to preview the motion of the animation. Once we are satisfied with the animation, the simulation data is sent to the ray tracer which produces a high resolution image with full color, shading, texturing, reflections and refractions.

This paper will concentrate on the structure of the three models defined above and will show how they are acted upon by different components of the animation system.

3 TOPOLOGICAL HIERARCHY

Our models are represented by a topological hierarchy of geometric primitives. At the lowest level of the hierarchy we describe points in the model and at the highest end we have groups of topological hierarchies that are combined to make one object. The structure we have developed gives us the greatest flexibility for manipulating objects in the animation system.

This hierarchy is maintained throughout the animation system. The geometric modeller first produces this hierarchy and then physical properties are attached in the physical attribute modeller. The hierarchy is maintained inside the simulator. The result of the simulation is the same hierarchy or a modification of the original hierarchy. The hierarchy is modified if an object fractures during simulation, i.e. the topological structure has changed. Preserving the structure during simulation enables us to use the output of the simulator as input to a new simulation. It also enables us to modify this new structure in the geometric or physical attribute modeller. The structures are output after a set number of time steps in the simulator. These structures can be used for recovery if the system crashes. Simulations can take days to to complete so the ability to restart is essential.

Objects of different dimensions are created by using subsets of the hierarchy. We could have a simple particle system, a two-dimensional sheet or a full three-dimensional object with multiple parts. An example of the latter is a swing from a playground. The hierarchical topology allows us to simulate the swing as a whole but model it as separate pieces. The swing could be described by two ropes and a seat. Each piece is easy to model separately. Figure 2 shows a representation of this swing in our system. It is also desirable to give different properties to each section of the swing. The ropes are more flexible and weigh less than the seat.

The topological hierarchy contains information about the internal and external structure of a model. For flexible objects external forces propagate through a model and deform it. Flexible objects need to have internal structure for this propagation to be realistic. Alternatively, one could ignore the internal structure and treat the object as a rigid body which simulates faster and then transform that object back into a flexible object. For example, a stiff object falling under

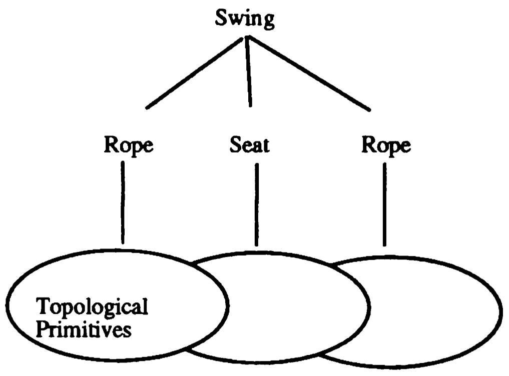

Figure 2: Hierarchy for Swing Model

gravity does not deform. The falling object can be treated as a rigid body until it strikes another object. At that point one needs to simulate the propagation of the force throughout object which could cause breaking.

The external structure of a model is extracted from the topological model for the rendering phase of the animation system. A hierarchy is still required to give different surface properties to each part of the model. In the swing example the ropes and the seat are assigned different colors and specularity.

This topological hierarchy is used for defining shape and dimension of objects. In order to simulate these objects we need to define physical attributes for each level of the hierarchy. The next section will describe the implementation of this hierarchy and the physical properties that are needed for simulations.

4 STRUCTURE OF MODELS

The model, whether it is a geometric, physical or surface model is the key element in our animation system. It is the input to or the output from all components in our system. Figure 1 shows the data flow of the animation system. We will define the structure of these models and show how the structure is effected by the different components of the animation system.

The following primitives are used in the hierarchy:

Point: a vector describing a position in three space. Physical properties associated with each point are velocity and mass. Velocity is represented as a vector.

Edge: a line segment connecting two points. Each edge in an object can represent a spring. A spring and damping constant is assigned to each edge that acts like a spring. There is also a spring threshold which determines the maximum amount the spring will stretch before breaking.

Face: a triangle defined by three points. Constants for air resistance or surface drag are assigned to each face. The constants define the smoothness or the resistance to wind of a face.

Cell: a list of connected edges and faces. Cells are defined for 2D and 3D objects. The boundary of a 2D cell is a polygon. The boundary of a 3D cell defines a closed unit of volume. In both cases, extra edges may be added diagonally across the surface or across the interior section of a cell to make the cell more rigid. In a physical environment a 3D cell with no internal structure will collapse easily. The cell is the only level of the hierarchy that does not contain physical attributes.

Figure 3 shows the interdependencies between the hierarchy primitives.

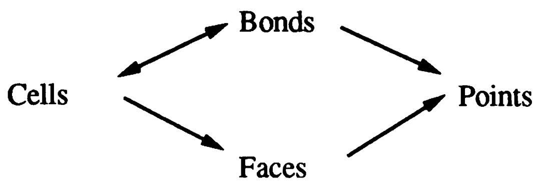

Figure 3: Interdependencies Between Hierarchy Primitives

A subobject is constructed by combining the above primitives in correct hierarchical order. The order is: points, edges, faces, and then cells. Adjacent edges in a subobject will share points. To avoid duplication, point information is stored only once. Edges then refer to points by using a pointer into the point structure. This is also true for faces; adjacent faces share points. Cells are a special case. Adjacent cells share bonds but not faces. Faces are not shared because we generate unique normals for each face of a cell. This will be explained in section 5.4.

We can now define more precisely the three types of models in our system using the above primitives. A geometric model is defined by one or more subobjects where the subobjects are ordered hierarchically. A physical model is a geometric model with all its associated physical attributes, including physical attributes which are assigned to the object as a whole, such as a coefficient of friction. A surface model simply contains points and faces. It does not contain all the points and faces of an object, it only contains those points and faces that create the exterior surfaces of the object. No physical attributes are included in the surface model.

5 OPERATORS

We consider the animation system to consist of a set of *operators* which act on our models and convert them into new models possibly of a different type. Our current system has four operators. We refer to these operators as the time, breaking, gluing and boundary operators. Three of the operators, time, breaking and gluing, are topological modifiers. They transform a physical model into a new physical model with different topology. The boundary operator transforms a physical model into a surface model for wind calculations and rendering.

5.1 The Time Operator

We consider time to be an operator since a time step is the action causes the forces to be applied to each object in the system. Forces are applied to points. We start by computing the spring forces. Each edge has a spring and damping constant associated with it. The length of the edge is determined from the position of its end points. The stretch and velocity of the spring scaled by the strength and damping of the spring determines one part of the force to be applied to the two end points. Collision detection is done between points. The collision detection subroutine uses a penalty method to repel points that are about to collide. This repulsion force gets added to the spring forces. Other forces added are gravity, weight, friction and forces due to wind fields and air resistance. The point's position is updated based integration of these computed forces over time. The levels of the hierarchy that are essential for this operator are points, edges and faces.

5.2 The Breaking Operator

The breaking operator is actually a suboperator of the time operator. Breakage starts when we are computing forces at a single time step. Before simulations are started, the resting or equilibrium length of each edge is computed and saved along with a breaking threshold. The new length of each edge computed during a time step is compared to the saved equilibrium length. If the difference exceeds the breaking threshold then the edge is marked as broken.

Marking the edge as broken is just the first step in the breaking process. The cell level of the hierarchy was created to be the unit of breaking for all 2 or

3 dimensional objects. If one edge in a cell breaks then that cell is no longer valid. All other edges in the invalid cell are broken unless the edge is shared by another cell which is valid. Therefore, the second step involves finding all cells that the broken edge belongs to and marking those cells invalid. All edges unique to the invalid cell will be marked as broken. Figure 3 shows a two way pointer between cells and bonds in the hierarchy. This two way pointer in essential for this step of the breaking operator.

The third step for this operation is to update the face boundary information for the broken object. The cell structure contains pointers to the faces that surround a cell. When a cell becomes invalid the face boundary information is updated. Breaking may expose some faces that were part of the internal structure.

5.3 The Gluing Operator

Gluing is an operation that takes two physical models and combines them into one. The new model that results from this operation can then be glued to another physical model. If we were only concerned about rendering composite objects, two objects could be positioned close enough together (and possibly interpenetrate one into the other) so that the result would look like one object instead of two. The two objects need to be attached in some way so that when the forces of the environment are applied the two objects will move together as one.

In the modelling environment two objects are positioned such that the portions of each object that are to be glued together are nearby without touching. The gluing operator takes as input the distance between points to be glued. The gluing operator searches for pairs of glue points that are this distance or less apart from each other. New edges are created between glue points. The two objects are combined into one object description structure. Physical properties are assigned to these new edges. For a tight bond between the two objects one could pick stiff spring and damping constants with a high breaking threshold. The glued edges can be stronger, weaker or the same as the other edges in the object.

In order for the glued area to be a strong bond it is important that the number of points and distance between points be similar. The new bonds formed by the gluing operator should be of similar lengths. If one object has a very fine grid structure and the other object has a very course grid structure the result could be that all the points in the specified region of the first object get bonded with one point in the second object.

Figure 4 shows two examples of two rectangular objects which are glued together. In the first example, the spacing of the points and the length of the edges for both rectangles is different. When glued, the points in the top rectangle are glued to just one point in the second rectangle. The resulting shape of the glued object is not the expected shape for gluing these two objects. The second example in figure 4 is a better example of a good glue bonding between objects. The spacing of points for both the objects is similar. If the grid structure of the two objects that are glued together is different, the motion of the object as a whole will not be uniform.

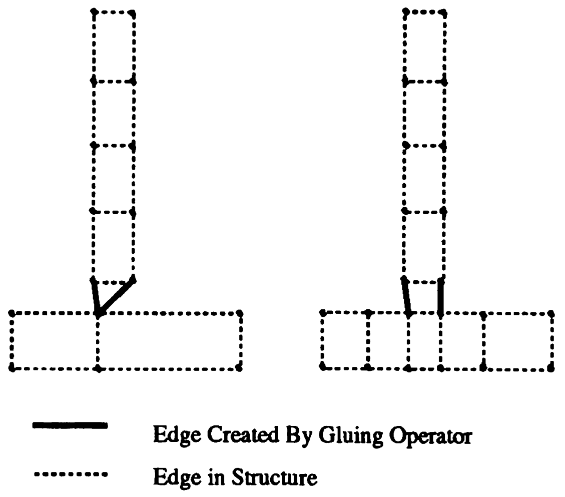

Figure 4: Gluing Two Objects Together

Although the two objects have been glued together to form one physical object, the identity of the two parts is not lost. Each physical model has a list of subobjects that are part of the model. For example, we construct a teapot by making the body, spout and handle as three separate objects. We can then position the body and spout and apply the gluing operator. We can take the result of that operation and position the handle and apply the gluing operator another time. The result of the gluing operations is one object called a teapot that has subobjects: handle, spout and body. A subobject is defined by simply giving a range of points, edges, faces and cells within the object definition structure.

5.4 The Boundary Operator

The boundary operator takes a physical model and produces a surface model for rendering. The boundary operator calculates external or boundary faces in a physical model. This operator only applies to 3D objects. In 2D objects all faces are external and for lower dimensional objects there are no surfaces.

For simplicity, the boundary operator is explained for a 3D physical model that is constructed from a cubic lattice of cells. Each cell has 12 faces, two triangles for each square side of the cube. Cells do not share faces. Figure 5 shows one side of a cubic cell that contains four points and two faces. The four points, labeled P1, P2, P3, and P4, are shared by cells A and B. Four unique faces are defined using these four points, two of the faces belonging to cell A and two faces belonging to cell B. The difference between the faces for these two cells is the ordering of the points for each face. The ordering of the points determines the direction of the normal for each face.

The normal is used for rendering when the face is an external face. The normals for internal faces are used when an object is broken and an internal face becomes an external face. The two faces for cell A are: (P1,P3,P2) and (P1,P4,P3). The two faces for cell B are: (P1,P2,P3) and (P1,P3,P4). The normal for the faces in cell A points in the direction of cell B and the normal for the faces in cell B points in the direction of cell A. If cell A becomes invalid and all edges are broken, the normal for the faces of cell B will be pointing outward which is correct for an external face.

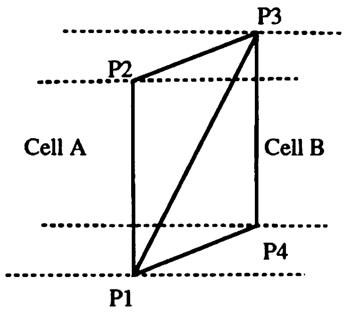

Figure 5: Faces Shared By Adjacent Cells

External faces can be found and marked by traversing the list of faces for an object and eliminating faces that have a reverse face. In the above example the pairs (P1,P3,P2) and (P1,P2,P3) cancel each other out making this face internal. We are assuming an specific ordering of the vertices. We assume that the ordering will always start with the same point (i.e. P1) for each set of faces.

Once all external faces have been marked we can extract them and their points from the physical model to create a surface model. This surface model is used for rendering.

6 CONCLUSIONS

We have presented a framework for creating models for simulated physical environments. We believe this framework gives us the flexibility to experiment with models of varying dimensions and physical properties. Various techniques are applied to these models to create animations of bouncing, breaking, and wind blown objects. Two animations have been created using our system and models: *Tipsy Turvy*(Bacon et al. 1989) is an animation of a teapot that flexes and fractures, and *Leaf Magic* (Arya et al. 1991) is an animation showing leaves floating and moving in various wind fields.

ACKNOWLEDGEMENTS

We wish to thank John Snyder for his ideas about modelling and for integrating his generative modelling program into our animation system. Thanks are also due to Jessica Hodgins and David W. Levine for their valuable ideas for structuring this paper.

REFERENCES

Arya, K., R. Bacon, A. Khorasani, D. Haumann, A. Norton, P. Sweeney, and J. Wejchert. 1991. Leaf Magic. *SIGGRAPH Electronic Theater.*

Bacon, R., J. Gerth, A. Norton, P. Sweeney, and G. Turk. 1989. Tipsy Turvy. *SIGGRAPH Electronic Theater.*

Barzel, R., and A. H. Barr. 1988. A Modeling System Based on Dynamic Constraints. *Computer Graphics* **22.4**: 179-188.

Hahn, J. 1988. Realistic Animation of Rigid Bodies. *Computer Graphics* **22.4**: 299-308.

Haumann, D., J. Wejchert, K. Arya, R. Bacon, A. Khorasani, A. Norton, and P. Sweeney. An Application of Motion Design and Control for Physically-Based Animation. 1991. *Proceedings of Graphics Interface '91*: 279-286.

Miller, G. S. P. 1988. The Motion Dynamics of Snakes and Worms. *Computer Graphics* **22.4**: 169-178.

Norton, A., G. Turk, R. Bacon, J. Gerth, and P. Sweeney. Animation and Fracture by Physical Modeling. to appear in *The Visual Computer.*

Platt, J., and A. Barr. 1988. Constraint Methods for Flexible Models. *Computer Graphics* **22.4**: 279-288.

Reeves W. 1983. Particle Systems - A Technique for Modeling a Class of Fuzzy Objects. *Computer Graphics* **17.3**: 359-376.

Sims, K. 1990. Particle Animation and Rendering Using Data Parallel Computation. *Computer Graphics* **24.4**: 405-413.

Terzopoulos, D., J. Platt, A. H. Barr, and K. Fleischer. 1987. Elastically Deformable Models. *Computer Graphics* **21.4**: 205-214.

Terzopoulos, D., and A. Witkin. 1988. Physically Based Models with Rigid and Deformable Components. *IEEE Computer Graphics and Applications.* **8.6**

Terzopoulos, D., and K. Fleischer. 1988. Modeling Inelastic Deformation: Viscoelasticity, Plasticity, Fracture. *Computer Graphics* **22.4**: 269-278.

Wejchert, J., and D. Haumann. 1991 Animation Aerodynamics. to appear in *Computer Graphics 8/91*

AUTHOR BIOGRAPHIES

PAULA SWEENEY has been working at IBM since 1984. She has worked on the design and implementation of operating systems and animation systems. Currently her interest is in realistic animation using physics and control theory. Paula has a BA in Mathematics from Manhattanville College and an MS in computer science from New York University.

ALAN NORTON is currently managing a research project in animation and image synthesis. He received his BA degree from the University of Utah in 1968, and the PhD from Princeton University in 1976, both in mathematics. He was instructor at the University of Utah, 1976-79, and Assistant Professor at Hamilton College 1979-80, before joining IBM Research. His research interests include computer graphics and animation,

ROBERT BACON only recently discovered that he has been working on visualization throughout most of his professional career. Following graduate studies at the University of Chicago, Mr. Bacon began working with computers in such diverse endeavors as process control, machine tool control, computer typesetting, image synthesis, and computer generated animation. He has been a frequent contributor to animation exhibits since 1985.

DAVID HAUMANN is currently a Research Staff Member at IBM. He received his Ph.D. in Computer Science at The Ohio State University in 1989, and his BS in Applied Mathematics from Brown University in 1977. His experience in computer graphics spans the fields of radiation treatment planning, flight simulation and commercial computer animation production. His research interests include computer graphics, animation, and physically-based modeling.

GREG TURK is a graduate student in the Computer Science Department at the University of North Carolina at Chapel Hill. At UNC he has been involved in the development of rendering algorithms for the Pixel-Planes graphics engine. He worked in the computer graphics group at IBM during the summer of 1988 and is currently supported by an IBM Graduate Fellowship. His interests include physically-based modelling, collision detection, synthetic texture generation and constructive solid geometry. Turk received a BA in mathematics from UCLA in 1984 and an MS in computer science from UNC in 1989.