Journal of Visualization and Computer Animation, Volume 12, Issue 2, May, 2001

Reproducing Works of Calder

Dongkyoo Lee*, Hee-Jung Bae*, Chang Tae Kim*, Dong-Chun Lee*,
Dae-Hyun Jung*, Nam-Kyung Lee*, Kyoo-Ho Lee*, Nakhoon Baek**,
J. Won Lee***, Kwan Woo Ryu* and James K. Hahn***

* Dept. of Computer Engineering, Kyungpook National University
** School of Electronic and Electrical Eng., Kyungpook National University
*** Dept. of Computer Science, The George Washington University

Journal of Visualization and Computer Animation, Volume 12, Issue 2, May, 2001

Abstract

Many fine art pieces have been reproduced in digital form. The digital reproductions have been used to store and transmit the original work. In contrast, mobiles, or moving sculptures, such as those designed by Alexander Calder cannot be reproduced realistically by photographs and/or static images. The real characteristics of mobiles come from the motions generated by interactive external forces applied to their structures. Hence people could not fully enjoy them through static images or even static three-dimensional models. We present a virtual mobile system where users can easily control the mobile and can feel the impressions that the artist originally intended to provide. Virtual winds are generated by blowing on a microphone which then exert external forces to the mobile. This microphone interface lets users control the mobile while they are watching it through a monitor. We introduce a linear time solution for the constraint dynamics and an improved impulse dynamics to speed up the simulation. Using these techniques, we achieve a real time simulation of the mobile on personal computers. The techniques presented can easily be extended to simulate other interactive dynamics systems.

Keywords: virtual mobile, constraint dynamics, virtual wind, impulse dynamics

1. Introduction

Recently, real world objects have been successfully reproduced in the computer systems, using computer graphics and virtual reality techniques. A good application example is digital museums, which display reproductions of real world fine art pieces on the computer.[1] For drawings, digital scanners and/or digital cameras can be used to generate the digital reproductions. In the case of sculptures, three-dimensional geometric and/or volume data can be used to create virtual sculptures.[2] Image-based rendering techniques including MCOP (multiple center of projection)[3] can also be used for this purpose.

Mobiles, which are also known as *moving sculptures*, however, cannot be represented successfully using these techniques. As an example, a real world mobile, “Steel Fish” by Alexander Calder is shown in Figure 1. Typical mobiles are dynamic systems: their components are usually dangling from the stems and move due to external forces such as from winds. Much of the people’s experience of the piece comes from real-time interactions with the piece.

In this paper, we present a physically-based virtual mobile system. To simulate a real world mobile,

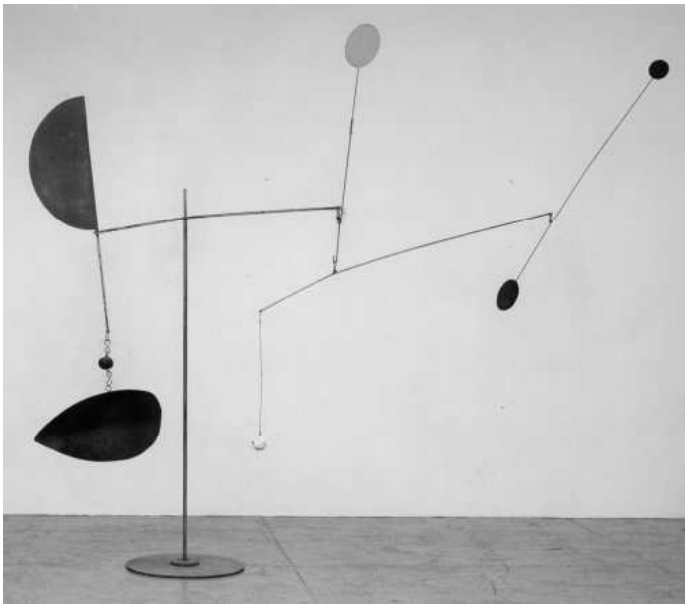


Figure 1. A real world mobile: *Steel Fish* by Alexander Calder.

we start by constructing its geometric shape. Physical properties such as masses and inertia tensors are then calculated. The mobile is simulated by a constraint dynamics system based on these geometric data and physical properties. Users can generate virtual winds, and our constraint dynamics solver simulates the motions and collisions of the components. An impulse dynamics system is used to simulate collisions among the components of the mobile. Using a simplified aerodynamics model for the virtual wind and other acceleration techniques, our system accomplished real time display of an example virtual mobile on Pentium chip-based personal computers. Although the system has value in reproducing mobiles, perhaps more importantly, the techniques presented can be applied to other real-time physically-based simulations.

The following sections describe the details of our virtual mobile system. In Section 2, we present the overview of the system. Section 3 explains how virtual mobiles are constructed from the real world mobiles. In Sections 4, 5, and 6, the virtual wind model, the constraint dynamics solver, and the impulse dynamics solver are presented, respectively. Example sequences of animation are shown in Section 7. Finally, conclusions and future work are given in Section 8.

2. System Overview

Our system is implemented in the C++ programming language and OpenGL graphics library on Pentium chip-based PC's. Figure 2 is the block diagram of the system. At the preprocessing step, the system calculates geometric and physical properties of a virtual mobile. Initially, the mobile is in its equilibrium state. The user can generate virtual winds through the microphone interface and these virtual winds act as external forces. After finding the surfaces of the mobile influenced by the virtual wind, the forces applied on these surfaces are calculated.

Using these forces, our system simulates the motions of the mobile, using constraint dynamics and impulse dynamics techniques. The constraint forces due to the connectivity among the components are first calculated, and the constraint dynamics solver generates new positions. These new positions may

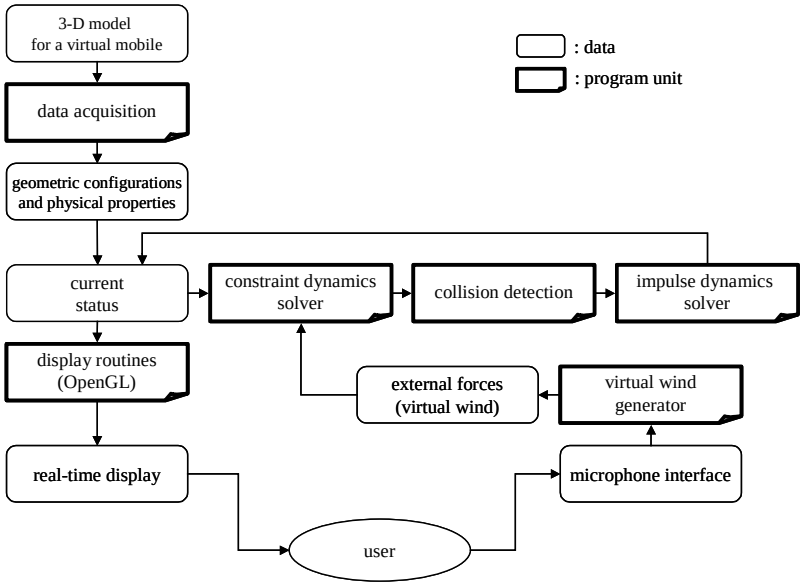


Figure 2. Block diagram of the virtual mobile system.

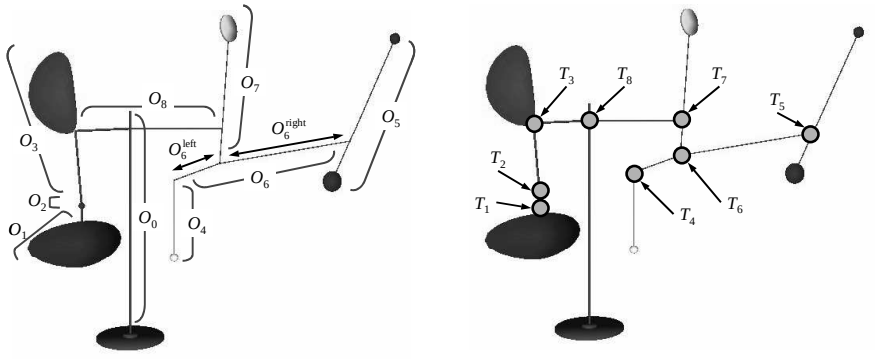
cause collisions among the components of the mobile. After detecting the collisions, the impulse dynamics solver is used to handle these collisions. The system repeats above steps and displays the virtual mobile for each iteration.

3. Data Acquisition

To represent a virtual mobile, we need its geometric configurations and physical properties. It is, however, difficult to extract these properties from a real world mobile. In this paper, we select Steel Fish by A. Calder as an example, and reproduce its geometric configurations as shown in Figure 3.

The example consists of nine *components*, each of which acts as an independent motion unit (Figure 3.a). Eight spherical *joints* with three degrees of freedom connect these components to each other (Figure 3.b). From the dynamics point of view, these joints play the role of constraints.

For the dynamic simulation, the virtual mobile requires some physical parameters such as the mass, the location of the center of mass, and the inertia tensor for each component. Our system adjusts these parameters from the given geometric configurations to achieve the equilibrium as shown in the original



(a) components (b) joints

Figure 3. Components and joints of the virtual mobile.

work.

Since the example has a tree-like shape, the mass ratios between components are computed in a bottom-up manner. As an example, the mass ratio between components O_4 and O_5 is calculated from the ratio of the length of O_6^{left} and O_6^{right} as shown in Figure 3.a. Notice that the geometric configuration only gives mass ratios. Hence the total mass of the mobile acts as a control parameter to finally calculate the mass of each component.

In addition to the mass of each component, we need to calculate the location of the center of mass and the inertia tensor. Assuming the components are uniform density rigid bodies, these physical parameters are calculated in a straightforward manner.[4] First, the volume of a component can be calculated from its geometric shape. From its mass and volume, the density of each component is specified. Then, we use Mirtich's integral equations to get other physical parameters. The geometric configurations and physical parameters are later used to simulate the dynamics behavior of the virtual mobile.

4. Virtual Wind

While natural wind causes the motions of real world mobiles, we need virtual wind to simulate the motions of the virtual mobile. To control the virtual wind, we may use traditional input devices such as

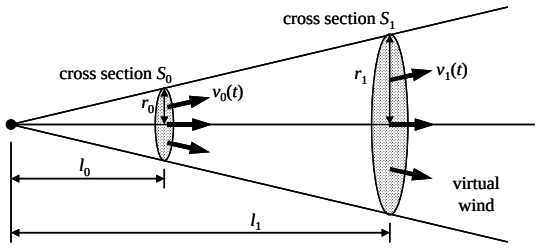


Figure 4. Virtual wind model.

keyboards and mice. Additionally, our system uses a microphone interface. The user blows on the microphone, and the speed of the generated virtual wind is proportional to the amplitude of the input sound. The direction of the wind is specified with the mouse or stereo glasses with ultrasound head tracker.

The microphone interface has some benefits. First, it is more intuitive for the user to generate the wind through the blowing action. Second, the microphone is more convenient to simulate the temporal variations of the wind speed. Another benefit is that the microphone is inexpensive and easily available even for personal computers.

Since it is generated from the human breath, the virtual wind is assumed to propagate in an infinite cone shape, as shown in Figure 4. A circular cross section S_0 with its radius r_0 plays the role of the source of wind. The vertex of the cone is located at the distance l_0 from the center of S_0. User can provide r_0 and l_0 to control the shape of the cone, and the wind propagates from S_0 in the direction opposite to the vertex. Using the microphone interface, user controls the wind speed $v_0(t)$ at S_0.

For simulating the wind, we need to calculate the speed of the wind at a distance $l_1 > l_0$ from the vertex. Although there are several results[5, 6, 7] for simulating aerodynamics in computer graphics applications, we use a simplified form to achieve real time display. We start with the assumption that the fluid (in this case, air) is not viscid and incompressible, and no fluid can cross the boundary of the cone shape. This is a reasonable model for air at normal speed.[6] Then, the *Equation of Continuity* in fluid

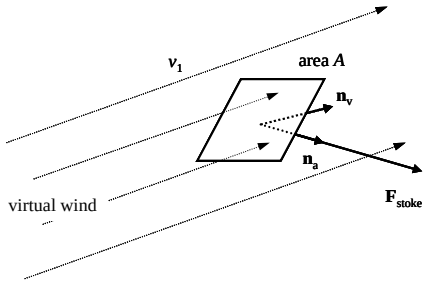


Figure 5. Force due to the virtual wind.

dynamics gives

$$A_0 v_0 = A_1 v_1, \tag{1}$$

where A and v represent the area of the cross-section and the fluid speed, respectively.[8] The subscript 0 and 1 corresponds to the distance l_0 and l_1, respectively.

From Equation (1) and the geometric configurations of the cone, it is easily found that

$$\frac{v_1}{v_0} = \frac{A_0}{A_1} = \frac{\pi r_0^2}{\pi r_1^2} = \frac{\pi l_0^2}{\pi l_1^2} = \frac{l_0^2}{l_1^2}. \tag{2}$$

Using the *Stoke drag equation*,[8, 9] the force acting on a face with its area A located on the cross section S_1 can be calculated as follows:

$$\mathbf{F}_{\text{stoke}} = \rho A v_1^2 (\mathbf{n}_v \cdot \mathbf{n}_a) \mathbf{n}_a, \tag{3}$$

where $\mathbf{n}_v$ and $\mathbf{n}_a$ are the unit directional vector of the wind and the face normal vector, respectively, as shown in Figure 5. The constant ρ is the density of fluid. Equations (2) and (3) give

$$\mathbf{F}_{\text{stoke}} = \alpha A \frac{v_0^2}{l_1^4} (\mathbf{n}_v \cdot \mathbf{n}_a) \mathbf{n}_a, \tag{4}$$

where α is a constant. For simulating the turbulent behavior of wind, we add a random noise term and the final force can be expressed as

$$\mathbf{F}_{\text{wind}} = \mathbf{F}_{\text{stoke}} + \mathbf{F}_{\text{random}}, \tag{5}$$

where the direction of $\mathbf{F}_{\text{random}}$ is randomly selected, and $|\mathbf{F}_{\text{random}}| < \beta |\mathbf{F}_{\text{stoke}}|$ for a user-selectable constant β.

Before applying the force calculated in Equation (5) to the face, we should check whether the wind is directly delivered to the face or not. When a face is occluded by another face in the air flow, we simply assume that the occluded face is not influenced by the wind. Without this assumption, it is hard to achieve the real-time display of the mobile. In this simplified wind model, the air flow can reach the faces which are directly visible from the vertex of the cone and which belong to the interior of the cone.

The faces affected by the wind can be identified by a visible surface detection method. We use the depth-buffer method for more speed-up. To simulate the partially occluded cases, faces are first partitioned into small areas on which sampling points are assigned. Then, the graphics pipeline is used to capture the image containing the cross section S_0 using the synthetic camera located at the vertex of cone. Each sampling point has its own identification number, and it is also stored in the alpha buffer through the graphics pipeline. Scanning the alpha buffer, we can easily detect whether each sampling point is visible from the vertex of the cone or not. Since S_0 corresponds to a circle on the image plane, it is also easy to check the point belongs to the interior of the cone. When a sampling point is visible and also belongs to the interior of the cone, the force calculated in Equation (5) is applied to its corresponding area. In the next section, we will present how the constraint dynamics techniques are used to apply these external forces to the mobile.

5. Constraint Dynamics

Since the components of typical mobiles are connected with joints, constraint dynamics simulation is required to generate physically correct motions. In constraint dynamics simulation, two different methods are frequently used: *the reduced coordinate method*[10] and *Lagrange multiplier method*.[11, 12] Although the reduced coordinate method can also be used, symbolic knowledge of the body-space to world-space mapping is required to parameterize the system's degree of freedom.[12] Our system is based on Lagrange

multiplier method. In this section, we will explain how Lagrange multiplier method can be applied to the mobile. See [11] and [12] for more details of Lagrange multiplier method itself.

At each joint of the mobile, we need to apply constraint force to preserve the connectivity due to the joint. Letting the position vectors of joints be $\mathbf{q}(t)$ at time t, Lagrange multiplier method calculates the constraint force $\hat{\mathbf{Q}} = \mathbf{J}^{\mathrm{T}}\lambda$ by solving the following constraint force equation with respect to the Lagrange multiplier λ:

$$\mathbf{J}\mathbf{W}\mathbf{J}^{\mathrm{T}}\lambda = -\dot{\mathbf{J}}\dot{\mathbf{q}} - \mathbf{J}\mathbf{W}\mathbf{Q}, \tag{6}$$

where $\mathbf{W}$ is the inverse of generalized mass matrix and $\mathbf{Q}$ is externally-applied force (virtual wind in this case). $\mathbf{J}$ is the Jacobian matrix $\partial\mathbf{C}/\partial\mathbf{q}$ where the vector function $\mathbf{C}(\mathbf{q})$ is the concatenation of all constraint functions. After solving the above equation, the calculation of the constraint force $\hat{\mathbf{Q}}$ is calculated in a straightforward manner.

Many numerical techniques can be used to solve the above equation.[11, 12, 13] Among them, Baraff's extension method[12] gives linear time solutions for usual constraint functions. However, this method requires complicated algorithms for auxiliary constraints, which can frequently occur in the case of virtual mobiles. For chain-like objects, Surles presented a linear time solution through permutating the rows of the matrices.[11] Our method is similar to Surles' method. However, we avoid the permutation process through proper numbering of the constraints. This numbering can be performed as a pre-processing step, as shown in the followings.

The joints in a mobile can be numbered in a bottom-up manner. Figure 3 shows the numbering of components and joints of our example mobile, *Steel Fish*. Notice that the component O_0 is fixed to the ground and thus joint T_8 act as a nail constraint while others act as point-to-point constraints. Using the bottom-up numbering of the joints, Jacobian matrix $\mathbf{J}$ in Equation (6) can be expressed as

$$\mathbf{J} = \begin{bmatrix} j_{11} & j_{12} & & & & & & & \\ & j_{22} & j_{23} & & & & & & \\ & & j_{33} & & & & & j_{38} & \\ & & & j_{44} & & j_{46} & & & \\ & & & & j_{55} & j_{56} & & & \\ & & & & & j_{66} & j_{67} & & \\ & & & & & & j_{77} & j_{78} & \\ & & & & & & & j_{88} & \end{bmatrix},$$

where the denoted elements are constants and others are all zeroes. The element j_{ij} is non-zero when the joint T_i connects the component O_j to another component. A tree-like object gives an upper triangular Jacobian matrix **J** with a bottom-up numbering of components and joints. Since **J** is upper triangular, the product of matrices $\mathbf{J}\,\mathbf{W}\,\mathbf{J}^{\mathrm{T}}$ in Equation (6) can be expressed as:

$$\mathbf{J}\mathbf{W}\mathbf{J}^{\mathrm{T}} = \begin{bmatrix} a_{11} & a_{12} & & & & & & & \\ a_{21} & a_{22} & a_{23} & & & & & & \\ & a_{32} & a_{33} & & & & & a_{37} & a_{38} \\ & & & a_{44} & a_{45} & a_{46} & & & \\ & & & a_{54} & a_{55} & a_{56} & & & \\ & & & a_{64} & a_{65} & a_{66} & a_{67} & & \\ & & a_{73} & & & a_{76} & a_{77} & a_{78} & \\ & & a_{83} & & & & a_{87} & a_{88} & \end{bmatrix}.$$

Letting the above matrix be $\mathbf{A} = [a_{ij}]$, we can easily find the following three characteristics:

(1) diagonal elements a_{ii}'s are all non-zeroes

(2) a_{ij} is non-zero if and only if a_{ji} is non-zero

(3) a_{ij} is non-zero if and only if a component is affected by both the i-th and the j-th constraints.

Although a few numerical methods can solve Equation (6) in $O(n)$ processing time, they are somewhat complex.[11, 12] In contrast, the above characteristics enable us to achieve the time complexity of $O(n)$ even with Gaussian elimination method, which is simple but requires $O(n^3)$ time for general matrix equations.

During Gaussian elimination process, we can always choose a_{ii} as the pivot element since all a_{ii}'s are non-zeroes. When subtracting the i-th row from the j-th row with $j > i$, no new non-zero elements in the j-

th row are introduced. The mobile has $O(n)$ constraints and thus, the matrix $\mathbf{A} = \mathbf{J}\,\mathbf{W}\,\mathbf{J}^{\mathrm{T}}$ has only $O(n)$ non-zero elements. Therefore, we need only $O(n)$ operations for the matrix solution, to finally achieve the linear-time solutions for our constraint-dynamics model.

6. Collision Handling

When simulating the motion of mobiles with constraint dynamics, collisions between its components will necessarily happen. Although there are many general collision detection methods,[14, 15] the collision detection can be achieved more efficiently through analyzing the characteristics of the mobile. First, some pairs of the components cannot collide with each other. As an example, the components O_3 and O_5 can never collide with each other in our example mobile. A preprocessing step detects all such pairs of components so that they can be excluded from the collision detection process. Additionally, some components are simple geometric shapes such as cylinders and spheres. Thus, we can use cylinder-to-cylinder, cylinder-to-sphere and sphere-to-sphere collision detection methods, which are much faster than the usual polyhedron-to-polyhedron collision detection methods.

After detecting collisions, we use the *impulse-based collision response method*.[16, 17] Since this method can calculate the new velocities instantaneously, it is suitable for real-time applications. The penalty method,[17] which is also widely used in the collision response, requires small time steps for accurate simulation, and is not appropriate for real-time applications.

In the case of a mobile, the collision response method should cooperate with the constraint dynamics model. Thus, the constraints at the joints and frictions at the collision points should also be handled during the collision response. Moore formulated the impulse equations for articulated figures, and his equation can be used for joint constraints.[17]

Letting the components of the mobile be O_i, $1 \le i \le n$, the mass and inertia tensor of O_i is denoted as m_i and $\mathbf{I}_i$, respectively. Due to the collision, the linear and angular velocity of O_i may be changed. Let $\mathbf{v}_i$

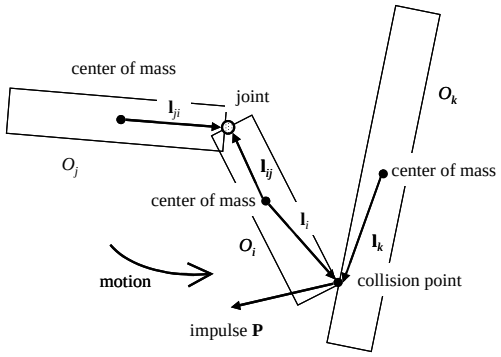


Figure 6. Impulse-based collision response.

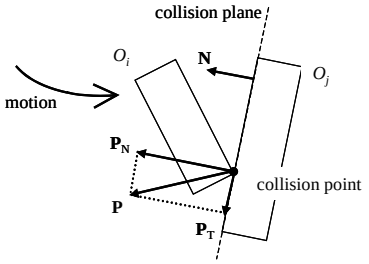


Figure 7. Collision plane.

and $\boldsymbol{\omega}_i$ be the linear and angular velocity of O_i with respect to the center of mass of O_i, before the collision. The impulse-based collision response method aims to calculate the linear velocity $\overline{\mathbf{v}}_i$ and the angular velocity $\overline{\boldsymbol{\omega}}_i$ of O_i after the collision.

The impulse-based collision response method starts from the law of momentum conservation. Since the change of momentum before and after the collision equals to the sum of impulses at the time of collision, the impulse equations for O_i can be expressed as follows:

$$m_i(\overline{\mathbf{v}}_i - \mathbf{v}_i) = \mathbf{P} + \sum_j \mathbf{P}_{ij}$$

and

$$\mathbf{I}_i(\overline{\boldsymbol{\omega}}_i - \boldsymbol{\omega}_i) = \mathbf{l}_i \times \mathbf{P} + \sum_j \mathbf{l}_{ij} \times \mathbf{P}_{ij} \ ,$$

where $\mathbf{P}$ is the impulse applied to O_i and $\mathbf{P}_{ij}$ is the attachment impulse on O_i due to O_j, which is connected

to O_i using a joint constraint. When O_i and O_j are not directly connected to each other, $\mathbf{P}_{ij}$ is a null vector. Notice that $\mathbf{P}$ can be zero for non-colliding components. The vectors $\mathbf{l}_i$ and $\mathbf{l}_{ij}$ are the distance vectors from the center of mass of O_i to the collision point and to the joint connecting O_i and O_j, respectively, as shown in Figure 6.

Joint constraints also give additional equations. When a spherical joint connects O_i and O_j, their relative velocity at the contact point should be equal to each other:

$$\overline{\mathbf{v}}_i + \overline{\boldsymbol{\omega}}_i \times \mathbf{l}_{ij} = \overline{\mathbf{v}}_j + \overline{\boldsymbol{\omega}}_j \times \mathbf{l}_{ji} .$$

In the case of nail constraints, a point on the component O_k has a fixed position. Thus, the linear velocity of the nailed point is zero:

$$\overline{\mathbf{v}}_k + \overline{\boldsymbol{\omega}}_k \times \mathbf{l}_{kk} = 0 ,$$

where $\mathbf{l}_{kk}$ is the vector from the center of mass of O_k to the nailed point.

Moore extended this formulation to cases with friction. However this requires solving the whole systems of linear equations repeatedly to determine the friction status of each collision point. We improve this method by combining it with Mirtich's conditional equation for friction.[15]

When a point of O_i is colliding with a face of O_j, the plane containing that face is defined as the collision plane, as shown in Figure 7. With respect to the normal vector $\mathbf{N}$ of the collision plane, the impulse $\mathbf{P}$ for O_i can be divided into two components: the normal component $\mathbf{P}_N$ and the tangential component $\mathbf{P}_T = \mathbf{P} - \mathbf{P}_N$. The state of friction can be classified into two cases: *sticking case* and *sliding case*. From the viewpoint of impulse, the sticking case means there is no slip at the collision point, which satisfies the condition of $|\mathbf{P}_T| \le \mu |\mathbf{P}_N|$ with the friction coefficient μ. When $|\mathbf{P}_T| > \mu |\mathbf{P}_N|$, the collision point will slip along the collision plane and it is the sliding case.

Since the sliding and sticking cases result in different equations, it is important to identify whether a collision point is sticking or sliding. For sticking cases, the collision point does not move along the

collision plane. In contrast, the friction force will act on the sliding collision points.

Using Moore's formulation, it is impossible to decide whether a collision point is sliding or sticking without calculating the impulse. Mirtich introduced a prediction equation for sliding conditions.[15] The friction at a collision point is sliding when it satisfies the following condition:

$$\left(\frac{1}{k_{13}}\right)^2 + \left(\frac{1}{k_{23}}\right)^2 > \mu^2 \left(\frac{1}{k_{33}}\right)^2 ,$$

where the 3-by-3 matrix $\mathbf{K} = [k_{ij}]$ equals to $(1/m_i + 1/m_j)\mathbf{E} - (\mathbf{l}_i \times \mathbf{I}_i^{-1} \times \mathbf{l}_i + \mathbf{l}_j \times \mathbf{I}_j^{-1} \times \mathbf{l}_j)$ with the 3-by-3 identity matrix **E**. We use this prediction equation to speed up the impulse calculation.

7. Example

Our example, *Steel Fish*, has 9 components connected by 8 joints. Our constraint dynamics solver uses total of 24 constraint functions for the three-directions of 8 joints. Thus, we need to solve the 24-by-24 matrix equation to get the constraint forces.

For the dynamics-based simulation, we should calculate linear velocities and angular velocities of 8 components excluding the fixed component O_0. Using the impulse-based collision handling, we also need to calculate the impulse **P** and the attachment impulses $\mathbf{P}_{ij}$'s of 8 joints. Thus, the number of unknowns is 81, and the impulse dynamics solver needs to solve the 81-by-81 matrix equation.

An example sequence of images generated by our virtual mobile system is shown in Figure 8. Our system was executed on a personal computer with 350MHz Pentium chip and 64Mbyte of main memory. We use software-implemented OpenGL libraries[18] for rendering, without any hardware acceleration. The simplified virtual wind model and customized dynamics solvers enable us to simulate the example mobile interactively. Figure 9 is another sequences of images for the mobile named "Southern Cross" which is also originally created by A. Calder.

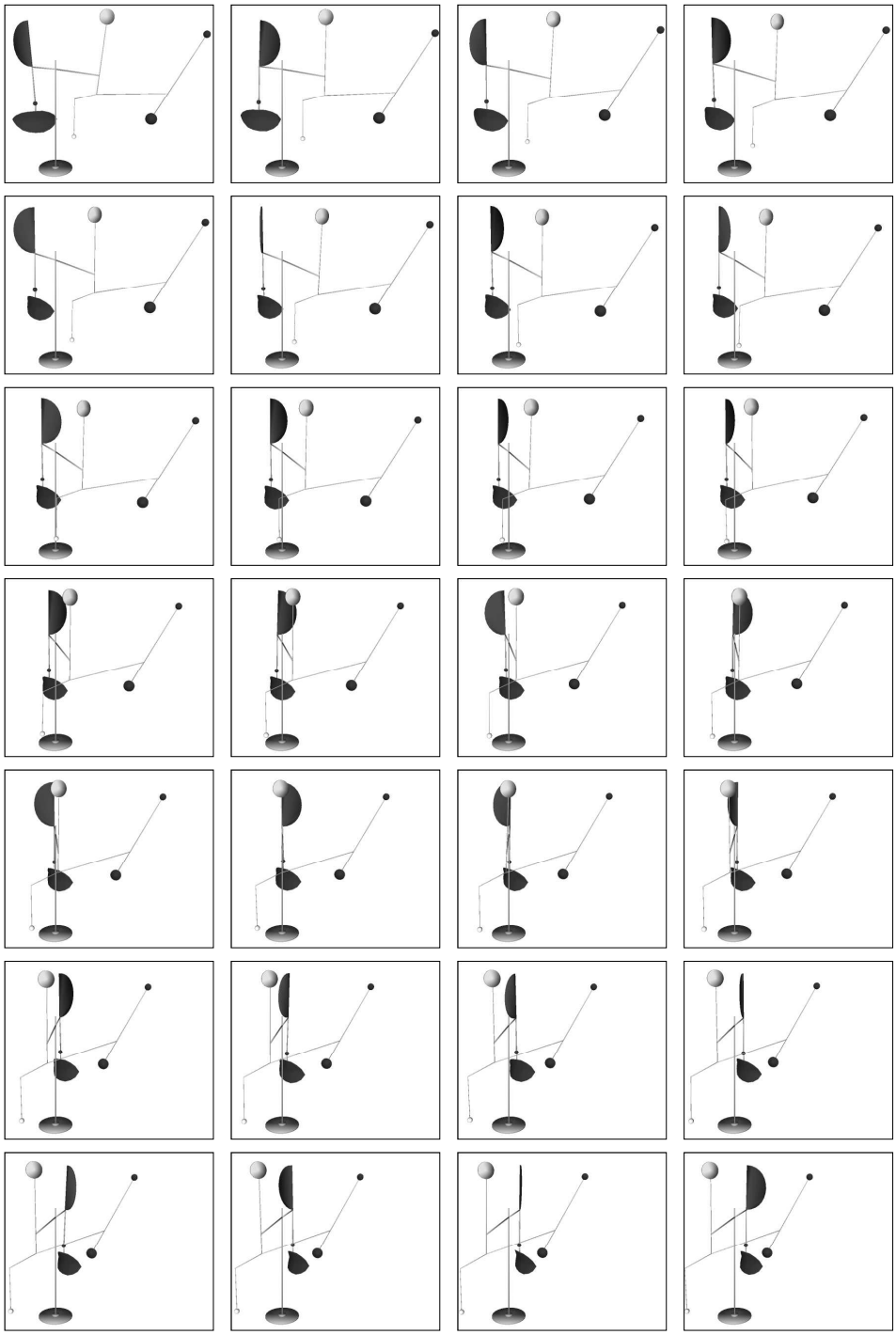


Figure 8. Example sequences of images.

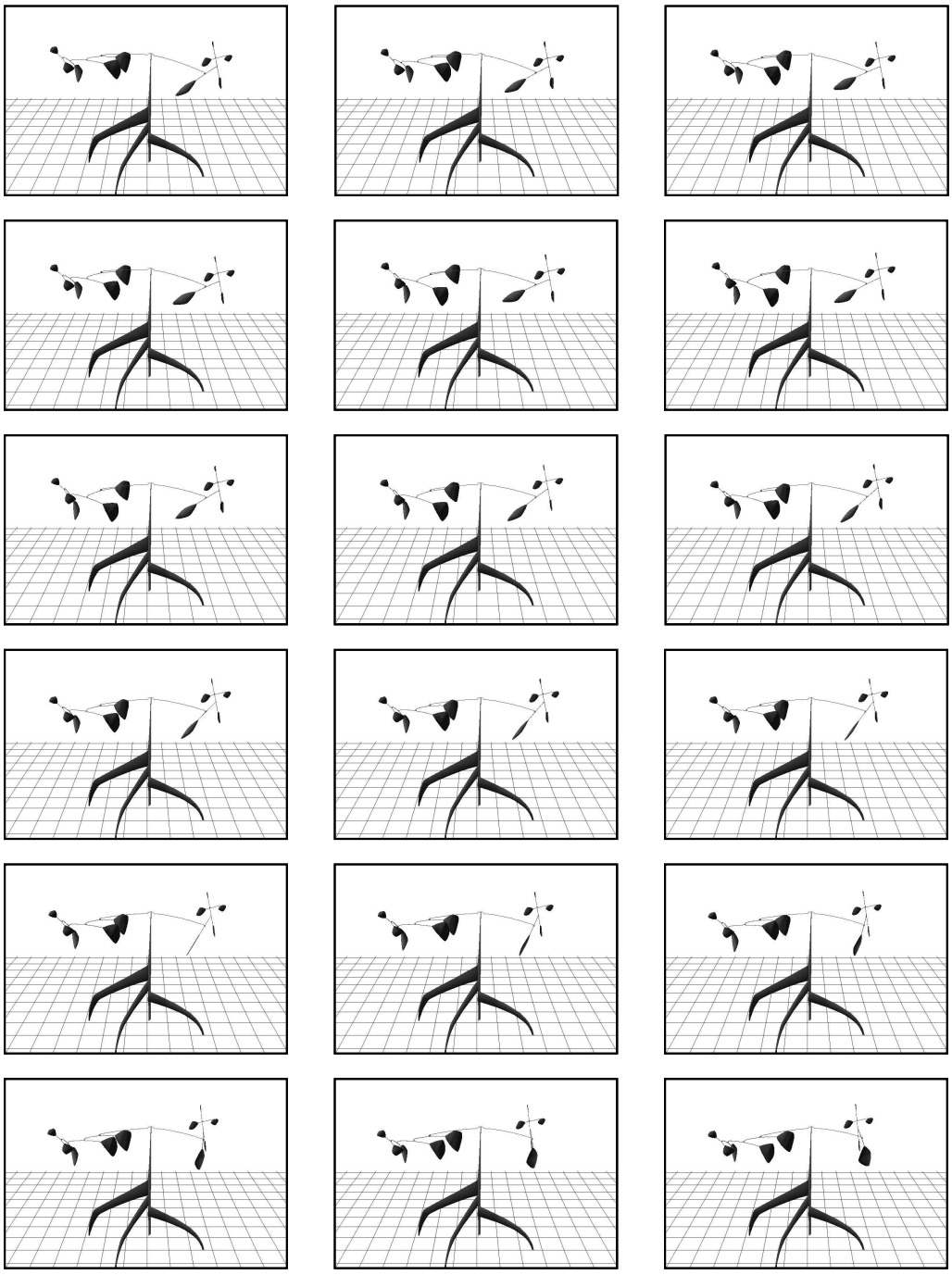


Figure 9. Another example sequences of images.

8. Conclusion

Our aim was to reproduce a real-world mobile on a low-end computer system. To achieve this goal, we developed a dynamics-based virtual mobile system. To interactively display the virtual mobile, our system concentrates on three improvements: the virtual wind model, constraint dynamics solver, and the impulse dynamics solver.

First, we suggested a wind model, which is simple but sufficient to simulate directional winds. Additionally, the microphone interface is developed for easy control of the virtual wind. Since this wind model can generate directional winds, it is suitable for simulating artificial winds generated by electric fans, air-conditioners, etc. Improving our wind model by combining with existing natural wind models[6, 7] will be a future work.

As a dynamics system, our virtual mobile system uses a constraint dynamics solver and an impulse dynamics solver. For real-time display, both are highly tuned for simulating typical mobiles. Since usual mobiles are tree-like shapes, this improvement can also be used for general tree-like objects.

The specific real example of the mobile allowed us to focus on the technical problems. While such systems have inherent value, perhaps more important is the fact that the improvements that have been made to existing techniques can be used to simulate other real-time systems.

References

1. H. McWhinnie, “The Electronic Museum”, *Computers and Graphics*, 12(2):269, 1988.

2. F. McGuire, “The origins of sculpture: Evolutionary 3D design”, *IEEE CG&A*, 13(1):9–11, 1993.

3. P. Rademacher and G. Bishop, “Multiple-Center-of-Projection Images”, *SIGGRAPH’98*, pp.199–206, 1998.

4. B. Mirtich, “V-Clip: Fast and Robust Polyhedral Collision Detection”, *ACM Trans. on Graphics*, 17(3):177–208, 1998.

5. W. Reeves, “Particle Systems – A Technique for Modeling a Class of Fuzzy Objects”, *ACM Trans. on Graphics*,

2(2):91–108, 1983.

6. J. Wejchert and D. Haumann, “Animation Aerodynamics”, *SIGGRAPH’91*, pp.19–22, 1991.

7. M. Shinya and A. Fournier, “Stochastic Motion – Motion Under the Influence of Wind”, *EUROGRAPHICS’92*, 11(3):119–128, 1992.

8. A. Patterson, *A First Course in Fluid Dynamics*, Cambridge University Press, 1989.

9. R. Feynmann, R. Leighton and M. Sands, *The Feynman Lectures on Physics*, Addison-Wesley, 1965.

10. P. Schröder and D. Zeltzer, “The Virtual Erector Set: Dynamic Simulation with Linear Recursive Constraint Propagation”, Computer Graphics (1990 Symposium on Interactive 3D Graphics), 24(2):23–31, 1990.

11. M. Surles, “An Algorithm with Linear Complexity for Interactive, Physically-based Modeling of Large Proteins”, *SIGGRAPH’92*, pp.221–230, 1992.

12. D. Baraff, “Linear-Time Dynamics using Lagrange Multipliers”, *SIGGRAPH’96*, pp.137–146, 1996.

13. M. Gleicher and A. Witkin, “Through-the-lens Camera Control”, *SIGGRAPH’92*, pp.331–340, 1992.

14. P. Hubbard, “Approximating Polyhedra with Spheres for Time-Critical Collision Detection”, *ACM Trans. on Graphics*, 15(3):179–210, 1996.

15. B. Mirtich, “Impulse-based Dynamic Simulations of Rigid Body Systems”, *Ph.D. thesis*, University of California, Berkeley, 1996.

16. J. Hahn, “Realistic Animation of Rigid Bodies”, *SIGGRAPH’88*, pp.299–308, 1988.

17. M. Moore and J. Wilhelms, “Collision Detection and Response for Computer Animation”, *SIGGRAPH’88*, pp.289–298, 1988.

18. J. Neider, T. Davis and M. Woo, *OpenGL Programming Guide*, Addison-Wesley Publishing Company, 1993.