Animating Sand, Mud, and Snow

Robert W. Sumner, James F. O'Brien, Jessica K. Hodgins

College of Computing and Graphics, Visualization, and Usability Center
Georgia Institute of Technology
801 Atlantic Drive
Atlanta, GA 30332-0280
[sumner|obrienj|jkh]@cc.gatech.edu

Accepted to *Graphics Interface '98*
Michael A. J. Sweeney Best Student Paper

Abstract

Computer animations often lack subtle environmental changes that should occur due to the actions of the characters. Squealing car tires usually leave no skid marks, airplanes rarely leave jet trails in the sky, and most runners leave no footprints. In this paper, we describe a simulation model of ground surfaces that can be deformed by the impact of rigid body models of animated characters. To demonstrate the algorithms, we show footprints made by a simulated runner in sand, mud, and snow as well as bicycle tire tracks, a bicycle crash, and a falling runner. The shapes of the footprints in the three surfaces are quite different, but the effects were controlled through only six essentially orthogonal parameters. To assess the realism of the resulting motion, we compare the simulated footprints to video footage of human footprints in sand.

Keywords: animation, physical simulation, ground interaction.

1 Introduction

To become a communication medium on a par with movies, computer animations must present a rich view into an artificial world. Texture maps applied to three-dimensional models of scenery help to create some of the required visual complexity. But static scenery is only part of the answer; subtle motion of many elements of the scene is also required. Trees and bushes should move in response to the wind created by a passing car, a runner should crush the grass underfoot, and clouds should drift across the sky. While simple scenery and sparse motion can sometimes be used effectively to focus the attention of the viewer, missing or inconsistent action may also distract the viewer from the plot or intended message of the animation. One of the principles of animation is that the viewer should never be unintentionally surprised by the motion or lack of it in a scene[22].

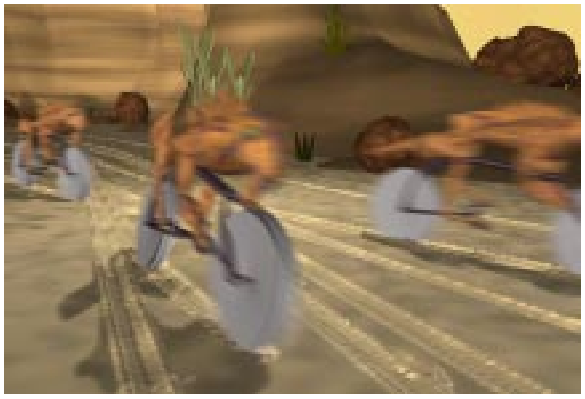
Figure 1: Image of tracks left in the sand by a group of fast moving, motion blurred, alien bikers.

Movie directors face a related problem because they must ensure that the viewer is presented with a consistent view of the world and the characters. An actor's clothing and makeup should not inexplicably change from scene to scene, lighting should be consistent across edits, and absent, unexpected, or anachronistic elements such as missing tire tracks, extra footprints, or jet trails in the background of a period piece must be avoided. The risk of distracting the viewer is so great that one member of the director's team known as a "continuity girl," "floor secretary," or "second assistant director," is responsible solely for maintaining consistency[17].

Maintaining consistency is both easier and harder in computer animation. Because we are creating an artificial world, we can control the lighting conditions, layout, and other scene parameters and recreate them if we need to "shoot" a fill-in scene later. Because the world is artificial, however, we may be tempted to rearrange objects between scenes for best effect, thereby creating a series of scenes that could not exist in a consistent world. Computer-generated animations and special effects add another facet to the consistency problem because making the models and motion appropriately responsive is a lot of work. For example, most animated figures do not leave tracks in the environment as a human actor would and special effects artists have had to work hard to create subtle but essential effects such as environment maps of flickering flames. Because each detail of the scene represents additional work, computer graphics environments are generally conspicuously clean and sparse. The approach presented here is a partial solution to this problem; we create a more interesting environment by allowing the character's actions to change a part of the environment.

In this paper, we describe a model of ground surfaces and explain how these surfaces can be deformed by characters in an animation. The ground material is modeled as a height field formed by vertical columns. After the impact of a rigid body model, the ground material is deformed by allowing compression of the material and movement of material between the columns. To demonstrate the algorithms, we show the creation of footprints in sand, mud, and snow. These surfaces are created by modifying only six essentially independent parameters of the simulation. We evaluate the results of the animation through comparison with video footage of human runners and through more dramatic patterns created by bicycle tire tracks (figure 1), a falling bicycle (figure 7), and a tripping runner (figure 8).

2 Background

Several researchers have investigated the use of procedural techniques for generating and animating background elements in computer-generated scenes. Although we are primarily interested in techniques that allow the state of the environment to be altered in response to the motions of an actor, methods for animating or modeling a part of the environment independent of the movements of the actors are also relevant because they can be modified to simulate reactive behavior.

The most closely related previous work is that of Li and Moshell[10]. They developed a model of soil that allows interactions between the soil and the blades of digging machinery. Soil spread over a terrain is modeled using an array of posts that represent the height of the soil at a given location. Soil that is pushed in front of a bulldozer's blade is modeled as discrete chunks. Although they discount several factors that contribute to soil behavior in favor of a more tractable model, their technique is physically based and they arrive at their simulation formulation after a relatively detailed analysis of soil dynamics. As these authors note, actual soil dynamics are complex and their model, therefore, focuses on a specific set of actions that can be performed on the soil, namely the effect of horizontal forces acting on the soil causing displacements and soil slippage. The method we present here has obvious similarities to that of Li and Moshell, but we focus on modeling a different set of phenomena at different scales. We also adopt a more appearance-based approach in the interest of developing a technique that can easily model a wide variety of ground materials for animation purposes.

Another environmental phenomenon that allows interaction is water. Early work by Peachey[14] and by Fournier and Reeves[7] used procedural models based on specially designed wave functions to model ocean waves as they travel and break on a beach. Later work by Kass and Miller[9] developed a more general approach using shallow water equations to model the behavior of water under a variety of conditions. Their model also modified the appearance of a sand texture as it became wet. O'Brien and Hodgins[13] extended the work of Kass and Miller to allow the behavior of the water simulation to be affected by the motion of other objects in the environment and to allow the water to affect the motion of the other objects. They included examples of objects floating on the surface and human actors diving into pools of water. More recently Foster and Metaxas[5] used a variation of the three-dimensional Navier-Stokes equations to model fluids. In addition to these surface and volumetric approaches, particle-based methods have been used to model water spray and other loosely packed materials. Supplementing particle models with inter-particle dynamics allows a wider range of phenomena to be modeled. Exam-

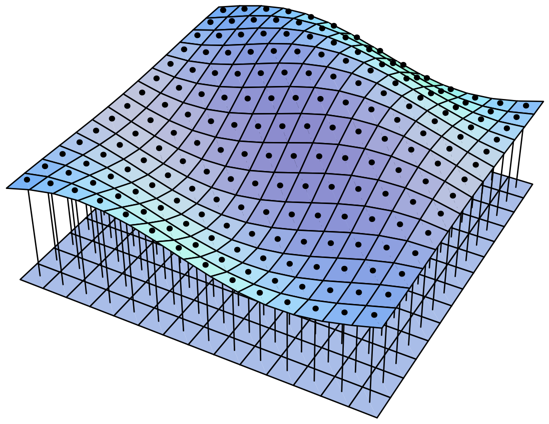

Figure 2: The uniform grid forms a height field that represents the surface of the ground. Each grid point within the height field represents a vertical column of ground material with the top of the column centered at the grid point.

ples of these systems include Reeves[16], Sims[18], Miller and Pearce[12], and Terzopoulos, Platt, and Fleischer[21].

Other environmental effects that have been animated include clouds and gases[4, 20, 6], fire[1, 20], lightning[15], and leaves blowing in the wind[23].

Simulation of interactions with the environment can also be used to generate still models. Several researchers have described techniques for generating complex plant models from grammars describing how the plant should develop or grow over time. Měch and Prusinkiewicz[11] developed techniques for allowing plants to affect and be affected by their environment as they develop. Dorsey and her colleagues[2, 3] used simulation to model how an object's surface changes over time as environmental factors act on it.

3 Simulation of Sand, Mud, Snow

In this paper, we present a general model of a deformable ground material. The model consists of a height field supported by vertical columns of material. Using displacement and compression algorithms, we animate the deformations that are created when rigid geometric objects impact the ground material. These models have been used to animate the creation of footprints, tire tracks, and other patterns on the ground. The properties of the model can be varied to produce deformations with the behavior of different ground materials such as sand, mud, and snow.

3.1 Model of Ground Material

Our simulation model discretizes a continuous volume of ground material by dividing the surface of the volume into a uniform rectilinear grid that defines a height field (figure 2). The resolution of the grid must be chosen appropriately for the size of the models that will collide with it and for the size of the features in the ground surface. For example, in figure 1 the resolution of the grid is 1 cm and the bicycles are approximately 2 meters long.

Initial conditions for the height of each grid point can be created procedurally or imported from a variety of sources. For illustrative purposes, we can assign all grid points a constant height. We also implemented more realistic initial conditions with noise generated on an integer lattice and interpolated with cubic Catmull-Rom splines (a variation of a two-dimensional Perlin noise function[4]). Terrain data or the output from a modeling program could also be used for the initial height field. Alternatively, the initial conditions could be the output of a previous simulation run. For example, the pockmarked surface of a public beach at the end of a busy summer day could be modeled by simulating many criss-crossing footfalls.

3.2 Motion of the Ground Material

The height field represented by the top of the columns is deformed as rigid geometric objects push into the grid. For the examples given in this paper, the geometric objects are a runner's shoe, a bicycle tire and frame, and a jointed human figure. The motion of the rigid bodies was computed using a dynamic simulation of a human running, bicycling, or falling down on a smooth, hard ground plane[8]. The resulting motion was given as input to the simulation of the ground material in the form of trajectories of positions and orientations of the geometric objects. Because of this generic specification of the motion, the input motion need not be dynamically simulated but can be keyframe or motion capture data.

The simulation approximates the motion of the columns of ground material by compressing or displacing the material under the rigid geometric objects. At each time step, a test is performed to determine whether any of the rigid objects have intersected the height field. The height of the affected columns is reduced until they no longer penetrate the surface of the rigid object. The material that was displaced is either compressed or forced outward to surrounding columns. A series of erosion steps are then performed to reduce the magnitude of the slopes between neighboring columns. Finally, particles can be generated from the contacting surface of the rigid object to mimic the spray of material that is often seen following an impact. We now discuss each step of the algorithm in more detail: collision, displacement, erosion, and particle generation.

Collision. The collision algorithm determines whether a rigid object has collided with the ground surface. For each column, a ray is cast from the bottom of the column through the vertex at the top. If the ray intersects a rigid object before it hits the vertex, then the rigid object has penetrated the surface and the top of the column is moved down to the intersection point. A flag is set to indicate that the column was moved, and the change in height is stored. The ray intersection tests are sped up by partitioning the polygons of the rigid body models using an axis-aligned bounding box hierarchy[19].

Using a vertex coloring algorithm, the simulation also computes the distance from each column that has collided with the object to the closest column that has not collided. This information is used when the material displaced by the collision is distributed. A representative map is shown in figure 3. The contour map is computed by iterating over the entire grid until all columns have been assigned a value. As an initialization step, columns not in contact with the object are assigned the value zero. During subsequent iterations, unlabeled columns adjacent to labeled columns are assigned a value equal to the value of the lowest adjacent column plus one (assuming eight-way connectivity).

Displacement. Ground material from the columns that are in contact with the object is either compressed or distributed to surrounding columns that are not in contact with the object. The compression ratio α is determined by the user and is one of the parameters that can control the visual appearance of the ground material. The material to be distributed, Δh, is computed by:

$$\Delta h = \alpha m \tag{1}$$

where m is the total amount of displaced material. The material that is not compressed is propagated down the contour map until columns that are not in contact with the object are reached. Material from a column is equally distributed among the neighbors with lower contour values. In this way, the ground material is redistributed to the closest ring of columns not in contact with the rigid object. The heights of the columns in this ring are increased to reflect the newly deposited material. This change in height may result in new collisions with the object that will be detected during the next time step.

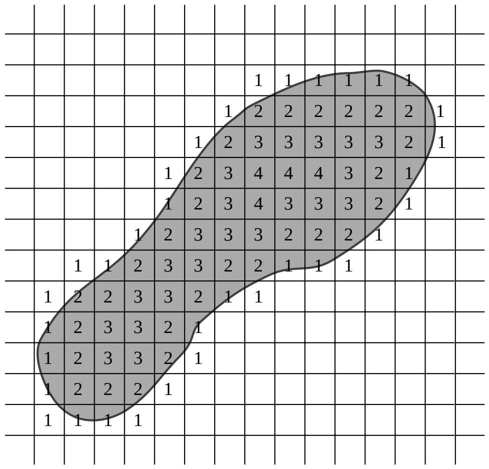
Figure 3: The contour map represents the distance from each column in contact with the foot to a column that is not in contact. For this illustration, we used columns that are four-way connected. However, in the examples in this paper we used eight-way connectivity because we found that the higher connectivity yielded smoother results.

Erosion. Because the displacement algorithm deposits material only in the first ring of columns not in contact with the object, the heights of these columns may be increased in an unrealistic fashion. An “erosion” algorithm is used to identify columns that form steep slopes with their neighbors and move material down the slope to form a more realistic mound. Several parameters allow the user to control the shape of the mound and mimic the motion of different ground materials.

The erosion algorithm examines the slope between each pair of adjacent columns in the grid (assuming eight-way connectivity). For a column ij and a neighboring column kl, the slope, s, is

$$s = \tan^{-1}(h_{ij} - h_{kl})/d \tag{2}$$

where h_{ij} is the height of the column, h_{kl} is the height of the neighboring column, and d is the distance to the neighbor. If the slope is greater than a threshold θ_{out}, then ground material is moved from the higher column down the slope to the lower column. Ground material is moved by computing the average difference in height, Δh_a, for all the neighboring columns with too great a slope:

$$\Delta h_a = \frac{\sum(h_{ij} - h_{kl})}{n} \tag{3}$$

where n is the number of neighbors with too great a downhill slope. The average difference in height is multiplied by a fractional constant, σ, and the resulting quantity is equally distributed among the downhill neighbors. The algorithm repeats until all slopes are below a threshold, θ_{stop}. In the special case that a neighboring column is in contact with the geometric object, a different threshold, θ_{in}, is used to control the slope. This threshold gives the user independent control over the inner slope of the mound of material around the geometric object.

Particles Generation. We use a particle system to model portions of the ground material that are thrown into the air by the motion of the geometric objects. The user controls the adhesiveness between the object and the material as well as the rate at which the particles fall from the object. Each triangle of the object that is in contact with the ground picks up a portion of the ground material during contact. The volume of material attached to a triangle, v, is determined by the area of the triangle multiplied by an adhesion constant for the material. After the triangle is no longer in contact with the ground, it gradually drops the attached material as particles. The rate at which the material is dropped is computed with an exponential decay. For each time step the volume of dropped material is

$$\Delta v = v(e^{(-t+t_c+\Delta t)/h} - e^{(-t+t_c)/h}) \tag{4}$$

where v is the initial volume attached to the triangle, t is the current time, t_c is the time at which the triangle left the ground, Δt is the size of the time step, and h is a half life parameter that controls how quickly the material falls off. The number of particles released on a given time step is determined by

$$n = \Delta v \phi \tag{5}$$

where $\frac{1}{\phi}$ is the volume of each particle.

The initial positions for the particles are randomly distributed over the surface of the triangle. The location of a particle is determined probabilistically as follows:

$$\mathbf{p_0} = b_a \mathbf{x}_a + b_b \mathbf{x}_b + b_c \mathbf{x}_c \tag{6}$$

where $\mathbf{x}_a$, $\mathbf{x}_b$, and $\mathbf{x}_c$ are the coordinates of the vertices of the triangle and b_a, b_b, and b_c are the barycentric coordinates of the point given by

$$\begin{aligned} b_a &= 1.0 - \sqrt{\rho_a} && (7) \\ b_b &= \rho_b(1.0 - b_a) && (8) \\ b_c &= 1.0 - (b_a + b_b) && (9) \end{aligned}$$

ρ_a and ρ_b are independent random variables evenly distributed between [0..1]. This results in a uniform distribution over the triangle.

The initial velocity is computed from the velocity of the rigid object

$$\dot{\mathbf{p}}_0 = \nu + \omega \times \mathbf{p}_0 \tag{10}$$

where ν, and ω are respectively the linear and angular velocity of the object. To give a more realistic and appealing look to the particle motion, the initial velocities are randomly perturbed.

The final component of the particle creation algorithm accounts for the greater probability that material will fall off fast moving objects. The particle is only created if $(|\dot{\mathbf{p}}_0|/s)^\gamma > \rho$, where s is the minimal speed at which all potential particles will be dropped, γ controls the variation of the probability of particle creation with speed, and ρ is a random variable evenly distributed in the range [0..1].

Once created, the particles fall under the influence of gravity. Upon striking the surface of the ground, their volume is added to the volume of the column on which they land.

3.3 Implementation and Optimization

Simulations of terrain generally span a large area. For example, we would like to be able to simulate a runner jogging on a beach, a skier gliding down a snow-covered slope, and a stampede of animals crossing a sandy valley. A naive implementation would be intractable because of the memory and computation requirements. The next two sections describe the optimizations that allow us to achieve reasonable performance by storing and simulating only the active portions of the surface and by parallelizing the computation.

Algorithm Complexity. Because the ground model is a two-dimensional rectilinear grid, the most straightforward implementation is a two-dimensional array of nodes containing the height and other information about the column. If an animation required a grid of i rows and j columns, $i \times j$ nodes would be needed, and computation time and memory would grow linearly with the number of grid points. Thus, a patch of sand 10 meters by 10 meters with a grid resolution of 1 cm yields a 1000×1000 grid with one million nodes. If each node requires 10 bytes of memory, the entire grid requires 10 Mbytes. Even this relatively small patch of sand requires significant system resources. However, most of the ground nodes remain static throughout

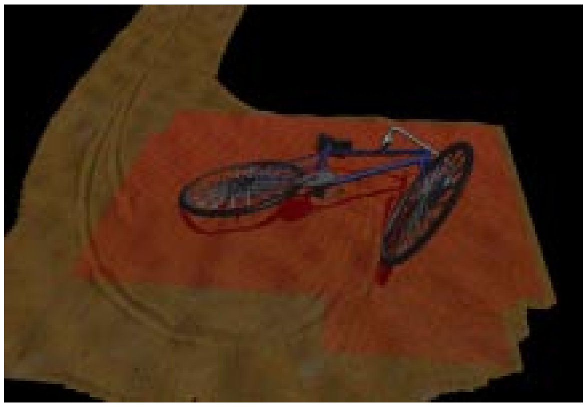
Figure 4: The active columns in the hash table are shown in red. The sand is rendered only for the area stored in the hash table (the area that was in contact with the bounding box of the bike at some time during the simulation). The number of cells in the red area is approximately 37,000 while the number of cells in the entire virtual grid is greater than 2 million.

the simulation. This observation allowed us to implement a much more efficient algorithm that creates only the active nodes.

The i, j position of a particular node is used as the index into a hash table allowing the algorithms to be implemented as if a simple array of nodes were being used. The hash table is initially empty and is filled with nodes as the rigid objects move through the scene. On each time step, nodes that are covered by a projection of the rigid objects onto the surface are marked as active. The actual projection is an enlarged bounding box for the rigid objects (figure 4). If the active nodes do not exist in the hash table, they are added. The collision detection, displacement, and erosion algorithms are applied, not to the entire grid, but only to the active grid points.

Because only the active grid points are processed, the computation time is now a function of the size of the rigid objects in the scene rather than the total grid size. Memory requirements are also significantly reduced. However, the state of all modified nodes must be stored even after they are no longer covered by a bounding box because a rigid object may impact those grid points at a later time. For example, in the scene depicted in figure 1, the bicyclists ride over the tire tracks of other bicyclists.

Parallel Implementation. Despite the optimization provided by simulating only active nodes, the computation time grows linearly with the projected area of the rigid objects. Adding the rigid objects for a second character will approximately double the active area. However, the computation time for multiple characters can be reduced by using parallel processing when the characters are affecting independent patches of ground.

In our parallel implementation, a parent process maintains the state of the grid and spawns a child process for every character in the animation. Each child process maintains a local copy of the grid and performs its computations in parallel. After each time step, the multiple copies of the grid are synchronized through a two stage communication routine. First, each child reports the changes in its copy of the grid to the parent process. The parent process then updates the master copy of the grid and reports all changes to the children. The data that must be sent to the parent and to the children is relatively small because the motion of any rigid object during a single time step is small.

We have implemented this design on a 12 processor SGI Power Challenge using UNIX pipes to handle communication. Because the parallel implementation does not rely on shared memory, it could easily be adapted for multiple machines using sockets instead of pipes. However, network delays between multiple machines would be more significant than the communication time on a single multiprocessor. This parallel implementation assumes that the projected bounding boxes of the rigid objects for different characters do not overlap. A more sophisticated implementation could handle this case by assigning characters with overlapping bounding boxes to the same processor.

4 Animation Parameters

Ground materials can behave in many different ways. One goal of this research is to create a tool that allows animators to easily generate a significant fraction of this variety. Six parameters of the simulation can be changed by the user in order to achieve different effects: liquidity, roughness, inside slope, outside slope, compression, and particle adhesion. The first four are used by the erosion algorithm, the fifth is used by the displacement algorithm, and the final parameter is used by the particle system.

Liquidity or θ_{stop} determines how watery the material appears by modifying how many times the erosion function is called per time step. With less erosion per time step, the surface appears to flow outward from the intersecting object; with more erosion, the surface moves to its final state more quickly.

Roughness or σ controls the irregularity of the ground deformations by changing the amount of ma-

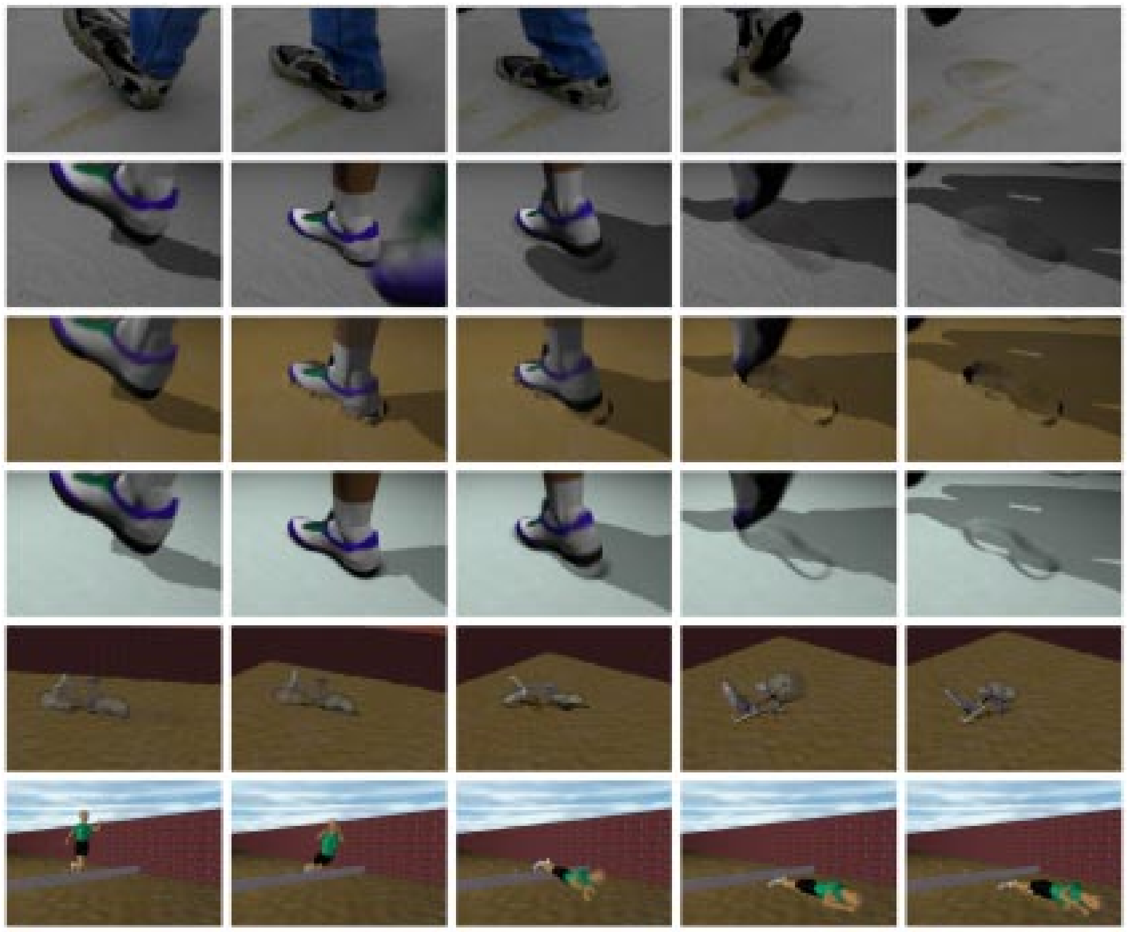

Figure 5: Images from video footage of a human runner stepping in sand and a simulated runner stepping in sand, mud, and snow. The human runner images are separated by 0.133 s; the simulated images are separated by 0.1 s. Images of a crashing bicycle (0.5 s spacing) and a tripping runner (0.166 s spacing).

Variable	Sand	Mud	Snow
liquidity (θ_{stop})	0.8	1.1	1.57
roughness (σ)	0.2	0.2	0.2
inside slope (θ_{in})	0.8	1.57	1.57
outside slope (θ_{out})	0.436	1.1	1.57
compression (α)	0.3	0.41	0.0

Figure 6: Table of parameters for the three ground materials.

terial that is moved from one column to another during erosion. Small values yield a smooth mound of material while larger values give a rough, irregular surface.

The inside and outside slope parameters (θ_{in} and θ_{out}) modify the shape of a mound of ground material by changing the slope adjacent to intersecting geometry and the slope on the outer part of the mound. Each parameter is a threshold that is used to determine if a particular column should erode to its neighbors. Small values lead to more erosion and a more gradual slope; large values yield less erosion and a steeper slope.

The compression or α parameter determines how much a given column compresses when it is pushed down by an object and, therefore, offers a way to model substances of different densities. In the displacement algorithm, this parameter determines how much displaced material is distributed outward from an object that has intersected the grid. A value of one causes all material to be displaced; a value less than one allows some of the material to be discarded.

Finally, the rate of creation of particles is controlled primarily by a parameter representing the adhesion between the ground material and the object. We included particles in the animations of sand but did not include them in the animations of mud or snow. Other more dynamic motions such as snow skiing might generate significant spray but running in snow appears to generate clumps of snow rather than particles.

5 Results and Discussion

Figure 5 shows images of a human runner stepping in sand and a simulated runner stepping in sand, mud, and snow. The parameters used for the simulations of the three ground materials are given in figure 6. The footprints left by the real and simulated runners in sand are quite similar.

Figures 5, 7, and 8 show more complicated patterns created in the sand by a falling bicycle and a tripping runner. For each of these simulations, we used a grid resolution of 1 cm by 1 cm yielding a virtual grid size of 2048×1024 for the bicycle and 4096×512 for the runner.

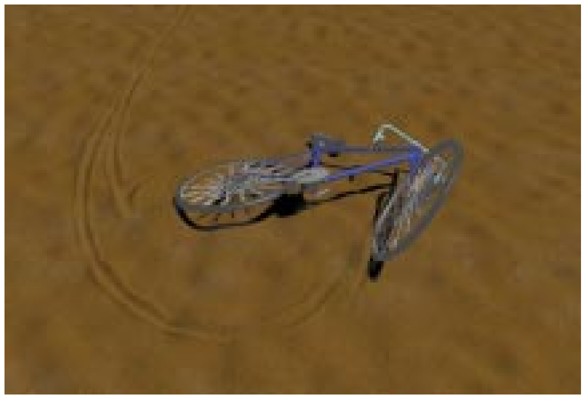
Figure 7: A closeup view of the bicycle at the end of the motion sequence shown in figure 5.

The simulation described in this paper allows us to capture with relative ease many of the behaviors of substances such as sand, mud, and snow. Only about fifteen iterations were required to hand tune the parameters for the desired effect with each material. The computation time is not burdensome: a 3-second simulation of the running figure interacting with a 1 cm by 1 cm resolution ground material required less than 2 minutes of computation time on a single MIPS R10000 processor.

Many effects, however, are missed by this model. For example, wet sand and crusty mud often crack and form large clumps, but our model can only generate smooth surfaces and particles. Actual ground material is not uniform but contains both small grains of sand or dirt as well as larger objects such as rocks, leaves, and seashells. More generally, many factors go into creating the appearance of a given patch of ground: water and wind erosion, plant growth, and the footprints of many people and animals. Some of these more subtle effects are illustrated by the human footprints in snow and mud shown in figure 9.

One significant approximation in this simulation system is that the motion of the human figure is not affected by the deformations of the surface. For the sequences presented here, each of the human simulations interacted with a flat, smooth ground plane. A more accurate and realistic simulation system would allow the bike and runner to experience the undulations in the initial terrain as well as the changes in friction caused by the deforming surfaces. For exam-

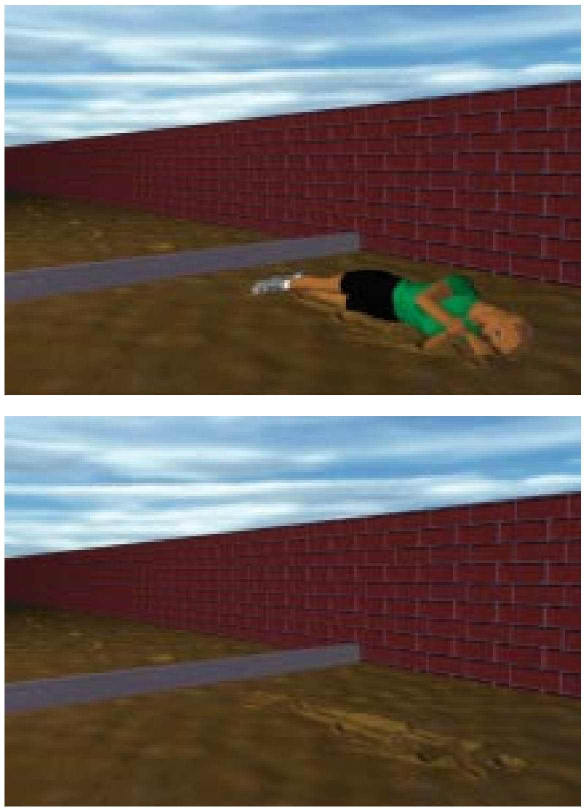

Figure 8: A closeup view of the tripping runner at the end of the motion sequence shown in figure 5 and the pattern that she made in the sand.

ple, a bike is slowed down significantly when rolling on sand and a runner's foot slips slightly with each step on soft ground.

We regard this simulation as appearance-based rather than engineering-based because most of the parameters bear only a scant resemblance to the physical parameters of the material being modeled. The liquidity parameter, for example, varies between 0.0 and $\pi/2$ rather than representing the quantity of water in a given amount of sand. It is our hope that this representation for the parameters allows for intuitive adjustment of the resulting animation without requiring a deep understanding of the simulation algorithms or soil mechanics. The evaluation is also qualitative or appearance-based in that we compare simulated and video images of the footprints rather than matching initial and final conditions quantitatively.

The motions of sand, mud, and snow that we generated are distinctly different from each other because of changes to the simulation parameters. Although much of the difference is due to the deformations determined by our simulations, part of the visual difference results from different surface properties used for rendering. To generate the images in this paper, we had not only to select appropriate parameters for the simulation but also to select parameters for rendering. A more complete investigation of techniques for selecting rendering parameters and texture maps might prove useful.

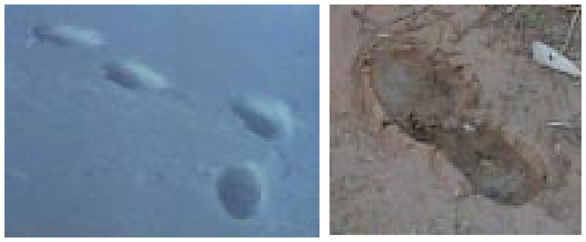

Figure 9: Images of actual human footprints in snow and in mud. The image of snow is from the opening scene of the recent movie *Smilla's Sense of Snow*.

References

[1] N. Chiba, S. Ohkawa, K. Muraoka, and M. Miura. Two-dimensional visual simulation of flames, smoke and the spread of fire. *The Journal of Visualization and Computer Animation*, 5(1):37–54, January–March 1994.

[2] Julie Dorsey and Pat Hanrahan. Modeling and rendering of metallic patinas. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 387–396. ACM SIGGRAPH, Addison Wesley, August 1996. Held in New Orleans, Louisiana, 04-09 August 1996.

[3] Julie Dorsey, Hans Køhling Pedersen, and Pat Hanrahan. Flow and changes in appearance. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 411–420. ACM SIGGRAPH, Addison Wesley, August 1996. Held in New Orleans, Louisiana, 04-09 August 1996.

[4] David Ebert, Kent Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. *Texturing and Modeling: A Procedural Approach.* Academic Press, October 1994. ISBN 0-12-228760-6.

[5] Nick Foster and Demitri Metaxas. Realistic animation of liquids. In *Proceedings of Graphics Interface '96*, pages 204–212, May 1996.

[6] Nick Foster and Dimitris Metaxas. Modeling the motion of a hot, turbulent gas. In *SIG-*

GRAPH 97 Conference Proceedings, Annual Conference Series, pages 181–189. ACM SIGGRAPH, Addison Wesley, August 1997. Held in Los Angeles, California, 03-08 August 1995.

[7] Alain Fournier and William T. Reeves. A simple model of ocean waves. In David C. Evans and Russell J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 75–84, August 1986.

[8] Jessica K. Hodgins, Wayne L. Wooten, David C. Brogan, and James F. O'Brien. Animating human athletics. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 71–78. ACM SIGGRAPH, Addison Wesley, August 1995. Held in Los Angeles, California, 06-11 August 1995.

[9] Michael Kass and Gavin Miller. Rapid, stable fluid dynamics for computer graphics. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 49–57, August 1990.

[10] Xin Li and J. Michael Moshell. Modeling soil: Realtime dynamic models for soil slippage and manipulation. In James T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 361–368, August 1993.

[11] Radomír Měch and Przemyslaw Prusinkiewicz. Visual models of plants interacting with their environment. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 397–410. ACM SIGGRAPH, Addison Wesley, August 1996. Held in New Orleans, Louisiana, 04-09 August 1996.

[12] Gavin Miller and Andrew Pearce. Globular dynamics: A connected particle system for animating viscous fluids. *Computers and Graphics*, 13(3):305–309, 1989.

[13] J. F. O'Brien and J. K. Hodgins. Dynamic simulation of splashing fluids. In *Computer Animation '95*, pages 198–205, April 1995. Held in Geneva, Switzerland, 19-21 April 1995.

[14] Darwyn R. Peachey. Modeling waves and surf. In David C. Evans and Russell J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 65–74, August 1986.

[15] Todd Reed and Brian Wyvill. Visual simulation of lightning. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 359–364. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[16] W. T. Reeves. Particle systems – a technique for modeling a class of fuzzy objects. *ACM Trans. Graphics*, 2:91–108, April 1983.

[17] Karel Reisz and Gavin Miller. *The Technique of Film Editing.* Focal Press, 1989.

[18] Karl Sims. Particle animation and rendering using data parallel computation. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 405–413, August 1990.

[19] John M. Snyder. An interactive tool for placing curved surfaces without interpenetration. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 209–218. ACM SIGGRAPH, Addison Wesley, August 1995. Held in Los Angeles, California, 06-11 August 1995.

[20] Jos Stam and Eugene Fiume. Depicting fire and other gaseous phenomena using diffusion processes. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 129–136. ACM SIGGRAPH, Addison Wesley, August 1995. Held in Los Angeles, California, 06-11 August 1995.

[21] Demetri Terzopoulos, John Platt, and Kurt Fleischer. Heating and melting deformable models (from goop to glop). In *Proceedings of Graphics Interface '89*, pages 219–226, June 1989.

[22] Frank Thomas and Ollie Johnston. *Disney Animation: The Illusion of Life.* Abbeville Press, New York, 1984.

[23] Jakub Wejchert and David Haumann. Animation aerodynamics. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 19–22, July 1991.