An Introduction to Physics-based Animation

ADAM W. BARGTEIL, University of Maryland, Baltimore County, United States
TAMAR SHINAR, University of California, Riverside, United States

Physics-based animation has emerged as a core area of computer graphics finding widespread application in the film and video game industries as well as in areas such as virtual surgery, virtual reality, and training simulations. This course introduces students and practitioners to fundamental concepts in physics-based animation, placing an emphasis on breadth of coverage and providing a foundation for pursuing more advanced topics and current research in the area. The course focuses on imparting practical knowledge and intuitive understanding rather than providing detailed derivations of the underlying mathematics. The course is suitable for someone with no background in physics-based animation—the only prerequisites are basic calculus, linear algebra, and introductory physics.

We begin with a simple, and complete, example of a mass-spring system, introducing the principles behind physics-based animation: mathematical modeling and numerical integration. From there, we systematically present the mathematical models commonly used in physics-based animation beginning with Newton's laws of motion and conservation of mass, momentum, and energy. We then describe the underlying physical and mathematical models for animating rigid bodies, soft bodies, and fluids. Then we describe how these continuous models are discretized in space and time, covering Lagrangian and Eulerian formulations, spatial discretizations and interpolation, and explicit and implicit time integration. In the final section, we discuss commonly used constraint formulations and solution methods.

CCS Concepts: • **Computing methodologies → Simulation by animation**; **Physical simulation**;

Additional Key Words and Phrases: Animation, Dynamics/Simulation, Research

ACM Reference Format:
Adam W. Bargteil and Tamar Shinar. 2019. An Introduction to Physics-based Animation. 1, 1 (May 2019), 57 pages. https://doi.org/10.1145/3305366.3328050

Intended Audience: Beginner (Beginning PhD students and industry professionals).

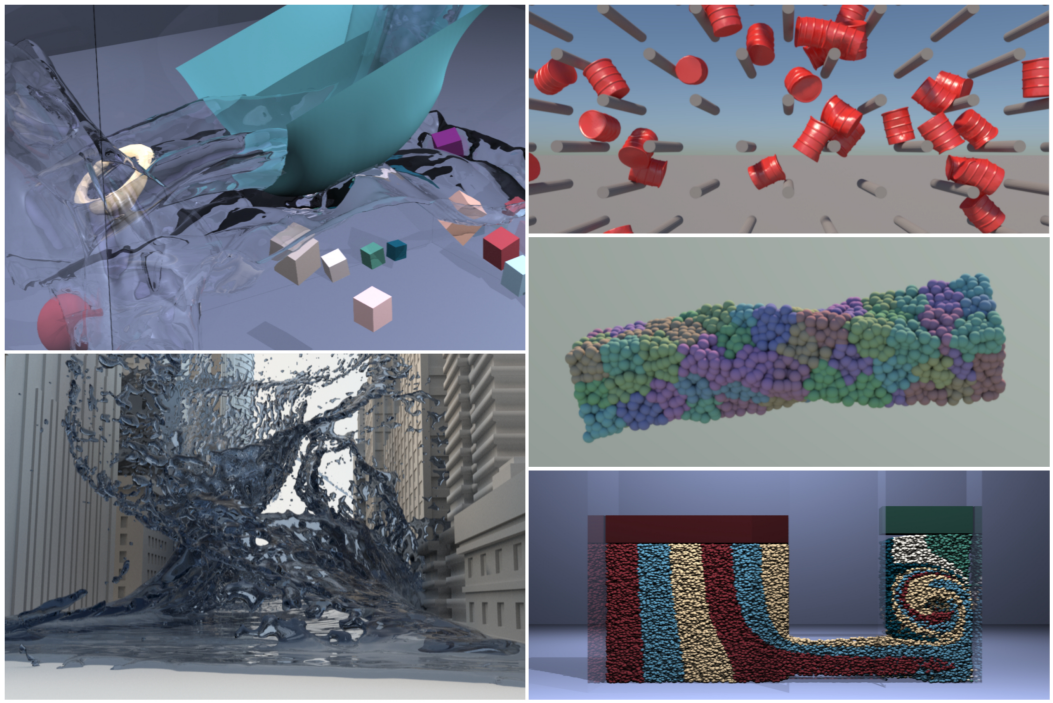
Prerequisites: Calculus, Linear Algebra, Introductory Physics.

Adam W. Bargteil is an assistant professor in the Computer Science and Electrical Engineering department at the University of Maryland, Baltimore County. His primary research interests lie in physics-based animation, an area he has worked in since 2001. His work spans a wide range of phenomena including rigid bodies, soft bodies, and fluids. He earned his Ph.D. in computer science from the University of California, Berkeley and spent two years as a postdoctoral fellow in the School of Computer Science at Carnegie Mellon University. From 2005 to 2007, he was a consultant at PDI/DreamWorks, developing fluid simulation tools that were used in "Shrek the Third" and "Bee Movie." He has offered full semester graduate courses on physics-based animation five times

Authors' addresses: Adam W. Bargteil, adamb@umbc.edu, University of Maryland, Baltimore County, Baltimore, United States; Tamar Shinar, shinar@cs.ucr.edu, University of California, Riverside, Riverside, United States.

SIGGRAPH '19 Courses, July 28 - August 01, 2019, Los Angeles, CA, USA

ACM ISBN 978-1-4503-6307-5/19/07.
https://doi.org/10.1145/3305366.3328050



and has twice taught an undergraduate course on computer animation, about half of which focuses on physics-based animation.

Tamar Shinar is an assistant professor of Computer Science and Engineering at the University of California, Riverside, where she co-leads the Riverside Graphics Lab. She completed her Ph.D. at Stanford University in 2008, where she developed techniques for physics-based animation of solids, fluids, and solid-fluid coupling. She was a postdoctoral fellow at the Courant Institute at NYU, where she developed mathematical and computational models of cell biomechanics. Her research interests center on physics-based animation, with a particular focus on solid-fluid coupling. At UCR, she teaches courses on computer graphics, scientific computing, and physics-based animation.

SCHEDULE

I. A Simple Start: Particle Dynamics
  A. A Passive Particle in a Velocity Field
  B. A Particle with Mass
  C. Spring-Mass Systems
II. Mathematical Models
  A. Physical Laws
    1. Newton's Laws of Motion
    2. Conservation of Mass, Momentum, Energy
  B. Materials
    1. Rigid Bodies
    2. Soft Bodies (Elasticity)
    3. Fluids
III. Spatial Discretization
  A. Lagrangian vs. Eulerian
  B. Grids, Meshes, Particles
  C. Interpolation
  D. Finite Differences
  E. Finite Elements
V. Temporal Discretization
  A. Explicit
    1. Trapezoidal Rule vs. Midpoint Method
    2. Symplectic Euler
  B. Implicit Integration
VI. Constraints
  A. Bilateral and Unilateral Constraints
  B. Soft vs. Hard Constraints
  C. Penalty Forces, Lagrange Multipliers, Generalized Coordinates
  D. Practical Rigid Body Systems
  E. Non-penetration Constraints, Collisions, and Contact

A Simple Start: Particle Dynamics. To begin, we develop a complete physics-based animation system by restricting ourselves to the simplest of models: particle systems. We introduce basic concepts of mass, energy, forces, and velocity. We then provide a refresher on first- and second-order differential equations and introduce numerical integration. Finally, we add springs between particles to allow for animation of soft bodies.

Mathematical Models. We continue by introducing the physical and mathematical models that underlie physics-based animation. We provide a refresher on Newton's laws of motion and then discuss the notions of conservation of mass, momentum, and energy. Then, still working in the continuous domain, we introduce the mathematical models and equations of motion that describe the most commonly used materials in physics-based computer animation: rigid bodies, soft elastic bodies, and fluids.

Spatial Discretization. An important component of physics-based animation is the discretization of spatially continuous quantities. We discuss a variety of techniques related to spatial discretization. First, we explain the difference between Eulerian and Lagrangian frames of reference, both of which are important and widely used and give some examples of their use. We introduce some commonly used grid-based data structures, e.g., collocated and staggered grids, and acceleration structures. We then describe the use of unstructured meshes for spatial discretization, as well as mesh-free particle-based discretizations. Interpolation techniques are critical for communicating between different spatial discretizations, and these are discussed. Finally, we describe two common approaches to discretization of spatial derivatives: finite differences and finite elements.

Temporal Discretization. We first introduce the explicit form of temporal discretization and integration. We discuss truncation error and order of accuracy and point out that methods with the same truncation error can behave differently by examining the trapezoidal rule and the midpoint method for first-order ordinary differential equations. We then discuss explicit integrators for

second-order ordinary differential equations and advocate for the symplectic Euler method, despite its lower order of accuracy. We then turn to implicit integration, which couples together a system of partial differential equations. We then derive the linearly implicit Euler method by explicitly linearizing the equations of motion for elastic bodies (avoiding the need for Taylor expansion). Then we briefly discuss alternatives for solving the resulting linear system and the underlying nonlinear system.

Constraints. In our final section, we discuss the use of constraints in physics-based animation. We discuss the difference between bilateral (equality) and unilateral (inequality) constraints and the resulting implications. We also discuss the difference between soft and hard constraints and the enforcement of constraints through penalty forces, Lagrange multipliers, and generalized coordinates. We then discuss how constraints are enforced in practical constrained rigid body systems. Finally, we discuss one of the most difficult constraint problems in computer animation: non-penetration constraints that result from collision and contact.

1 A SIMPLE START: PARTICLE DYNAMICS

In this section, we develop a complete physics-based animation system by restricting ourselves to the simplest of models: particle systems. We introduce basic concepts of mass, energy, forces, and velocity. We also provide a refresher on first- and second-order differential equations and introduce numerical integration. Finally, we add springs between particles to allow for animation of soft bodies.

1.1 A passive particle in a velocity field

In the simplest particle systems, mass-less particles are advected through given velocity fields. As simple as this case is, talented animators can create compelling animations by varying the velocity fields over space and time. Sometimes the velocity fields are even the result of another simulation.

When advecting a passive particle through a given velocity field, we only need to track the particle's position through time. We must also be able to evaluate the velocity field at a given point in space and at a given instant of time. Thus, we assume that we have a function $\mathbf{v}(\mathbf{x}, t)$ that will provide us with a velocity $\mathbf{v}$ at a position, $\mathbf{x}$, and time, t. A particle's position, which varies over time, $\mathbf{x}_p(t)$ is then the solution of an *Initial Value Problem*:

$$\begin{aligned} \mathbf{x}_p(0) &= \mathbf{x}_0 \\ \frac{d\mathbf{x}_p(t)}{dt} &= \dot{\mathbf{x}}_p(t) = \mathbf{v}(\mathbf{x}_p, t), \end{aligned} \tag{1}$$

where the the initial position of the particle, at time 0, is given as $\mathbf{x}_0$ and the change in position is determined by the velocity field. Here the overdot is a common shorthand to denote a time derivative. Because we described the particle's position by describing how it *changes*, the second equation is a *differential equation*. This differential equation is known as an *Oridinary Differential Equation* (ODE) because it only involves a single derivative, in this case, with respect to time. Moreover, this equation is a *First-order* ODE because we are only taking a single derivative with respect to time. Later, we will see both second-order systems and partial differential equations (PDE).

For very simple velocity fields, we could solve this initial value problem analytically, but in the general case we will need to use numerical integration to solve this system. The intuition behind the simplest method, known as Euler integration, can be found by looking at the continuous definition of the derivative,

$$\frac{d\mathbf{x}_p(t)}{dt} = \lim_{\epsilon \to 0} \frac{\mathbf{x}_p(t+\epsilon) - \mathbf{x}_p(t)}{\epsilon}. \tag{2}$$

To approximate this equation discretely, we simply replace the limit as $\epsilon \longrightarrow 0$ with a finite value $\epsilon = \Delta t$,

$$\frac{d\mathbf{x}_p(t)}{dt} \approx \frac{\mathbf{x}_p(t+\Delta t) - \mathbf{x}_p(t)}{\Delta t}. \tag{3}$$

Substituting into Equation (1)

$$\frac{\mathbf{x}_p(t+\Delta t) - \mathbf{x}_p(t)}{\Delta t} = \mathbf{v}(\mathbf{x}_p, t) \tag{4}$$

and performing some manipulation, we have

$$\mathbf{x}_p(t+\Delta t) = \mathbf{x}_p(t) + \Delta t \cdot \mathbf{v}(\mathbf{x}_p, t). \tag{5}$$

Given that the initial value problem gives us $\mathbf{x}_p(0)$, we can now use Equation (5) to trace the particle forward through time. More advanced and accurate techniques exist, of course, and we will postpone the discussion of these until Section 4. One thing to note here, is that as the timestep, Δt,

is decreased the approximation in Equation (3) becomes more accurate, resulting in a better solution to the initial value problem. However, that better solution comes at the cost of longer computation time as a larger number of timesteps are required to cover the same amount of simulated time.

1.2 A particle with mass

A real-world object's motion is governed by internal and external forces that affect the object's acceleration through Newton's second law of motion, $\mathbf{f} = m\mathbf{a}$. Thus to introduce forces, such as gravity, into our particle systems we must also introduce mass. Our initial value problem then becomes

$$\begin{aligned} \mathbf{x}_p(0) &= \mathbf{x}_0 \\ \frac{d^2\mathbf{x}_p(t)}{dt^2} &= \ddot{\mathbf{x}}_p(t) = \frac{\mathbf{f}(\mathbf{x}_p, t)}{m_p}. \end{aligned} \tag{6}$$

It is often convenient to rewrite this second-order differential equation as a system of coupled first-order equations,

$$\begin{aligned} \mathbf{x}_p(0) &= \mathbf{x}_0 \\ \mathbf{v}_p(0) &= \mathbf{v}_0 \\ \frac{d\mathbf{x}_p(t)}{dt} &= \dot{\mathbf{x}}_p(t) = \mathbf{v}_p(t) \\ \frac{d\mathbf{v}_p(t)}{dt} &= \dot{\mathbf{v}}_p(t) = \frac{\mathbf{f}(\mathbf{x}_p, t)}{m_p}. \end{aligned} \tag{7}$$

As before we can numerical solve this initial value problem by integrating forward in time using Euler's method:

$$\begin{aligned} \mathbf{v}_p(t + \Delta t) &= \mathbf{v}_p(t) + \Delta t \cdot \frac{\mathbf{f}(\mathbf{x}_p, t)}{m_p} \\ \mathbf{x}_p(t + \Delta t) &= \mathbf{x}_p(t) + \Delta t \cdot \mathbf{v}_p(t + \Delta t). \end{aligned} \tag{8}$$

This particular integration scheme is commonly referred to as *symplectic Euler* and is, in many cases, the best choice among the simple integration schemes.

The particle system designer can specify a wide variety of forces such as gravitational forces around a point, line, or (ground) plane, damping, and penalties for violating constraints. An early example of such systems was detailed in 1990 by Sims [1990] and was used to create the animated short *Particle Dreams*.

In a particle system implementation, a `Particle` structure would typically store, at a minimum: mass, position, velocity, and a force accumulator.[1] Most implementations allow particles to have arbitrary attributes such as color, size, age, type, etc. A typical simulation will have an outer loop that takes timesteps. Each timestep typically contains three loops. The first iterates over the particles initializing the force accumulators. The second iterates over the forces, applying them to the affected particles. The third updates the particles velocities and positions using Equation (8).

[1] While it is conceptually convenient to imagine a `Particle` structure, many practical implementations do not store all a particle's data contiguously. Instead, a `ParticleSystem` will store multiple single-datum arrays (i.e. separate arrays for positions, velocities, and force accumulators). Doing so increases efficiency because it avoids loading unnecessary data into the cache. For example, a particle's position is not needed to zero out its force accumulator.

1.3 Spring-mass systems

In the sort of particle systems thus far described, the particles can be subjected to a wide range of forces, but they do not interact. Allowing the particles to interact enables the creation of particle systems that can be used to animate a wide range of physical phenomena from hair to water to clothing. The simplest mechanism for particle interaction is to apply spring forces between particles. The resulting systems are typically called *spring-mass* or *mass-spring systems*.

The simplest formulation of a spring force involves a mass attached to the origin with a zero length spring and is often expressed as

$$\mathbf{f} = -k\mathbf{d} \tag{9}$$

where k is Hooke's spring constant and $\mathbf{d}$ is the *displacement*. Using the notation we have been using, we would replace $\mathbf{d}$ with $\mathbf{x}_p$

$$\mathbf{f}_p = -k\mathbf{x}_p. \tag{10}$$

This can be generalized to a spring with a given rest length r as

$$\mathbf{f}_p = k\left(\|\mathbf{x}_p\| - r\right)\frac{-\mathbf{x}_p}{\|\mathbf{x}_p\|}. \tag{11}$$

Here we have a force that is proportional to the difference between the spring's current length and its rest length, in the direction from the particle to the origin. Unfortunately, in this form, the term $\left(\|\mathbf{x}_p\| - r\right)$ has units of meters, which means that if the resolution increases the forces get smaller. That is, if a single spring is replaced by two springs the force on the particle is half as large and k must be adjusted to maintain similar behavior. It is better to divide by r and use the dimension-less quantity $\left(\|\mathbf{x}_p\|/r - 1\right)$. This dimension-less quantity is known as a *strain* measure and will be discussed in more detail in Section 2.2.2. Because strain is dimension-less, our parameter k is now a *material parameter* independent of the discretization. Our force then becomes

$$\mathbf{f}_p = k\left(\frac{\|\mathbf{x}_p\|}{r} - 1\right)\frac{-\mathbf{x}_p}{\|\mathbf{x}_p\|}. \tag{12}$$

We can generalize further by placing the other end point at an arbitrary point in space $\mathbf{x}_q$. Then our force becomes

$$\mathbf{f}_p = k\left(\frac{\|\mathbf{x}_q - \mathbf{x}_p\|}{r} - 1\right)\frac{\mathbf{x}_q - \mathbf{x}_p}{\|\mathbf{x}_q - \mathbf{x}_p\|}. \tag{13}$$

From Newton's third law we know that the force on particle q must be $\mathbf{f}_q = -\mathbf{f}_p$.

So far we have a model for ideal elastic bodies, but in nature internal friction converts some elastic energy into heat. This phenomenon is called *damping*. We can enhance our model by including an additional damping force, which depends on the relative velocities between the particles:

$$\mathbf{f}_p = k_d\left(\frac{\mathbf{v}_q - \mathbf{v}_p}{r}\cdot\frac{\mathbf{x}_q - \mathbf{x}_p}{\|\mathbf{x}_q - \mathbf{x}_p\|}\right)\frac{\mathbf{x}_q - \mathbf{x}_p}{\|\mathbf{x}_q - \mathbf{x}_p\|}. \tag{14}$$

Putting together the elastic and damping forces and rearranging terms we get,

$$\mathbf{f}_p = \left[k_s\left(\frac{\|\mathbf{x}_q - \mathbf{x}_p\|}{r} - 1\right) + k_d\left(\frac{\left(\mathbf{v}_q - \mathbf{v}_p\right)\cdot\left(\mathbf{x}_q - \mathbf{x}_p\right)}{r\|\mathbf{x}_q - \mathbf{x}_p\|}\right)\right]\frac{\mathbf{x}_q - \mathbf{x}_p}{\|\mathbf{x}_q - \mathbf{x}_p\|}. \tag{15}$$

In code, the simulation loop might look something like the listing in Algorithm 1

Algorithm 1 Spring-Mass System Timestep Loop

```
for Particle p : particles do
    p.frc = 0
    p.frc += p.mass*gravity
end for
for Spring s : springs do
    Vec3 d = particles[s.j].pos - particles[s.i].pos
    double l = mag(d)
    Vec3 v = particles[s.j].vel - particles[s.i].vel
    Vec3 frc = (k_s*((l / s.r) - 1.0) + k_d*dot(v / s.r, d / l)) * (d / l)
    particles[s.i].frc += frc
    particles[s.j].frc -= frc
end for
for Particle p : particles do
    p.vel += dt*(p.frc / p.mass)
    p.pos += dt*(p.vel)
end for
```

2 MATHEMATICAL MODELS

In this section we introduce the mathematical models that govern motion. This field of study is often referred to as *Continuum Mechanics* because the models assume that the underlying materials are continuous, which is a reasonable assumption at the scales and for the phenomena we are generally concerned with. In the following sections we will discuss how numerical methods are employed to discretize these models to obtain computational solutions.

2.1 Physical Laws

Much of physics-based simulation is based on the the formulation of classical mechanics developed by Isaac Newton, which he published in his tome *The Principia* in 1687. In that work, he described several laws of motion that remain fundamental in the modeling, analysis, and simulation of the types of mechanical systems that we are interested in here. Another distinct formulation of classical mechanics is *variational* or *analytical mechanics*, which is based on the principle of least action, and also has significant use in physics-based animation. We will revisit concepts from variational mechanics in discussing rigid bodies and constraints. Here, we focus on the fundamental laws of Newtonian mechanics.

2.1.1 Newton's Laws of Motion. Newton's laws describe the dynamics of a body that can be idealized as a particle, or point mass. Newton used these laws to study many systems, including the motion of planets over large distances. While Newton's laws do not address the extent of the body, the distribution of its mass, and its internal forces, they can be applied to systems such as soft bodies or fluids by viewing these as collections of particles and characterizing all of the inter-particle forces.

Newton's first law. Newton's first law states that a body remains at rest or moves with a constant velocity unless acted upon by a force. This is also called the law of inertia and the motion described is called *uniform motion*.

Newton's second law. Newton's second law is fundamental in computing the evolution in time of the state of a body. Newton's famous second law, $\mathbf{f} = m\mathbf{a}$, states that the instantaneous change

in the momentum of a body is equal to the net force acting on the body. The momentum of a body is given by

$$\mathbf{p}(t) = m\mathbf{v}(t), \tag{16}$$

where m is the mass of the body and $\mathbf{v}$ is the velocity of the body. The mathematical expression of Newton's second law is

$$\mathbf{f}(t) = \frac{d}{dt}\mathbf{p}(t) = m\frac{d}{dt}\mathbf{v}(t) = m\mathbf{a}(t). \tag{17}$$

Therefore, application of Newton's second law to a physical system is done by identifying all of the bodies in the system and identifying the forces acting on each body. For example, in Figure 1,

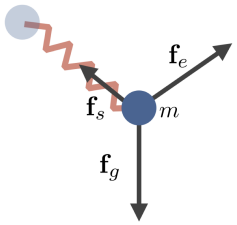


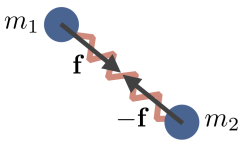
Fig. 1. *A particle of mass m is subject to a spring force $\mathbf{f}_s$, a graviational force $\mathbf{f}_g$, and an unspecified external force $\mathbf{f}_e$. Given the forces, Newton's second law gives the acceleration of the particle, which is then used to compute a new velocity and position of the particle.*

a particle of mass m experiences a spring force $\mathbf{f}_s$ from a neighboring particle, the gravitational force $\mathbf{f}_g$, and another unspecified external force $\mathbf{f}_e$. By Newton's second law, the acceleration of the particle is given by

$$m\mathbf{a} = \mathbf{f}_s + \mathbf{f}_g + \mathbf{f}_e. \tag{18}$$

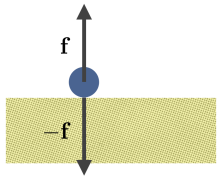
Given the forces, the acceleration is used to update the velocity of the body, and the velocity is used to update the position of body.

Newton's third law. Newton's third law states that for every action, there is an equal and opposite reaction. Specifically, in an interaction between two particles or bodies, if body A applies force $\mathbf{f}$ to body B, then body B applies force $-\mathbf{f}$ to body A. We can use this law to help identify all of the forces acting on the bodies in a given scenario. For example, in the image below, a particle of mass m_1 is connected by spring to a particle of mass m_2. If the spring force experienced by the



particle of mass m_1 is $\mathbf{f}$, then Newton's third law requires that the spring force experienced by the particle of mass m_2 is $-\mathbf{f}$. Note that by Newton's second law, if their masses are unequal, then their accelerations due to the spring force will necessarily be unequal. In particular, the lighter particle will experience a larger acceleration, while the heavier particle experiences a smaller acceleration.

Consider another example, illustrated below, where a particle of mass m is resting on the ground. The particle exerts a force on the ground equal to $\mathbf{f} = -m\mathbf{g}$. By Newton's third law, the ground

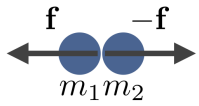


exerts a reaction force on the particle equal to $-\mathbf{f} = m\mathbf{g}$. Therefore, forces acting on the particle are the force of gravity $-m\mathbf{g}$ and the reaction force from the ground $m\mathbf{g}$. By Newton's second law,

$$m\mathbf{a} = -m\mathbf{g} + m\mathbf{g} = \mathbf{0}, \tag{19}$$

so that the particle remains at rest on the ground.

Finally, consider an example where a moving particle of mass m_1 collides with a moving particle of mass m_2, illustrated below. Due to the collision, each particle feels a force, such that after the



collision the relative velocity of the particles is at least zero to prevent their interpenetration. By Newton's third law, the forces on the particles must be a pair of action/reaction forces. Notably, since the change in momentum of each particle is equal to the force acting on it, Newton's third law implies here that the total momentum of the particles before the collision will be equal to the total momentum of the particles after the collision, i.e., the total momentum of the system will be conserved.

2.1.2 Conservation of Mass, Momentum, Energy. The physical principles of conservation of mass, linear and angular momentum, and energy are also of central importance in physics-based animation. A conserved quantity is one which cannot be created or destroyed.

Newton's second law tells us that if there is no net force acting on a particle, then

$$\frac{d}{dt} m\mathbf{v}(t) = \mathbf{0} \implies m\mathbf{v}(t) = \text{constant}, \tag{20}$$

i.e., the momentum of the particle is conserved. By Newton's third law, every action has an equal and opposite reaction, which also leads to conservation of linear and angular momentum in a closed system of interacting particles.

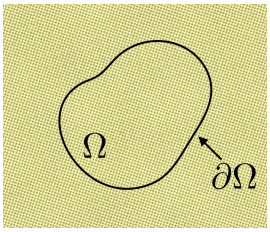
The principle of conservation of energy tells us that energy cannot be created or destroyed, but can only change form, so that the total energy in a closed system remains constant. For example, consider an object of mass m dropped with zero initial velocity in a vacuum (so that there is no loss of energy to air friction) from a height h_0. Initially the object has potential energy mgh_0 due to the force of gravity. As the object falls, the potential energy is converted to kinetic energy $\frac{1}{2}mv^2$, where $v = \|\mathbf{v}\|$ is the speed of the object. Since total energy is conserved and equal to the initial energy mgh_0, we have

$$mgh(t) + \frac{1}{2}mv(t)^2 = mgh_0 \tag{21}$$

$$\implies v(t) = \sqrt{2(gh_0 - gh(t))}. \tag{22}$$

Therefore, using the principle of conservation of energy, we can conclude, for example, that the object will be traveling with speed $\sqrt{2gh_0}$ when in hits the ground.

Conservation laws for continua. The principles of conservation of mass, momentum, and energy are also used in the field of *continuum mechanics* to derive the equations describing the behavior of continua. These are materials such as soft bodies and fluids, modeled as being made of up of a continuum of matter rather than discrete particles, an assumption that is appropriate for the spatial scales of interest in physics-based animation. In this case, given a continuum of material, we consider the rate of change of the conserved quantity in an arbitrary *control volume* Ω with boundary $\partial\Omega$, as illustrated below. Imagine that the material is a fluid, and that $\mathbf{v}(\mathbf{x}, t)$ is the



time-varying velocity field of the fluid. The total mass in the region Ω is given by $\int_\Omega \rho dV$ where ρ is the fluid density and dV is a volume element. Conservation of mass tells us that the total amount of mass in the fixed region Ω can only change in time due to flow of mass into or out of Ω through its boundary $\partial\Omega$. Mathematically, this is expressed as

$$\frac{d}{dt}\int_\Omega \rho dV = -\oint_{\partial\Omega} \rho \mathbf{v}(\mathbf{x}, t)\cdot \mathbf{n}(\mathbf{x}) dS, \tag{23}$$

where $\mathbf{n}(\mathbf{x})$ is the outward unit normal on $\partial\Omega$. Applying the divergence theorem to the right hand side, and moving all terms to the left hand side gives

$$\int_\Omega \frac{\partial \rho}{\partial t} + \nabla\cdot(\rho\mathbf{v})\, dV = 0. \tag{24}$$

Since the control volume was arbitrary and this must hold for any control volume, we get that

$$\frac{\partial \rho}{\partial t} + \nabla\cdot(\rho\mathbf{v}) = 0. \tag{25}$$

More details can be found in [Landau and Lifshitz 1959].

These conservation laws are *local* rather than global, so that if we lose mass in one part of the domain but add it in another, such that the total global mass is conserved, the conservation law would still be violated. In a simulation, this would amount to seeing a volume of fluid disappear in one part of the domain and magically appear in another, a situation that is not desirable. The local nature of the conservation laws means that mass, momentum, and energy should be locally conserved everywhere.

In general, a local conservation law takes the form

$$\frac{\partial u}{\partial t} + \nabla\cdot\mathbf{f}(u) = 0, \tag{26}$$

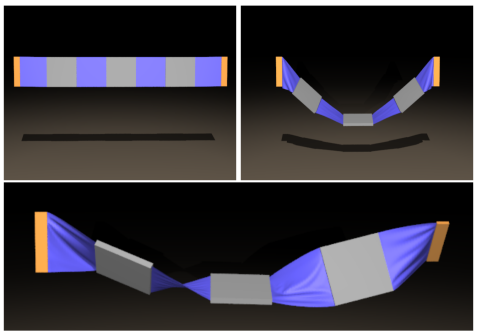
where u is the conserved quantity, and $\mathbf{f}(u)$ is the flux function, representing the flow through the boundary of the conserved quantity.

It is also possible to include *sources* or *sinks* on the right hand side of (23). These functions are used to model the addition or removal of the conserved quantity through interaction with an external system.

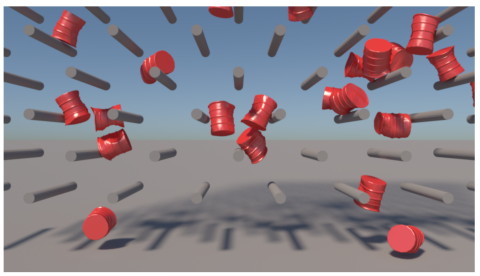
2.2 Materials

Physics-based animation is used for a wide variety of materials: rigid bodies which are idealized to have no deformation, soft bodies that deform elastically and plastically, fluids such as air, water, honey, and others materials such as sand and snow. In this section, we introduce the mathematical models describing soft bodies, rigid bodies, and fluids. These models and their combinations form the basis for most materials considered in physics-based animation.

2.2.1 Rigid Bodies.



(a) *Simulation of rigid bodies coupled to elastic cloth. Reproduced from [Sifakis et al. 2007].*



(b) *Simulation of rigid barrels, augmented with example-based deformations. Reproduced from [Jones et al. 2016].*

Fig. 2. *Examples of rigid body simulations.*

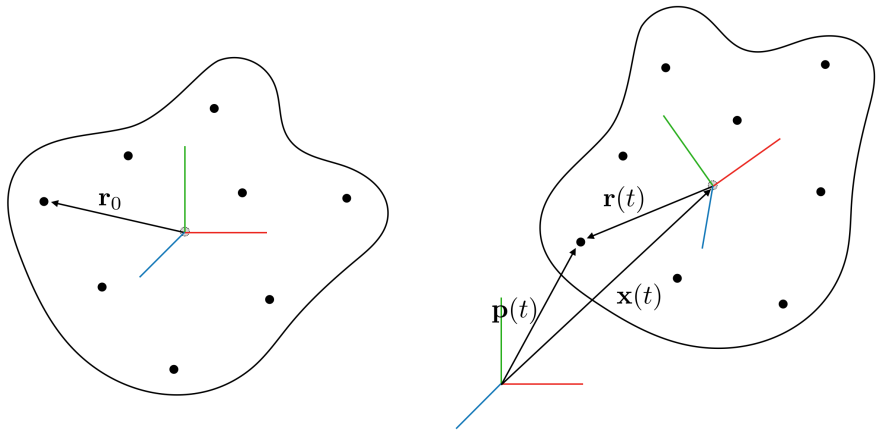
Introduction. We saw previously that a system of particles connected by springs can be used to model an elastic object. In many cases, we are interested in modeling very stiff objects for which we are not concerned with elastic deformation. We could use a particle system connected with very stiff springs, but this representation is inefficient compared to a *rigid body* approximation. A rigid body is a object whose points are constrained to be at a fixed distance from one another, i.e., it is assumed to not undergo any deformation. In this idealized limit, the position of all points on the object can be describe with six degrees of freedom. First, the center of mass of the object can be at any point in three-dimensional space, giving three positional degrees of freedom. Second, the object can be oriented in any way about that center of mass, given three orientational degrees of freedom. Using the constraint that all points in the rigid body remain a fixed distance from one another, we arrive at a special form for the laws of motion of a rigid body.

For further reading, a thorough introduction to rigid body simulation can be found in [Baraff 2001].

Kinematics. Though we will use rigid bodies to model very stiff continuous media, for the purposes of deriving the rigid body properties and laws, it is convenient to think of the rigid body as composed of discrete particles labeled $i = 1, \cdots, N$, each with mass m_i. In what follows, discrete sums over the constituent particles can be replaced with continuous integrals to derive the results.

The *center of mass*, $\mathbf{x}_{com}$ of the rigid body is the mass-weighted average position of the rigid body's constituent particles,

$$\mathbf{x}_{com} = \frac{\sum_{i=1}^{N} m_i \mathbf{p}_i}{\sum_{i=1}^{N} m_i}. \tag{27}$$



(a) *Object space.* **(b)** *World space.*

Fig. 3

It is convenient to fix a coordinate system to the body, with its origin at the body's center of mass. Coordinates of points on the body relative to this fixed coordinate system are said to be in *object space*. In constrast, the usual common space in which all objects are defined is now referred to as *world space*. As the body translates and rotates over time, its center of mass and constituent particles will move in space. Hence, while the object space coordinates of the body particles are fixed, the world space coordinates generally change over time. A rigid body in object space and world space is depicted in Figure 3.

Consider a particle of the rigid body with object space position $\mathbf{r}_0$. At time t, the position of the body center of mass is given by $\mathbf{x}(t)$, and the orientation of the body can be described by a rotation matrix $\mathbf{R}(t)$ about the center of mass. Therefore, the world space position of the particle is given by

$$\mathbf{p}(t) = \mathbf{x}(t) + \mathbf{R}(t)\mathbf{r}_0. \tag{28}$$

We will also write

$$\mathbf{r}(t) = \mathbf{R}(t)\mathbf{r}_0, \tag{29}$$

to represent the rotated particle position about the center of mass, so that $\mathbf{p}(t) = \mathbf{x}(t) + \mathbf{r}(t)$ as shown in Figure 3b.

The velocity of a particle of the rigid body can be found by differentiating (28) with respect to time, to get

$$\mathbf{v}(t) = \dot{\mathbf{p}}(t) = \dot{\mathbf{x}}(t) + \dot{\mathbf{R}}(t)\mathbf{r}_0. \tag{30}$$

Linear and Angular Velocity. The velocity of the rigid body center of mass, $\dot{\mathbf{x}}(t)$, is referred as the rigid body *linear velocity*, and is analogous to the velocity of a single particle. Notice that the motion of a particle on the body in Eq. (30) contains this term. It is the component of the motion of the particle due to the linear velocity of the body. If the body is not rotating, i.e., $\dot{\mathbf{R}}(t) = 0$, then $\dot{\mathbf{p}}(t) = \dot{\mathbf{x}}(t)$, and the velocity of the particle is identical to the velocity of the body center of mass.

For a rigid body that is rotating, $\dot{\mathbf{R}}(t) \neq 0$. Let us examine this term further. This term $\dot{\mathbf{R}}(t)\mathbf{r}_0$ in Eq. (30) represents the component of motion of the particle due to the instantaneous rotation of the body about its center of mass. Euler's rotation theorem tells us that this instantaneous rotation

must be equivalent to a rotation about a single axis that runs through the center of mass. We let $\boldsymbol{\omega}$ be the vector whose direction is the instantaneous axis of rotation and whose length is the rate of rotation in radians per second. Figure 4 illustrates the motion of the point $\mathbf{p}$ due to the rotation. Let

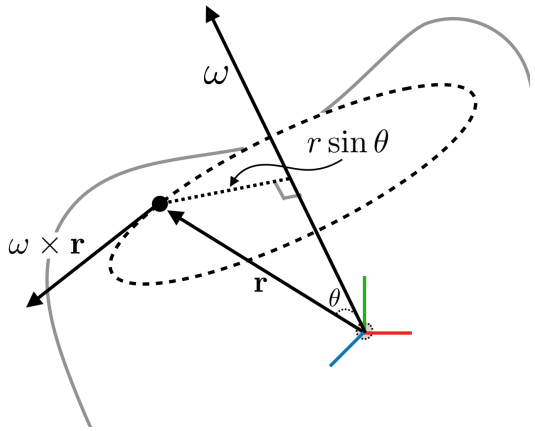


Fig. 4. *For a body with angular velocity $\boldsymbol{\omega}$, a point at $\mathbf{r}$ on the body relative to the body center of mass rotates about about the center of mass with velocity $\boldsymbol{\omega} \times \mathbf{r}$.*

θ be the angle between $\boldsymbol{\omega}$ and $\mathbf{r}$, and $r = \|\mathbf{r}\|$. Since the point rotates $\|\boldsymbol{\omega}\|$ radians/sec on a circle of radius $r \sin\theta$, the speed of motion of the particle is $\|\boldsymbol{\omega}\|\|\mathbf{r}\| \sin\theta$, and the direction of motion of the particle is perpendicular to $\boldsymbol{\omega}$ and $\mathbf{r}$ and given by the right-hand rule. Hence, we have that

$$\dot{\mathbf{R}}(t)\mathbf{r}_0 = \boldsymbol{\omega}(t) \times \mathbf{r}(t). \tag{31}$$

Therefore, we can rewrite the velocity of the particle in Eq. (30) as

$$\mathbf{v}(t) = \dot{\mathbf{p}}(t) = \dot{\mathbf{x}}(t) + \boldsymbol{\omega}(t) \times \mathbf{r}(t). \tag{32}$$

The vector $\boldsymbol{\omega}(t)$ is called the *angular velocity* of the body.

Note also that since $\mathbf{r}(t) = \mathbf{R}(t)\mathbf{r}_0$,

$$\dot{\mathbf{R}}(t)\mathbf{r}_0 = \boldsymbol{\omega}(t) \times \mathbf{R}(t)\mathbf{r}_0, \tag{33}$$

and as this holds for any vector $\mathbf{r}_0$, we have

$$\dot{\mathbf{R}}(t) = \boldsymbol{\omega}(t) \times \mathbf{R}(t), \tag{34}$$

where the cross product is understood to apply to each column of the matrix.

Linear Momentum. The momentum of the rigid body is given by the sum of the momenta of its constituent particles,

$$\mathbf{P}(t) = \sum_{i=1}^{N} m_i \mathbf{v}_i(t). \tag{35}$$

Using Eq. (32) to substitute for $\mathbf{v}_i(t)$, we get

$$\mathbf{P}(t) = \sum_{i=1}^{N} m_i \left(\dot{\mathbf{x}}(t) + \boldsymbol{\omega}(t) \times \mathbf{r}_i(t)\right) \tag{36}$$

$$= \sum_{i=1}^{N} m_i \dot{\mathbf{x}}(t) + \boldsymbol{\omega}(t) \times \left(\sum_{i=1}^{N} m_i \mathbf{r}_i(t)\right). \tag{37}$$

Because the $\mathbf{r}_i$ are defined relative to the center of mass of the object, the last sum in the above equation is zero. Let $M = \sum_{i=1}^{N} m_i$ be the total mass of the body. Then the linear momentum of the body is simply

$$\mathbf{P}(t) = M\dot{\mathbf{x}}(t). \tag{38}$$

Angular Momentum. The angular momentum of the body is given by the sum of the angular momenta of its constituent particles,

$$\mathbf{L}(t) = \sum_{i=1}^{N} \mathbf{r}_i(t) \times m_i \mathbf{v}_i(t). \tag{39}$$

Using Eq. (32) to substitute for $\mathbf{v}_i(t)$, we get

$$\mathbf{L}(t) = \sum_{i=1}^{N} m_i \mathbf{r}_i(t) \times (\dot{\mathbf{x}}(t) + \boldsymbol{\omega}(t) \times \mathbf{r}_i(t)) \tag{40}$$

$$= \sum_{i=1}^{N} m_i \mathbf{r}_i(t) \times \dot{\mathbf{x}}(t) + \sum_{i=1}^{N} m_i \mathbf{r}_i(t) \times \boldsymbol{\omega}(t) \times \mathbf{r}_i(t). \tag{41}$$

The first sum is again equal to zero, simplifying the expression for the body angular velocity to

$$\mathbf{L}(t) = \sum_{i=1}^{N} m_i \mathbf{r}_i(t) \times (\boldsymbol{\omega}(t) \times \mathbf{r}_i(t)). \tag{42}$$

Since $\boldsymbol{\omega}$ does not depend on i, we'd like to pull it out of the sum. First, we'll use the fact that $\boldsymbol{\omega} \times \mathbf{r} = -\mathbf{r} \times \boldsymbol{\omega}$, to write

$$\mathbf{L}(t) = \sum_{i=1}^{N} m_i \mathbf{r}_i(t) \times (-\mathbf{r}_i(t) \times \boldsymbol{\omega}(t)). \tag{43}$$

To further simplify the remaining term, it is helpful to switch from cross product notation to matrix notation, after which we can use more familiar matrix algebra. For a vector $\mathbf{r} = (r_x, r_y, r_z)^T$, and any vector $\boldsymbol{\omega}$, the result of computing $\mathbf{r} \times \boldsymbol{\omega}$ is equivalent to premultiplying $\boldsymbol{\omega}$ by the matrix[2]

$$\mathbf{r}^\star = \begin{pmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{pmatrix}. \tag{44}$$

Note that like the cross product operator, this matrix is skew-symmetric, i.e., $-\mathbf{r}^\star = \mathbf{r}^{\star T}$. Therefore, we can replace the expression $\mathbf{r}_i\times$ with $\mathbf{r}_i^\star$, and, $-\mathbf{r}_i\times$ with $\mathbf{r}_i^{\star T}$ in the above to get

$$\mathbf{L}(t) = \sum_{i=1}^{N} m_i \mathbf{r}_i^\star(t)(\mathbf{r}_i^{\star T}(t)\boldsymbol{\omega}(t)) \tag{45}$$

$$= \left(\sum_{i=1}^{N} m_i \mathbf{r}_i^\star(t)\mathbf{r}_i^{\star T}(t)\right) \boldsymbol{\omega}(t). \tag{46}$$

$$\tag{47}$$

[2]Other authors and communities use a variety of other symbols for $\mathbf{r}^\star$: $\mathbf{r}^*$, $\mathbf{r}^\times$, $\hat{\mathbf{r}}$, $[\mathbf{r}]$, and $\tilde{\mathbf{r}}$

Notice that the term in parentheses depends only on the masses of the particle and their geometric distributions. This 3 × 3 matrix is called the *inertia tensor* of the body, and is often denoted by

$$\mathbf{I}(t) = \sum_{i=1}^{N} m_i \mathbf{r}_i^\star(t) \mathbf{r}_i^{\star T}(t). \tag{48}$$

The inertia tensor describes the distribution of mass in a rigid body and can be readily computed directly from a closed triangle mesh. With the inertia tensor, we can write

$$\mathbf{L}(t) = \mathbf{I}(t)\boldsymbol{\omega}(t). \tag{49}$$

Though it depends on time, we can also express the inertia tensor as a transformation of the object space inertia tensor by noting that $\mathbf{r}^\star \mathbf{r}^{\star T} = \mathbf{r}^T \mathbf{r} \boldsymbol{\delta} - \mathbf{r}\mathbf{r}^T$, where $\boldsymbol{\delta}$ is the 3 × 3 identity matrix, and substituting $\mathbf{r} = \mathbf{R}\mathbf{r}_0$ to get

$$\begin{aligned}
\mathbf{I}(t) &= \sum_{i=1}^{N} m_i \mathbf{r}_i^\star(t) \mathbf{r}_i^{\star T}(t) && (50)\\
&= \sum_{i=1}^{N} m_i \left(\mathbf{r}_i^T \mathbf{r}_i \boldsymbol{\delta} - \mathbf{r}_i \mathbf{r}_i^T \right) && (51)\\
&= \mathbf{R}(t) \sum_{i=1}^{N} m_i \left(\mathbf{r}_{0i}^T \mathbf{r}_{0i} \boldsymbol{\delta} - \mathbf{r}_{0i} \mathbf{r}_{0i}^T \right) \mathbf{R}(t)^T && (52)\\
&= \mathbf{R}(t) \mathbf{I}_0 \mathbf{R}(t)^T. && (53)
\end{aligned}$$

Thus, the inertia tensor at time t is the constant object-space inertia tensor $\mathbf{I}_0$ transformed to account for the current body orientation.

Force and Torque. Newton's second law relates the time rate of change of momentum of a body to the net force on the body. In the case of a rigid body, Newton's second law takes the form

$$\frac{d}{dt} \begin{pmatrix} \mathbf{P}(t) \\ \mathbf{L}(t) \end{pmatrix} = \begin{pmatrix} \mathbf{f}(t) \\ \boldsymbol{\tau}(t) \end{pmatrix}, \tag{54}$$

where $\mathbf{f}$ is the net force on the body, and $\boldsymbol{\tau}$ is the net *torque* on the body.

If a force $\mathbf{f}$ is applied to a rigid body at its center of mass, then the body responds as if it was a particle with mass M, that is, its particles undergo an acceleration $\mathbf{a} = \mathbf{f}/M$. When a force is applied to the body at a point other than its center of mass, this may generate a torque as well. The torque due to an applied force at a point $\mathbf{p}$ located at $\mathbf{r}$ from the center of mass is equal to

$$\boldsymbol{\tau} = \mathbf{r} \times \mathbf{f}. \tag{55}$$

Since the magnitude of the torque is $\|\boldsymbol{\tau}\| = \|\mathbf{r}\|\|\mathbf{f}\| \sin\theta$, where θ is the angle between $\mathbf{r}$ and $\mathbf{f}$, the torque can be increased by increasing either the applied force or the distance at which it is applied.

Summary. In summary, if we define the auxiliary quantities, $\mathbf{v}(t)$, $\mathbf{I}(t)$, and $\boldsymbol{\omega}(t)$ as,

$$\mathbf{v}(t) = \frac{\mathbf{P}(t)}{M}, \qquad \mathbf{I}(t) = \mathbf{R}(t)\mathbf{I}_0\mathbf{R}(t)^T, \qquad \text{and} \qquad \boldsymbol{\omega}(t) = \mathbf{I}(t)^{-1}\mathbf{L}(t), \tag{56}$$

then the update to the state of the rigid body is given by,

$$\frac{d}{dt} \begin{pmatrix} \mathbf{x}(t) \\ \mathbf{R}(t) \\ \mathbf{P}(t) \\ \mathbf{L}(t) \end{pmatrix} = \begin{pmatrix} \mathbf{v}(t) \\ \boldsymbol{\omega}^\star(t)\mathbf{R}(t) \\ \mathbf{f}(t) \\ \boldsymbol{\tau}(t) \end{pmatrix}. \tag{57}$$

2.2.2 Soft Bodies.

Introducing Elasticity into Newton's Second Law. To model soft bodies we incorporate elastic and damping forces into Newton's second law, $\mathbf{f} = M\mathbf{a}$. Elastic forces are often expressed as a function of displacements, $\mathbf{d}$, so that zero displacement results in zero force, but can also be written as a function of position, $\mathbf{x}$. Damping forces are usually a function of velocity, which is the time deriviative of $\mathbf{d}$ or $\mathbf{x}$,

$$\mathbf{v} = \frac{d\mathbf{x}}{dt} = \frac{d\mathbf{d}}{dt} = \dot{\mathbf{x}} = \dot{\mathbf{d}}, \tag{58}$$

where an overdot is shorthand for a time derivative. The resulting dynamic system is

$$\mathbf{K}(\mathbf{d}) + \mathbf{D}(\dot{\mathbf{d}}) + \mathbf{M}\ddot{\mathbf{d}} = \mathbf{f}_{ext}, \tag{59}$$

where $\mathbf{K}(\mathbf{d})$ represents internal elastic forces, $\mathbf{D}(\dot{\mathbf{d}})$ represent damping forces, $\mathbf{M}$ is the mass matrix, and $\mathbf{f}_{ext}$ is external forces (such as gravity or wind). After discretization both $\mathbf{K}$ and $\mathbf{D}$ will be matrices as well, with $\mathbf{K}$ referred to as the "stiffness" matrix and $\mathbf{D}$ as the "damping" matrix. The stiffness matrix, which determines the magnitude of elastic forces, is the negative of the Jacobian of the the elastic forces with respect to position, $\mathbf{K} = -\partial f/\partial x$ and is positive semi-definite. It is also the negative Hessian of the elastic energy, $\mathbf{K} = -\partial^2 \eta/\partial x_i \partial x_j$, which makes clear that $\mathbf{K}$ encodes the elastic relationships between particles in the system. In the commonly adopted Rayleigh damping model, $\mathbf{D} = \lambda\mathbf{K} + \mu\mathbf{M}$. [3] Notice the similarity to the damped zero-length Hookean spring,

$$\mathbf{f} = -k\mathbf{d} - c\dot{\mathbf{d}}. \tag{60}$$

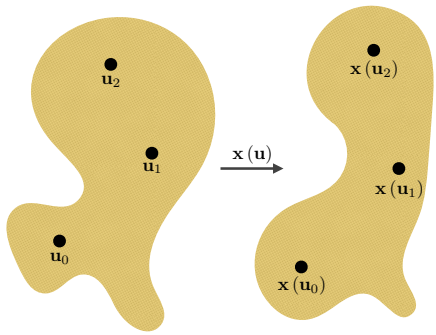


Fig. 5. *A function* $\mathbf{x}(\mathbf{u})$ *maps points in rest (or material or object) space* $\mathbf{u}_i$ *to points in world space,* $\mathbf{x}_i$.

Deformation Gradient. We begin by defining a deformation function $\mathbf{x}(\mathbf{u})$ that maps points in a rest space, $\mathbf{u}_i$, to points in world space, $\mathbf{x}_i$. This mapping may be arbitrarily complicated, but we can always linearize about a point, $\mathbf{u}_0$, to arrive at a familiar affine transformation,

$$\mathbf{x}(\mathbf{u}) = \mathbf{x}(\mathbf{u}_0) + \mathbf{A}\left(\mathbf{u} - \mathbf{u}_0\right), \tag{61}$$

where $\mathbf{A}$ is an arbitrary $d \times d$ (in d-dimensional space) transformation matrix and $\mathbf{x}(\mathbf{u_0})$ is the world space position corresponding to $\mathbf{u}_0$, which can be thought of as a translation. Elastic forces work to ensure that this mapping is a rigid transformation—a global rotation and translation—and are determined by the gradient of this mapping, the *deformation gradient*, $\partial\mathbf{x}/\partial\mathbf{u}$, which is commonly denoted $\mathbf{F}$ and for this linearized mapping is simply the matrix $\mathbf{A}$. This $d \times d$ matrix describes how infinitesimal vectors/lengths/displacements in rest space are mapped to world space (ignoring

[3] We have adopted the convention that $\mathbf{K}$ is positive semi-definite, other authors choose $\mathbf{K}$ to be negative semi-definite.

translations). This bears repeating: intuitively, $\mathbf{F}$ is the rest→world transformation. $\mathbf{F}$ measures stretch—if a soft body is stretched by a factor of two, then displacements in rest space lead to displacements in world space that are twice as large. If $\mathbf{F}$ is a rotation, then displacements in rest space lead to the same size displacements in world space. Similarly, if $\det(\mathbf{F}) = 1$ then volume is preserved between rest and world space.

The Singular Value Decomposition (SVD) of $\mathbf{F} = \mathbf{U}\Sigma\mathbf{V}^T$, where $\mathbf{U}$ and $\mathbf{V}$ are orthonormal matrices and Σ is a diagonal matrix[4], provides a great deal of insight. Intuitively, $\mathbf{V}^T$ maps from rest space to a perfectly aligned space where the deformation is described by a non-uniform scale Σ, and $\mathbf{U}$ maps from this perfectly aligned space to world space. The, related, polar decomposition is used later to avoid elastic forces resulting from global rotations.

Strain. From the deformation gradient we define a *strain* metric. Strain is a dimension-less (or unit-less) quantity that measures the amount of deformation. Because strain is dimension-less it does not depend on the scale of an object. There are several strain metrics commonly used in computer graphics: Green's finite strain, Cauchy's infinitesimal strain, and the co-rotated strain. Green's finite strain, also known as the right Cauchy-Green strain, is given by

$$\epsilon_{ij} = \frac{1}{2}\left(\frac{\partial \mathbf{x}}{\partial u_i} \cdot \frac{\partial \mathbf{x}}{\partial u_j} - \mathbf{I}\right) = \frac{1}{2}\left(\mathbf{F}^T\mathbf{F} - \mathbf{I}\right). \tag{62}$$

This strain metric is straightforward and simple and has the significant property that it does not penalize world-space rotations, which led to widespread early adoption in computer graphics. Unfortunately, this metric is quadratic in positions, leading to a quartic energy function, which in turn leads to a *stiffness matrix*, $\mathbf{K}$—the Hessian (second derivative) of the energy—that is quadratic in positions. The result is that Green's strain always results in a non-constant $\mathbf{K}$, which makes various pre-computations and pre-factorizations invalid and results in significant computational cost.

To derive Cauchy's infinitesimal strain let us re-write the deformation gradient as $\mathbf{F} = \mathbf{I} + \mathbf{D}$. Then Green's strain is

$$\begin{aligned}
\epsilon &= \frac{1}{2}\left((\mathbf{I}+\mathbf{D})^T(\mathbf{I}+\mathbf{D}) - \mathbf{I}\right) && (63)\\
&= \frac{1}{2}\left(\mathbf{I}^T\mathbf{I} + \mathbf{D}^T + \mathbf{D} + \mathbf{D}^T\mathbf{D} - \mathbf{I}\right) && (64)\\
&= \frac{1}{2}\left(\mathbf{D}^T + \mathbf{D} + \mathbf{D}^T\mathbf{D}\right) && (65)\\
&= \frac{1}{2}\left((\mathbf{D}+\mathbf{I})^T + (\mathbf{D}+\mathbf{I}) + \mathbf{D}^T\mathbf{D}\right) - \mathbf{I} && (66)\\
&= \frac{1}{2}\left(\mathbf{F}^T + \mathbf{F} + \mathbf{D}^T\mathbf{D}\right) - \mathbf{I} && (67)\\
& && (68)
\end{aligned}$$

[4]Note that the Singular Value Decomposition is not unique. Singular values *are* unique and, if they are distinct, singular vectors are unique, up to the sign. For repeated singular values, the associated vectors form an orthonormal basis for the subspace associated with the singular value, so are not unique. Also, the singular values and vectors can be permuted and still form a valid decomposition, though it is common to order them from largest to smallest. It is sometimes helpful to view the SVD as $\mathbf{U}\Sigma\mathbf{V}^T = \sum_i \boldsymbol{\sigma}_i \mathbf{u}_i \mathbf{v}_i^T$, where we are using the subscript to select column of the matrices. That is, the original matrix is a sum of outer products of the singular vectors, weighted by their singular values.

Now if deformation is small (i.e. infinitesimal), then the $\mathbf{D}^T\mathbf{D}$ term is negligable compared to the other terms and can be dropped, resulting in Cauchy's infinitesimal strain,

$$\epsilon = \frac{1}{2}\left(\mathbf{F}^T + \mathbf{F}\right) - \mathbf{I} = \frac{1}{2}\left(\mathbf{F} + \mathbf{F}^T\right) - \mathbf{I}, \tag{69}$$

or using Einstein notation,

$$\boldsymbol{\epsilon}_{ij} = \frac{1}{2}\left(\frac{\partial \mathbf{x}}{\partial u_i} + \frac{\partial \mathbf{x}}{\partial u_j}\right) - \delta_{ij}. \tag{70}$$

The primary advantage of this strain metric is that it is linear, which leads to a quadratic energy and a *constant* Hessian that can be pre-factored or otherwise analyzed. The entire field of modal analysis is derived from this assumption that elastic deformations will occur, but will be very small. Unfortunately, this strain metric does penalize world-space rotations, leading to a variety of unpleasant artifacts under large deformations.

By far the most common strain model in computer graphics since its introduction in the mid-2000s is the co-rotated strain metric. Intuitively, this model is Cauchy's linear strain with the rotation explicitly removed through the polar decomposition. Once the deformation gradient, $\mathbf{F}$, is computed, we compute the polar decomposition $\mathbf{F} = \mathbf{Q}\tilde{\mathbf{F}}$ and replace $\mathbf{F}$ with $\tilde{\mathbf{F}}$ in Equation (69),

$$\epsilon = \frac{1}{2}\left(\tilde{\mathbf{F}} + \tilde{\mathbf{F}}^T\right) - \mathbf{I}. \tag{71}$$

By explicitly discarding the rotation the metric does not penalize rotations. The changing rotations mean that stiffness matrix is no longer constant, though some pre-computation is still possible and implementations can be quite efficient. It is worth noting that since $\tilde{\mathbf{F}}$ is symmetric, $\tilde{\mathbf{F}} = 1/2(\tilde{\mathbf{F}} + \tilde{\mathbf{F}}^T)$, but, while the values of these functions are equivalent, their derivatives are not.

Stress. Unlike strain, stress is not a dimension-less quantity. In three dimensions, stress has units of Newton's per meter squared (N/m^2). Instead of measuring the amount of deformation, it measures the materials reaction to that deformation. Different materials have different stress-strain relationships, which are often specified with parameterized models. Entire research careers in material science have been devoted to developing stress-strain models and fitting their parameters to real-world materials. The most common models in computer graphics are linear,

$$\boldsymbol{\sigma} = \mathbf{C}\boldsymbol{\epsilon}, \tag{72}$$

where $\mathbf{C}$ is a rank-four tensor containing 81 entries. If we take into account that both $\boldsymbol{\sigma}$ and $\boldsymbol{\epsilon}$ are symmetric, this reduces to 36 unique entries; further assuming that the material is iso-tropic (doesn't resist deformation in different directions differently), then there are just two parameters. There are many ways for specifying these two coefficients (see wikipedia) but a convenient one is the Lamé coefficients. In this case,

$$\sigma_{ij} = \lambda\epsilon_{kk}\delta_{ij} + 2\mu\epsilon_{ij}, \tag{73}$$

or in matrix notation

$$\boldsymbol{\sigma} = \lambda \mathrm{Tr}\left(\epsilon\right)\mathbf{I} + 2\mu\epsilon. \tag{74}$$

Here the stress is some constant times the strain plus a scaled identity matrix times the trace of the strain. The second term approximately preserves volume. Far more sophisticated stress-strain relationship have been developed, especially in the context of organic materials and elasto-plasticity for, e.g. granular materials and snow.

Elastic Potential, Traction, and Force. The elastic potential energy density is the product of stress and strain

$$\eta = \frac{1}{2}\sigma_{ij}\epsilon_{ij} = \frac{1}{2}\boldsymbol{\sigma} : \boldsymbol{\epsilon} = \frac{1}{2}\sum_{i,j}\sigma_{ij}\epsilon_{ij}. \tag{75}$$

The total elastic energy in a soft body is found by integrating the energy density over the volume of the body. Another important quantity is the traction, or force per unit area,

$$\boldsymbol{\tau} = \boldsymbol{\sigma}\mathbf{n}, \tag{76}$$

where $\mathbf{n}$ is the unit-magnitude surface normal to the area being integrated over.

Elastic forces will seek to reduce the energy of the system, thus they will be in the direction of the negative gradient of the energy. That is the force at a point, $\mathbf{x}_i$, will be

$$\mathbf{f}_i = -\frac{\partial \eta}{\partial \mathbf{x}_i}. \tag{77}$$

Forces can also be defined by integrating tractions over the boundary of a region, R, (the foundation of finite volume methods),

$$\mathbf{f} = \oint_{\partial R} \boldsymbol{\sigma}\mathbf{n}\, dS. \tag{78}$$

Stress Revisited. The definition of traction makes clear that stress maps from normals to forces (per unit area). However, it is important to distinguish where these normals and forces are defined. If both normals and forces are in world space, the stress is known as a Cauchy stress and often written as $\boldsymbol{\sigma}$. If the stress maps normals in material space to forces in material space it is known as a second Piola-Kirchhoff stress and sometimes written as $\mathbf{S}$. A first Piola-Kirchhoff stress maps normals in material space to forces in world space and is written $\mathbf{P}$. Now, watch out, when only one stress is being considered it is often written with $\boldsymbol{\sigma}$ regardless of its definition. The stress we defined earlier was actually a second Piola-Kirchhoff stress. Now its easy to convert between these stresses since they all measure the same thing just in different coordinate systems. Letting $J = \det(\mathbf{F})$ we have

$$\mathbf{P} = J\boldsymbol{\sigma}\mathbf{F}^{-T} = \mathbf{FS} \tag{79}$$

By having different ways of specifying stress, we can choose whichever one is most convenient for a given application. The first Piola-Kirchhoff is particularly attractive since it works with normals in the material space (where they are often constant) and maps directly to forces in world space (where they will be applied). But many material models (stress-strain relationships) yield a second Piola-Kirchhoff stress.

Plasticity. Plasticity refers to permanent deformation that typically occurs when a material fails. While sophisticated plasticity models have been developed in the graphics literature, incorporating basic plastic effects is straightforward. We begin by breaking the deformation gradient $\mathbf{F}$ into two parts, an elastic part and a plastic part,

$$\mathbf{F} = \mathbf{F}_e\mathbf{F}_p. \tag{80}$$

Then when computing elastic strain, stress, forces, etc, we ignore the plastic part and only use the elastic part, $\mathbf{F}_e$. Intuitively, $\mathbf{F}_p$ measures how the rest shape is (permanently) changing and $\mathbf{F}_e$ is measuring how the mapping from rest to world space. Elastic forces will seek to undo the latter deformation. The sophistication we referred to earlier has to do with how the decomposition in Equation (80) is formulated and is beyond the scope of these notes.

2.2.3 Fluids. Fluids surround us in our daily lives, from the air that we breathe, to water droplets, oceans, and fire. Other materials that flow include sand, toothpaste, and putty. Intuitively, we can think of a fluid as a material that flows to conform to the shape of its container. Physically, a fluid is a material that, unlike a solid, cannot support a shear force and instead rearranges itself in response. Fluids do support compressive forces, as can be seen by considering a balloon filled with air, where the elastic balloon exerts compressive forces that are balanced by air pressure.

Here, we give a brief overview of the quantities and equations describing fluid motion. We refer the interested reader to the excellent comprehensive text on fluid simulation in computer graphics by Bridson [2015]. Many of the fluids of interest in physics-based animation can be described by the incompressible Navier-Stokes equations,

$$\rho(\mathbf{u}_t + \mathbf{u} \cdot \nabla \mathbf{u}) = -\nabla p + \mu \Delta \mathbf{u} + \mathbf{f}, \tag{81}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{82}$$

where ρ is the fluid density, $\mathbf{u}(\mathbf{x}, t) = (u(\mathbf{x}, t), v(\mathbf{x}, t), w(\mathbf{x}, t))^T$ is the three-dimensional fluid velocity, $\mathbf{u}_t$ is shorthand for $\partial \mathbf{u}/\partial t$, p is the pressure, μ is the dynamic viscosity, and $\mathbf{f}$ represents body forces such as gravity. Note that the mathematics community tends to use Δ to denote the Laplacian operator, while the engineering community tends to favor ∇^2; $\nabla \cdot \nabla$ is also sometimes used. The first equation is derived using the principle of conservation of momentum [Landau and Lifshitz 1959; Bridson 2015]. In the form given it has units of force per unit volume (N/m^3), and is analogous to Newton's second law $ma = f$.

Material derivative. On the left hand side, we identify the density ρ with the mass m, while the acceleration term takes the form

$$\mathbf{u}_t + \mathbf{u} \cdot \nabla \mathbf{u}. \tag{83}$$

To understand where the acceleration form in Eq. (83) comes from, consider a fluid particle with position $\mathbf{x}_p(t)$. Above, we described the fluid velocity field $\mathbf{u}(\mathbf{x}, t)$ as a function of position $\mathbf{x}$ and time t. The fluid particle will be moving through the fluid velocity field. At any given time, the velocity of the fluid particle $\mathbf{v}_p(t)$ is equal to the velocity at the particle's position,

$$\mathbf{v}_p(t) = \mathbf{u}(\mathbf{x}(t), t).$$

Therefore, the acceleration of the fluid particle is

$$\mathbf{a}_p(t) = \frac{d}{dt}\mathbf{v}_p(t) \tag{84}$$

$$= \frac{d}{dt}\mathbf{u}(\mathbf{x}_p(t), t) \tag{85}$$

$$= \left(\frac{\partial \mathbf{u}}{\partial t} + \frac{\partial \mathbf{u}}{\partial \mathbf{x}} \frac{d\mathbf{x}_p}{dt} \right) (\mathbf{x}_p(t), t), \tag{86}$$

where we have used the chain rule. With the notation $\mathbf{u}_t = \frac{\partial \mathbf{u}}{\partial t}$, $\nabla \mathbf{u} = \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$, and using the fact that $\frac{d\mathbf{x}_p}{dt} = \mathbf{u}(\mathbf{x}_p, t)$, we can see that the acceleration of a fluid particle at $\mathbf{x}$ is given by

$$\frac{D\mathbf{u}}{Dt} = \mathbf{u}_t + \mathbf{u} \cdot \nabla \mathbf{u},$$

the form that appears in the Navier-Stokes equations. This quantity goes by many names, commonly the *material derivative* or *substantial derivative* of the velocity field. To clarify, it is worth writing

out the derivative in terms of its components, recalling that $\mathbf{u} = (u, v, w)^T$:

$$\frac{D\mathbf{u}}{Dt} = \mathbf{u}_t + \mathbf{u} \cdot \nabla \mathbf{u} = \begin{pmatrix} u_t + \mathbf{u} \cdot \nabla u \\ v_t + \mathbf{u} \cdot \nabla v \\ w_t + \mathbf{u} \cdot \nabla w \end{pmatrix}.$$

Forces. On the right hand side of Eq. (81) are the forces acting on the fluid. The first term is the negative pressure gradient in the flow. The pressure $p(\mathbf{x}, t)$ is a scalar field, that in an incompressible fluid acts everywhere to maintain the volume of the flow, resisting any compression or expansion forces. Since the gradient of a function points uphill in the direction of steepest increase, and hence the negative gradient points downhill, this forcing term drives the flow from areas of high pressure to areas of low pressure. In an incompressible fluid, the pressure forces arise internally to resist compression or expansion due to other forces. The pressure field will take on exactly the values needed to ensure that the fluid in any arbitrary region does not compress or expand. For this reason, it is sometimes called a "Lagrange multiplier" for the divergence-free constraint on the fluid, a concept we will discuss Section 5.

The second term on the right hand side of Eq. (81) represents the viscous forces in the fluid. The *dynamic viscosity* of the fluid, μ, is a material parameter that depends on the type of fluid. Generally, "thicker" fluids like honey have a high viscosity. The viscosity of water is one to two orders of magnitude larger than the viscosity of air, while honey is three to four orders of magnitude more viscous than water. Viscosity is also sometimes specified in terms of the *kinematic viscosity*, $\nu = \mu/\rho$. Viscous forces arise from drag between layers of fluid flowing at different velocities. The Laplacian operator, Δ, can be thought of as measuring the difference between the value of a function at a point and the average surrounding values, and the viscous force $\mu\Delta\mathbf{u}$ in Eq. (81) penalizes velocity differences, acting to equalize velocities throughout the fluid. When a viscous fluid flows past a solid wall, the fluid particles adjacent to the wall experience drag from the wall and stick to it. This is called the *no-slip* boundary condition and says that the fluid should not slip at the surface of the wall. Mathematically, this is expressed as

$$\mathbf{u}(\mathbf{x}, t) = \mathbf{V}(\mathbf{x}, t), \quad \mathbf{x} \in \Gamma$$

where $\mathbf{V}$ is the velocity of the solid wall, and Γ is the boundary of the fluid domain in contact with the wall.

Finally, the $\mathbf{f}$ in Eq. (81) represents other forces such as gravity. It may also be used to model a variety of other effects, such as surface tension forces, control forces for driving the flow based on artistic goals, or forces due to elastic stuctures embedded in the fluid.

Incompressibility. As we saw in Section 2.1.2, conservation of mass leads to the conservation law

$$\rho_t + \nabla \cdot (\rho \mathbf{u}) = 0.$$

Applying the product rule to the spatial derivative term gives

$$\begin{aligned} \nabla \cdot (\rho \mathbf{u}) &= \partial_x(\rho u) + \partial_y(\rho v) + \partial_z(\rho w) && (87) \\ &= (\partial_x \rho)u + (\partial_y \rho)v + (\partial_z \rho)w + \rho(\partial_x u + \partial_y v + \partial_z w) && (88) \\ &= \mathbf{u} \cdot \nabla \rho + \rho \nabla \cdot \mathbf{u}. && (89) \end{aligned}$$

Therefore, conservation of mass can be written as

$$\frac{D\rho}{Dt} + \rho \nabla \cdot \mathbf{u} = 0,$$

using the notation for the material derivative. Since the fluid should not compress or expand under the incompressibility assumption, $\frac{D\rho}{Dt} = 0$, so the conservation of mass equation reduces to the

divergence-free condition seen in Eq. (81)

$$\nabla \cdot \mathbf{u} = 0. \tag{90}$$

This condition holds everywhere in the fluid. Integrating the divergence-free condition over an arbitrary control volume, and applying the divergence theorem, we have

$$0 = \int_{\Omega} \nabla \cdot \mathbf{u} \; dV = \int_{\partial\Omega} \mathbf{u} \cdot \mathbf{n} \; dS, \tag{91}$$

which says that the total flow through the boundary of any region in the flow must be zero. If other forces attempt to compress or expand the fluid, an internal pressure arises to counteract those forces and prevent the change in volume.

3 SPATIAL DISCRETIZATION

In this section we review essential concepts for spatially discretizing continuous equations of motion. We first discuss Lagrangian and Eulerian reference frames, then discuss the most common spatial data structures used to store simulation variables before discussing how quantities stored in the structures are interpolated at arbitrary points in space. Finally, we discuss two of the most common approaches to discretizing equations of motion: finite differences and finite element methods.

3.1 Lagrangian vs Eulerian

A fundamental concept when discussing spatial discretization is the distinction between Lagrangian and Eulerian reference frames. In fact, the concept is so fundamental that it is often only implicitly mentioned.

As a concrete example, imagine we wanted to know how fast a river was flowing. There are two ways we could measure its flow rate: we could wade out into the river and see how quickly the water moved by us or we could jump on a raft and see how quickly we moved. In the first approach the point of measurement, the reference frame, is fixed in space. We refer to such a fixed reference frame as *Eulerian*. In the second approach, the reference frame moves with the material and we refer to the reference frame as *Lagrangian*. Eulerian reference frames are often used for fluids and often employ regular grids and finite differences and Lagrangian reference frames are often used for solids and employ tetrahedral meshes or particles and finite element methods. These trends are so common that it is easy to confuse the underlying distinction that Eulerian frames are fixed and Lagrangian frames move with the material.

The reason Lagrangian frames are often used for soft bodies is that these simulations need a mapping ($\mathbf{x}(\mathbf{u})$) from rest or material space to deformed or world space (see Section 2.2.2) in order to compute elastic forces. This mapping is easily constructed if we explicitly track points in the material through time. Eulerian approaches are well-suited to fluids because we do not need such a mapping and fixed reference frames offer many computational advantages.

There is also a difference in the mathematical notation we use when working in an Eulerian frame vs. a Lagrangian frame. In a Lagrangian frame we will often write the material derivative of a field y

$$\frac{Dy}{Dt}, \tag{92}$$

while when working in an Eulerian frame we will apply the chain rule and write

$$\frac{\partial y}{\partial t} + \mathbf{u} \cdot \nabla y, \tag{93}$$

breaking out the rate of change at a fixed point and the advection through the flow field (see Section 2.2.3.

3.2 Grids, Meshes, Particles

Grids. Perhaps the most common spatial data structure is the regular grid. In this structure all edges have the same length, called the *grid spacing*, which is often denoted h or Δx. Individual cubes in the grid are referred to as *cells* which have eight *vertices*, twelve *edges*, and six *faces*. The grid can be described by a few redundant parameters: the grid spacing, the grid resolution (i.e. the number of cells in each dimension), and the upper and lower extent of the gird. Grids are typically fixed in space and do not change shape, thus typically an Eulerian frame is adopted. In many implementations the grid is abstract, with only part of it every being allocated. In this case a

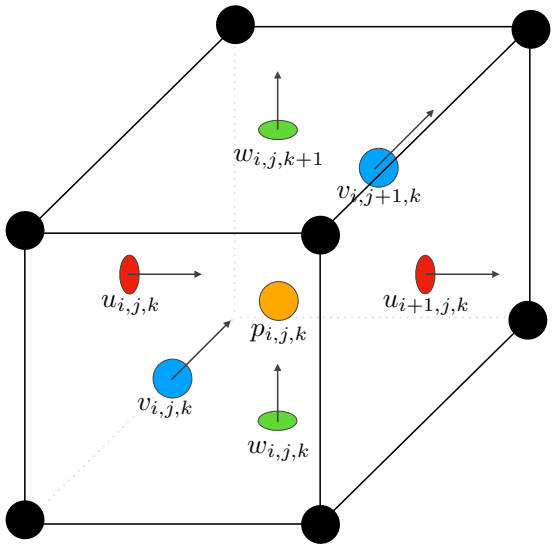


Fig. 6. *Staggered grid*

hierarchical structure, such as an octree or B-tree, is typically used to represent the full domain. OpenVDB [Museth 2013] is an example of such a structure.

A question that arises when considering grids is where to store simulation variables. It is equally natural to store variables at cell centers or at vertices and either convention is frequently adopted. A special case, known as the *staggered grid* (sometimes less-usefully referred to as the *MAC grid*, because it was introduced by Harlow and Welch [1965] with their Marker-and-Cell fluid simulation method) stores different variables at different locations. This structure is commonly used in fluid simulation to achieve second-order accuracy when using finite differences at small additional computational cost. The resulting computer code is more complicated than when variables are co-located. In this case individual component of the velocity $\mathbf{u}$ are stored as scalars at face centers, while pressure is stored at cell centers, see Figure 6. Note that some authors prefer using *half-index* notation so that a velocity may be referred to as $u_{i-1/2,j,k}$, but this maps less cleanly to computer code.

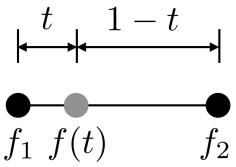
Meshes. "Meshes" itself is an ill-defined term, but typically the term is used to refer to *simplicial complexes*. A k-simplex contains $k+1$ vertices that are all connected. A 0-simplex is a point, a 1-simplex a line segment or edge, a 2-simplex a triangle, and a 3-simplex a tetrahedron. A simplicial complex is decomposes a domain into a set of disjoint simplices, triangles in 2D and tetrahedra in 3D. These simplices meet at lower-dimensional simplices forming the *complex*. For example, two tetrahedra meet at a face, which will be a triangle. Similarly, two triangles meet at an edge, and two edges meet at a vertex. Automatic tetrahedral meshing is still a difficult problem that continues to receive attention from the SIGGRAPH and Computational Geometry research communities. In fact, the difficulty of generating high quality tetrahedral meshes for arbitrary geometry remains a significant impediment to their widespread use.

Particles. Given the difficulty of generating tetrahedral meshes, some choose to represent geometry as a set of particles. Particles then typically interact with nearby particles. The primary advantage of this approach is simplicity. The primary drawback is that, because space is not broken into disjoint pieces, integration becomes more difficult.

Hybrid Structures. In recent years hybrid data structures have become increasingly common. In particular combining Lagrangian particles with regular *background* grids, such as in the Fluid Implicit Particle (FLIP) and Material Point Method (MPM) approaches, has proven very successful at combining the various strengths of these two approaches.

3.3 Interpolation

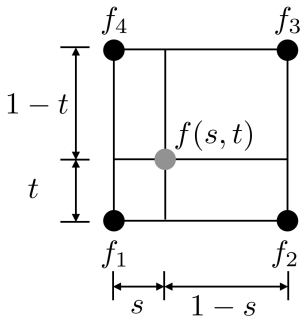
As we saw previously, the values of scalar, vector, or tensor fields that we are tracking are often sampled at discrete points on a grid, mesh, or set of particles. When the value of the field is required at a location other than the sample points, it must be *interpolated* from the discrete samples to the desired location. A simple example of interpolation is finding the value of some quantity on a line segment between two endpoints for which we have samples, illustrated below. The samples at the



endpoints are f_1 and f_2, t is the fraction of the distance from f_1 to f_2 at which we want to determine the value $f(t)$. *Linear interpolation* between the endpoints gives us

$$f(t) = (1 - t)f_1 + tf_2.$$

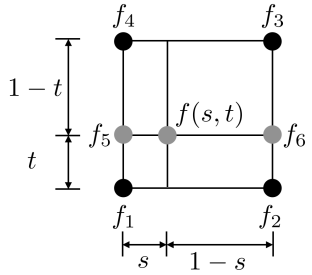
As a sanity check, we see that this formula gives $f(0) = f_1$ and $f(1) = f_2$ as desired. Notice that the formula is computing a weighted average of the two endpoints, with the weights of each endpoint being the length ratio of the line sub-segment *opposite* the sample. Importantly, the weights in any interpolation formula should always sum to 1, forming a *partition of unity*. This is indeed the cases here as $1 - t + t = 1$. The formula above can also be rearranged into $f(t) = f_1 + t(f_2 - f_1)$, but the former expression is more useful for generalizing to bilinear, trilinear, and barycentric interpolation, as we do next.



We now move to two dimensions, where we have a function sampled at the four corners of a rectangle or square, with samples f_1, f_2, f_3, f_4. If the function value is desired at a point inside the rectangle, we can use *bilinear interpolation*. We first assign the coordinates (s, t) to the point in the rectangle, where s is the fraction of the distance from f_1 to f_2, and t is the fraction of the distance from f_1 to f_4, as illustrated in the figure. With bilinear interpolation, we take a weighted average of the function values at the four corner samples. As with linear interpolation, the weight of each sample point is equal to the relative area of the region opposite that sample point, so that the formula for bilinear interpolation is

$$f(s,t) = (1-s)(1-t)f_1 + s(1-t)f_2 + stf_3 + (1-s)tf_4. \tag{94}$$

Notice that, as required, the weights sum to 1. The formula can also be derived by using two linear interpolation steps to intermediate values f_5 and f_6, and then a final linear interpolation step to $f(s,t)$ as illustrated below. From here, we have



$$f_5 = (1-t)f_1 + tf_4 \tag{95}$$

$$f_6 = (1-t)f_2 + tf_3 \tag{96}$$

$$f(s,t) = (1-s)f_5 + sf_6 \tag{97}$$

$$= (1-s)\left((1-t)f_1 + tf_4\right) + s\left((1-t)f_2 + tf_3\right). \tag{98}$$

Rearranging gives Eq. (94). The rule that each weight is the relative area of the region opposite the sample is easier to recall and generalizes nicely.

In three dimensions, we use *trilinear interpolation* to interpolate values from the eight corners of a rectangle to a point inside the rectangle. If the point is at coordinate (s, t, u) inside the rectangle, then by using the relative volumes of the rectangle opposite the sample point, trilinear interpolation gives

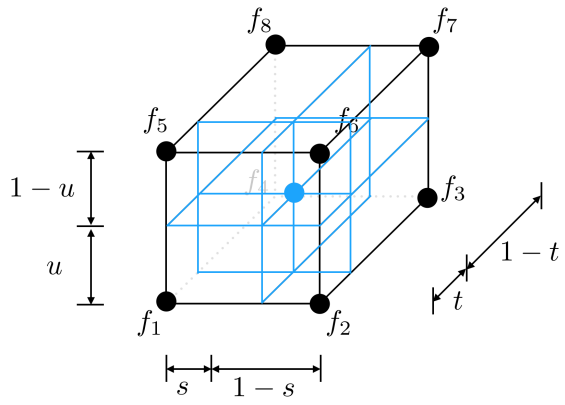
$$f(s,t,u) = (1-s)(1-t)(1-u)f_1 + s(1-t)(1-u)f_2 + st(1-u)f_3 + (1-s)t(1-u)f_4 \tag{99}$$

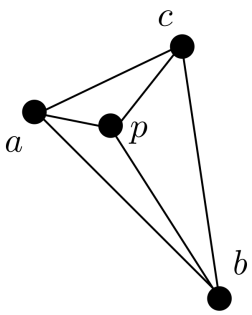
$$+ (1-s)(1-t)uf_5 + s(1-t)uf_6 + stuf_7 + (1-s)tuf_8. \tag{100}$$

As before, we could also derive this formula by first doing bilinear interpolation to intermediate points on two opposite faces using two of the three coordinates, and then doing a linear interpolation between those two using the third coordinate.

We saw previously that triangles and more generally simplicial complexes are used frequently in physics-based animation. Similiar to the above, we can linearly interpolate between the three nodes of a triangle using *barycentric coordinates*. To interpolate to a point p, we associate with the vertices a, b, and c the weights α, β, and γ, respectively, so that a quantities f_a, f_b, and f_c are interpolated as

$$f_p = f(\alpha, \beta, \gamma) = \alpha f_a + \beta f_b + \gamma f_c. \tag{101}$$





As before, $\alpha + \beta + \gamma = 1$, and can be determined by finding the ratio of the area of the opposite subtriangle to the whole triangle. Specifically,

$$\alpha = \frac{\text{area}(p, b, c)}{\text{area}(a, b, c)}, \tag{102}$$

$$\beta = \frac{\text{area}(p, c, a)}{\text{area}(a, b, c)}, \tag{103}$$

$$\gamma = \frac{\text{area}(p, a, b)}{\text{area}(a, b, c)}. \tag{104}$$

Notice that at each vertex a, b, or c, the corresponding weight α, β, or γ, respectively, is 1, while the other weights are 0. Along any of the three edges of the triangle, the weight of the opposite vertex is 0 while the other two weights can be determined by linear interpolation between the edge vertices. The areas above are signed areas, with the convention that counter-clockwise orientation has positive sign, and clockwise orientation has negative sign. Some of the barycentric coordinates of points in the plane of a triangle can be negative, but they must always sum to one. In addition to being used for interpolation of quantities throughout the triangle, barycentric coordinate are sometimes determined for a point to test whether the point lies inside the triangle or on its boundary (all coordinates non-negative), or outside the triangle (one or more coordinates negative). Alternate ways of finding barycentric coordinates exist (see e.g., [Marschner and Shirley 2015]).

Sometimes it is convenient to remove the redundancy introduced by the constraint that $\alpha+\beta+\gamma = 1$ and write

$$f_p = f(\alpha,\beta) = (1-\alpha-\beta)f_a + \alpha f_b + \beta f_c = f_a + \alpha(f_b - f_a) + \beta(f_c - f_a) \tag{105}$$

where the definitions of α and β have changed. This formulation often leads to more efficient code.

Polynomial interpolation. Given $n+1$ distinct data points $(t_1, f_1), \ldots, (t_{n+1}, f_{n+1})$, there is a unique polynomial of degree n that interpolates the data points. One approach to writing down the interpolating polynomial is via *Lagrange interpolation*, where

$$f(t) = \sum_{i=1}^{i=1} f_i l_i(t) \tag{106}$$

$$l_i(t) = \frac{\Pi_{k\neq i}(t - t_k)}{\Pi_{k\neq i}(t_i - t_k)}. \tag{107}$$

The polynomial can also be determined using monomial basis elements, leading to the ill-conditioned Vandermonde matrix, or using Newton interpolation. Further details as well as advantages and disadvantages of the different approaches can be found in a textbook such as [Heath 2002].

Approximating functions. While interpolating functions go through a set of discrete samples and can be used to determine function values between the samples, approximating functions do not necessarily go through the data points. One of the most common approaches to approximating a set of sample data is *least squares* interpolation. When data is noisy or a system is overdetermined, it is often undesirable to require interpolation of the data points, as very poor approximations can result. Instead, we seek to find unknown parameters for a model that is thought to fit the data, while minimizing the resulting error between the model and data. In least squares, given m data points (x_i, f_i), $i = 1, \ldots, m$, a set of n basis functions $\phi_1(x), \ldots, \phi_n(x)$ are assumed to model the data, and n coefficients $\alpha_1, \ldots, \alpha_n$ are sought so that the resulting approximating function

$$\phi(x) = \alpha_1\phi_1(x) + \alpha_2\phi_2(x) + \ldots + \alpha_n\phi_n(x)$$

minimizes the sum of the squared errors,

$$\sum_{i=1}^{m} |\phi(x_i) - f_i|^2. \tag{108}$$

Once the basis functions are chosen, the approximation is completed by fitting the coefficients $\alpha_1, \ldots, \alpha_n$ to the data, resulting in the optimization

$$\text{argmin}_{\boldsymbol{\alpha}} \|A\boldsymbol{\alpha} - \mathbf{f}\|^2 \tag{109}$$

where

$$A = \begin{pmatrix} \phi_1(x_1) & \phi_2(x_1) & \cdots & \phi_n(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \cdots & \phi_n(x_2) \\ \vdots & & & \\ \phi_1(x_m) & \phi_2(x_m) & \cdots & \phi_n(x_m) \end{pmatrix}, \boldsymbol{\alpha} = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{pmatrix}, \mathbf{f} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix}. \tag{110}$$

The normal equations $A^TA\boldsymbol{\alpha} = A^T\mathbf{f}$ can be formed and used to solve the nonsingular least squares problem, although this method is poorly conditioned and better alternatives exist [Heath 2002].

Sometimes a weighted version of Eq. (108) is used, so that the error to be minimized is

$$\sum_{i=1}^{m} w_i|\phi(x_i) - f_i|^2, \tag{111}$$

where the w_i are weights that modify the relative contribution of each error term to the total error. More generally, these weights can vary over the domain. Such *moving least squares* approaches are

used in scattered data interpolation to reconstruct functions from unstructured data [Shen et al. 2005].

Approximating piecewise polynomial curves such as Bezier curves and B-splines are also sometimes used in physics-based animation to reconstruct functions from sample data [Marschner and Shirley 2015].

Finally, we note that in the context of geometric deformation of character models, specialized coordinates for interpolation have been developed that work with more general structures.These include mean-value coordinates [Floater 2003; Ju et al. 2005] harmonic coordinate [Joshi et al. 2007], green coordinates [Lipman et al. 2008], and bounded biharmonic weights [Jacobson et al. 2011].

3.4 Finite Differences

The evolution equations that we have encountered often involve first and second order spatial derivatives of functions. *Finite difference* methods are commonly used to discretize the differential operators for numerical solution. Recall that the derivative of a function $f(x)$ is

$$\frac{d}{dx}f(x) = \lim_{h\to 0}\frac{f(x+h)-f(x)}{h} \tag{112}$$

if this limit exists. By choosing h to be some small yet finite number, a finite difference approximation to the derivative of f at a point x is

$$\frac{d}{dx}f(x) \approx \frac{f(x+h)-f(x)}{h}. \tag{113}$$

The error in this approximation decreases with decreasing h. The exact dependence between h and the error can be investigated using the *Taylor series* of the function f about the point x (when the Taylor series converges in some interval about x), specifically

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + \ldots + \frac{h^n}{n!}f^{(n)}(x) + \ldots. \tag{114}$$

Rearranging and dividing through by h, we see that

$$f'(x) = \frac{f(x+h)-f(x)}{h} - \frac{h}{2}f''(x) + \frac{h^2}{6}f'''(x) + \ldots + \frac{h^{n-1}}{n!}f^{(n)}(x) + \ldots, \tag{115}$$

or

$$f'(x) = \frac{f(x+h)-f(x)}{h} + O(h), \tag{116}$$

since the $O(h)$ term in the infinite series is the dominant term as $h \to 0$. This choice of finite difference approximation to $f'(x)$ is called a *forward difference*. Since its dominant error term is $O(h)$, it is said to be *first order accurate*. Another choice of finite difference approximation to $f'(x)$ is the *backward difference*

$$f'(x) \approx \frac{f(x)-f(x-h)}{h}. \tag{117}$$

Taylor series expansion about x gives

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + \ldots,$$

which can be rearranged to

$$f'(x) = \frac{f(x)-f(x-h)}{h} + \frac{h}{2}f''(x) - \frac{h^2}{6}f'''(x) + \ldots, \tag{118}$$

or

$$f'(x) = \frac{f(x) - f(x-h)}{h} + O(h). \tag{119}$$

Like the forward difference, the backward different is first order accurate, although notice that their dominant error terms differ by a sign. The significance of first order accuracy is that if the spacing between sample points, h, is halved, the dominant error term will also be halved as it is linear in the size of the mesh spacing.

A common *second order accurate* finite difference discretization is the *central difference*,

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}. \tag{120}$$

We can again use Taylor series to study the order of accuracy, using

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + O(h^4), \tag{121}$$

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + O(h^4). \tag{122}$$

Taking the difference of these two expressions and dividing by $2h$, we get

$$\frac{f(x+h) - f(x-h)}{2h} = \frac{1}{2h}\left(2hf'(x) + \frac{h^3}{3}f'''(x) + O(h^4)\right) \tag{123}$$

$$= f'(x) + \frac{h^2}{6}f'''(x) + O(h^3) \tag{124}$$

$$= f'(x) + O(h^2) \tag{125}$$

This shows that the dominant error term for the central difference is $O(h^2)$, making it second order accurate. This means if we decrease the mesh spacing from h to $h/2$, the dominant error term will scale by a factor of $1/4$. Thus, asymptotically, the error decreases more quickly with decreasing h in a higher order method.

When choosing a finite difference scheme, order of accuracy is only one of several considerations. Other considerations include the overall stability of the scheme, the nature of the errors produced by the scheme (e.g., dissipative or dispersive), and the conservation properties of the scheme. For example, when discretizing advective terms in a partial differential equation, the central difference scheme above can lead to undesirable oscillations or instabilities when combined with certain time discretization methods. *Upwind discretizations* are often preferred for advection. These schemes use the local flow direction to choose between the forward or backward differences, or higher-order one-sided discretizations.

We can also approximate higher derivatives with finite difference approximations. For example, we can write a central difference approxmation to $f''(x)$ as

$$f''(x) = \frac{f'(x + \frac{h}{2}) - f'(x - \frac{h}{2})}{h} + O(h^2) \tag{126}$$

$$= \frac{\frac{f(x+h)-f(x)}{h} - \frac{f(x)-f(x-h)}{h}}{h} + O(h^2) \tag{127}$$

$$= \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + O(h^2), \tag{128}$$

where Taylor series analysis of this finite difference approximation shows that the odd errors terms in the approximations to $f(x-h)$ and $f(x+h)$ cancel perfectly, leaving an $O(h^4)$ dominant error term in the numerator, and thus giving second order accuracy.

Laplacian operator. Now consider approximating the Laplacian operator applied to a function u, i.e., Δu using finite differences. In two dimensions, $\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$, and each of the two second order derivatives can be approximated using the centered difference scheme in (128) as

$$\frac{\partial^2 u(x,y)}{\partial x^2} \approx \frac{u(x+h,y) - 2u(x,y) + u(x-h,y)}{h^2} \tag{129}$$

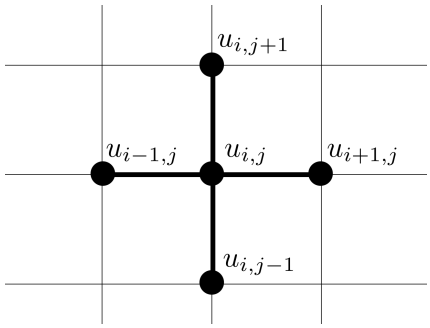
$$\frac{\partial^2 u(x,y)}{\partial y^2} \approx \frac{u(x,y+h) - 2u(x,y) + u(x,y-h)}{h^2} \tag{130}$$

Combining these, we get

$$\frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2} \approx \frac{u(x+h,y) - 2u(x,y) + u(x-h,y)}{h^2} + \frac{u(x,y+h) - 2u(x,y) + u(x,y-h)}{h^2} \tag{131}$$

$$= \frac{u(x+h,y) + u(x-h,y) + u(x,y+h) + u(x,y-h) - 4u(x,y)}{h^2} \tag{132}$$

If we consider our function u to be sampled at the nodes of a uniform two-dimensional grid, as illustrated below and associate with the location x, $x+h$, y, $y-h$ the grid indices i, $i+1$, j, $j-1$,



etc., respectively, we can write the finite difference approximation in a the commonly used form

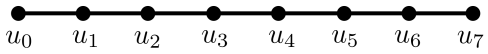
$$\Delta u(x,y) \approx \frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}}{h^2} \tag{133}$$

The figure above illustrates which surrounding sample points are used in the finite difference approximation, and is called the *stencil* of the finite difference scheme. The particular stencil illustrated for the Laplacian is very commonly used and is called the *five-point stencil*. In one dimension and three dimensions, the analogous stencils are the three- and seven-point stencils, respectively.

We now consider discretizing the one-dimensional Poisson equation

$$u_{xx} = f, \quad x \in \Omega \tag{134}$$

on a uniform one-dimensional grid of size 8. We use the finite difference approximation in (128) at each point in the grid. Discretizing the Laplacian over the grid gives rise to a matrix that is sparse yet globally couples all variables. Finite difference discretization of differential operators generally gives rise to sparse matrices, since differential operators give local information about functions.



We number the n samples from the left to right $i = 0, 1, \ldots, n-1$. At node i on the grid, the finite difference approximation gives the equation

$$\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} = f_i.$$

We apply this discretization at each of the 8 grid nodes and assemble the resulting equations into a linear system. Here we notice that at the two endpoints of the grid, we do not have all the neighbors required by the stencil. Mathematically, we require boundary conditions for the solution of Eq. (134) to be fully determined. The boundary conditions are then used in discretizing the equation at the boundary points near the boundary. If we assume, for example, the Dirichlet boundary condition

$$u(x) = \bar{u}(x), \quad x \in \partial\Omega, \tag{135}$$

then the discretized Laplace equation for our example becomes the linear system

$$\frac{1}{h^2}\begin{pmatrix} -2 & 1 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 1 & -2 \end{pmatrix}\begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{pmatrix} = \begin{pmatrix} f_1 - \frac{\bar{u}(x_0)}{h^2} \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 - \frac{\bar{u}(x_7)}{h^2} \end{pmatrix} \tag{136}$$

Note how the first and last rows in the matrix have only 2 nonzero terms, while the rows corresponding to interior samples have 3. For the first and last rows, the third term is a boundary term and not an unknown, and hence appears on the right hand side of the equation.

Neumann boundary conditions, which specify derivatives are also common. If we assume that the derivative at the boundary is zero,

$$\frac{\partial u(x)}{\partial x} = 0 \in \partial\Omega, \tag{137}$$

as we would for pressure at a solid boundary, then the discretized Laplace equation for our example becomes the linear system

$$\frac{1}{h^2}\begin{pmatrix} -1 & 1 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix}\begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \end{pmatrix} \tag{138}$$

Note how the first and last rows in the matrix have only 2 nonzero terms and that the diagonal entry has been reduced. For the first and last rows, the third entry must be equal to the solution and the matrix entries cancel.

3.5 Finite Elements

The *Finite Element Method* can seem a little intimidating and it is true that a rigorous mathematical derivation is a bit involved, requiring the introduction of arbitrary *test functions* and *integration by parts*. However, the underlying ideas and the linear finite elements that are commonly encountered in graphics are actually quite straightforward.

As seen in the last section, finite differences discretize derivatives directly by replacing infinitesimal differences with finite ones. The key idea underlying the finite element method is to, instead, discretize the space of functions that can be represented and solve problems in that limited space. As the name implies, we begin by breaking an object up into a finite set of elements. The elements are disjoint (non-overlapping) and provide a spatial discretization of the object. Common elements are simplices (triangles in 2D and tetrahedra in 3D) and deformed hypercubes (quadrilaterals in 2D and hexahedra in 3D). We then define *basis functions* over these elements. Unsurprisingly, the most common basis function space is piecewise linear and we will limit our discussion to this case. Arbitrary functions in the equations we wish to solve are then *projected* onto this piecewise linear space. This projection is often referred to as a *Galerkin* projection. Intuitively, given a spatial discretization, we find the closest piecewise linear function to the function of interest. We then solve the problem in this limited space, resulting in a piecewise linear solution, which, of course, just an approximation to the solution to the original problem.

Application to Soft Bodies. Finite elements are commonly used in graphics for animating elastic bodies. Recall that to do so we must compute the deformation gradient, $\mathbf{F}$. Because we are using a piecewise linear basis to represent the deformation function $\mathbf{x}(\mathbf{u})$, the gradient of this function will be piecewise constant. That is $\mathbf{F}$ is constant over an element. Recall that an arbitrary point inside a triangle can be represented with barycentric coordinates as

$$\mathbf{u} = \mathbf{u}_0 + \alpha\left(\mathbf{u}_1 - \mathbf{u}_0\right) + \beta\left(\mathbf{u}_2 - \mathbf{u}_0\right). \tag{139}$$

That same point, in the deformed element will have the same barycentric coordinates and can be computed as

$$\mathbf{x} = \mathbf{x}_0 + \alpha\left(\mathbf{x}_1 - \mathbf{x}_0\right) + \beta\left(\mathbf{x}_2 - \mathbf{x}_0\right). \tag{140}$$

Letting $\mathbf{u}_{ab}$ be the vector $\mathbf{u}_a - \mathbf{u}_b$ and similarly for $\mathbf{x}$, we can write these in matrix form as

$$\mathbf{u} = \mathbf{u}_0 + \begin{pmatrix}\mathbf{u}_{10} & \mathbf{u}_{20}\end{pmatrix}\begin{pmatrix}\alpha\\ \beta\end{pmatrix} \tag{141}$$

and

$$\mathbf{x} = \mathbf{x}_0 + \begin{pmatrix}\mathbf{x}_{10} & \mathbf{x}_{20}\end{pmatrix}\begin{pmatrix}\alpha\\ \beta\end{pmatrix}, \tag{142}$$

where the columns of the matrices are vectors along the edges of the element. Putting these together we can write the deformation function that takes $\mathbf{u}$ to $\mathbf{x}(\mathbf{u})$ as

$$\mathbf{x}(\mathbf{u}) = \mathbf{x}_0 + \begin{pmatrix}\mathbf{x}_{10} & \mathbf{x}_{20}\end{pmatrix}\begin{pmatrix}\mathbf{u}_{10} & \mathbf{u}_{20}\end{pmatrix}^{-1}\left(\mathbf{u} - \mathbf{u}_0\right). \tag{143}$$

The gradient of this function with respect to $\mathbf{u}$, the deformation gradient is,

$$\mathbf{F} = \frac{\partial \mathbf{x}}{\partial \mathbf{u}} = \begin{pmatrix}\mathbf{x}_{10} & \mathbf{x}_{20}\end{pmatrix}\begin{pmatrix}\mathbf{u}_{10} & \mathbf{u}_{20}\end{pmatrix}^{-1}. \tag{144}$$

Thus, the finite element method, using linear basis functions, allows us to compute the deformation gradient for a single element by simply assembling matrices composed of vectors along edges of the elements, inverting one of the matrices and then performing a matrix multiply. This simple process is remarkably powerful because once we have the deformation gradient we compute any stress measure we would like and from there compute elastic forces. Also, note that if the

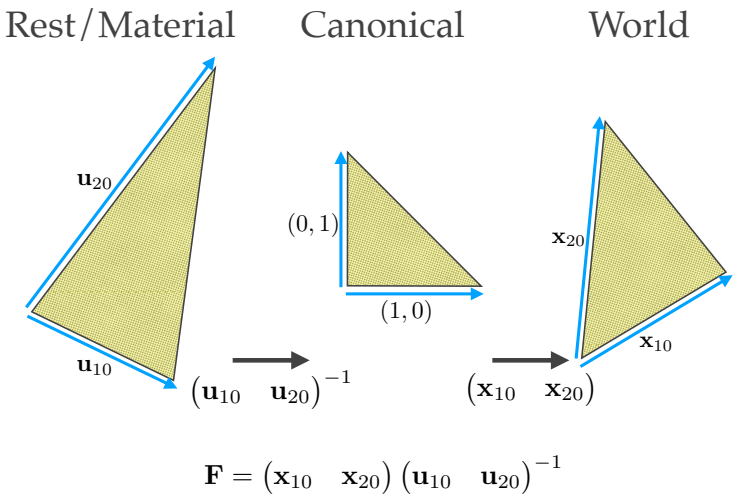


Fig. 7. *The finite element method. To compute the deformation gradients we assemble matrices composed of vectors along the edges of the elements in rest and world space. We invert the rest space matrix and premultiple by the world space matrix.*

object's rest shape does not change, the matrix that needs to be inverted is constant and can be precomputed. See Figure 7.

Probably the most common strain model in computer graphics, as mentioned in Section 2.2.2, is the co-rotated model where we compute the polar decomposition $\mathbf{F} = \mathbf{Q}\tilde{\mathbf{F}}$ and then compute strain as

$$\epsilon = \frac{1}{2}\left(\tilde{\mathbf{F}} + \tilde{\mathbf{F}}^T\right) - \mathbf{I}. \tag{145}$$

We can then compute stress as

$$\boldsymbol{\sigma} = \lambda \mathrm{Tr}\left(\epsilon\right)\mathbf{I} + 2\mu\epsilon. \tag{146}$$

And finally forces are given by

$$\mathbf{f} = \mathbf{Q}\boldsymbol{\sigma}\mathbf{n}_i, \tag{147}$$

where $\mathbf{n}_i$ is the area weighted normal of the edge (2D) or face (3D) opposite the node the force is acting upon in the rest configuration. The simplicity of the finite element approach can be seen in Algorithm 2[5], which differs little from Algorithm 1.

[5]There are a couple of obvious optimizations: inverse(Matrix3x3(u1-u0, u2-u0, u3-u0)) in line 6 can be precomputed and line 8 can be simplified because Ftilde is symmetric.

Algorithm 2 Finite Element Timestep Loop

```
for Particle p : particles do
    p.frc = 0
    p.frc += p.mass*gravity
end for
for Element e : elements do
    Matrix3x3 F = Matrix3x3 (x1-x0, x2-x0, x3-x0) * inverse(Matrix3x3(u1-u0, u2-u0, u3-u0))
    PolarDecomp (F, Q, Ftilde)
    Matrix3x3 strain = 1/2 * (Ftilde + transpose (Ftilde)) - I
    Matrix3x3 stress = lambda * trace(strain) * I + 2 * mu * strain
    for i = 0 to 3 do
        particles[e.node[i]].frc += Q * stress * e.normal[i]
    end for
end for
for Particle p : particles do
    p.vel += dt*(p.frc / p.mass)
    p.pos += dt*(p.vel)
end for
```

4 TEMPORAL DISCRETIZATION

In addition to discretizing space, we must also discretize time. For the purposes of animation we typically march forward through time outputing frames at some regular frequency. Though the timestep can vary in size, a fixed timestep is often adopted for simplicity and to avoid unwanted changes in material behavior, which may occur with changes in timestep size. Typically there are several timesteps per frame of animation, but most modern animation systems try to take as few steps per frame as possible.

4.1 Explicit Integration

Explicit integration refers to integration techniques where the updated state at time $t + \Delta t$ is described solely in terms of quantities computed at time t. The Euler integrator discussed in Section 1,

$$\mathbf{x}_p(t + \Delta t) = \mathbf{x}_p(t) + \Delta t \cdot \mathbf{v}(\mathbf{x}_p, t). \tag{148}$$

is an explicit integrator. We have an explicit formula for $\mathbf{x}_p(t + \Delta t)$ in terms of quantities evaluated at time t, $\mathbf{x}_p(t)$ and $\mathbf{v}(\mathbf{x}_p, t)$. There are infinitely many such integration techniques all with different properties. In studying such integrators it is important to remember that properties that are important for particular applications, may not be important for computer animation and that integration techniques that may seem very similar can have very different behavior in practice.

4.1.1 Trapezoidal Rule vs. Midpoint Method. Let's take a closer look at the particular cas of the Trapezoidal Rule and the Midpoint Method. You may remember these from your first calculus class where you learned to integrate the area under a curve by computing the area of various rectagles. See Figure 8. It turns out that these methods can also be applied to our problem of moving massless particles through a velocity field from Section 1.1.

For the trapezoidal rule, we evaluate the velocity at the particle's position, pretend to move the particle a full timestep and evaluate the velocity again, then use the average of these two

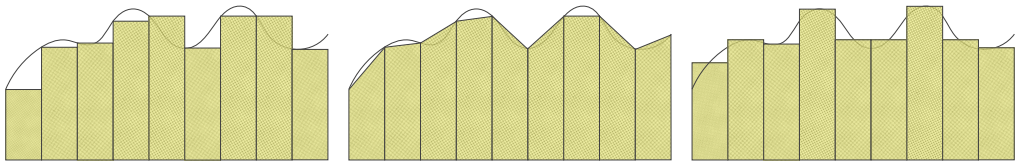


Fig. 8. *From left to right: Forward Euler, Trapazoidal Rule, and Midpoint Method.*

evaluations to update the particle's position, see Figure 9. Mathematically,

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \frac{\Delta t}{2}\left(\mathbf{v}(\mathbf{x}(t), t) + \mathbf{v}(\mathbf{x} + \Delta t \mathbf{v}(\mathbf{x}, t), t)\right). \tag{149}$$

For the midpoint method, we evaluate the velocity at the particle's position, pretend to move the particle a *half* step and evaluate the velocity again, then use the second evaluation to update the particle's position, see Figure 9. Mathematically,

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \Delta t\left(\mathbf{v}(\mathbf{x} + \frac{\Delta t}{2}\mathbf{v}(\mathbf{x}, t), t)\right). \tag{150}$$

These are both second order Runge-Kutta (RK2) methods and Taylor series analysis easily reveals that these methods both have local truncation errors of $O(\Delta t^3)$ and are second-order accurate. They are both frequently used in computer animation as the improvement in accuracy over the Euler integrator does seem to be worth the cost of the additional evaluation in many contexts. But, they do behave quite differently. Because the trapezoidal rule averages to velocity estimates it tends to produce a smoother solution, while the midpoint method is more susceptible to noise or aliasing. In some cases the additional smoothing of the trapezoidal rule is desired, in other cases these solutions appear overly damped and the midpoint method is preferred.

The take away point is that there is more to a numerical method than its accuracy and that differences that can seem quite subtle during analysis can be significant in practice.

4.1.2 Symplectic Euler. We already saw the Symplectic Euler integrator in Section 1, but it is important enough to revisit. For a second-order ordinary differential equation, e.g. Newton's Second Law, there are three obvious numerical methods for updating the position:

$$\mathbf{x}_p(t + \Delta t) = \mathbf{x}_p(t) + dt \cdot \mathbf{v}_p(t) \tag{151}$$

$$\mathbf{x}_p(t + \Delta t) = \mathbf{x}_p(t) + \frac{dt}{2} \cdot \left(\mathbf{v}_p(t) + \mathbf{v}_p(t + \Delta t)\right) \tag{152}$$

$$\mathbf{x}_p(t + \Delta t) = \mathbf{x}_p(t) + dt \cdot \mathbf{v}_p(t + \Delta t). \tag{153}$$

The first one doesn't seem to have a name associated with it, but the middle one is sometimes called *Improved Euler* and the third one is called *Symplectic Euler*. Obviously the only difference is where the velocity is evaluated, at the beginning of the step, end of the step, or an average of both. From the discussion above, the reader may guess that improved Euler is second-order accurate and that is partially true. It turns out that depending on the problem improved Euler may not converge at all. In particular, for pure elasticity problems one can fairly easily show that improved Euler is *unconditionally unstable* for open-ended (i.e. infinite) time integration. That means no matter

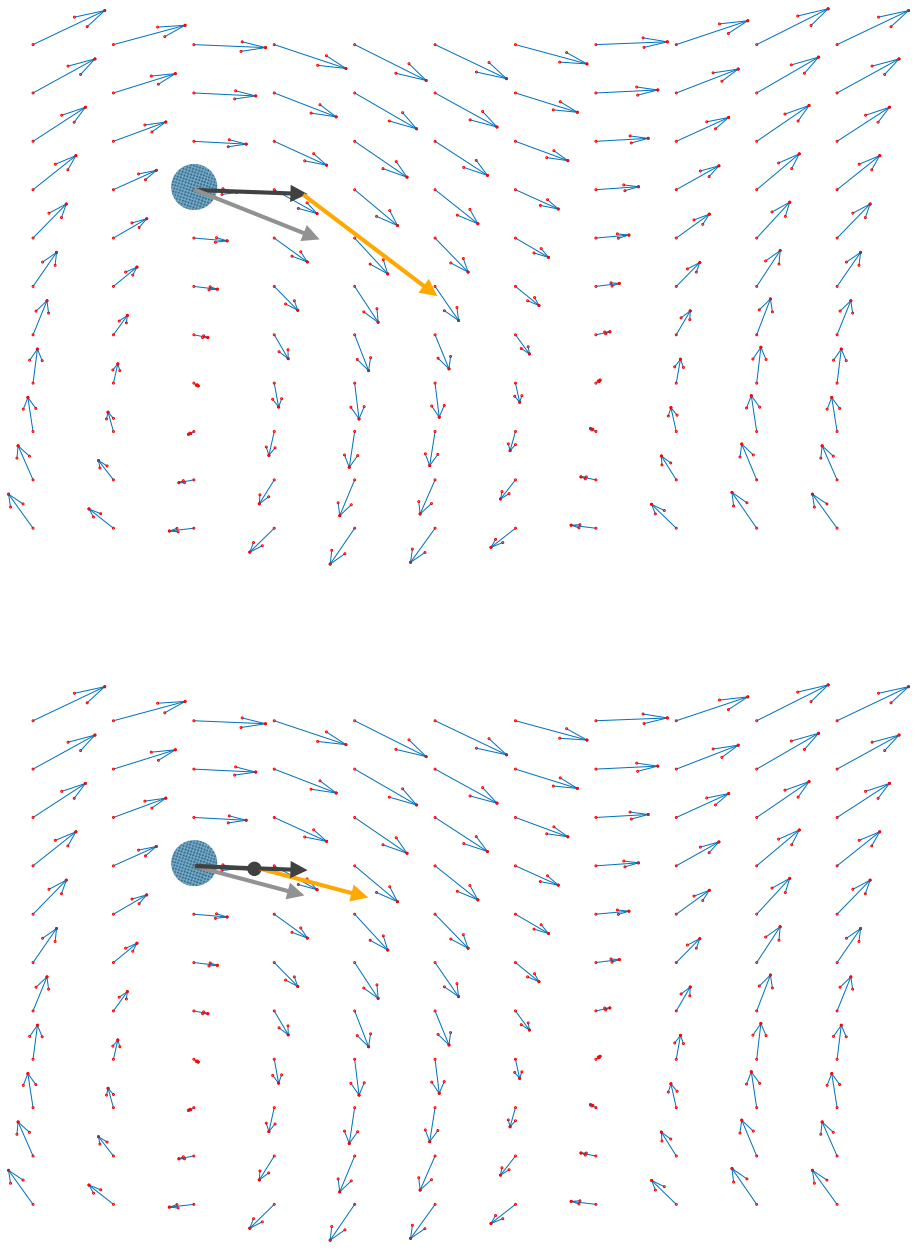


Fig. 9. *Trapazoidal Rule (top) vs. Midpoint Method (bottom) for integrating a path through a velocity field. The black arrow is the first velocity evaluation, the orange arrow is the second, and the gray arrow is the velocity used to upate the particles position. Note that the midpoint mehtod moves only halfway along the path from the first estimate and that the trapazoidal rule uses the average of the two velocity evaluations to update the position.*

how small the timestep the solution will still diverge. In practice, for most problems in computer animation symplectic Euler is the preferred explicit integrator, despite its lower order accuracy. Again, this demonstrates that there is more to chosing a numerical method than simple order of accuracy and numerical experiments are often necessary to determine the best choice.

4.2 Implicit Integration

Sometimes we wish to solve *stiff* problems. "Stiff problem" is not particularly well defined, but they occur, for example, when materials have very strong resistance to deformation.As problems become more stiff, explicit integrators require smaller and smaller timesteps in order to remain stable. In the simple case of a zero-length spring where

$$\mathbf{f} = -k\mathbf{x} \tag{154}$$

with initial displacement $\mathbf{x_0}$, initial velocity of 0, and mass m. If we use our symplectic Euler integrator we after the first step our new position will be

$$\mathbf{x}(\Delta t) = \left(1 - \frac{\Delta t^2 k}{m}\right)\mathbf{x_0}. \tag{155}$$

If $\Delta t > \sqrt{\frac{m}{k}}$ we will overshoot the origin. If $\Delta t > \sqrt{\frac{2m}{k}}$ the spring will be more extended then when it started and our solution will diverge. So, as k increases the size of our timestep must decrease to maintain stability. If we wish to avoid this timestep restriction, we can switch to *implicit integration.*

Here is our explicit symplectic Euler integrator again:

$$\begin{aligned}\mathbf{v}(t+\Delta t) &= \mathbf{v}(t) + \Delta t \cdot \mathbf{M}^{-1}\mathbf{f}(\mathbf{x}(t), t)\\ \mathbf{x}(t+\Delta t) &= \mathbf{x}(t) + \Delta t \cdot \mathbf{v}(t+\Delta t).\end{aligned} \tag{156}$$

and here is the implicit formulation:

$$\begin{aligned}\mathbf{v}(t+\Delta t) &= \mathbf{v}(t) + \Delta t \cdot \mathbf{M}^{-1}\mathbf{f}(\mathbf{x}(t+\Delta t), t+\Delta t)\\ \mathbf{x}(t+\Delta t) &= \mathbf{x}(t) + \Delta t \cdot \mathbf{v}(t+\Delta t).\end{aligned} \tag{157}$$

The only difference is that in the implicit version, forces are evaluated at $t + \Delta t$. But, because we can't directly compute the forces at $t + \Delta t$ without knowing the system state at $t + \Delta t$ the new state is defined *implicitly* as one which satisfies this equation. This definition is analogous to the definition of implicit surfaces being the set of points that satisfy an equation (e.g $x^2 + y^2 = r^2$ defines a circle). Generally, this is a complex non-linear problem and Newton's method is often employed in conjunction with a variety of performance enhancing approaches. However, we will consider the simplest version of the problem, which can be thought of as a single Newton iteration, where we linearize the system around the current state and solve the linearized problem. We take our non-linear dynamic system for soft bodies,

$$\mathbf{K}(\mathbf{x} - \mathbf{x}_0) + \mathbf{D}(\dot{\mathbf{x}}) + \mathbf{M}\ddot{\mathbf{x}} = \mathbf{f}_{ext}, \tag{158}$$

where $\mathbf{x}_0$ is the rest state. We then linearize to get

$$\mathbf{K}\mathbf{x} - \mathbf{K}\mathbf{x}_0 + \mathbf{D}\dot{\mathbf{x}} + \mathbf{M}\ddot{\mathbf{x}} = \mathbf{f}_{ext}. \tag{159}$$

Thus our forces are

$$\mathbf{f}_{ext} - \mathbf{K}\mathbf{x} + \mathbf{K}\mathbf{x}_0 - \mathbf{D}\dot{\mathbf{x}}. \tag{160}$$

Substituting these forces and the second equation of Equation (157) into the first equation of Equation (157) we have,

$$\mathbf{v}(t+\Delta t) = \mathbf{v}(t) + \Delta t \cdot \mathbf{M}^{-1}\left[-\mathbf{K}\left(\mathbf{x}(t) + \Delta t\mathbf{v}(t+\Delta t)\right) + \mathbf{K}\mathbf{x}_0 - \mathbf{D}\mathbf{v}(t+\Delta t) + \mathbf{f}_{ext}\right]. \tag{161}$$

Rearranging terms we arrive at the linear system

$$\left(\mathbf{M} + \Delta t^2 \mathbf{K} + \Delta t \mathbf{D}\right) \mathbf{v}(t + \Delta t) = \mathbf{M}\mathbf{v}(t) + \Delta t \left(-\mathbf{K}\left(\mathbf{x}(t) - \mathbf{x}_0\right) + \mathbf{f}_{ext}\right). \tag{162}$$

This integrator is often referred to as *Linearly Implicit Euler* and can also be derived from a first-order Taylor expansion. It is worth noting that the right hand side corresponds to an explicit Euler step that ignores damping forces. The matrix on the left hand side acts as a smoothing filter on this explicit Euler step. Traditionally, this system is solved with preconditioned conjugate gradients to take advantage of the sparse structure of the matrix on the left hand side. This integrator is also the first step in a Newton's method solution of a fully implicit backwards Euler integrator for the non-linear problem. As such it is sometimes referred to as semi-implicit Euler or semi-implicit Backward Euler.

5 CONSTRAINTS

Newton's second law describes the evolution of a body under applied forces. However, in many scenarios, it is more natural and convenient to specify conditions on the positions or velocities of the body that should be satisfied, rather than the forces acting on the body. These conditions are referred to as *constraints*. For example, an object resting on a table is subject to gravity and possibly other external forces, while constrained to not penetrate the table. How is this constraint maintained physically? By Newton's third law, the table pushes on the object with a force equal in magnitude and opposite in direction to the force of the object on the table. The net effect on the object will be to prevent penetration. The force associated with the non-penetration constraint will be exactly the force necessary to maintain the constraint. In this scenario and others, the *constraint force* arises instantaneously in response to other forces in the system to maintain the constraint. It is thus more natural to specify the constraint and determine the necessary force than to specify the constraint force. There are a many types of constraints as well as a variety of methods for solving problems of constrained dynamics.

5.1 Bilateral / Unilateral constraints

The number of *degrees of freedom* (DOF) in a system is the number of independent parameters needed to fully specify the configuration of the system. For example, in three-dimensional space, a single particle has three degrees of freedom, and a single rigid body has six degrees of freedom. A *holonomic* constraint is a relationship between the positional degrees of freedom in the system of the form

$$g(x_1, x_2, \ldots, x_n, t) = 0. \tag{163}$$

Note that it may depend on time. The constraint defines a relationship between the positional degrees of freedom that must be satisfied, therefore it implicitly removes degrees of freedom, since given $n - 1$ of the n positions, the remaining position can be determined using Eq. (163).

Implicitly, Eq. (163) defines a manifold embedded in the n-dimensional space. The constraint in (163) is a *bilateral* or *equality* constraint, as it requires that the positions be on the manifold and prohibits them from being off the manifold to either side.

As an example of such a constraint, consider two rigid bodies in two dimensions that are attached at their ends, but free to rotate about the point of attachment, as illustrated below. The degrees of

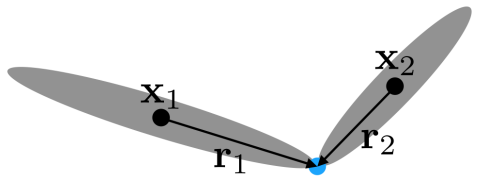


Fig. 10

freedom of the rigid bodies ignoring the constraint are $\mathbf{x}_1, \theta_1$, and $\mathbf{x}_2, \theta_2$, and the constraint can be written as

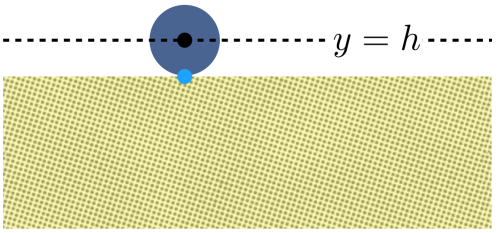
$$\mathbf{g}(\mathbf{x}_1, \theta_1, \mathbf{x}_2, \theta_2) = (\mathbf{x}_1 + \mathbf{r}_1) - (\mathbf{x}_2 + \mathbf{r}_2) = \mathbf{0}. \tag{164}$$

Here $\mathbf{g} : \mathbb{R}^6 \longrightarrow \mathbb{R}^2$ and implicitly removes two DOF, so that the constrained problem has $6 - 2 = 4$ degrees of freedom. We can also see this as the system configuration is fully determined by specifying the position of the attachment point in $\mathbb{R}^2$, and the angle of each rigid body, giving a total of 4 DOF.

In a *unilateral* or *inequality* constraint, the equality sign in (163) is replaced by an inequality, so that the constraint takes the form

$$g(x_1, x_2, \ldots, x_n, t) \geq 0. \tag{165}$$

This form of constraint arises when one body is in contact with another. In such cases, the bodies are free to move apart, but are prohibited from interpenetrating. An example is illustrated below, where a ball rests on a surface. The ball is constrained from moving into the surface, but can be freely lifted off the surface. The point of contact is shown in light blue. The constraint can be



written as

$$\mathbf{g}(x_1, y_1, \theta_1) = y_1 - h \geq 0, \tag{166}$$

where y_1 is the height of the center of the ball. This constraint allows the ball to slide, so that its lateral position x_1 and orientation θ_1 are independent. If the ball is further restricted to rolling, then x_1 and θ_1 would not evolve independently during contact.

Inequality constraints arise in other cases as well, for example when limits are set on the legal range of joint angles in an articulated rigid body.

5.2 Soft vs. Hard

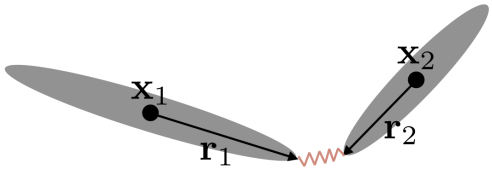
Constraints can also be said to be either hard or soft. A *hard constraint* is one which should be satsified exactly. Thus the forces maintaining it should be able to overcome any competing forces that act to violate the constraint. In constrast, a *soft constraint* need not be satisfied exactly; some amount of error is tolerated. The forces maintaining the soft constraint are not arbitrarily strong as in the case of hard constraints, and competing forces can cause the constraint to be violated until those forces subside.

5.3 Solution Methods: Penalty Forces, Lagrange Multipliers, Generalized Coordinates

There are several approaches to simulation of constrained dynamics problems. Most broadly, they fall into two categories: approaches which use *maximal coordinates*, the coordinates of the bodies disregarding the constraints, and impose auxiliary conditions defining the constraints, and approaches which use *generalized coordinates* (sometimes referred to as reduced coordinates, perhaps because they are reduced in number as compared with maximal coordinates), which are parameters representing directly the degrees of freedom after accounting for the constraints, and from which the maximal coordinates can then be deduced. Penalty methods and the method of

Lagrange multipliers are examples of approaches that use maximal coordinates. We discuss these first.

Penalty methods. Penalty methods attempt to enforce constraints by penalizing constraint violation using spring-like restoring forces. As an example, consider the scenario illustrated in Figure 10 where the ends of two rigid bodies are attached. As a simple illustration of a penalty method, we add a zero restlength spring that connects the two ends of the rigid bodies, as illustrated below. The first rigid body then feels a force at its attachment point equal to



$$\mathbf{f}_{21}(\mathbf{x}_1, \theta_1, \mathbf{x}_2, \theta_2) = -k\left((\mathbf{x}_2 + \mathbf{r}_2(\theta_2)) - (\mathbf{x}_1 + \mathbf{r}_1(\theta_1))\right), \tag{167}$$

while the second rigid body feels a force at its attachment point equal to

$$\mathbf{f}_{12}(\mathbf{x}_1, \theta_1, \mathbf{x}_2, \theta_2) = -\mathbf{f}_{21}(\mathbf{x}_1, \theta_1, \mathbf{x}_2, \theta_2). \tag{168}$$

More generally, penalty forces may be formulated based on an energy associated with the constraint violation as in [Faure et al. 2008] and may include damping terms that depend on derivatives of the constraint [Bender et al. 2014].

While penalty terms for constraint violation are relatively simple to implement, they do suffer from several drawbacks [Drumwright 2008; Bender et al. 2014]. As with the parameter k in our example above, penalty force terms contain one or more strength parameters. Tuning stiffness parameters in complex systems with many competing forces can be challenging. When k is relatively large, the constraint will be better maintained. However, this introduces a stiff force into the system which will incur a severe time step restriction or necessitate implicit integration. Penalty forces can also lead to undesirable oscillations and make scenes with resting contact of many bodies difficult to achieve [Bender et al. 2014]. Nevertheless, penalty methods are still widely used, and researchers continue to develop approaches to mitigate their drawbacks [Harmon et al. 2009; Tang et al. 2012; Xu et al. 2014].

Lagrange multipliers. The method of Lagrange multipliers is an approach that explicitly includes in the equations of motion the *constraint forces* necessary to maintain the constraints. Unlike a penalty force, the constraint force is exactly that force needed to counteract any other forces violating the constraint. Although mathematically this is sufficient to maintain a constraint that is satisfied initially, in practice additional *stabilization* techniques are required to counteract drift in the constraint satisfaction.

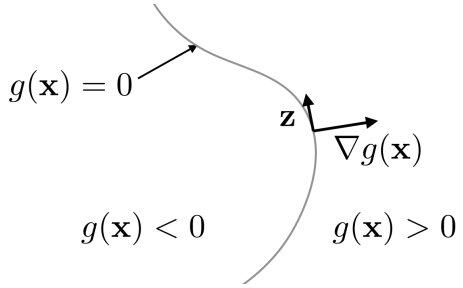
Consider the constraint equation

$$g(\mathbf{x}) = 0, \tag{169}$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. This is an implicit surface equation defining all the legal positions $\mathbf{x}$. The gradient of the constraint is the vector

$$\nabla g(\mathbf{x}) = \begin{pmatrix} \frac{\partial g}{\partial x_1}(\mathbf{x}) \\ \frac{\partial g}{\partial x_2}(\mathbf{x}) \\ \vdots \\ \frac{\partial g}{\partial x_n}(\mathbf{x}) \end{pmatrix}. \tag{170}$$

As illustrated in the figure above, the gradient vector is perpendicular to the surface. A vector $\mathbf{z}$



which is tangent to the constraint surface at a point $\mathbf{x}$ will satisfy $\mathbf{z}^T \nabla g(\mathbf{x}) = 0$.

Consider an arbitrary force $\mathbf{F}$ applied to a constrained particle. If $\mathbf{F}$ attempts to move the particle off the constraint surface, a counteracting constraint force $\mathbf{F}_c$ will arise. With the notable exception of frictional contact forces, the constraint forces we consider act only in the direction $\nabla g(\mathbf{x})$, and are therefore *workless* [Lanczos 2012]. This is expressed by considering a *virtual* displacement δr that is consistent with the constraints, as

$$\mathbf{F}_c^T \delta \mathbf{r} = 0. \tag{171}$$

Displacements consistent with the constraint are those such that $\delta \mathbf{r} = \alpha \mathbf{z}$, so that $\delta \mathbf{r}^T \nabla g = 0$. Therefore, constraint forces that satisfy (171) are exactly those forces of the form

$$\mathbf{F}_c = \lambda \nabla g \tag{172}$$

for some scalar λ. The scalar λ is known as the Lagrange multiplier for the constraint, and determines the strength of the constraint force.

More generally, we consider m constraint equations,

$$\mathbf{g}(\mathbf{x}) = \begin{pmatrix} g_1(\mathbf{x}) \\ g_2(\mathbf{x}) \\ \vdots \\ g_m(\mathbf{x}) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \mathbf{0}. \tag{173}$$

The *Jacobian* of the constraints is the $m \times n$ matrix

$$J(\mathbf{x}) = \frac{\partial \mathbf{g}}{\partial \mathbf{x}}(\mathbf{x}) = \begin{pmatrix} \frac{\partial g_1}{\partial x_1}(\mathbf{x}) & \frac{\partial g_1}{\partial x_2}(\mathbf{x}) & \ldots & \frac{\partial g_1}{\partial x_n}(\mathbf{x}) \\ \frac{\partial g_2}{\partial x_1}(\mathbf{x}) & \frac{\partial g_2}{\partial x_2}(\mathbf{x}) & \ldots & \frac{\partial g_2}{\partial x_n}(\mathbf{x}) \\ \vdots & & & \\ \frac{\partial g_m}{\partial x_1}(\mathbf{x}) & \frac{\partial g_m}{\partial x_2}(\mathbf{x}) & \ldots & \frac{\partial g_m}{\partial x_n}(\mathbf{x}) \end{pmatrix} = \begin{pmatrix} \nabla g_1(\mathbf{x})^T \\ \nabla g_2(\mathbf{x})^T \\ \vdots \\ \nabla g_m(\mathbf{x})^T \end{pmatrix}. \tag{174}$$

In this case, the total constraint forces is comprised of m terms, one for each constraint, $\lambda_1 \nabla g_1 + \lambda_2 \nabla g_2 + \ldots + \lambda_m \nabla g_m$, which we write

$$\mathbf{F}_c = J^T \boldsymbol{\lambda}, \tag{175}$$

for $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_m)$.

Given this form for the constraint forces, the equations of motion, including applied forces and constraint forces, are written as

$$M\mathbf{a} = \mathbf{F} + \mathbf{F}_c = \mathbf{F} + J^T \boldsymbol{\lambda}. \tag{176}$$

We follow a derivation similar to [Baraff 1996; Witkin 2001]. The unknown $\boldsymbol{\lambda}$ can be determined by requiring the acceleration $\mathbf{a}$ of the system to be compatible with the constraint. To get this condition, we differentiate the constraint equation with respect to time to get

$$\begin{aligned} 0 &= \dot{\mathbf{g}}(\mathbf{x}) && (177)\\ &= \frac{\partial \mathbf{g}}{\partial \mathbf{x}}^T \dot{\mathbf{x}}(t) && (178)\\ &= J(\mathbf{x})\mathbf{v}(t), && (179) \end{aligned}$$

This shows us that the legal velocities are those that are orthogonal to the constraint gradient, parallel to $\mathbf{z}$ in the figure above.

To get an expression involving the accelerations, we differentiate once more with respect to time to get

$$\begin{aligned} 0 &= \ddot{\mathbf{g}}(\mathbf{x}) && (180)\\ &= \dot{J}\mathbf{v}(t) + J\dot{\mathbf{v}}(t) && (181)\\ &= \dot{J}\mathbf{v}(t) + J\mathbf{a}(t) && (182) \end{aligned}$$

This is the condition that must be satisfied by the accelerations to maintain the position and velocity level constraints. Plugging in $\mathbf{a}$ from Eq. (176) into (182), we get

$$JM^{-1}\left(\mathbf{F} + J^T\boldsymbol{\lambda}\right) = -\dot{J}\mathbf{v}(t). \tag{183}$$

Isolating the terms involving $\boldsymbol{\lambda}$ on the left hand side, we get

$$(JM^{-1}J^T)\boldsymbol{\lambda} = -JM^{-1}\mathbf{F} - \dot{J}\mathbf{v}(t). \tag{184}$$

In practice constraints often couple only a few degrees of freedom in the system. Therefore, the factors J and M in (184) are typically sparse. However, the product $JM^{-1}J^T$ is not generally sparse. Baraff [Baraff 1996] reformulated Eq. (184) into the sparse KKT system

$$\begin{pmatrix} M & -J^T \\ -J & 0 \end{pmatrix} \begin{pmatrix} \mathbf{y} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ -\mathbf{b} \end{pmatrix}, \tag{185}$$

where $\mathbf{y} = \mathbf{a} - M^{-1}\mathbf{F}$, and $\mathbf{b} = -JM^{-1}\mathbf{F} - \dot{J}\mathbf{v}(t)$ is the left-hand side of (184), and described a linear-time algorithm for its solution which is applicable in many practical cases. KKT systems appear regularly in solution methods for a variety of constrained dynamics problems.

Generalized coordinates. So far we have treated constraints in maximal coordinates and used Newton's second law to define the equations of motion. Another approach to constrained dynamics is based on finding a minimal set of independent coordinates, or *generalized coordinates*,

$$q_1, q_2, \ldots, q_N,$$

and expressing the equations of motion in terms of these coordinates and their derivatives. The formulation of the equations of motion in terms of generalized coordinates is part of the subject of variational mechanics, developed after Newton's time by Euler, Lagrange, Hamilton, and others [Lanczos 2012]. The choice of q_i is not unique, but in some cases there may be a particular choice which is natural and convenient.

The best-known example of generalized coordinates are those for a rigid body, which can be viewed as a collection of particles where each pair of particles are constrained to be a fixed distance apart. In three dimensions, a minimal set of six degrees of freedom, three for position and three for orientation, is used to describe a rigid body. The maximal coordinates of any point on the body can be calculated from the generalized coordinates using the well known formula in Section 2.2.1.

The n ($n > N$) maximal coordinates are given as functions of the generalized coordinates,

$$x_1 = x_1(q_1, q_2, \dots, q_N) \tag{186}$$
$$x_2 = x_2(q_1, q_2, \dots, q_N) \tag{187}$$
$$\vdots \tag{188}$$
$$x_n = x_n(q_1, q_2, \dots, q_N). \tag{189}$$

Concatening all the maximal and generalized coordinates, we denote this $\mathbf{x} = \mathbf{x}(\mathbf{q})$. In general, Equations (186)-(189) may also depend on time, but for simplicity we do not consider that case here. See [Murray et al. 1986; Lanczos 2012] for further details.

These expressions can be used to define the kinetic energy and potential energy of the system in terms of the $\mathbf{q}$ and $\dot{\mathbf{q}}$, the generalized velocity. The velocity of the system is

$$\dot{\mathbf{x}}(t) = \frac{\partial \mathbf{x}}{\partial \mathbf{q}} \dot{\mathbf{q}}. \tag{190}$$

Let $J(\mathbf{q}) = \frac{\partial \mathbf{x}}{\partial \mathbf{q}}$ be the Jacobian of the coordinate functions. The kinetic energy is given by

$$T = \frac{1}{2} \dot{\mathbf{x}}^T M \dot{\mathbf{x}} = \frac{1}{2} (J\dot{\mathbf{q}})^T M J \dot{\mathbf{q}} = \frac{1}{2} \dot{\mathbf{q}}^T J^T M J \dot{\mathbf{q}}. \tag{191}$$

Analogous to the force in the system of maximal coordinates, we can define *generalized forces* for the system of generalized coordinates. For example, for a rigid body the generalized forces are the six components of the force and torque and on the body. As described in detail in [Lanczos 2012], two sets of forces for two sets of coordinates will induce equivalent dynamics if they do the same virtual work on a virtual displacement consistent with the constraints. Therefore, the generalized forces for any system can be easily found by equating the virtual work done by both sets of forces on virtual displacements consistent with the constraints as

$$\mathbf{F} \cdot \delta \mathbf{x} = \mathbf{G} \cdot \delta \mathbf{q}, \tag{192}$$

where we have used $\mathbf{G} \in \mathbb{R}^N$ to represent the generalized forces. The displacements $\delta \mathbf{x}$ are required to be consistent with the constraints, so $\delta \mathbf{x} = J \delta \mathbf{q}$. Using this, we get

$$\mathbf{F}^T \delta \mathbf{x} = \mathbf{G}^T \delta \mathbf{q} \tag{193}$$
$$\mathbf{F}^T J \delta \mathbf{q} = \mathbf{G}^T \delta \mathbf{q} \tag{194}$$

The coordinates $\delta \mathbf{q}$ are unconstrained and so are arbitrary. This means we must have

$$\mathbf{F}^T J = \mathbf{G}^T, \text{ or } \mathbf{G} = J^T \mathbf{F}. \tag{195}$$

We can verify this result in the case of a rigid body, where the velocity of a particle is

$$\dot{\mathbf{x}} = \begin{pmatrix} I & \mathbf{r}^{*T} \end{pmatrix} \begin{pmatrix} \mathbf{v} \\ \boldsymbol{\omega} \end{pmatrix}, \tag{196}$$

so $J = \begin{pmatrix} I & \mathbf{r}^{*T} \end{pmatrix}$. A force $\mathbf{F}$ applied to the particle results in a generalized force on the body

$$\mathbf{G} = J^T \mathbf{F} = \begin{pmatrix} I \\ \mathbf{r}^* \end{pmatrix} \mathbf{F} = \begin{pmatrix} \mathbf{F} \\ \mathbf{r} \times \mathbf{F} \end{pmatrix}, \tag{197}$$

which are the familiar expressions for force and torque.

The Lagrange equations of motion, analogous to Newton's second law, can be written as

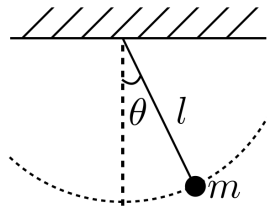
$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\mathbf{q}}}\right) = \mathbf{G}, \tag{198}$$

where G are the generalized forces. For conservative forces, we can define the Lagrangian, $L = T - V$, where V is the potential energy associated with the forces, and express the above as the *Euler-Lagrange* equations

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\mathbf{q}}}\right) - \frac{\partial L}{\partial \mathbf{q}} = \mathbf{0}. \tag{199}$$

Additional generalized forces accounting for non-conservative forces or constraint forces not respected by the generalized coordinates can be included on the right hand side of the Euler-Lagrange equations.

As an example we consider the pendulum in two dimensions illustrated in the figure below. A



single particle of mass m with maximal coordinates $\mathbf{x} = (x, y)$ is suspended at a length l. Although $\mathbf{x}$ has two coordinates, the system has only one degree of freedom due to the constraint imposed by the pendulum. We use the angle of pendulum with the vertical, θ, as the generalized coordinate. With the origin at the attachment point, the maximal coordinates are

$$x(\theta) = l \sin\theta \tag{200}$$

$$y(\theta) = -l \cos\theta \tag{201}$$

The velocity is

$$\begin{pmatrix} \dot{x}(\theta) \\ \dot{y}(\theta) \end{pmatrix} = \begin{pmatrix} l\cos\theta\dot{\theta} \\ l\sin\theta\dot{\theta} \end{pmatrix} = J\dot{\theta}. \tag{202}$$

The kinetic energy is

$$T = \frac{1}{2} m(\dot{x}^2 + \dot{y}^2) = \frac{1}{2} m l^2 \dot{\theta}^2.$$

We have

$$\frac{\partial T}{\partial \dot{\theta}} = m l^2 \dot{\theta},$$

and thus

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\theta}}\right) = ml^2\ddot{\theta}. \tag{203}$$

The only force we need to explicitly account for is gravity, given in maximal coordinates as $(0, -mg)^T$. The generalized force is therefore

$$J^T \begin{pmatrix} 0 \\ -mg \end{pmatrix} = \begin{pmatrix} l\cos\theta & l\sin\theta \end{pmatrix} \begin{pmatrix} 0 \\ -mg \end{pmatrix} = -lmg\sin\theta. \tag{204}$$

Putting together Equations (203) and (204), we get

$$ml^2\ddot{\theta} = -lmg\sin\theta,$$

or the well-known equations of motion for a pendulum

$$\ddot{\theta} + \frac{g}{l}\sin\theta = 0.$$

5.4 Practical Constrained Rigid Body Systems

Practical rigid body simulation systems must deal with a variety of constraints. These include articulation constraints, or joints (e.g., ball and socket, revolute, prismatic), collisions, and resting and sliding contact. For ragdoll physics, where the connectivity prescribed by the articulation constraints is that of a tree (no closed loops), generalized coordinates are often efficiently employed [Bender et al. 2014], using e.g., Featherstone's algorithm [Featherstone 1983]. For other types of constraints, constraint forces are widely used. In these cases, several modifications are made to the formulation of Lagrange multipliers presented in Section 5.3.

In Section 5.3, we took the second time derivative of the constraint equation, which gave us an acceleration-level condition for constraint satisfaction. We then derived the linear system for the Lagrange multipliers by plugging in the acceleration from Newton's second law into this condition and solving for $\boldsymbol{\lambda}$. This effectively determines the forces needed to maintain a legal acceleration. Assuming that the position and velocity are already legal, legal accelerations should result in maintenance of the position and velocity level constraints over the time. In practice, there are issues with acceleration-level constraints that make them unsuitable as a general-purpose approach in rigid body systems. Newton's second law describes a continuous evolution of the velocity. However, collision of rigid bodies results in a *discontinuous* change in velocity, as when a rigid block hits the ground and immediately bounces back up. Acceleration-level resolution of constraints has also been found to be problematic in modeling frictional contact scenarios, as when an object is pushed over a rough surface (as in the Painlevé paradox).

For these reasons, many systems model rigid body constraints on the velocity level, using the integrated Newton's equations that describe finite velocity changes [Mirtich 1996; Guendelman et al. 2003; Weinstein et al. 2006]. In the velocity-level formulation, *impulses* change the velocity of a body discontinuously. An impulse is defined as the integral of a force acting over time,

$$\mathbf{j} = \int_{t_1}^{t_2} \mathbf{f}\, dt. \tag{205}$$

It therefore has units of momentum, $\mathrm{N}\cdot\mathrm{s} = \mathrm{kg\ m/s}$. An impulse applied to a particle of mass m will result in a change of momentum of the particle. Integrating Newton's second law for a particle

with mass m with respect to time, we get the momentum-impulse equation

$$\int_{t_1}^{t_2} m\mathbf{a}\, dt = \int_{t_1}^{t_2} \mathbf{f}\, dt, \tag{206}$$

$$\implies m(\mathbf{v}(t_2) - \mathbf{v}(t_1)) = \mathbf{j}. \tag{207}$$

To reformulate the system (185) on the velocity level, we consider the discretized Newton's law

$$M\mathbf{V}^{n+1} = M\mathbf{V}^n + \Delta t\mathbf{F} + \Delta t J^T \boldsymbol{\lambda}^{n+1}. \tag{208}$$

The velocity constraint (179) is $J\mathbf{V}^{n+1} = \mathbf{0}$. Defining $\boldsymbol{\mu}^{n+1} = \Delta t\boldsymbol{\lambda}^{n+1}$, moving the unknowns $\mathbf{V}^{n+1}$ and $\boldsymbol{\mu}^{n+1}$ to the left hand side, and combining the two equations gives the linear system [Tournier et al. 2015]

$$\begin{pmatrix} M & -J^T \\ -J & 0 \end{pmatrix} \begin{pmatrix} \mathbf{V}^{n+1} \\ \boldsymbol{\mu}^{n+1} \end{pmatrix} = \begin{pmatrix} M\mathbf{V}^n + \Delta t\mathbf{F} \\ \mathbf{0} \end{pmatrix}. \tag{209}$$

This linear system is similar to Eq. (185), but has units of momentum instead of force. Notice also that instead of $\boldsymbol{\lambda}$, we have the impulse form $\boldsymbol{\mu} = \Delta t\boldsymbol{\lambda}$ as the unknown. If we eliminate $\mathbf{V}^{n+1}$ from the equations, we get the system

$$JM^{-1}J^T\boldsymbol{\mu}^{n+1} = -J\mathbf{V}^n - \Delta t JM^{-1}\mathbf{F}. \tag{210}$$

Equation (210) is an equation that couples the solution of the constraints impulses, which occurs because when a body's velocity is adjusted to satisfy a constraint, it may result in the violation of a different constraint. The constraints in the system must be taken into account simultaneously to give a solution that satisfies them all. In practice, many systems use an impulse-based approach [Mirtich 1996; Guendelman et al. 2003; Weinstein et al. 2006], computing an approximate solution to (210) using a limited number of Gauss-Seidel iterations, where each constraint is updated independently using the most recently computed state of the other constraints. Interactive applications may severely limit the number of allowed Gauss-Seidel iterations, and require techniques such as warm-starting the solution (with the solution from the previous time step) to achieve acceptable results [Catto 2014]. Gauss-Seidel approaches can be very slow to converge when a light object couples the interaction between two heavier objects, as the light object will not succeed in moving the heavy object very much during a single pairwise iteration. This *messenger problem* can be partially mitigated by solving some constraints simultaneously, a compromise between solving the full global system and local pairwise constraints [Shinar et al. 2008; Catto 2014].

Another issue which occurs in practice due to discretization and numerical errors is that of *drift*. In the acceleration-level formulation of Section 5.3, the position and velocity are assumed to satisfy the constraint initially, and only accelerations are required to be legal thereafter. Over time, this will cause velocities and positions to drift away from legal states, creating noticeable artifacts such as erroneous overlaps or gaps. The velocity-level formulation as presented so far in this section suffers from a similar problem because it does not explicitly correct any position errors in the constraints. To combat drift, *stabilization* techniques are employed [Bender et al. 2014]. A widely used approach is *Baumgarte stabilization* [Baumgarte 1972], where the acceleration constraint equation $\ddot{\mathbf{g}}(\mathbf{x}) = 0$ is replaced with [Bender et al. 2014]

$$\ddot{\mathbf{g}}(\mathbf{x}) + 2\alpha\dot{\mathbf{g}}(\mathbf{x}) + \beta^2\mathbf{g}(\mathbf{x}) = 0, \tag{211}$$

for some parameters α and β which must be tuned. For the velocity-level formulation, the velocity constraint equation $\dot{\mathbf{g}}(\mathbf{x}) = 0$ is replaced with

$$\dot{\mathbf{g}}(\mathbf{x}) + \gamma\mathbf{g}(\mathbf{x}) = 0, \tag{212}$$

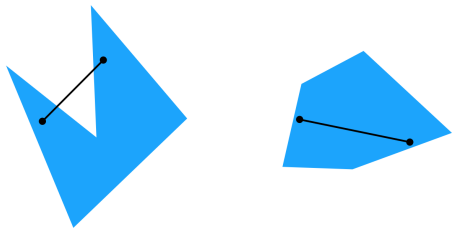


Fig. 11. *A convex polygon is one that fully contains the line segment joining any two points in the polygon. The polygon on the left is nonconvex, while the polygon on the right is convex.*

for some tunable parameter γ. When the position-level constraint $\mathbf{g}(\mathbf{x}) = 0$ is already satisfied, this will result in the same equation as before. If there is error in the position constraint, the computed velocities will drive the error to zero.

Many practical rigid body systems (e.g., Bullet [Coumans 2010], ODE [Smith et al. 2005], and Box2D [Catto 2011a]) add *compliance* to hard constraints by replacing $J\mathbf{v} = 0$ with

$$J\mathbf{v} + \frac{\beta}{\Delta t}\mathbf{g}(\mathbf{x}) + \gamma\boldsymbol{\lambda} = 0, \tag{213}$$

for constants β and γ [Catto 2011b]. The linear system (209) then becomes

$$\begin{pmatrix} M & -J^T \\ -J & \gamma I \end{pmatrix} \begin{pmatrix} \mathbf{V}^{n+1} \\ \boldsymbol{\mu}^{n+1} \end{pmatrix} = \begin{pmatrix} M\mathbf{V}^n + \Delta t\mathbf{F} \\ -\frac{\beta}{\Delta t}\mathbf{g}(\mathbf{X}^n) \end{pmatrix}. \tag{214}$$

This makes the constraint a bit softer, while countering drift. Furthermore, Eq. (214) is easier to solve numerically as the new diagonal block regularizes the equations, which may become poorly conditioned in the case of redundant or nearly redundant constraints. See also [Catto 2011b; Tournier et al. 2015; Lacoursiere 2007] for further details.

5.5 Collisions

Dynamic scenarios with many rigid or deformable solids will typically lead to many collisions and contacts between objects. Soft bodies also experience self-collision, where one part of the body deforms and collides with another part of the body. Cloth simulations in particular can undergo many simultaneous self-collisions that can be challenging to resolve. In all of these cases, a significant part of the computational cost of a solver is in detecting and resolving collisions and contacts.

Collision detection. Before collisions can be resolved, they must first be detected by checking for interpenetration of object geometry. Collision detection algorithms make use of a variety of geometric representations.

Polygonal geometry is commonly used to represent both rigid and deformable bodies. Many algorithms have been developed for finding intersections of polytopes, particularly convex polytopes (illustrated above). The Separating Axis Theorem says that if a separating hyperplane can be found between two convex shapes, they are disjoint and hence do not intersect. Faces of potentially colliding geometry are natural candidates for separating hyperplanes. *Convex decompositions* are often computed for nonconvex bodies for collision detection purposes. When using polygonal geometry, various intersection cases must be considered, such as point-face intersections and edge-edge intersections.

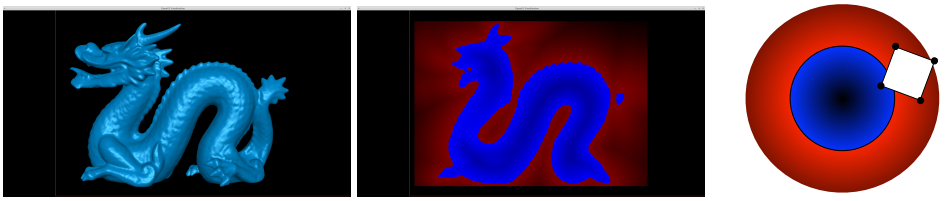


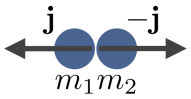
Fig. 12. *Left and middle: Three-dimensional dragon model and two-dimensional slice of its signed distance field. Right: Signed distance field for a circle. The vertices of a polygonal shape, illustrated in white, can be efficiently tested against the signed distance field to detect the interpenetration.*

Another geometric representation commonly used in collision detection is the *signed distance field*. The signed distance field $\phi(\mathbf{x})$ of an object assigns to each point in space $\mathbf{x}$ the signed distance from $\mathbf{x}$ to the nearest point on the surface of the object. Points inside the object will have $\phi < 0$ while points outside have $\phi > 0$, or the other way around depending on the convention used. The surface is then implicitly defined as the zero level set of ϕ, the points $\mathbf{x}$ such that $\phi(\mathbf{x}) = 0$. Typically ϕ will be sampled on a structured grid containing the object. Surface points $\mathbf{p}$ from other objects can query $\phi(\mathbf{p})$ to determine in constant time whether they are inside or outside the object, as illustrated in the figure below. Though inside/outside tests are fast, with many objects there can be significant memory overhead in storing the distance fields. Also, while ϕ can be computed once for rigid bodies in a preprocessing step, for deformable geometry it may need to be recomputed intermittently. Another advantage of a signed distance representation is that the surface normals needed for collision and contact resolution can be readily computed from the signed distance field since $\mathbf{n}(\mathbf{x}) = \nabla\phi(\mathbf{x})$.

For n objects, there are $O(n^2)$ pairs of objects. To make collision detection practical in simulations with many interacting objects, acceleration structures must be employed. A common approach is to create *bounding volumes* for the geometry, specifically volumes for which intersection tests can be made very fast. The bounding volumes are then used to quickly prune the set of possible intersections, since two objects do not intersect if their bounding volumes do not intersect. Spheres, axis-aliged bounding boxes, and oriented bounding boxes are commonly used. Furthermore, *hierarchical* bounding volumes can computed, potentially allowing for more efficient culling of a large number of candidate intersections. In addition to bounding volumes, *spatial partitions* can accelerate collision detection by indicating which objects are simultaneously present in a particular region of space. The must be updated throughout the simulation as the objects evolve.

In *discrete collision detection* we consider the geometric state of objects in the scene at a fixed instant in time, ignoring their trajectories. For large volumetric objects this is often sufficient. However, thin objects such as cloth, or very small objects could experience intersections during a time step that would not be evident when looking at a fixed instant of time at the beginning or end of the time step. For example, a small object thrown at a piece of cloth could be fully on one side of the cloth at the start of the time step and fully on the other side at the end of the time step. However, between the two instants of time the object passed through the cloth, so in principle a collision should be detected and resolved. *Continuous collision detection* takes into account the trajectory of motion over the time step when checking for intersections. Analytical methods for solving the intersection problem exist in some cases. Ray casting and numerical solvers are also used. This is harder and more computationally intensive than discrete collision detection and some systems may avoid continuous collision detection by limiting time steps.

Collision response (1D). In collision response, the solver resolves a previously detected collision, typically by computing and applying a pair of impulses to the colliding bodies so that their updated velocities are valid. Let us consider a simple example where two particles in one dimension with masses m_1, m_2 and velocities v_1, v_2 collide, as illustrated below. By Newton's third law, they



will exert equal and opposite forces on each other. The collision is instantaneous, so we find a pair of impulses $\pm j$ that effect the change in momenta of the particles. Let v_1' and v_2' be the velocities of the particles after the collision. We add impulses j and $-j$ to the system as

$$m_1 v_1' = m_1 v_1 + j \tag{215}$$

$$m_2 v_2' = m_2 v_2 - j, \tag{216}$$

and require that the collision be *inelastic*, or *sticking*, which means that after the collision the particles travel together, i.e.,

$$v_1' = v_2'. \tag{217}$$

We now have three equations and three unknowns. We solve for v_1' and v_2' in Equations (215) and (216) and plug the expressions into Eq. (217) to get

$$\frac{1}{m_1}(m_1 v_1 + j) = \frac{1}{m_2}(m_2 v_2 - j). \tag{218}$$

We can now solve for j:

$$\left(\frac{1}{m_1} + \frac{1}{m_2}\right) j = v_2 - v_1 \tag{219}$$

$$\implies j = \left(\frac{1}{m_1} + \frac{1}{m_2}\right)^{-1} (v_2 - v_1). \tag{220}$$

For example, if $v_1 = v$, $v_2 = -v$, and $m_1 = m_2 = m$, then the above tells us that $j = -mv$ and after the collision $v_1' = v_2' = 0$. If $v_1 = v$, $v_2 = -v$, and $m_1 = 3$, $m_2 = 1$, then after the collision $v_1' = v_2' = \frac{1}{2}v$, so that the particles continue to travel together in the direction that the more massive particle was traveling, with reduced speed.

It is also common to employ *elastic* collisions, especially in rigid body simulation. In that case, instead of being zero, the relative velocity of the particles after collision is given by

$$(v_2' - v_1') = -\epsilon(v_2 - v_1), \tag{221}$$

where $\epsilon \in [0, 1]$ is the *coefficient of restitution*. Note that $\epsilon = 0$ gives the inelastic case above. Substituting the expressions for v_1' and v_2' from Eqs. (215) and (216) as before, and solving for j we get

$$j = \left(\frac{1}{m_1} + \frac{1}{m_2}\right)^{-1} (1 + \epsilon)(v_2 - v_1). \tag{222}$$

For example, if $v_1 = v$, $v_2 = -v$, and $m_1 = m_2 = m$, then for a perfectly elastic collision with $\epsilon = 1$, $j = -2mv$, and $v_1' = -v$, $v_2' = v$.

Deformable object collisions. Physically, the bounce resulting from a collision occurs due to deformation of the colliding objects in some region around the collision. As the bodies collide, they compress, and elastic energy is stored. The release of the energy results in the rebound seen following the collision. Therefore, for deformable objects, one can use the inelastic collision equations described above. The one-dimensional case generalizes straightforwardly to two or three dimensions to give

$$m_1\mathbf{v}_1' = m_1\mathbf{v}_1 + \mathbf{j} \tag{223}$$

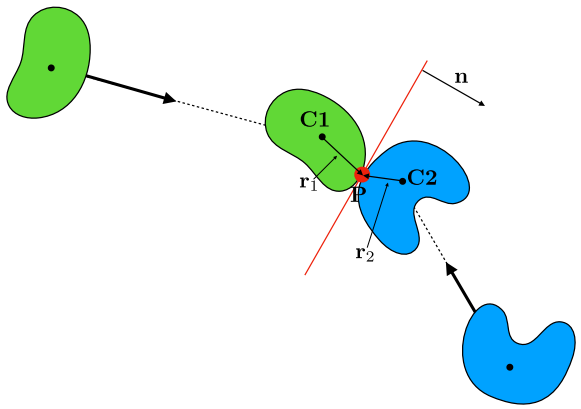
$$m_2\mathbf{v}_2' = m_2\mathbf{v}_2 - \mathbf{j} \tag{224}$$

$$\mathbf{v}_2' = \mathbf{v}_1', \tag{225}$$

for a total of six equations and unknowns in two dimensions, or nine equations and unknowns in three dimensions. The velocities are measured at the point of collision, and the impulses are applied at the point of collision. As the objects deform, there may be many points of collision.

Rigid body collisions. For rigid bodies the situation is somewhat different than for deformable bodies. Physically, such objects will also undergo deformation in a collision in the process described above for deformable bodies. But given the rigid body idealization which prohibits any deformation, a different approach is needed to model rigid body collisions. Typically, *collision laws* are used to relate quantities before and after the collision [Chatterjee and Ruina 1998]. These are derived similarly to the above derivations, but consider also the angular components of velocity and momentum. The treatment of friction in the tangential directions of motion significantly complicates rigid body collisions. We first consider the simpler cases of sticking and frictionless collisions.

Following [Ruina and Pratap 2009], we consider the collision of two rigid bodies with masses m_1, m_2, inertia tensors I_1, I_2, linear velocities $\mathbf{v}_1$, $\mathbf{v}_2$, and angular velocities $\boldsymbol{\omega}_1$, $\boldsymbol{\omega}_2$. The velocities after collision are denoted with a prime as before. The scenario is illustrated below. The figure identifies



a plane tangent to the bodies at the time of collision, with $\mathbf{n}$ a unit normal the plane.

We write the linear momenta of the bodies before and after the collision,

$$m_1\mathbf{v}_{C1}' = m_1\mathbf{v}_{C1} + \mathbf{j} \tag{226}$$

$$m_2\mathbf{v}_{C2}' = m_2\mathbf{v}_{C2} - \mathbf{j}, \tag{227}$$

and the angular momenta of the bodies before and and after collisions as

$$I_1 \boldsymbol{\omega}_1' = I_1 \boldsymbol{\omega}_1 + \mathbf{r}_1 \times \mathbf{j}, \tag{228}$$

$$I_2 \boldsymbol{\omega}_2' = I_2 \boldsymbol{\omega}_1 - \mathbf{r}_2 \times \mathbf{j}. \tag{229}$$

For a sticking collisions, the remaining equations state that the objects move at the same velocity at the point of contact immediately following the collision:

$$\mathbf{v}_{P1}' = \mathbf{v}_{P2}', \tag{230}$$

where

$$\mathbf{v}_{P1}' = \mathbf{v}_{C1}' + \boldsymbol{\omega}_1' \times \mathbf{r}_1, \tag{231}$$

$$\mathbf{v}_{P2}' = \mathbf{v}_{C2}' + \boldsymbol{\omega}_2' \times \mathbf{r}_2 \tag{232}$$

are the velocities of the first and second body, respectively, at the collision point. The number of equations/unknowns in (226)-(230) are eight in two dimensions and fifteen in three dimensions.

For a frictionless elastic collision, we assume

$$\mathbf{j} = j\mathbf{n}, \tag{233}$$

and replace Eq. (230) with a condition on the normal component of the relative velocities after collisions, analogous to Eq. (221):

$$(\mathbf{v}_{P2}' - \mathbf{v}_{P1}') \cdot \mathbf{n} = -\epsilon(\mathbf{v}_{P2} - \mathbf{v}_{P1}) \cdot \mathbf{n}, \tag{234}$$

where $\epsilon \in [0, 1]$. Given the linear and angular velocities of the bodies before the collision and the collision plane at collision, these equations can be solved to determine the normal impulse and the velocities of the bodies after the collision.

Frictional collisions. In frictional collisions, the impulse $\mathbf{j}$ consists of both a normal component and a tangential component that prevents or damps the relative tangential velocities of the bodies at the point of collision.

We first describe the *Coulomb friction* model, which is an empirical model describing the frictional forces that arise in sticking or sliding contact. According to the Coulomb friction model, the magnitude of the tangential force cannot exceed $\mu\|\mathbf{f}_n\|$, but may be less if the contact point is sticking. The direction of the tangential force is directly opposing the tangential direction of motion. The coefficient μ is called the coefficient of friction. When the contact is sliding, the tangential force is at its maximum possible value, and is equal to

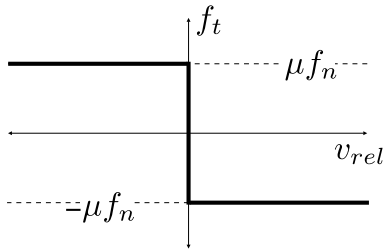
$$\mathbf{f}_t = -\mu\|\mathbf{f}_n\|\mathbf{t},$$

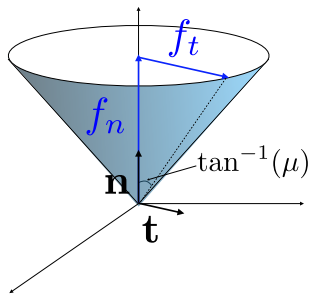
where $\mathbf{t}$ is the direction of motion. When the contact is sticking, the tangential force is just enough to counteract other tangential forces. This is summarized in this plot, adapted from [Ruina and Pratap 2009]: When the relative velocity at the contact point is positive, the frictional force is $-\mu f_n \mathbf{t}$. When the relative motion is negative, the frictional force is $\mu f_n \mathbf{t}$. When the relative velocity is zero, the frictional force is between these two values.

The restrictions on the forces or impulses in the Coulomb friction model can be illustrated by the *friction cone*, defined by the inequality

$$\|\mathbf{f}_t\| \leq \mu\|\mathbf{f}_n\|,$$

and shown here.





In [Guendelman et al. 2003], tangential frictional impulses were included in rigid body collisions by first assuming sticking in the tangential direction, as

$$(I - \mathbf{n}\mathbf{n}^T)(\mathbf{v}'_{P2} - \mathbf{v}'_{P1}) = \mathbf{0}, \tag{235}$$

and solving this along with Eqs. (226)-(229) and (234). If the solution $\mathbf{j}$ is in the friction cone, then the sticking assumption is admissible and the impulse $\mathbf{j}$ is used. If $\mathbf{j}$ is not in the friction cone, then sliding friction is applied. In practice, algebraic collision laws such as this one can give rise to some nonphysical results. A detailed analysis of several collision laws is given in [Chatterjee and Ruina 1998].

REFERENCES

[Baraff 1996] David Baraff. 1996. Linear-time dynamics using Lagrange multipliers. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM, 137–146.

[Baraff 2001] David Baraff. 2001. Physically based modeling: Rigid body simulation. *SIGGRAPH Course Notes, ACM SIGGRAPH* 2, 1 (2001), 2–1.

[Baumgarte 1972] Joachim Baumgarte. 1972. Stabilization of constraints and integrals of motion in dynamical systems. *Computer methods in applied mechanics and engineering* 1, 1 (1972), 1–16.

[Bender et al. 2014] Jan Bender, Kenny Erleben, and Jeff Trinkle. 2014. Interactive simulation of rigid body dynamics in computer graphics. In *Computer Graphics Forum*, Vol. 33. Wiley Online Library, 246–270.

[Bridson 2015] Robert Bridson. 2015. *Fluid simulation for computer graphics*. AK Peters/CRC Press.

[Catto 2011a] Erin Catto. 2011a. Box2d: A 2d physics engine for games. http://box2d.org/

[Catto 2011b] Erin Catto. 2011b. Soft Constraints: Reinventing the Spring. http://box2d.org/files/GDC2011/GDC2011_Catto_Erin_Soft_Constraints.pdf

[Catto 2014] Erin Catto. 2014. Physics for Game Programmers: Understanding Constraints. https://www.youtube.com/watch?v=SHinxAhv1ZE

[Chatterjee and Ruina 1998] Anindya Chatterjee and Andy Ruina. 1998. A new algebraic rigid-body collision law based on impulse space considerations. *Journal of Applied Mechanics* 65, 4 (1998), 939–951.

[Coumans 2010] Erwin Coumans. 2010. Bullet physics engine. *Open Source Software: http://bulletphysics. org* 1 (2010), 3.

[Drumwright 2008] Evan Drumwright. 2008. A fast and stable penalty method for rigid body simulation. *IEEE Transactions on Visualization and Computer Graphics* 14, 1 (2008), 231–240.

[Faure et al. 2008] François Faure, Sébastien Barbier, Jérémie Allard, and Florent Falipou. 2008. Image-based collision detection and response between arbitrary volume objects. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Eurographics Association, 155–162.

[Featherstone 1983] Roy Featherstone. 1983. The calculation of robot dynamics using articulated-body inertias. *The International Journal of Robotics Research* 2, 1 (1983), 13–30.

[Floater 2003] Michael S. Floater. 2003. Mean Value Coordinates. *Comput. Aided Geom. Des.* 20, 1 (March 2003), 19–27. https://doi.org/10.1016/S0167-8396(02)00002-5

[Guendelman et al. 2003] Eran Guendelman, Robert Bridson, and Ronald Fedkiw. 2003. Nonconvex rigid bodies with stacking. In *ACM Transactions on Graphics (TOG)*, Vol. 22. ACM, 871–878.

[Harlow and Welch 1965] Francis H. Harlow and J. Eddie Welch. 1965. Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface. *Physics of Fluids* 8, 12 (1965), 2182–2189. https://doi.org/10.1063/1.1761178

[Harmon et al. 2009] David Harmon, Etienne Vouga, Breannan Smith, Rasmus Tamstorf, and Eitan Grinspun. 2009. Asynchronous contact mechanics. *ACM Transactions on Graphics (TOG)* 28, 3 (2009), 87.

[Heath 2002] Michael T Heath. 2002. *Scientific computing*. McGraw-Hill New York.

[Jacobson et al. 2011] Alec Jacobson, Ilya Baran, Jovan Popović, and Olga Sorkine. 2011. Bounded Biharmonic Weights for Real-time Deformation. *ACM Trans. Graph.* 30, 4, Article 78 (July 2011), 8 pages. https://doi.org/10.1145/2010324.1964973

[Jones et al. 2016] Ben Jones, Nils Thuerey, Tamar Shinar, and Adam W Bargteil. 2016. Example-based plastic deformation of rigid bodies. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 34.

[Joshi et al. 2007] Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. 2007. Harmonic coordinates for character articulation. In *ACM Transactions on Graphics (TOG)*, Vol. 26. ACM, 71.

[Ju et al. 2005] Tao Ju, Scott Schaefer, and Joe Warren. 2005. Mean Value Coordinates for Closed Triangular Meshes. *ACM Trans. Graph.* 24, 3 (July 2005), 561–566. https://doi.org/10.1145/1073204.1073229

[Lacoursiere 2007] Claude Lacoursiere. 2007. *Ghosts and machines: regularized variational methods for interactive simulations of multibodies with dry frictional contacts*. Ph.D. Dissertation. Datavetenskap.

[Lanczos 2012] Cornelius Lanczos. 2012. *The variational principles of mechanics*. Courier Corporation.

[Landau and Lifshitz 1959] LD Landau and EM Lifshitz. 1959. *Course of theoretical physics. vol. 6: Fluid mechanics*. London.

[Lipman et al. 2008] Yaron Lipman, David Levin, and Daniel Cohen-Or. 2008. Green coordinates. *ACM Transactions on Graphics (TOG)* 27, 3 (2008), 78.

[Marschner and Shirley 2015] Steve Marschner and Peter Shirley. 2015. *Fundamentals of computer graphics*. CRC Press.

[Mirtich 1996] Brian Vincent Mirtich. 1996. *Impulse-based dynamic simulation of rigid body systems*. Citeseer.

[Murray et al. 1986] R Murray, R Murray, and Spiegal. 1986. *Theory and Problem of Theoretical Mechanics*. Shaum's Outline Series.

[Museth 2013] Ken Museth. 2013. VDB: High-resolution Sparse Volumes with Dynamic Topology. *ACM Trans. Graph.* 32, 3, Article 27 (July 2013), 22 pages. https://doi.org/10.1145/2487228.2487235

[Ruina and Pratap 2009] Andy L Ruina and Rudra Pratap. 2009. *Introduction to statics and dynamics*. Pre-print for Oxford University Press. http://ruina.tam.cornell.edu/Book/RuinaPratap1-31-11.pdf

[Shen et al. 2005] Chen Shen, James F O'Brien, and Jonathan R Shewchuk. 2005. Interpolating and approximating implicit surfaces from polygon soup. In *ACM Siggraph 2005 Courses*. ACM, 204.

[Shinar et al. 2008] Tamar Shinar, Craig Schroeder, and Ronald Fedkiw. 2008. Two-way coupling of rigid and deformable bodies. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Eurographics Association, 95–103.

[Sifakis et al. 2007] Eftychios Sifakis, Tamar Shinar, Geoffrey Irving, and Ronald Fedkiw. 2007. Hybrid simulation of deformable solids. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association, 81–90.

[Sims 1990] Karl Sims. 1990. Particle Animation and Rendering Using Data Parallel Computation. *SIGGRAPH Comput. Graph.* 24, 4 (Sept. 1990), 405–413. https://doi.org/10.1145/97880.97923

[Smith et al. 2005] Russell Smith et al. 2005. Open dynamics engine. (2005).

[Tang et al. 2012] Min Tang, Dinesh Manocha, Miguel A Otaduy, and Ruofeng Tong. 2012. Continuous penalty forces. *ACM Trans. Graph.* 31, 4 (2012), 107–1.

[Tournier et al. 2015] Maxime Tournier, Matthieu Nesme, Benjamin Gilles, and François Faure. 2015. Stable constrained dynamics. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 132.

[Weinstein et al. 2006] Rachel Weinstein, Joseph Teran, and Ronald Fedkiw. 2006. Dynamic simulation of articulated rigid bodies with contact and collision. *IEEE Transactions on Visualization and Computer Graphics* 12, 3 (2006), 365–374.

[Witkin 2001] Andrew Witkin. 2001. Physically Based Modeling–Constraint Dynamics. *ACM SIGGRAPH 2001 Course Notes* (2001).

[Xu et al. 2014] Hongyi Xu, Yili Zhao, and Jernej Barbic. 2014. Implicit multibody penalty-baseddistributed contact. *IEEE Transactions on Visualization & Computer Graphics* 9 (2014), 1266–1279.