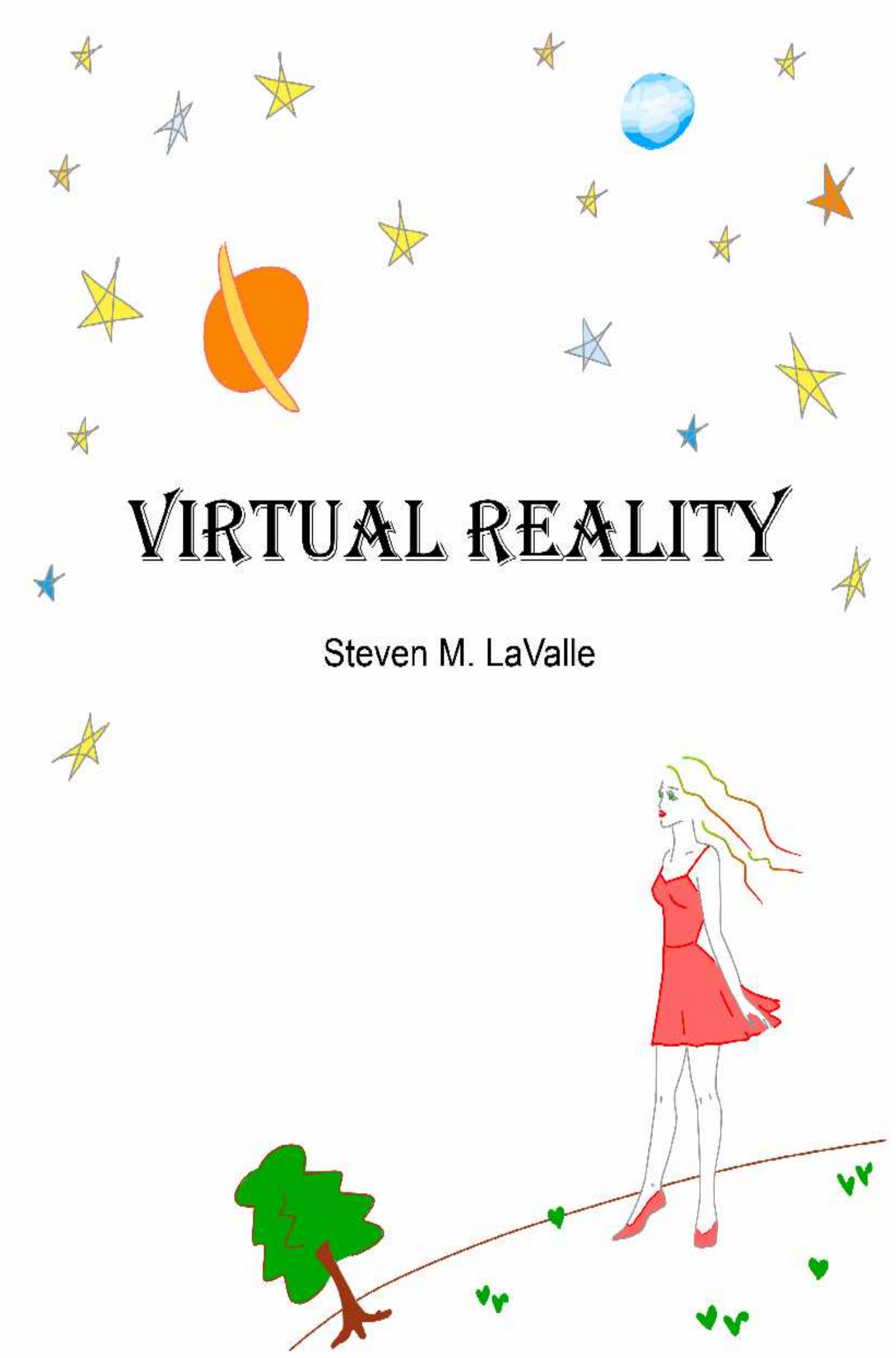


Chapter 8

Motion in Real and Virtual Worlds

Steven M. LaValle

University of Illinois

Chapter 8

Motion in Real and Virtual Worlds

Up to this point, the discussion of movement has been confined to specialized topics. Section 5.3 covered eye movements and Section 6.2 covered the perception of motion. The transformations from Chapter 3 indicate how to place bodies and change viewpoints, but precise mathematical descriptions of motions have not yet been necessary. We now want to model motions more accurately because the physics of both real and virtual worlds impact VR experiences. The accelerations and velocities of moving bodies impact simulations in the VWG and tracking methods used to capture user motions in the physical world. Therefore, this chapter provides foundations that will become useful for reading Chapter 9 on tracking, and Chapter 10 on interfaces.

Section 8.1 introduces fundamental concepts from math and physics, including velocities, accelerations, and the movement of rigid bodies. Section 8.2 presents the physiology and perceptual issues from the human vestibular system, which senses velocities and accelerations. Section 8.3 then describes how motions are described and produced in a VWG. This includes numerical integration and collision detection. Section 8.4 focuses on vection, which is a source of VR sickness that arises due to sensory conflict between the visual and vestibular systems: The eyes may perceive motion while the vestibular system is not fooled. This can be considered as competition between the physics of the real and virtual worlds.

8.1 Velocities and Accelerations

8.1.1 A one-dimensional world

We start with the simplest case, which is shown in Figure 8.1. Imagine a 1D world in which motion is only possible in the vertical direction. Let y be the coordinate of a moving point. Its position at any time t is indicated by $y(t)$, meaning that y actually defines a function of time. It is now as if y were an animated point, with

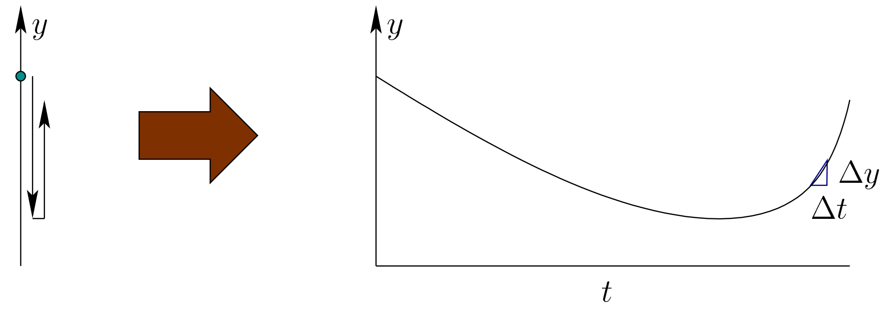


Figure 8.1: A point moving in a one-dimensional world.

an infinite number of frames per second!

Velocity How fast is the point moving? Using calculus, its *velocity*, v, is defined as the derivative of y with respect to time:

$$v = \frac{dy(t)}{dt}. \tag{8.1}$$

Using numerical computations, v is approximately equal to $\Delta y/\Delta t$, in which Δt denotes a small change in time and

$$\Delta y = y(t + \Delta t) - y(t). \tag{8.2}$$

In other words, Δy is the change in y from the start to the end of the time change. The velocity v can be used to estimate the change in y over Δt as

$$\Delta y \approx v\Delta t. \tag{8.3}$$

The approximation quality improves as Δt becomes smaller and v itself varies less during the time from t to $t + \Delta t$.

We can write $v(t)$ to indicate that velocity may change over time. The position can be calculated for any time t from the velocity using integration as[1]

$$y(t) = y(0) + \int_0^t v(s)ds, \tag{8.4}$$

which assumes that y was known at the starting time $t = 0$. If $v(t)$ is constant for all time, represented as v, then $y(t) = y(0) + vt$. The integral in (8.4) accounts for $v(t)$ being allowed to vary.

[1]The parameter s is used instead of t to indicate that it is integrated away, much like the index in a summation.

Acceleration The next step is to mathematically describe the change in velocity, which results in the *acceleration*, a; this is defined as:

$$a = \frac{dv(t)}{dt}. \tag{8.5}$$

The form is the same as (8.1), except that y has been replaced by v. Approximations can similarly be made. For example, $\Delta v \approx a \Delta t$.

The acceleration itself can vary over time, resulting in $a(t)$. The following integral relates acceleration and velocity (compare to (8.4)):

$$v(t) = v(0) + \int_0^t a(s) ds. \tag{8.6}$$

Since acceleration may vary, you may wonder whether the naming process continues. It could, with the next derivative called *jerk*, followed by *snap*, *crackle*, and *pop*. In most cases, however, these higher-order derivatives are not necessary. One of the main reasons is that motions from classical physics are sufficiently characterized through forces and accelerations. For example, Newton's Second Law states that $F = ma$, in which F is the force acting on a point, m is its mass, and a is the acceleration.

For a simple example that should be familiar, consider acceleration due to gravity, $g = 9.8 \text{m/s}^2$. It is as if the ground were accelerating upward by g; hence, the point accelerates downward relative to the Earth. Using (8.6) to integrate the acceleration, the velocity over time is $v(t) = v(0) - gt$. Using (8.4) to integrate the velocity and supposing $v(0) = 0$, we obtain

$$y(t) = y(0) - \frac{1}{2} g t^2. \tag{8.7}$$

8.1.2 Motion in a 3D world

A moving point Now consider the motion of a point in a 3D world $\mathbb{R}^3$. Imagine that a geometric model, as defined in Section 3.1, is moving over time. This causes each point (x, y, z) on the model to move, resulting a function of time for each coordinate of each point:

$$(x(t), y(t), z(t)). \tag{8.8}$$

The velocity v and acceleration a from Section 8.1.1 must therefore expand to have three coordinates. The velocity v is replaced by (v_x, v_y, v_z) to indicate velocity with respect to the x, y, and z coordinates, respectively. The magnitude of v is called the *speed*:

$$\sqrt{v_x^2 + v_y^2 + v_z^2} \tag{8.9}$$

Continuing further, the acceleration also expands to include three components: (a_x, a_y, a_z).

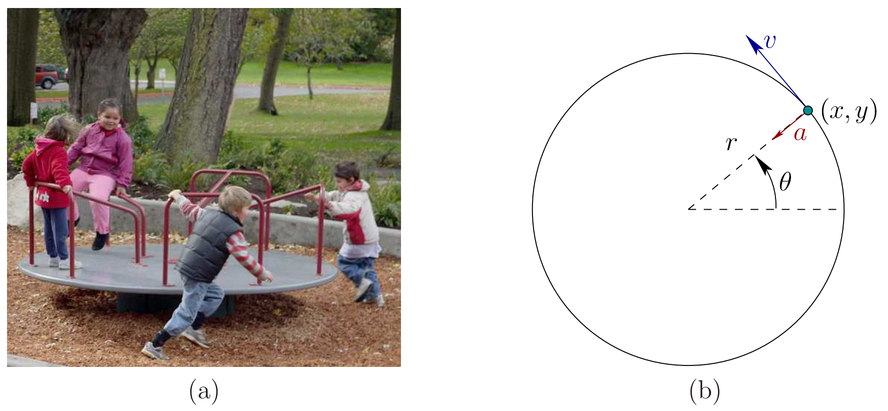


Figure 8.2: (a) Consider a merry-go-round that rotates at constant angular velocity ω. (Picture by Seattle Parks and Recreation.) (b) In a top-down view, the velocity vector, v, for a point on the merry-go-round is tangent to the circle that contains it; the circle is centered on the axis of rotation and the acceleration vector, a, points toward its center.

Rigid-body motion Now suppose that a rigid body is moving through $\mathbb{R}^3$. In this case, all its points move together. How can we easily describe this motion? Recall from Section 3.2 that translations or rotations may be applied. First, consider a simple case. Suppose that rotations are prohibited, and the body is only allowed to translate through space. In this limited setting, knowing the position over time for one point on the body is sufficient for easily determining the positions of all points on the body over time. If one point has changed its position by some (x_t, y_t, z_t), then *all* points have changed by the same amount. More importantly, the velocity and acceleration of every point would be identical.

Once rotation is allowed, this simple behavior breaks. As a body rotates, the points no longer maintain the same velocities and accelerations. This becomes crucial to understanding VR sickness in Section 8.4 and how tracking methods estimate positions and orientations from sensors embedded in the world, which will be discussed in Chapter 9.

Angular velocity To understand the issues, consider the simple case of a spinning *merry-go-round*, as shown in Figure 8.2(a). Its orientation at every time can be described by $\theta(t)$; see Figure 8.2(b). Let ω denote its *angular velocity*:

$$\omega = \frac{d\theta(t)}{dt}. \tag{8.10}$$

By default, ω has units of radians per second. If $\omega = 2\pi$, then the rigid body returns to the same orientation after one second.

Assuming $\theta(0) = 0$ and ω is constant, the orientation at time t is given by $\theta = \omega t$. To describe the motion of a point on the body, it will be convenient to use polar coordinates r and θ:

$$x = r \cos\theta \text{ and } y = r \sin\theta. \tag{8.11}$$

Substituting $\theta = \omega t$ yields

$$x = r \cos\omega t \text{ and } y = r \sin\omega t. \tag{8.12}$$

Taking the derivative with respect to time yields[2]

$$v_x = -r\omega \sin\omega t \text{ and } v_y = r\omega \cos\omega t. \tag{8.13}$$

The velocity is a 2D vector that when placed at the point is tangent to the circle that contains the point (x, y); see Figure 8.2(b).

This makes intuitive sense because the point is heading in that direction; however, the direction quickly changes because it must move along a circle. This change in velocity implies that a nonzero acceleration occurs. The acceleration of the point (x, y) is obtained by taking the derivative again:

$$a_x = -r\omega^2 \cos\omega t \text{ and } a_y = -r\omega^2 \sin\omega t. \tag{8.14}$$

The result is a 2D acceleration vector that is pointing toward the center (Figure 8.2(b)), which is the rotation axis. This is called *centripetal acceleration*. If you were standing at that point, then you would feel a pull in the opposite direction, as if nature were attempting to fling you away from the center. This is precisely how artificial gravity can be achieved in a rotating space station.

3D angular velocity Now consider the rotation of a 3D rigid body. Recall from Section 3.3 that Euler's rotation theorem implies that every 3D rotation can be described as a rotation θ about an axis $v = (v_1, v_2, v_3)$ though the origin. As the orientation of the body changes over a short period of time Δt, imagine the axis that corresponds to the change in rotation. In the case of the merry-go-round, the axis would be $v = (0, 1, 0)$. More generally, v could be any unit vector.

The 3D angular velocity is therefore expressed as a 3D vector:

$$(\omega_x, \omega_y, \omega_z), \tag{8.15}$$

which can be imagined as taking the original ω from the 2D case and multiplying it by the vector v. This weights the components according to the coordinate axes. Thus, the components could be considered as $\omega_x = \omega v_1$, $\omega_y = \omega v_2$, and $\omega_z = \omega v_3$. The ω_x, ω_y, and ω_z components also correspond to the rotation rate in terms of pitch, roll, and yaw, respectively. We avoided these representations in Section 3.3 due to noncommutativity and kinematic singularities; however, it turns out that for velocities these problems do not exist [26]. Thus, we can avoid quaternions at this stage.

[2]If this is unfamiliar, then look up the derivatives of sines and cosines, and the chain rule, from standard calculus sources (for example, [27]).

Angular acceleration If ω is allowed to vary over time, then we must consider *angular acceleration*. In the 2D case, this is defined as

$$\alpha = \frac{d\omega(t)}{dt}. \tag{8.16}$$

For the 3D case, there are three components, which results in

$$(\alpha_x, \alpha_y, \alpha_z). \tag{8.17}$$

These can be interpreted as accelerations of pitch, yaw, and roll angles, respectively.

8.2 The Vestibular System

As mentioned in Section 2.3, the *balance sense* (or *vestibular sense*) provides information to the brain about how the head is oriented or how it is moving in general. This is accomplished through *vestibular organs* that measure both linear and angular accelerations of the head. These organs, together with their associated neural pathways, will be referred to as the *vestibular system*. This system plays a crucial role for bodily functions that involve motion, from ordinary activity such as walking or running, to activities that require substantial talent and training, such as gymnastics or ballet dancing. Recall from Section 5.3 that it also enables eye motions that counteract head movements via the VOR.

The vestibular system is important to VR because it is usually neglected, which leads to a mismatch of perceptual cues (recall this problem from Section 6.4). In current VR systems, there is no engineered device that renders vestibular signals to a display that precisely stimulates the vestibular organs to values as desired. Some possibilities may exist in the future with *galvanic vestibular stimulation*, which provides electrical stimulation to the organ [8, 7]; however, it may take many years before such techniques are sufficiently accurate, comfortable, and generally approved for safe use by the masses. Another possibility is to stimulate the vestibular system through low-frequency vibrations, which at the very least provides some distraction.

Physiology Figure 8.4 shows the location of the vestibular organs inside of the human head. As in the cases of eyes and ears, there are two symmetric organs, corresponding to the right and left sides. Figure 8.3 shows the physiology of each vestibular organ. The *cochlea* handles hearing, which is covered in Section 11.2, and the remaining parts belong to the vestibular system. The *utricle* and *saccule* measure linear acceleration; together they form the *otolith system*. When the head is not tilted, the sensing surface of the utricle mostly lies in the horizontal plane (or xz plane in our common coordinate systems), whereas the corresponding surface of the saccule lies in a vertical plane that is aligned in the forward direction

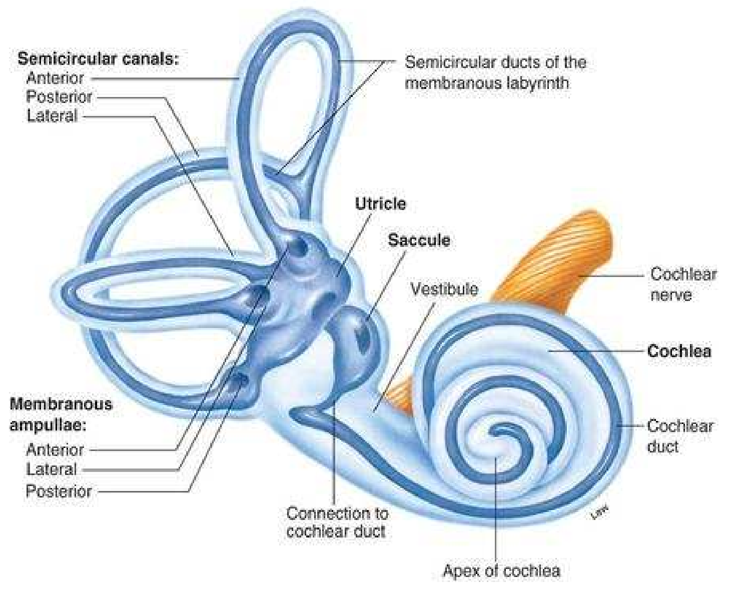


Figure 8.3: The vestibular organ.

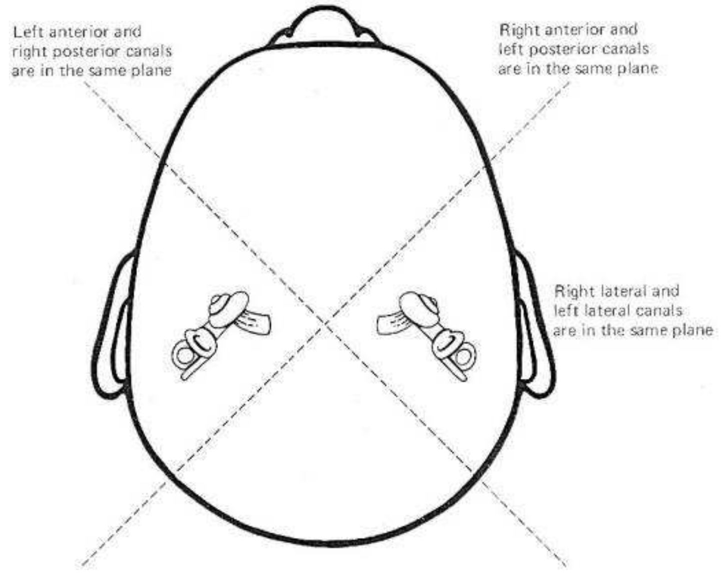

Left anterior and right posterior canals are in the same plane
Right anterior and left posterior canals are in the same plane
Right lateral and left lateral canals are in the same plane

Figure 8.4: The vestibular organs are located behind the ears. (Figure from CNS Clinic Jordan.)

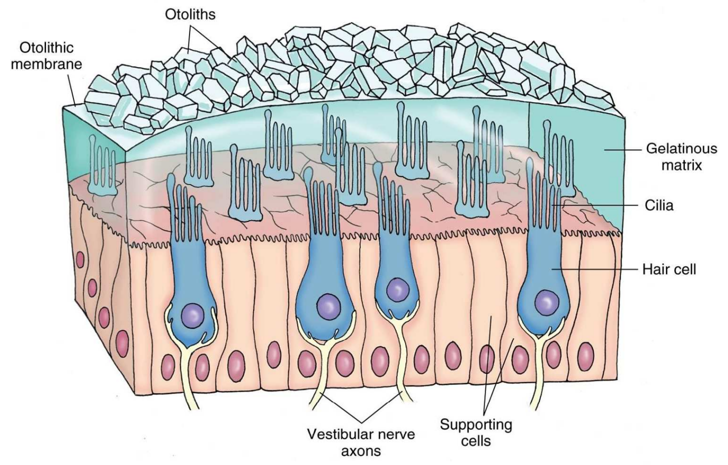


Figure 8.5: A depiction of an otolith organ (utricle or saccule), which senses linear acceleration. (Figure by Lippincott Williams & Wilkins.)

(this is called the *sagittal plane*, or yz plane). As will be explained shortly, the utricle senses acceleration components a_x and a_z, and the saccule senses a_y and a_z (a_z is redundantly sensed).

The *semicircular canals* measure angular acceleration. Each canal has a diameter of about 0.2 to 0.3mm, and is bent along a circular arc with a diameter of about 2 to 3mm. Amazingly, the three canals are roughly perpendicular so that they independently measure three components of angular velocity. The particular canal names are *anterior canal*, *posterior canal*, and *lateral canal*. They are not closely aligned with our usual 3D coordinate coordinate axes. Note from Figure 8.4 that each set of canals is rotated by 45 degrees with respect to the vertical axis. Thus, the anterior canal of the left ear aligns with the posterior canal of the right ear. Likewise, the posterior canal of the left ear aligns with the anterior canal of the right ear. Although not visible in the figure, the lateral canal is also tilted about 30 away from level. Nevertheless, all three components of angular acceleration are sensed because the canals are roughly perpendicular.

Sensing linear acceleration To understand how accelerations are sensed, we start with the case of the otolith system. Figure 8.5 shows a schematic representation of an otolith organ, which may be either the utricle or saccule. Mechanoreceptors, in the form of *hair cells*, convert acceleration into neural signals. Each hair cell has *cilia* that are embedded in a gelatinous matrix. Heavy weights lie on top of the matrix so that when acceleration occurs laterally, the shifting weight applies a shearing force that causes the cilia to bend. The higher the acceleration magnitude, the larger the bending, and a higher rate of neural impulses become

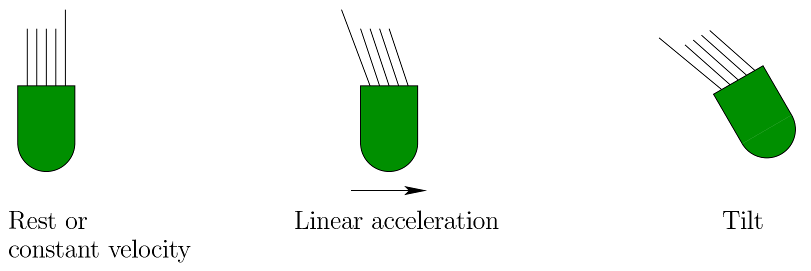


Figure 8.6: Because of the Einstein equivalence principle, the otolith organs cannot distinguish linear acceleration of the head from tilt with respect to gravity. In either case, the cilia deflect in the same way, sending equivalent signals to the neural structures.

transmitted. Two dimensions of lateral deflection are possible. For example, in the case of the utricle, linear acceleration in any direction in the xz plane would cause the cilia to bend. To distinguish between particular directions inside of this plane, the cilia are *polarized* so that each cell is sensitive to one particular direction. This is accomplished by a thicker, lead hair called the *kinocilium*, to which all other hairs of the cell are attached by a ribbon across their tips so that they all bend together.

One major sensing limitation arises because of a fundamental law from physics: The *Einstein equivalence principle*. In addition to the vestibular system, it also impacts VR tracking systems (see Section 9.2). The problem is gravity. If we were deep in space, far away from any gravitational forces, then linear accelerations measured by a sensor would correspond to pure accelerations with respect to a fixed coordinate frame. On the Earth, we also experience force due to gravity, which feels as if we were on a rocket ship accelerating upward at roughly 9.8m/s^2. The equivalence principle states that the effects of gravity and true linear accelerations on a body are indistinguishable. Figure 8.6 shows the result in terms of the otolith organs. The same signals are sent to the brain whether the head is tilted or it is linearly accelerating. If you close your eyes or wear a VR headset, then you should not be able to distinguish tilt from acceleration. In most settings, we are not confused because the vestibular signals are accompanied by other stimuli when accelerating, such as vision and a revving engine.

Sensing angular acceleration The semicircular canals use the same principle as the otolith organs. They measure acceleration by bending cilia at the end of hair cells. A viscous fluid moves inside of each canal. A flexible structure called the *cupula* blocks one small section of the canal and contains the hair cells; see Figure 8.7. Compare the rotation of a canal to the merry-go-round. If we were to place a liquid-filled tube around the periphery of the merry-go-round, then the fluid would remain fairly stable at a constant angular velocity. However, if angular

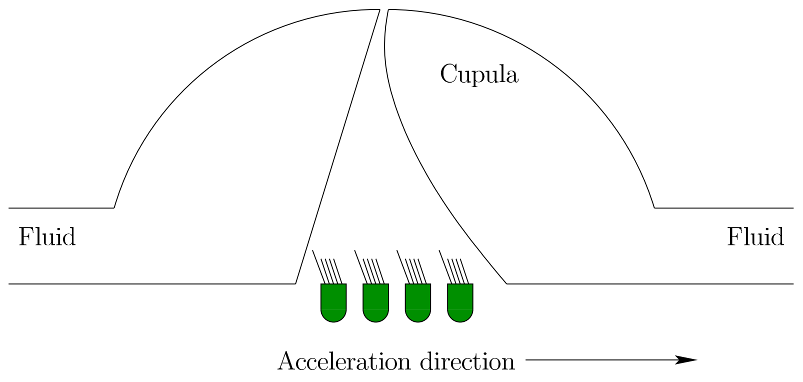


Figure 8.7: The cupula contains a center membrane that houses the cilia. If angular acceleration occurs that is aligned with the canal direction, then pressure is applied to the cupula, which causes the cilia to bend and send neural signals.

acceleration is applied, then due to friction between the fluid and the tube (and also internal fluid viscosity), the fluid would start to travel inside the tube. In the semicircular canal, the moving fluid applies pressure to the cupula, causing it to deform and bend the cilia on hair cells inside of it. Note that a constant angular velocity does not, in principle, cause pressure on the cupula; thus, the semicircular canals measure angular *acceleration* as opposed to velocity. Each canal is polarized in the sense that it responds mainly to rotations about an axis perpendicular to the plane that contains the entire canal.

Impact on perception Cues from the vestibular system are generally weak in comparison to other senses, especially vision. For example, a common danger for a skier buried in an avalanche is that he cannot easily determine which way is up without visual cues to accompany the perception of gravity from the vestibular system. Thus, the vestibular system functions well when providing consistent cues with other systems, including vision and proprioception. Mismatched cues are problematic. For example, some people may experience *vertigo* when the vestibular system is not functioning correctly. In this case, they feel as if the world around them is spinning or swaying. Common symptoms are nausea, vomiting, sweating, and difficulties walking. This may even impact eye movements because of the VOR. Section 8.4 explains a bad side effect that results from mismatched vestibular and visual cues in VR.

8.3 Physics in the Virtual World

8.3.1 Tailoring the Physics to the Experience

If we expect to fool our brains into believing that we inhabit the virtual world, then many of our expectations from the real world should be matched in the virtual world. We have already seen this in the case of the physics of light (Chapter 4) applying to visual rendering of virtual worlds (Chapter 7). Motions in the virtual world should also behave in a familiar way.

This implies that the VWG contains a *physics engine* that governs the motions of bodies in the virtual world by following principles from the physical world. Forces acting on bodies, gravity, fluid flows, and collisions between bodies should be handled in perceptually convincing ways. Physics engines arise throughout engineering and physics in the context of any simulation. In video games, computer graphics, and film, these engines perform operations that are very close to our needs for VR. This is why popular game engines such as Unity 3D and Unreal Engine have been quickly adapted for use in VR. As stated in Section 2.2, we have not yet arrived at an era in which general and widely adopted VR engines exist; therefore, modern game engines are worth understanding and utilizing at present.

To determine what kind of physics engine needs to be borrowed, adapted, or constructed from scratch, one should think about the desired VR experience and determine the kinds of motions that will arise. Some common, generic questions are:

- Will the matched zone remain fixed, or will the user need to be moved by locomotion? If locomotion is needed, then will the user walk, run, swim, drive cars, or fly spaceships?

- Will the user interact with objects? If so, then what kind of interaction is needed? Possibilities include carrying weapons, opening doors, tossing objects, pouring drinks, operating machinery, drawing pictures, and assembling structures.

- Will multiple users be sharing the same virtual space? If so, then how will their motions be coordinated or constrained?

- Will the virtual world contain entities that appear to move autonomously, such as robots, animals, or humans?

- Will the user be immersed in a familiar or exotic setting? A familiar setting could be a home, classroom, park, or city streets. An exotic setting might be scuba diving, lunar exploration, or traveling through blood vessels.

In addition to the physics engine, these questions will also guide the design of the interface, which is addressed in Chapter 10.

Based on the answers to the questions above, the physics engine design may be simple and efficient, or completely overwhelming. As mentioned in Section 7.4, a key challenge is to keep the virtual world frequently updated so that interactions between users and objects are well synchronized and renderers provide a low-latency projection onto displays.

Note that the goal may not always be to perfectly match what would happen in the physical world. In a familiar setting, we might expect significant matching; however, in exotic settings, it often becomes more important to make a comfortable experience, rather than matching reality perfectly. Even in the case of simulating oneself walking around in the world, we often want to deviate from real-world physics because of vection, which causes VR sickness (see Section 8.4).

The remainder of this section covers some fundamental aspects that commonly arise: 1) numerical simulation of physical systems, 2) the control of systems using human input, and 3) collision detection, which determines whether bodies are interfering with each other.

8.3.2 Numerical simulation

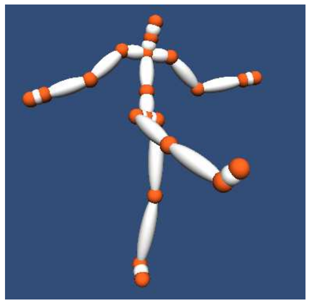
The state of the virtual world Imagine a virtual world that contains many moving rigid bodies. For each body, think about its degrees of freedom (*DOFs*), which corresponds to the number of independent parameters needed to uniquely determine its position and orientation. We would like to know the complete list of parameters needed to put every body in its proper place in a single time instant. A specification of values for all of these parameters is defined as the *state* of the virtual world.

The job of the physics engine can then be described as calculating the virtual world state for every time instant or "snapshot" of the virtual world that would be needed by a rendering system. Once the state is determined, the mathematical transforms of Chapter 3 are used to place the bodies correctly in the world and calculate how they should appear on displays.

Degrees of freedom How many parameters are there in a virtual world model? As discussed in Section 3.2, a free-floating body has 6 DOFs which implies 6 parameters to place it anywhere. In many cases, DOFs are lost due to constraints. For example, a ball that rolls on the ground has only 5 DOFs because it can achieve any 2D position along the ground and also have any 3D orientation. It might be sufficient to describe a car with 3 DOFs by specifying the position along the ground (two parameters) and the direction it is facing (one parameter); see Figure 8.8(a). However, if the car is allowed to fly through the air while performing stunts or crashing, then all 6 DOFs are needed.

For many models, rigid bodies are attached together in a way that allows relative motions. This is called *multibody kinematics* [16, 26]. For example, a car usually has 4 wheels which can roll to provide one rotational DOF per wheel

(a)



(b)

Figure 8.8: (a) A virtual car (Cheetah) that appears in the game Grand Theft Auto; how many degrees of freedom should it have? (b) A human skeleton, with rigid bodies connected via joints, commonly underlies the motions of an avatar. (Figure from SoftKinetic).

. Furthermore, the front wheels can be steered to provide an additional DOF. Steering usually turns the front wheels in unison, which implies that one DOF is sufficient to describe both wheels. If the car has a complicated suspension system, then it cannot be treated as a single rigid body, which would add many more DOFs.

Similarly, an animated character can be made by attaching rigid bodies to form a skeleton; see Figure 8.8(b). Each rigid body in the skeleton is attached to one or more other bodies by a *joint*. For example, a simple human character can be formed by attaching arms, legs, and a neck to a rigid torso. The upper left arm is attached to the torso by a shoulder joint. The lower part of the arm is attached by an elbow joint, and so on. Some joints allow more DOFs than others. For example, the shoulder joint has 3 DOFs because it can yaw, pitch, and roll with respect to the torso, but an elbow joint has only one DOF.

To fully model the flexibility of the human body, 244 DOFs are needed, which are controlled by 630 muscles [31]. In many settings, this would be too much detail, which might lead to high computational complexity and difficult implementation. Furthermore, one should always beware of the uncanny valley (mentioned in Section 1.1), in which more realism might lead to increased perceived creepiness of the character. Thus, having more DOFs is not clearly better, and it is up to a VR content creator to determine how much mobility is needed to bring a character to life, in a way that is compelling for a targeted purpose.

In the extreme case, rigid bodies are not sufficient to model the world. We might want to see waves rippling realistically across a lake, or hair gently flowing in the breeze. In these general settings, *nonrigid models* are used, in which case the state can be considered as a continuous function. For example, a function of the form $y = f(x, z)$ could describe the surface of the water. Without making

some limiting simplifications, the result could effectively be an infinite number of DOFs. Motions in this setting are typically described using *partial differential equations* (*PDEs*), which are integrated numerically to obtain the state at a desired time. Usually, the computational cost is high enough to prohibit their use in interactive VR experiences, unless shortcuts are taken by precomputing motions or dramatically simplifying the model.

Differential equations We now introduce some basic differential equations to model motions. The resulting description is often called a *dynamical system*. The first step is to describe rigid body velocities in terms of state. Returning to models that involve one or more rigid bodies, the state corresponds to a finite number of parameters. Let

$$x = (x_1, x_2, \ldots, x_n) \tag{8.18}$$

denote an n-dimensional *state vector*. If each x_i corresponds to a position or orientation parameter for a rigid body, then the state vector puts all bodies in their place. Let

$$\dot{x}_i = \frac{dx_i}{dt} \tag{8.19}$$

represent the time derivative, or velocity, for each parameter.

To obtain the state at any time t, the velocities need to be integrated over time. Following (8.4), the integration of each state variable determines the value at time t:

$$x_i(t) = x_i(0) + \int_0^t \dot{x}_i(s) ds, \tag{8.20}$$

in which $x_i(0)$ is the value of x_i at time $t = 0$.

Two main problems arise with (8.20):

1. The integral almost always must be evaluated numerically.

2. The velocity $\dot{x}_i(t)$ must be specified at each time t.

Sampling rate For the first problem, time is discretized into steps, in which Δt is the *step size* or *sampling rate*. For example, Δt might be 1ms, in which case the state can be calculated for times $t = 0, 0.001, 0.002, \ldots$, in terms of seconds. This can be considered as a kind of frame rate for the physics engine. Each Δt corresponds to the production of a new frame.

As mentioned in Section 7.4, the VWG should synchronize the production of virtual world frames with rendering processes so that the world is not caught in an intermediate state with some variables updated to the new time and others stuck at the previous time. This is a kind of tearing in the virtual world. This does not, however, imply that the frame rates are the same between renderers and the physics engine. Typically, the frame rate for the physics engine is much higher to improve numerical accuracy.

Using the sampling rate Δt, (8.20) is approximated as

$$x_i((k+1)\Delta t) \approx x_i(k\Delta t) + \sum_{j=1}^{k} \dot{x}_i(j\Delta t)\Delta t, \tag{8.21}$$

for each state variable x_i.

It is simpler to view (8.21) one step at a time. Let $x_i[k]$ denote $x_i(k\Delta t)$, which is the state at time $t = k\Delta t$. The following is an update law that expresses the new state $x_i[k+1]$ in terms of the old state $x_i[k]$:

$$x_i[k+1] \approx x_i[k] + \dot{x}_i(k\Delta t)\Delta t, \tag{8.22}$$

which starts with $x_i[0] = x_i(0)$.

Runge-Kutta integration The approximation used in (8.21) is known as Euler integration. It is the simplest approximation, but does not perform well enough in many practical settings. One of the most common improvements is the fourth-order *Runge-Kutta integration* method, which expresses the new state as

$$x_i[k+1] \approx x_i[k] + \frac{\Delta t}{6}(w_1 + 2w_2 + 2w_3 + w_4), \tag{8.23}$$

in which

$$\begin{aligned} w_1 &= f(\dot{x}_i(k\Delta t)) \\ w_2 &= f(\dot{x}_i(k\Delta t + \tfrac{1}{2}\Delta t) + \tfrac{1}{2}\Delta t\ w_1) \\ w_3 &= f(\dot{x}_i(k\Delta t + \tfrac{1}{2}\Delta t) + \tfrac{1}{2}\Delta t\ w_2) \\ w_4 &= f(\dot{x}_i(k\Delta t + \Delta t) + \Delta t\ w_3). \end{aligned} \tag{8.24}$$

Although this is more expensive than Euler integration, the improved accuracy is usually worthwhile in practice. Many other methods exist, with varying performance depending on the particular ways in which $\dot{x}$ is expressed and varies over time [13].

Time-invariant dynamical systems The second problem from (8.20) is to determine an expression for $\dot{x}(t)$. This is where the laws of physics, such as the acceleration of rigid bodies due to applied forces and gravity. The most common case is *time-invariant dynamical systems*, in which $\dot{x}$ depends only on the current state and not the particular time. This means each component x_i is expressed as

$$\dot{x}_i = f_i(x_1, x_2, \ldots, x_n), \tag{8.25}$$

for some given vector-valued function $f = (f_1, \ldots, f_n)$. This can be written in compressed form by using x and $\dot{x}$ to represent n-dimensional vectors:

$$\dot{x} = f(x). \tag{8.26}$$

The expression above is often called the *state transition equation* because it indicates the state's rate of change.

Here is a simple, one-dimensional example of a state transition equation:

$$\dot{x} = 2x - 1. \tag{8.27}$$

This is called a *linear* differential equation. The velocity $\dot{x}$ roughly doubles with the value of x. Fortunately, linear problems can be fully solved "on paper". The solution to (8.27) is of the general form

$$x(t) = \frac{1}{2} + ce^{2t}, \tag{8.28}$$

in which c is a constant that depends on the given value for $x(0)$.

The phase space Unfortunately, motions are usually described in terms of accelerations (and sometimes higher-order derivatives), which need to be integrated twice. This leads to higher-order differential equations, which are difficult to work with. For this reason, *phase space* representations were developed in physics and engineering. In this case, the velocities of the state variables are themselves treated as state variables. That way, the accelerations become the velocities of the velocity variables.

For example, suppose that a position x_1 is acted upon by gravity, which generates an acceleration $a = -9.8\text{m/s}^2$. This leads to a second variable x_2, which is defined as the velocity of x_1. Thus, by definition, $\dot{x}_1 = x_2$. Furthermore, $\dot{x}_2 = a$ because the derivative of velocity is acceleration. Both of these equations fit the form of (8.25). Generally, the number of states increases to incorporate accelerations (or even higher-order derivatives), but the resulting dynamics are expressed in the form (8.25), which is easier to work with.

Handling user input Now consider the case in which a user commands an object to move. Examples include driving a car, flying a spaceship, or walking an avatar around. This introduces some new parameters, called the *controls*, actions, or inputs to the dynamical system. Differential equations that include these new parameters are called *control systems* [4].

Let $u = (u_1, u_2, \ldots, u_m)$ be a vector of controls. The state transition equation in (8.26) is simply extended to include u:

$$\dot{x} = f(x, u). \tag{8.29}$$

Figure 8.9 shows a useful example, which involves driving a car. The control u_s determines the speed of the car. For example, $u_s = 1$ drives forward, and $u_s = -1$ drives in reverse. Setting $u_s = 10$ drives forward at a much faster rate. The control u_ϕ determines how the front wheels are steered. The state vector is (x, z, θ), which corresponds to the position and orientation of the car in the horizontal, xz plane.

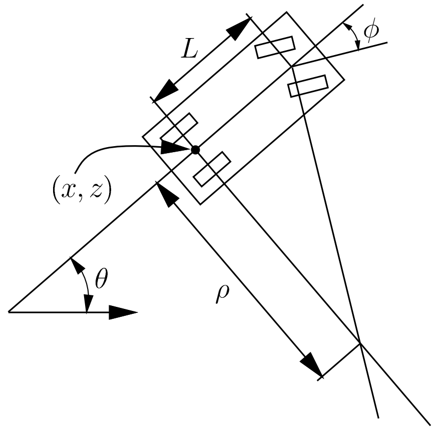


Figure 8.9: A top-down view of a simple, steerable car. Its position and orientation are given by (x, y, θ). The parameter ρ is the minimum turning radius, which depends on the maximum allowable steering angle ϕ. This model can also be used to "steer" human avatars, by placing the viewpoint above the center of the rear axle.

The state transition equation is:

$$\begin{aligned} \dot{x} &= u_s \cos\theta \\ \dot{z} &= u_s \sin\theta \\ \dot{\theta} &= \frac{u_s}{L} \tan u_\phi. \end{aligned} \tag{8.30}$$

Using Runge-Kutta integration, or a similar numerical method, the future states can be calculated for the car, given that controls u_s and u_ϕ are applied over time.

This model can also be used to steer the virtual walking of a VR user from first-person perspective. The viewpoint then changes according to (x, z, θ), while the height y remains fixed. For the model in (8.30), the car must drive forward or backward to change its orientation. By changing the third component to $\theta = u_\omega$, the user could instead specify the angular velocity directly. This would cause the user to rotate in place, as if on a merry-go-round. Many more examples like these appear in Chapter 13 of [16], including bodies that are controlled via accelerations.

It is sometimes helpful conceptually to define the motions in terms of discrete points in time, called *stages*. Using numerical integration of (8.29), we can think about applying a control u over time Δt to obtain a new state $x[k+1]$:

$$x[k+1] = F(x[k], u[k]). \tag{8.31}$$

The function F is obtained by integrating (8.29) over Δt. Thus, if the state is $x[k]$, and $u[k]$ is applied, then F calculates $x[k+1]$ as the state at the next stage.

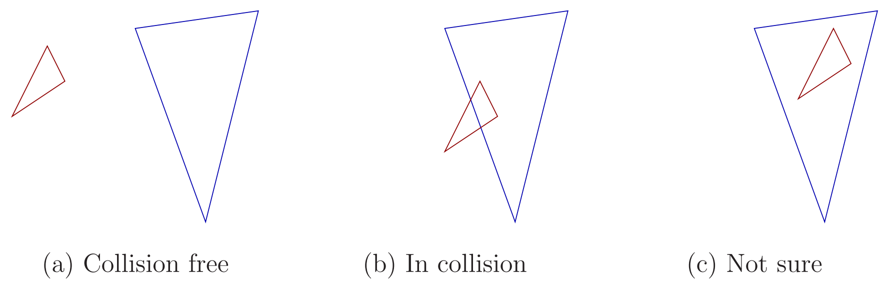


Figure 8.10: Three interesting cases for collision detection (these are 2D examples). The last case may or not cause collision, depending on the model.

8.3.3 Collision detection

One of the greatest challenges in building a physics engine is handling collisions between bodies. Standard laws of motion from physics or engineering usually do not take into account such interactions. Therefore, specialized algorithms are used to detect when such collisions occur and respond appropriately. Collision detection methods and corresponding software are plentiful because of widespread needs in computer graphics simulations and video games, and also for motion planning of robots.

Solid or boundary model? Figure 8.10 shows one the first difficulties with collision detection, in terms of two triangles in a 2D world. The first two cases (Figures 8.10(a) and 8.10(b)) show obvious cases; however, the third case, Figure 8.10(c), could be ambiguous. If one triangle is wholly inside of another, then is this a collision? If we interpret the outer triangle as a solid model, then YES. If the outer triangle is only the boundary edges, and is meant to have an empty interior, then the answer is NO. This is why emphasis was placed on having a coherent model in Section 3.1; otherwise, the boundary might not be established well enough to distinguish the inside from the outside.

Distance functions Many collision detection methods benefit from maintaining a distance function, which keeps track of how far the bodies are from colliding. For example, let A and B denote the set of all points occupied in $\mathbb{R}^3$ by two different models. If they are in collision, then their intersection $A \cap B$ is not empty. If they are not in collision, then the *Hausdorff distance* between A and B is the Euclidean distance between the closest pair of points, taking one from A and one from B.[3]

[3]This assumes models contain all of the points on their boundary and that they have finite extent; otherwise, topological difficulties arise [12, 16]

Let $d(A, B)$ denote this distance. If A and B intersect, then $d(A, B) = 0$ because any point in $A \cap B$ will yield zero distance. If A and B do not intersect, then $d(A, B) > 0$, which implies that they are not in collision (in other words, collision free).

If $d(A, B)$ is large, then A and B are mostly likely to be collision free in the near future, even if one or both are moving. This leads to a family of collision detection methods called *incremental distance computation*, which assumes that between successive calls to the algorithm, the bodies move only a small amount. Under this assumption the algorithm achieves "almost constant time" performance for the case of convex polyhedral bodies [19, 21]. Nonconvex bodies can be decomposed into convex components.

A concept related to distance is *penetration depth*, which indicates how far one model is poking into another [20]. This is useful for setting a threshold on how much interference between the two bodies is allowed. For example, the user might be able to poke his head two centimeters into a wall, but beyond that, an action should be taken.

Simple collision tests At the lowest level, collision detection usually requires testing a pair of model primitives to determine whether they intersect. In the case of models formed from 3D triangles, then we need a method that determines whether two triangles intersect. This is similar to the ray-triangle intersection test that was needed for visual rendering in Section 7.1, and involves basic tools from analytic geometry, such as cross products and plane equations. Efficient methods are given in [11, 23].

Broad and narrow phases Suppose that a virtual world has been defined with millions of triangles. If two complicated, nonconvex bodies are to be checked for collision, then the computational cost may be high. For this complicated situation, collision detection often becomes a two-phase process:

1. **Broad Phase:** In the *broad phase*, the task is to avoid performing expensive computations for bodies that are far away from each other. Simple bounding boxes can be placed around each of the bodies, and simple tests can be performed to avoid costly collision checking unless the boxes intersect. Hashing schemes can be employed in some cases to greatly reduce the number of pairs of boxes that have to be tested for intersect [22].

2. **Narrow Phase:** In the *narrow phase* individual pairs of model parts are each checked carefully for collision. This involves the expensive tests, such as triangle-triangle intersection.

In the broad phase, *hierarchical methods* generally decompose each body into a tree. Each vertex in the tree represents a *bounding region* that contains some subset of the body. The bounding region of the root vertex contains the whole

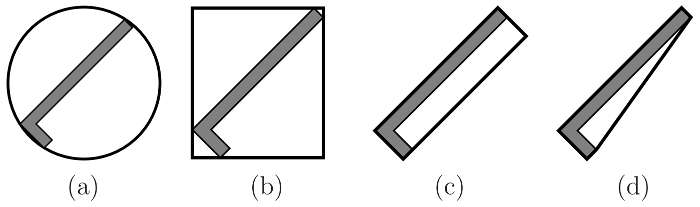


Figure 8.11: Four different kinds of bounding regions: (a) sphere, (b) axis-aligned bounding box (AABB), (c) oriented bounding box (OBB), and (d) convex hull. Each usually provides a tighter approximation than the previous one but is more expensive to test for intersection with others.

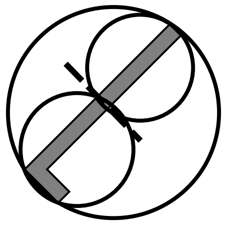


Figure 8.12: The large circle shows the bounding region for a vertex that covers an L-shaped body. After performing a split along the dashed line, two smaller circles are used to cover the two halves of the body. Each circle corresponds to a child vertex.

body. Two opposing criteria that guide the selection of the type of bounding region:

1. The region should fit the intended model points as tightly as possible.

2. The intersection test for two regions should be as efficient as possible.

Several popular choices are shown in Figure 8.11, for the case of an L-shaped body. Hierarchical methods are also useful for quickly eliminating many triangles from consideration in visual rendering, as mentioned in Section 7.1.

The tree is constructed for a body, A (or alternatively, B) recursively as follows. For each vertex, consider the set X of all points in A that are contained in the bounding region. Two child vertices are constructed by defining two smaller bounding regions whose union covers X. The split is made so that the portion covered by each child is of similar size. If the geometric model consists of primitives such as triangles, then a split could be made to separate the triangles into two sets of roughly the same number of triangles. A bounding region is then computed for each of the children. Figure 8.12 shows an example of a split for the case of an L-shaped body. Children are generated recursively by making splits until very simple sets are obtained. For example, in the case of triangles in space, a split is made unless the vertex represents a single triangle. In this case, it is easy to test for the intersection of two triangles.

Consider the problem of determining whether bodies A and B are in collision. Suppose that the trees T_a and T_b have been constructed for A and B, respectively. If the bounding regions of the root vertices of T_a and T_b do not intersect, then it is known that T_a and T_b are not in collision without performing any additional computation. If the bounding regions do intersect, then the bounding regions of the children of T_a are compared to the bounding region of T_b. If either of these intersect, then the bounding region of T_b is replaced with the bounding regions of its children, and the process continues recursively. As long as the bounding regions intersect, lower levels of the trees are traversed, until eventually the leaves are reached. At the leaves the algorithm tests the individual triangles for collision, instead of bounding regions. Note that as the trees are traversed, if a bounding region from the vertex v_1 of T_a does not intersect the bounding region from a vertex, v_2, of T_b, then no children of v_1 have to be compared to children of v_2. Usually, this dramatically reduces the number of comparisons, relative to a naive approach that tests all pairs of triangles for intersection.

Mismatched obstacles in VR Although collision detection is a standard, well-solved problem, VR once again poses unusual challenges. One of the main difficulties is the matched zone, in which the real and virtual worlds share the same space. This leads to three interesting cases:

1. **Real obstacle only:** In this case, an obstacle exists in the real world, but not in the virtual world. This is potentially dangerous! For example, you could move your arm and knock over a real, hot cup of coffee that is not represented in the virtual world. If you were walking with a VR headset, then imagine what would happen if a set of real downward stairs were not represented. At the very least, the boundary of the matched zone should be rendered if the user gets close to it. This mismatch motivated the introduction of the Chaperone system in the HTC Vive headset, in which an outward-facing camera is used to detect and render real objects that may obstruct user motion.

2. **Virtual obstacle only:** This case is not dangerous, but can be extremely frustrating. The user could poke her head through a wall in VR without feeling any response in the real world. This should not be allowed in most cases. The VWG could simply stop moving the viewpoint in the virtual world as the virtual wall is contacted; however, this generates a mismatch between the real and virtual motions, which could be uncomfortable for the user. It remains a difficult challenge to keep users comfortable while trying to guide them away from interference with virtual obstacles.

3. **Real and virtual obstacle:** If obstacles are matched in both real and virtual worlds, then the effect is perceptually powerful. For example, you might stand on a slightly raised platform in the real world while the virtual world shows you standing on a building rooftop. If the roof and platform edges align perfectly, then you could feel the edge with your feet. Would you be afraid to step over the edge? A simpler case is to render a virtual chair that matches the real chair that a user might be sitting in.

8.4 Mismatched Motion and Vection

Vection was mentioned in Section 2.3 as an illusion of self motion that is caused by varying visual stimuli. In other words, the brain is tricked into believing that the head is moving based on what is seen, even though no motion actually occurs. Figure 2.20 showed the haunted swing illusion, which convinced people that were swinging upside down; however, the room was moving while they were stationary. Vection is also commonly induced in VR by moving the user's viewpoint while there is no corresponding motion in the real world.

Vection is a prime example of mismatched cues, which were discussed in Section 6.4. Whereas the McGurk effect has no harmful side effects, vection unfortunately leads many people to experience sickness symptoms, such as dizziness, nausea, and occasionally even vomiting. Thus, it should be used very sparingly, if at all, for VR experiences. Furthermore, if it is used, attempts should be made to alter the content so that the side effects are minimized. Industry leaders often proclaim that their latest VR headset has beaten the VR sickness problem; however, this neglects the following counterintuitive behavior:

> **If a headset is better in terms of spatial resolution, frame rate, tracking accuracy, field of view, and latency, then the potential is higher for making people sick through vection and other mismatched cues.**

Put simply and intuitively, if the headset more accurately mimics reality, then the sensory cues are stronger, and our perceptual systems become more confident about mismatched cues. It may even have the ability to emulate poorer headsets, resulting in a way to comparatively assess side effects of earlier VR systems. In some cases, the mismatch of cues may be harmless (although possibly leading to a decreased sense of presence). In other cases, the mismatches may lead to greater fatigue as the brain works harder to resolve minor conflicts. In the worst case, VR sickness emerges, with vection being the largest culprit based on VR experiences being made today. One of the worst cases is the straightforward adaptation of first-person shooter games to VR, in which the vection occurs almost all the time as the avatar explores the hostile environment.

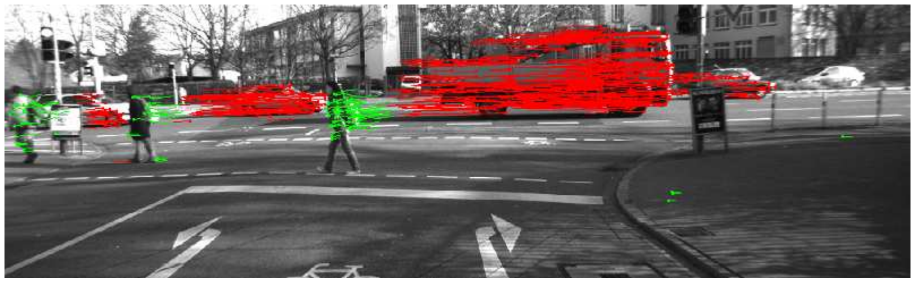


Figure 8.13: The optical flow of features in an image due to motion in the world. These were computed automatically using image processing algorithms. (Image by Andreas Geiger, from Max Planck Institute for Intelligent Systems in Tübingen.)

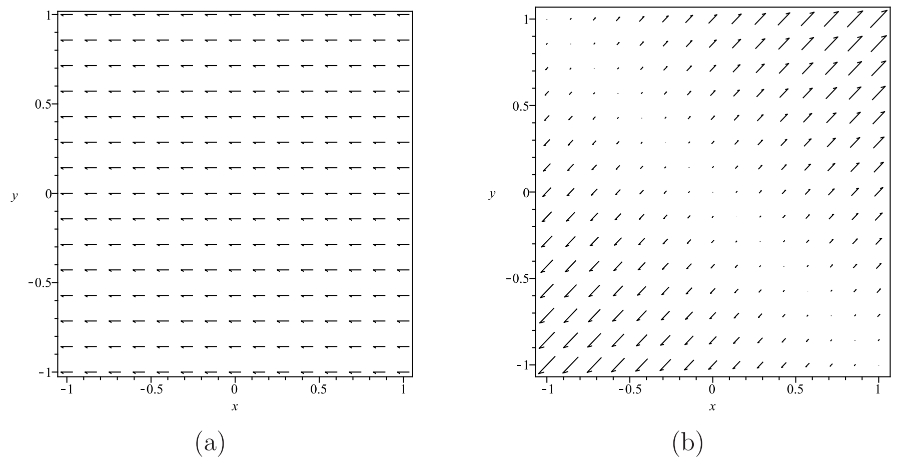


Figure 8.14: Example vector fields: (a) A constant vector field, for which every vector is $(-1, 0)$, regardless of the location. (b) In this vector field, $(x, y) \mapsto (x + y, x + y)$, the vectors point away from the diagonal line from $(-1, 1)$ to $(1, -1)$, and their length is proportional to the distance from it.

Optical flow Recall from Section 6.2, that the human visual system has neural structures dedicated to detecting the motion of visual features in the field of view; see Figure 8.13. It is actually the images of these features that move across the retina. It is therefore useful to have a mathematical concept that describes the velocities of moving points over a surface. We therefore define a *vector field*, which assigns a velocity vector at every point along a surface. If the surface is the xy plane, then a velocity vector

$$(v_x, v_y) = \left(\frac{dx}{dt}, \frac{dy}{dt}\right) \tag{8.32}$$

is assigned at every point (x, y). For example,

$$(x, y) \mapsto (-1, 0) \tag{8.33}$$

is a *constant vector field*, which assigns $v_x = -1$ and $v_y = 0$ everywhere; see Figure 8.14(a). The vector field

$$(x, y) \mapsto (x + y, x + y) \tag{8.34}$$

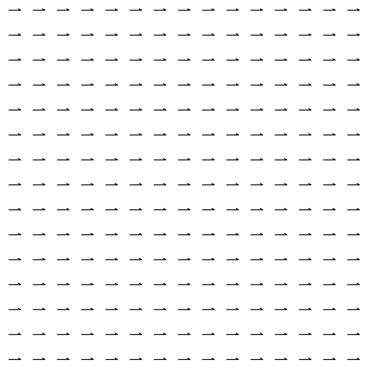
is non-constant, and assigns $v_x = v_y = x + y$ at each point (x, y); see Figure 8.14(b). For this vector field, the velocity direction is always diagonal, but the length of the vector (speed) depends on $x + y$.

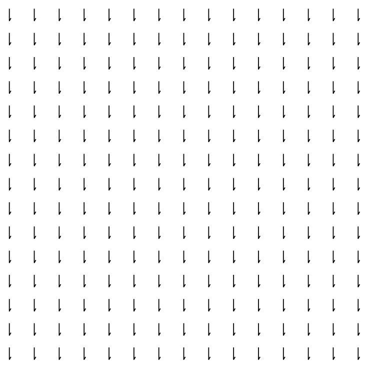
To most accurately describe the motion of features along the retina, the vector field should be defined over a spherical surface that corresponds to the locations of the photoreceptors. Instead, we will describe vector fields over a square region, with the understanding that it should be transformed onto a sphere for greater accuracy.

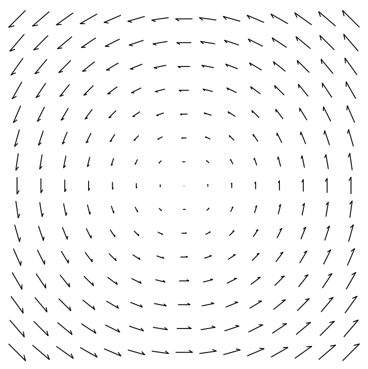
Types of vection Vection can be caused by any combination of angular and linear velocities of the viewpoint in the virtual world. To characterize the effects of different kinds of motions effectively, it is convenient to decompose the viewpoint velocities into the three linear components, v_x, v_y, and v_z, and three angular components, ω_x, ω_y, and ω_z. Therefore, we consider the optical flow for each of these six cases (see Figure 8.15):

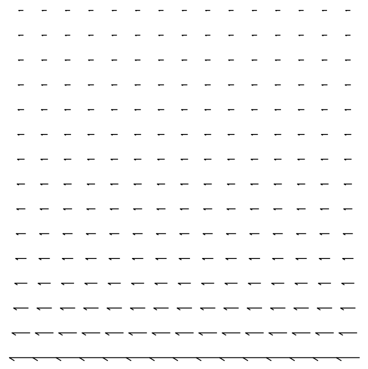
1. **Yaw vection:** If the viewpoint is rotated counterclockwise about the y axis (positive ω_y), then all visual features move from right to left at the same velocity, as shown in Figure 8.15(a). Equivalently, the virtual world is rotating clockwise around the user; however, self motion in the opposite direction is perceived. This causes the user to feel as if she is riding a merry-go-round (recall Figure 8.2).

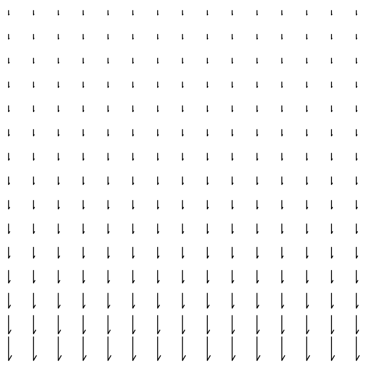
2. **Pitch vection:** By rotating the viewpoint counterclockwise about the x axis (positive ω_x), all features move downward at the same velocity, as shown in Figure 8.15(b).


(a) yaw


(b) pitch


(c) roll


(d) lateral


(e) vertical

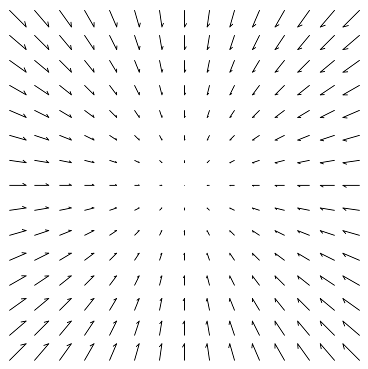

(f) forward/backward

Figure 8.15: Six different types of optical flows, based on six degrees of freedom for motion of a rigid body. Each of these is a contributing component of vection.

3. **Roll vection:** Rotating the viewpoint counterclockwise about z, the optical axis (positive ω_z), causes the features to rotate clockwise around the center of the image, as shown in Figure 8.15(c). The velocity of each feature is tangent to the circle that contains it, and the speed is proportional to the distance from the feature to the image center.

4. **Lateral vection:** In this case, the viewpoint is translated to the right, corresponding to positive v_x. As a result, the features move horizontally; however, there is an important distinction with respect to yaw vection: Features that correspond to closer objects move more quickly than those from distant objects. Figure 8.15(d) depicts the field by assuming vertical position of the feature corresponds to its depth (lower in the depth field is closer). This is a reappearance of *parallax*, which in this case gives the illusion of lateral motion and distinguishes it from yaw motion.

5. **Vertical vection:** The viewpoint is translated upward, corresponding to positive v_x, and resulting in downward flow as shown i Figure 8.15(e). Once again, parallax causes the speed of features to depend on the distance of the corresponding object. This enables vertical vection to be distinguished from pitch vection.

6. **Forward/backward vection:** If the viewpoint is translated along the optical axis away from the scene (positive v_z), then the features flow inward toward the image center, as shown in Figure 8.15(f). Their speed depends on *both* their distance from the image center and the distance of their corresponding objects in the virtual world. The resulting illusion is backward motion. Translation in the negative z direction results in perceived forward motion (as in the case of the Millennium Falcon spaceship after its jump to hyperspace in the Star Wars movies).

The first two are sometimes called *circular vection*, and the last three are known as *linear vection*. Since our eyes are drawn toward moving features, changing the viewpoint may trigger smooth pursuit eye movements (recall from Section 5.3). In this case, the optical flows shown in Figure 8.15 would not correspond to the motions of the features on the retina. Thus, our characterization so far ignores eye movements, which are often designed to counteract optical flow and provide stable images on the retina. Nevertheless, due the proprioception, the brain is aware of these eye rotations, which results in an equivalent perception of self motion.

All forms of vection cause perceived velocity, but the perception of acceleration is more complicated. First consider pure rotation of the viewpoint. Angular acceleration is perceived if the rotation rate of yaw, pitch, and roll vection are varied. Linear acceleration is also perceived, even in the case of yaw, pitch, or roll vection at constant angular velocity. This is due to the merry-go-round effect, which was shown in Figure 8.2(b).

Now consider pure linear vection (no rotation). Any linear acceleration of the viewpoint will be perceived as an acceleration. However, if the viewpoint moves at constant velocity, then this is the only form of vection in which there is no perceived acceleration. In a VR headset, the user may nevertheless perceive accelerations due to optical distortions or other imperfections in the rendering and display.

Vestibular mismatch We have not yet considered the effect of each of these six cases in terms of their mismatch with vestibular cues. If the user is not moving relative to the Earth, then only gravity should be sensed by the vestibular organ (in particular, the otolith organs). Suppose the user is facing forward without any tilt. In this case, any perceived acceleration from vection would cause a mismatch. For example, yaw vection should cause a perceived constant acceleration toward the rotation center (recall Figure 8.2(b)), which mismatches the vestibular gravity cue. As another example, downward vertical vection should cause the user to feel like he is falling, but the vestibular cue would indicate otherwise.

For cases of yaw, pitch, and roll vection at constant angular velocity, there may not be a conflict with rotation sensed by the vestibular organ because the semicircular canals measure angular accelerations. Thus, the angular velocity of the viewpoint must change to cause mismatch with this part of the vestibular system. Sickness may nevertheless arise due to mismatch of perceived linear accelerations, as sensed by the otolith organs.

If the head is actually moving, then the vestibular organ is stimulated. This case is more complicated to understand because vestibular cues that correspond to linear and angular accelerations in the real world are combined with visual cues that indicate different accelerations. In some cases, these cues may be more consistent, and in other cases, they may diverge further.

Factors that affect sensitivity The intensity of vection is affected by many factors:

- **Percentage of field of view:** If only a small part of the visual field is moving, then people tend to perceive that it is caused by a moving object. However, if most of the visual field is moving, then they perceive *themselves* as moving. The human visual system actually includes neurons with receptive fields that cover a large fraction of the retina for the purpose of detecting self motion [5]. As VR headsets have increased their field of view, they project onto a larger region of the retina, thereby strengthening vection cues.

- **Distance from center view:** Recall from Section 5.1 that the photoreceptors are not uniformly distributed, with the highest density being at the innermost part of the fovea. Thus, detection may seem stronger near the center. However, in the cases of yaw and forward/backward vection, the optical flow vectors are stronger at the periphery, which indicates that detection may be stronger at the periphery. Sensitivity to the optical flow may therefore be strongest somewhere between the center view and the periphery, depending on the viewpoint velocities, distances to objects, photoreceptor densities, and neural detection mechanisms.

- **Exposure time:** The perception of self motion due to vection increases with the time of exposure to the optical flow. If the period of exposure is very brief, such as a few milliseconds, then no vection may occur.

- **Spatial frequency:** If the virtual world is complicated, with many small structures or textures, then the number of visual features will be greatly increased and the optical flow becomes a stronger signal. As the VR headset display resolution increases, higher spatial frequencies can be generated.

- **Contrast:** With higher levels of contrast, the optical flow signal is stronger because the features are more readily detected. Therefore, vection typically occurs with greater intensity.

- **Other sensory cues:** Recall from Section 6.4 that a perceptual phenomenon depends on the combination of many cues. Vection can be enhanced by providing additional consistent cues. For example, forward vection could be accompanied by a fan blowing in the user's face, a rumbling engine, and the sounds of stationary objects in the virtual world racing by. Likewise, vection can be weakened by providing cues that are consistent with the real world, where no corresponding motion is occurring.

- **Prior knowledge:** Just by knowing beforehand what kind of motion should be perceived will affect the onset of vection. This induces a prior bias that might take longer to overcome if the bias is against self motion, but less time to overcome if it is consistent with self motion. The prior bias could be from someone telling the user what is going to happen, or it could simply by from an accumulation of similar visual experiences through the user's lifetime. Furthermore, the user might expect the motion as the result of an action taken, such as turning the steering wheel of a virtual car.

- **Attention:** If the user is distracted by another activity, such as aiming a virtual weapon or selecting a menu option, then vection and its side effects may be mitigated.

- **Prior training or adaptation:** With enough exposure, the body may learn to distinguish vection from true motion to the point that vection becomes comfortable. Thus, many users can be trained to overcome VR sickness through repeated, prolonged exposure.

Due to all of these factors, and the imperfections of modern VR headsets, it becomes extremely difficult to characterize the potency of vection and its resulting side effects on user comfort.

Further Reading

For basic concepts of vectors fields, velocities, and dynamical systems, see [2]. Modeling and analysis of mechanical dynamical systems appears in [24]. The specific problem of human body movement is covered in [29, 30]. See [10] for an overview of game engines, including issues such as simulated physics and collision detection. For coverage of particular collision detection algorithms, see [9, 20].

A nice introduction to the vestibular system, including its response as a dynamical system is [15]. Vection and visually induced motion sickness are thoroughly surveyed in [14], which includes an extensive collection of references for further reading. Some key articles that address sensitivities to vection include [1, 3, 6, 17, 18, 25, 28].

Bibliography

[1] D. Alais, C. Morrone, and D. Burr. Separate attentional resources for vision and audition. *Proceedings of the Royal Society B: Biological Sciences*, 273(1592):1339–1345, 2006.

[2] D. K. Arrowsmith and C. M. Place. *Dynamical Systems: Differential Equations, Maps, and Chaotic Behaviour*. Chapman & Hall/CRC, New York, 1992.

[3] K. W. Arthur. *Effects of Field of View on Performance with Head-Mounted Displays*. PhD thesis, University of North Carolina at Chapel Hill, 2000.

[4] K. J. Astrom and R. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, Princeton, NJ, 2008.

[5] D. C. Burr, M. C. Morrone, and L. M. Vaina. Large receptive fields for optic flow detection in humans. *Vision Research*, 38(12):1731–1743, 1998.

[6] M. Emoto, K. Masaoka, M. Sugawara, and F. Okano. Viewing angle effects from wide field video projection images on the human equilibrium. *Displays*, 26(1):9–14, 2005.

[7] R. C. Fitzpatrick and B. L. Day. Probing the human vestibular system with galvanic stimulation. *Journal of Applied Physiology*, 96(6):2301–2316, 2004.

[8] R. C. Fitzpatrick, J. Marsden, S. R. Lord, and B. L. Day. Galvanic vestibular stimulation evokes sensations of body rotation. *NeuroReport*, 13(18):2379–2383, 2002.

[9] S. Gottschalk, M. C. Lin, and D. Manocha. Obbtree: A hierarchical structure for rapid interference detection. In *Proceedings ACM SIGGRAPH*, 1996.

[10] J. Gregory. *Game Engine Architecture, 2nd Ed.* CRC Press, Boca Raton, FL, 2014.

[11] P. Guigue and O. Devillers. Fast and robust triangle-triangle overlap test using orientation predicates. *Journal of Graphics Tools*, 8(1):25–32, 2003.

[12] J. G. Hocking and G. S. Young. *Topology.* Dover, New York, 1988.

[13] A. Iserles. *A First Course in the Numerical Analysis of Differential Equations, 2nd Ed.* Cambridge University Press, Cambridge, U.K., 2008.

[14] B. Keshavarz, H. Hecht, and B. D. Lawson. Visually induced motion sickness: Causes, characteristics, and countermeasures. In K. S. Hale and K. M. Stanney, editors, *Handbook of Virtual Environments, 2nd Edition*, pages 647–698. CRC Press, Boca Raton, FL, 2015.

[15] H. Kingma and M. Janssen. Biophysics of the vestibular system. In A. M. Bronstein, editor, *Oxford Textbook of Vertigo and Imbalance*. Oxford University Press, Oxford, UK, 2013.

[16] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[17] J.-C. Lepecq, I. Giannopulu, and P.-M. Baudonniere. Cognitive effects on visually induced body motion in children. *Perception*, 24(4):435–449, 1995.

[18] J.-C. Lepecq, I. Giannopulu, S. Mertz, and P.-M. Baudonniere. Vestibular sensitivity and vection chronometry along the spinal axis in erect man. *Perception*, 28(1):63–72, 1999.

[19] M. C. Lin and J. F. Canny. Efficient algorithms for incremental distance computation. In *Proceedings IEEE International Conference on Robotics & Automation*, 1991.

[20] M. C. Lin and D. Manocha. Collision and proximity queries. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, 2nd Ed.*, pages 787–807. Chapman and Hall/CRC Press, New York, 2004.

[21] B. Mirtich. V-Clip: Fast and robust polyhedral collision detection. Technical Report TR97-05, Mitsubishi Electronics Research Laboratory, 1997.

[22] B. Mirtich. Efficient algorithms for two-phase collision detection. In K. Gupta and A.P. del Pobil, editors, *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, pages 203–223. Wiley, Hoboken, NJ, 1998.

[23] T. Möller. A fast triangle-triangle intersection test. *Journal of Graphics Tools*, 2(2):25–30, 1997.

[24] A. Ruina and R. Pratap. *Introduction to Statics and Dynamics*. Oxford University Press, Oxford, UK, 2015. Available at http://ruina.tam.cornell.edu/Book/.

[25] X. M. Sauvan and C. Bonnet. Spatiotemporal boundaries of linear vection. *Perception and Psychophysics*, 57(6):898–904, 1995.

[26] M. W. Spong, S. Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*. Wiley, Hoboken, NJ, 2005.

[27] Thomas and Finney. *Calculus and Analytic Geomtry, 9th Ed.* Addison-Wesley, Boston, MA, 1995.

[28] W. H. Warren and K. J. Kurtz. The role of central and peripheral vision in perceiving the direction of self-motion. *Perception and Psychophysics*, 51(5):443–454, 1992.

[29] V. M. Zatsiorsky. *Kinematics of Human Motion*. Human Kinetics, Champaign, IL, 1997.

[30] V. M. Zatsiorsky. *Kinetics of Human Motion*. Human Kinetics, Champaign, IL, 2002.

[31] V. M. Zatsiorsky and B. I. Prilutsky. *Biomechanics of Skeletal Muscles*. Human Kinetics, Champaign, IL, 2012.