The Wiimote and Beyond: Spatially Convenient Devices for 3D User Interfaces

Chadwick A. Wingrave, Brian Williamson, Paul Varcholik, Jeremy Rose, Andrew Miller, Emiko Charbonneau, Jared Bott, and Joseph J. LaViola Jr. ■ *University of Central Florida*

Experience with the Nintendo Wii Remote indicates that it's an imperfect harbinger of a new class of spatially convenient devices. This tutorial presents techniques for using the Wiimote in 3D user interfaces. It discusses the device's strengths and how to compensate for its limitations, with implications for future spatially convenient devices.

Devices such as the Nintendo Wii Remote, or Wiimote, (see Figure 1) provide an inexpensive and robust packaging of several useful but limited sensors, with the ability to rapidly relay information to a computer. They offer such an impressive array of sensors and hardware that 3D user interface (3DUI) researchers have largely been caught off-balance. How do we use and classify these devices? Are they the sought-after silver bullet for 3D tracking or just a step on the path to a common hardware platform for ubiquitous 3DUIs?

Although the Wiimote can be useful for 3DUI input, in almost every important way it differs from general-purpose input hardware[1] found in research labs and commercial applications (see the sidebar). Despite this, no one has systematically evaluated the device in terms of what it offers 3DUIs. This might be due to the high quality of Wiimote hacking (for example, as head trackers, finger trackers, and whiteboards[2]), which has led people to overestimate the Wiimote's capabilities.

Our experience with the Wiimote and related hardware has led us to conclude that it's an imperfect harbinger of a new class of *spatially convenient* devices. This tutorial discusses techniques for exploiting the Wiimote's capabilities as such a device. We also discuss its strengths and methods to compensate for its limitations in 3DUIs, arguing that you can extrapolate these methods to future spatially convenient devices.

The Wiimote as a Spatially Convenient Device

Spatially convenient devices involve three important aspects:

- *Spatial data*. The device provides 3D input data, be it partial, error-prone, or conditional.
- *Functionality*. The device packs a range of useful sensors, emitters, and interface implements.
- *Commodity design*. The device is inexpensive, durable, easily configurable, and robust.

Regarding spatial data, traditional 3D hardware trackers present information in 6 degrees of freedom (DOF) in a tracked space with relatively good precision. In contrast, a Wiimote presents three axes of acceleration data in no particular frame of reference, with intermittent optical sensing. (The Wii MotionPlus gyroscope attachment can add three axes of orientation change, or angular velocity.) Although this means the Wiimote's spatial data doesn't directly map to a real-world position, the device can be employed effectively under constrained use.[3–8]

Regarding functionality, traditional 3D hardware might come with a few buttons. The Wiimote incorporates several buttons, some in a gamepad and trigger configuration, and has a speaker, programmable LEDs, and a rumble device.

Regarding commodity design, 3D hardware can require extensive installations and environment

 Published by the IEEE Computer Society

instrumentation and can be difficult to work with. The Wiimote is easy to set up, turn on, and maintain.

Overall, the Wiimote incorporates many useful input and output features in an inexpensive, consumer-oriented, easy-to-replace, and easy to repurchase package. This lets 3DUI developers (researchers as well as homebrew engineers) use and modify it to best serve their needs.

Wiimote Basics

To use the Wiimote in 3DUIs, you need to know about

- connecting the Wiimote to the system,
- its frames of reference (FORs),
- the sensor bar connection (SBC),
- dealing with accelerometer and gyroscope data, and
- other design considerations.

For technical details and specs, see the WiiBrew Web site (www.wiibrew.org).

Connecting the Wiimote

The Wiimote connects to a Wii game console or computer wirelessly through Bluetooth. When you press both the 1 and 2 buttons simultaneously or press the red Sync button in the battery case, the Wiimote's LEDs blink, indicating it's in discovery mode. Make sure your computer has a Bluetooth adapter and proper drivers. Currently, each OS and library handles the connection process differently.

PCs. Here are the basic steps (for extra assistance, seek online help such as at www.brianpeek.com):

1. Open Bluetooth Devices in the Control Panel.
2. Under the Devices tab, click the Add button.
3. Put the Wiimote into discovery mode (continuously reenter this mode during this process as it times out).
4. Select Next, select the Wiimote to connect to, and select Next again.
5. Don't use a passkey, and select Next, then Finish.

If a failure occurs, repeat this process until the PC connects to the Wiimote. A simple start for a PC to retrieve data from a Wiimote is the WiimoteLib (www.wiimotelib.org).

Macs. The setup needs to run only once, and future Wiimote connections are simpler. To set up the Wiimote, run the Bluetooth Setup Assistant utility and follow these steps:

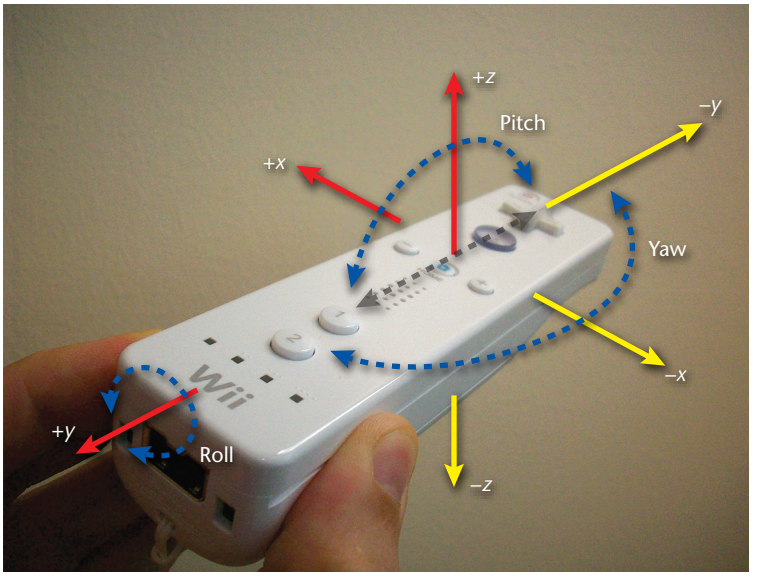


Figure 1. The Wiimote, with labels indicating the Wiimote's coordinate system. Multiple coordinate systems and partial spatial data make the Wiimote difficult to design for.

1. Select the Any Device option.
2. Put the Wiimote into discovery mode.
3. Select the device (Nintendo RVL-CNT-01) and set the passkey options to "Do not use a passkey."
4. Press Continue until the Mac is connected; then quit the Setup Assistant.

To connect in the future, in the Bluetooth section of System Preferences, select the Wiimote entry and set it to connect. Then, put the Wiimote into discovery mode, and the Wiimote will connect. The OS can keep some applications from connecting to the Wiimote. In those cases, disconnect the Wiimote and let the application connect to the Wiimote.

Frames of Reference

Figure 1 shows the first FOR—the Wiimote's own. The *x*, *y* and *z* axes are labeled, along with the rotation about each axis: pitch, roll, and yaw, respectively. A second FOR is the Earth's, important because the Wiimote's accelerometers detect the Earth's gravity. A third FOR is the Wiimote's relationship to the sensor bar.

Three examples clarify how these FOR interact. To begin, a user is considered holding a Wiimote naturally when +*z* is up in both the Wiimote's and the Earth's FOR and the Wiimote's front points away from the user and toward a sensor bar, which is usually on top of the display.

In the first example, the user moves the Wiimote toward the sensor bar. This results in acceleration reported in the *y*-axis of both the Earth and Wiimote FORs and the sensor bar reporting decreased distance between the Wiimote and it.

A Comparison of Input Devices

Input devices use several different approaches to tracking, none of which is the silver bullet to solve all tracking needs.[1] Table A compares a few representative commodity and traditional input devices.[2] Commodity devices include those designed for wide distribution and uptake, and as such are built more robustly, with harder cost constraints. Traditional 3D input devices have fewer cost constraints and require more operator care, but they offer more functionality.

As the table shows, each system has its limitations and advantages, suggesting that when choosing an input device, you should take into account the requirements of the system you're developing. The 3Dconnexion SpaceExplorer is akin to a trackball. However, instead of using a ball, the user grasps and manipulates a stubby column, transforming his or her force on the column into 6-degree-of-freedom (DOF) data. The SpaceExplorer is cheap and good at 6-DOF input, but it has only one tracked point and is isomorphic.

The Polhemus Fastrak uses an emitter to produce a magnetic field and receivers to identify their position in relation to the emitter. Each tracked point has a cable going back to the tracker. The Fastrak is limited in its tracked points, update, range and is susceptible to magnetic distortion, but it gives 6-DOF data for lower cost.

The InterSense IS-900 is a hybrid tracking system, using ultrasonic emitters placed in carefully measured positions to emit sounds that are triangulated by microphones on receivers. As a hybrid system, it also uses accelerometers to provide faster updates and incremental updates when the microphones are occluded. Each tracked point requires a cable, but a costlier wireless option exists. The IS-900 costs more than the other devices, its receivers are larger and fragile, and installation can be complicated, but it gives good 6-DOF input.

Vicon's system uses multiple cameras that emit and capture infrared light reflected off retroreflective markers. The system is costly, has occlusion issues, and has a complicated setup, but it tracks many 6-DOF points at fast updates with small, nonwired markers.

Finally, the Wiimote is a commodity device originally shipped with the Nintendo Wii gaming system; it uses accelerometers and a single infrared camera for tracking. It transmits data wirelessly using Bluetooth. The Wiimote tracks only a single point with nonpositional data but is cheap, robust, and wireless.

References

1. G. Welch and E. Foxlin, "Motion Tracking: No Silver Bullet, but a Respectable Arsenal," *IEEE Computer Graphics and Applications*, vol. 22, no. 6, 2002, pp. 24–38.
2. E. Foxlin, "Motion Tracking Requirements and Technologies," *Handbook of Virtual Environments: Design, Implementation, and Applications*, K. Stanney, ed., Lawrence Erlbaum Associates, 2002, pp. 163–210.

Table A. Analyzing commodity and traditional input devices.

	3Dconnexion SpaceExplorer	Polhemus Fastrak	InterSense IS-900	Vicon	Wiimote
Data completeness	6 degree-of freedom (DOF) isometric forces for one point; fast updates	6 DOF for one to four points in a 5-ft. range; 120-Hz update divided by the number of tracked points	6 DOF for many points in potentially large spaces; fast updates	6 DOF for many nonoccluding points in potentially large spaces; fast updates	3D acceleration up to 30 ft. and a single camera up to 16 ft. for one point; 100-Hz update
Device robustness	Commodity design	Sensitive to magnetic distortion and breakage; moderate jitter	Susceptible to breakage	Susceptible to occlusion	Designed for durability; can be sensitive to sunlight
Setup	Install drivers	Set up emitter and configure software	Carefully calibrate emitter bars and configure software	Carefully calibrate cameras and configure software	Install software and Bluetooth hardware
Encumbrance and space usage	Uses space commonly reserved for a mouse	Requires holding or wearing receivers and a cable for each tracked point	Requires holding or wearing larger, fragile trackers with cables (a wireless option also exists)	Requires holding or wearing tracked points	Hold in hand or strap to a limb
Cost (US$)	~300	A few thousand	Many thousands	Many thousands	40 and dropping

In the second example, the user rotates the Wiimote down to point toward the earth (a 90° pitch). When the user again moves the Wiimote directly toward the sensor bar in front of him or her, the Wiimote reports acceleration in its *z*-axis. However, the Earth's FOR has acceleration in its *y*-axis because the 90° downward pitch didn't change the Earth's FOR. The sensor bar has no FOR because as the Wiimote rotates away from the sensor bar, the Wiimote loses contact with it.

The third example has the same configuration as the second example, but with the sensor bar on the ground directly below the Wiimote, at the user's feet. When the user repeats that forward motion, the sensor bar reports the Wiimote moving up in the sensor bar's *z*-axis, whereas the Earth and Wiimote data are the same as in the previous example.

These examples show that each FOR captures important and different information. The following sections explain this in more detail.

The Sensor Bar Connection

The SBC is one of the Wiimote's two primary spatial sensors. It occurs when the Wiimote's infrared optical camera points at a sensor bar and sees the infrared (IR) light emitted by its LEDs. A sensor bar has LEDs on each side (see Figure 2), with a known width between them. This produces IR blobs that the Wiimote tracks and reports in *x* and *y* coordinates, along with the blob's width in pixels. To improve tracking distance (the Wiimote can sense blobs up to 16 feet away), the sensor bar LEDs are spread in a slight arc, with the outer LEDs angled out and the inner LEDs angled in. Actually, any IR source will work, such as custom IR emitters, candles, or multiple sensor bars (provided you have the means to differentiate between the sensor bars).

When you point the Wiimote at the sensor bar, it picks up two points, $P_L = (x_L, y_L)$ and $P_R = (x_R, y_R)$, from the LED arrays. You can easily calculate the midpoint between P_L and P_R and, with some smoothing, use it as a 2D cursor or pointer on the display. In addition, if you rotate the Wiimote about its *y*-axis (when held naturally with +z up), you can calculate its roll with respect to its *x*-axis, using

$$roll = \arccos\left(\vec{x} \cdot \frac{\vec{v}}{\|\vec{v}\|}\right),$$

where $\vec{x} = (1,0)$ and $\vec{v} = P_L - P_R$.

Combining information from the sensor bar and the Wiimote's optical sensor also lets you determine the Wiimote's distance from the sensor bar, using triangulation.[9] You calculate the distance *d* between the sensor bar and Wiimote using

$$d = \frac{w/2}{\tan(\theta/2)}$$

$$w = \frac{m \cdot w_{\text{img}}}{m_{\text{img}}}$$

$$m_{\text{img}} = \sqrt{(x_L - x_R)^2 + (y_L - y_R)^2},$$

Figure 2. A Wiimote sensor bar has two groups of infrared LEDs at fixed widths. With this, the Wiimote's IR camera can determine its position relative to the sensor bar.

where θ is the optical sensor's viewing angle, m is the distance between the sensor bar's left and right LEDs, w_{img} is the width of the image taken from the optical sensor, and m_{img} is the distance between P_L and P_R taken from the optical sensor's image. θ, w_{img}, and m are all constants based on the device hardware specifications.

The previous calculation works only when the Wiimote points directly at the sensor bar (orthogonally). When the Wiimote is off-axis from the sensor bar, you need more information to find depth. To calculate depth, you can use the relative sizes of the points on the optical sensor's image. To find *d* in this case, compute the distances corresponding to the left and right points on the optical sensor's image:

$$d_L = \frac{w_L/2}{\tan(\theta/2)}$$

$$d_R = \frac{w_R/2}{\tan(\theta/2)},$$

using

$$w_L = \frac{w_{\text{img}} \cdot diam_{\text{LED}}}{diam_L}$$

$$w_R = \frac{w_{\text{img}} \cdot diam_{\text{LED}}}{diam_R},$$

where $diam_{\text{LED}}$ is the diameter of the actual LED marker from the sensor bar, and $diam_L$ and $diam_R$ are the diameters of the points on the optical sensor's image. With d_L and d_R, you can calculate the Wiimote's distance to the sensor bar as

$$d = \sqrt{d_L^2 + (m/2)^2 - 2d_L(m/2)^2\cos(\phi)},$$

where

$$\cos(\phi) = \frac{d_L^2 m^2 - d_R^2}{2md_L}.$$

With d, d_L, and m, you can also find the Wiimote's angular position with respect to the sensor bar:

$$\alpha = \arccos\left(\frac{d^2m^2 - d_L^2}{md_L}\right).$$

The Three-Axis Accelerometer

The Wiimote's other primary spatial sensor is its

three-axis accelerometer, common in many devices such as cell phones, laptops, camcorders, and unmanned aerial vehicles. The accelerometer reports acceleration in the device's *x*, *y*, and *z* dimensions, expressed in the unit of gravity *g*. This reported acceleration vector is composed of the Wiimote's actual acceleration a' and the Earth's gravity a_g, so obtaining the actual acceleration requires subtracting the gravity vector from the Wiimote's reported acceleration, *a*. Additionally, the gravity vector is constantly oriented toward the Earth (0, 0, 1 in the Earth's FOR), so you can use it to discover part of the Wiimote's orientation in the Earth's FOR:

$$pitch = \arctan\left(a_{gz} / a_{gy}\right)$$

$$roll = \arctan\left(a_{gz} / a_{gx}\right).$$

A MotionPlus-augmented Wiimote provides information on changes to the Wiimote's orientation, alleviating many of the device's data limitations.

Unfortunately, determining yaw in the Earth's FOR isn't possible because the Earth's gravity vector aligns with its *z*-axis.

Another unfortunate issue is that determining the actual acceleration of the Wiimote is problematic owing to the confounding gravity vector. For you to determine the actual acceleration, one of the following must take place:

- the Wiimote must be under no acceleration other than gravity so that you can accurately measure the gravity vector (in which case, you already know the actual acceleration is zero),
- you must make assumptions about the Wiimote's orientation, thus allowing room for errors, or
- you must determine the orientation by other means, such as by the SBC or a gyroscope (we discuss this in more detail later).

The implications for orientation tracking by the accelerometers are that the Wiimote's orientation is only certain when it is under no acceleration. For this reason, many Wii games require that users either hold the Wiimote steady for a short period of time before using it in a game trial or have it pointed at the screen and oriented by the SBC.

MotionPlus

This attachment uses two gyroscopes to report angular velocity along all three axes (one dual-axis gyro for *x* and *y* and a single-axis gyro for *z*). Mechanical gyroscopes would typically be too large and expensive for a Wiimote. So, it uses MEMS (microelectromechanical system) gyroscopes that operate using a vibrating structure, are inexpensive, use little power, and are fairly accurate.[7] A MotionPlus-augmented Wiimote provides information on changes to the Wiimote's orientation, alleviating many of the device's data limitations. With this, Nintendo is attempting to improve the orientation accuracy of the device.

The MotionPlus was released only recently and at this time isn't yet fully reverse engineered. It reports orientation changes in two granularities—fast and slow—with fast being roughly four times the rate per bit. The gyroscope manufacturer reports that the two gyroscopes have a linear gain but that the different gyroscopes report values in two different scales, so there's no single scaling factor. Additionally, temperature and pressure changes can impact this scale factor and change the value associated with zero orientation change.

Merging the acceleration and gyroscopic data isn't simple; both sensors have accuracy and drift errors that, albeit small, amount to large errors over short time periods. When using the SBC, you can compensate for these accumulating errors by providing an absolute orientation and position. Researchers have improved orientation by merging accelerometer and gyroscopic data but didn't test a system under translational motion.[6] Other research has shown that you can combine accelerometers and gyroscopes for accurate position and orientation tracking.[10] In addition, researchers have successfully used Kalman filters to merge accelerometer and gyroscopic data.[3,11]

Design Considerations

Because of the Wiimote's many limitations, you must take into account three design considerations.

Waggling motions. "Cheating" motions (or less fatiguing and comfortable movements, depending on your perspective) are a side effect of Wiimote input limitations. Game designers might intend for games to encourage exercise, breaking the stereotype of the lazy video game player. However, the Wiimote's inability to detect actual position change lets users make only small or limited "waggling" motions, which the Wiimote interprets as full movement. The result is boxing games played by tapping the Wiimote, tennis games played with wrist flicks, and

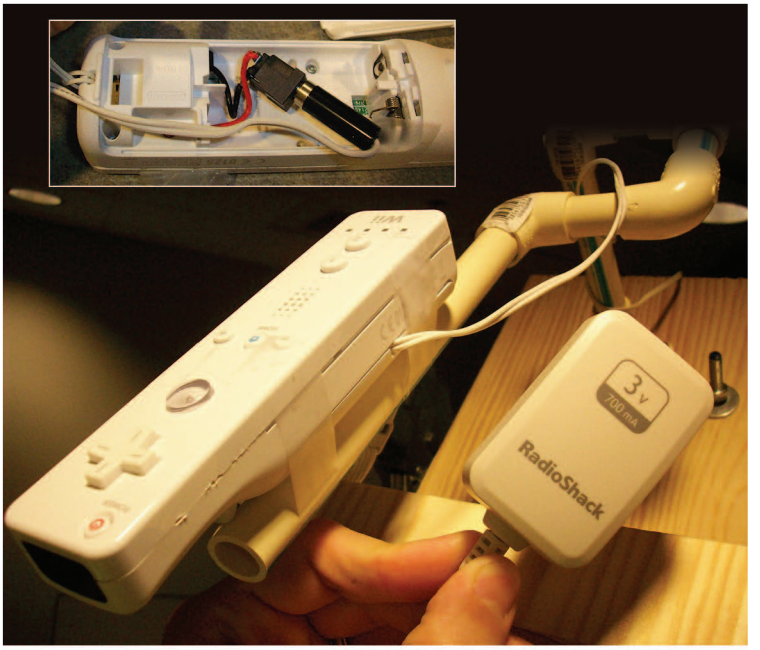
many games winnable by simply moving the device randomly. Although this is still fun for gamers, it limits the Wiimote's utility for 3DUIs and exercise and health gaming, unless you employ better hardware and data-interpretation methods.

Compensation by "story." Whenever possible, the easiest means of compensating for input hardware limitations is through the use of story. By "use of story," we mean that by careful manipulation of the users' tasks and their goals, the shortcomings of the hardware can be avoided. This is common in Wii gaming, in which accuracy is second to enjoyment and playability. As a first example, consider the inherent drift in Wiimotes. You can compensate for the drift by requiring the user to return to a known position from which you can assume the Wiimote's orientation, or you can create an SBC by requiring the user to point at an on-screen button to begin a task.

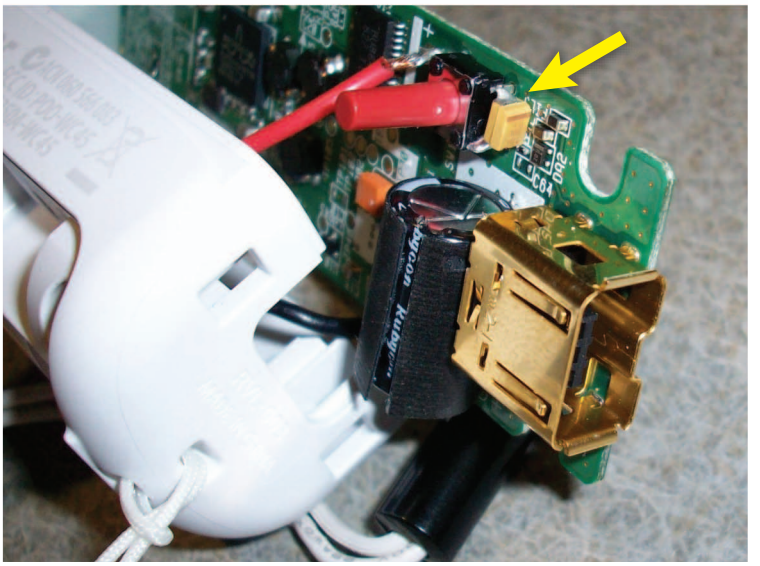
As a second example, story can engage users, instructing them to freeze their hand at a known orientation. From this orientation assumption, the gravity vector can be assumed and actual acceleration calculated. Several examples come from games. The We Cheer game instructs players to hold the Wiimote as if they're holding pom-poms, WarioWare tells players how to hold the Wiimote for each minigame, and Wii Sports Boxing makes players raise their hands when preparing for a fight. Researchers commonly guide participants by story, instructing them to enter start positions before a trial. For example, in studies of 3DUI selection techniques, researchers often have users select an object to begin a trial.

Mounting as a camera. Nintendo designed the Wiimote for use with a console device. When used in 3DUIs as a camera, Wiimotes are often mounted in the environment, usually up high or at specific orientations, so that they can observe IR-emitting LEDs. Consequently, changing batteries or hitting the Sync button becomes a problem. To remedy this, you can solder a power adapter plug directly to the Wiimote's board and place the plug and adapter in the battery compartment (see Figure 3).

So that the Wiimote enters discovery mode on power-up, you can solder a capacitor between the x-y pins on the Sync button (see Figure 3b). The capacitor fits nicely between the pins, which allows for easy soldering. The capacitor is a polar 10-microfarad (μf) part, so it's important to mount it with the red stripe toward the board's outside edge.



(a)



(b)

Figure 3. Mounting the Wiimote as a camera. (a) You can more easily mount the Wiimote if it uses an adapter for power, which you can achieve by soldering the adapter plug to the board. (b) A small capacitor soldered to the red Sync button starts the Wiimote in discovery mode.

Interpreting Wiimote Data

Here we review three basic types of Wiimote data interpretation in terms of our experiences: integration, identifying heuristics, and gesture recognition. We also contrast our experiences with the Wiimote, a spatially convenient device, and with a typical 3D tracking system.

Integration

Because acceleration is the rate of velocity change, which is the rate of positional change, you can theoretically integrate the Wiimote acceleration data to find velocity and reintegrate it to find the Wiimote's actual position change. You should also

be able to integrate the gyroscope's angular velocity to achieve absolute orientation. However, this approach has three significant limitations.[12]

First, acceleration jitter can lead to significant positional deviations during a calculation. So, you can ignore accelerations below a threshold because they're most likely the result of jitter. Alternatively, a smoothing function can help reduce jitter.

Second, you must remove the gravity vector from the reported acceleration before computing velocity. If the SBC or a gyroscope is available, you can possibly infer the Wiimote's orientation and realize the gravity vector as a 1-g downward force; otherwise, you must determine the gravity vector on the fly.

One Man Band used a Wiimote to simulate the movements necessary to control the rhythm and pitch of several musical instruments.

One way to compute the gravity vector is by watching the derivative of the acceleration (or jerk data). When this is close to zero (that is, no acceleration change is computed) and the reported acceleration magnitude is close to 1 g, thereby eliminating cases of user-induced constant acceleration, you can assume the Wiimote is reporting only the gravity vector. Subtracting this vector from future accelerations results in the acceleration being directly attributable to user action, assuming the Wiimote maintains its orientation.

Third, orientation must be accurate; even slight errors produce large positional errors after movement. For example, one meter of travel after a five-degree orientation change (well within the variance of the hand) results in an error of nearly 9 cm. A larger 30-degree orientation change results in an error of 50 cm. Additionally, unaccounted-for orientation changes can give very wrong results. For example, raising the Wiimote a foot, inverting it, and lowering it to its original location will result in no actual positional change, but as computed will result in a two-foot upward movement in the Wiimote's FOR. You can address this only by using the SBC or a gyroscope, because the Wiimote's accelerometers can't easily provide a gravity vector while the Wiimote is moving, as we discussed earlier. In our work with a locomotion interface for American football, double integration let users control the application using a Wiimote and have their body movements maneuver the quarterback.[11] We achieved this in three steps. First, we attached a Wiimote to the center of the user's chest, which was close to the body's center of mass. Although this improved the reported acceleration data, it broke the SBC. Second, we passed the Wiimote acceleration data through an exponential smoothing filter:

$$\vec{a}_{current} = \alpha\vec{a}_i + (1-\alpha)\vec{a}_{i-1}, \quad (1)$$

where $\alpha = 0.9$. An alternate approach is to use a Kalman filter, but this would require more computation.[13] Third, we performed the double-integration step on the smoothed acceleration data. Owing to these three steps, the Wiimote was responsive, and the user seemed to move relatively accurately for that application.

To further evaluate the control's accuracy, we performed tests in which the user moved from a starting location and then back. These tests showed little error in position over short time periods (5 to 10 seconds), but only when users maneuvered in an unnaturally upright and stiff fashion.

Identifying Heuristics

The Wiimote provides a raw data stream, but you can use heuristics to interpret and classify the data. Whether you use heuristics on their own or as features for gesture recognition (which we discuss later), their higher-level meaning is more useful for 3DUI design and implementation.

We used such heuristics in two of our projects. One Man Band used a Wiimote to simulate the movements necessary to control the rhythm and pitch of several musical instruments.[14] Careful attention to the Wiimote data enabled seamless transitions between all musical instruments. RealDance explored spatial 3D interaction for dance-based gaming and instruction.[15] By wearing Wiimotes on the wrists and ankles, players followed an on-screen avatar's choreography and had their movements evaluated on the basis of correctness and timing.

Poses and underway intervals. A *pose* is a length of time during which the Wiimote isn't changing position. Poses can be useful for identifying held positions in dance, during games, or possibly even in yoga. An *underway interval* is a length of time during which the Wiimote is moving but not accelerating. Underway intervals can help identify smooth movements and differentiate between, say, strumming on a guitar and beating on a drum.

Because neither poses nor underway intervals have an acceleration component, you can't dif-

ferentiate them by accelerometer data alone. To differentiate the two, an SBC can provide an FOR to identify whether the Wiimote has velocity. Alternatively, you can use context, tracking Wiimote accelerations over time to gauge whether the device is moving or stopped. This approach can be error prone, but you can successfully use it until you reestablish the SBC.

Poses and underway intervals have three components. First, the *time span* is the duration in which the user maintains a pose or an underway interval. Second, the *gravity vector's orientation* helps verify that the user is holding the Wiimote at the intended orientation. Of course, unless you use an SBC or a gyroscope, the Wiimote's yaw won't be reliably comparable. Third, the *allowed variance* is the threshold value for the amount of Wiimote acceleration allowed in the heuristic before rejecting the pose or underway interval.

In RealDance (see Figure 4), poses were important for recognizing certain dance movements. For a pose, the user was supposed to stand still in a specific posture beginning at time t_0 and lasting until $t_0 + N$, where N is a specified number of beats. So, we represented the score as the percentage of the time interval during which the user successfully maintained the correct posture.

Figure 4. RealDance uses Wiimotes to recognize dance moves, without requiring the sensor bar connection. These are not simple "waggling" movements but actual movements with a one-to-one correspondence to reality.

Impulse motions. An impulse motion is characterized by a rapid change in acceleration, easily measured by the Wiimote's accelerometers. A good example is a tennis or golf club swing in which the Wiimote motion accelerates through an arc or a punching motion, which contains a unidirectional acceleration.

An impulse motion has two components, which designers can tune for their use. First, the time span of the impulse motion specifies the window over which the impulse is occurring. Shorter time spans increase the interaction speed, but larger time spans are more easily separable from background jitter. The second component is the maximum magnitude reached. This is the acceleration bound that must be reached during the time span in order for the Wiimote to recognize the impulse motion.

You can also characterize impulse motions by their direction. The acceleration in a punch is basically a straight impulse motion, a tennis swing has an angular acceleration component, and a golf swing has both angular acceleration and even increasing acceleration during the follow-through when the elbow bends. All three of these impulse motions, however, are indistinguishable to the Wiimote, which doesn't easily sense these orientation changes. For example, the punch has an acceleration vector along a single axis, as does the tennis swing as it roughly changes its orientation as the swing progresses. You can differentiate the motions only by using an SBC, using a gyroscope, or assuming that the Wiimote orientation doesn't change.

RealDance used impulse motions to identify punches. A punch was characterized by a rapid deceleration occurring when the arm was fully extended. In a rhythm game, this instant should line up with a strong beat in the music. We scored an impulse motion by considering a one-beat interval centered on the expected beat. For the Wiimote corresponding to the relevant limb, we then selected the time sample in the time span corresponding to the maximal acceleration in the Wiimote's longitudinal axis. If this maximal acceleration was below a threshold, we concluded that no punch occurred, and the score was zero. Otherwise, we computed the score from the distance to the expected beat. If the gesture involved multiple limbs, the maximal acceleration value had to be greater than the threshold for all Wiimotes involved. The average of all individual limb scores served as the gesture's overall score.

Impact events. An impact event is an immediate halt to the Wiimote due to a collision, characterized by an easily identifiable acceleration bursting across all three dimensions. Examples of this event

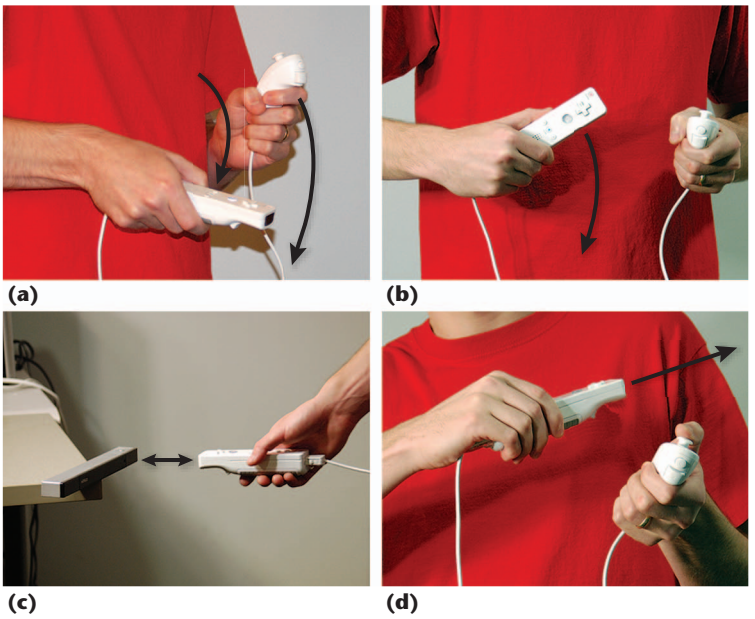


Figure 5. One Man Band differentiated between multiple Wiimote gestures using mostly simple modal differentiations for (a) drums, (b) guitar, (c) theremin, and (d) violin. To the player, changing instruments only required orienting the Wiimote to match how an instrument would be played.

include the user tapping the Wiimote on a table or a dropped Wiimote hitting the floor. To identify an impact event, compute the change in acceleration (jerk) vectors for each pair of adjacent time samples. Here, t_k corresponds to the largest magnitude of jerk:

$$t_k = \arg\max_T \left| A_t - A_{t-1} \right|.$$

If the magnitude is larger than a threshold value, an impact has occurred.

RealDance used impact motions to identify stomps. If the interval surrounding a dance move had a maximal jerk value less than a threshold, we concluded that no impact occurred, and the score was zero. Otherwise, we calculated the score the same way as for an impulse motion. One Man Band also used impact events to identify when a Nintendo Nunchuk controller and Wiimote collided, which is how users played hand cymbals.

Modal differentiation. You can use easily recognized modes in an interface to differentiate between functionality. For example, a button's semantics can change as the Wiimote changes orientation, or the Wiimote's pitch might differentiate between a system's states. Although modes can lead to errors when users are unaware of them, constant user action such as holding a button or pose can lead to quasimodal states that are easily understood and useful. This is important because the Wiimote has several attachments, such as the Nunchuk, and multiple buttons that you can use to create quasimodes.

In One Man Band, the multi-instrument musical interface (MIMI) differentiated between five different instruments by implementing modal differences based on the Wiimote's orientation. Figure 5 shows four of these. If the user held the Wiimote on its side and to the left, as if playing a guitar, the application interpreted impulse motions as strumming motions. If the user held the Wiimote to the left, as if playing a violin, the application interpreted the impulse motions as violin sounds.

To achieve this, the MIMI's modal-differentiation approach used a normalization step on the accelerometer data to identify the most prominent orientation:

$$\vec{a}_{norm} = \frac{\vec{a}}{\|\vec{a}\|},$$

followed by two exponential smoothing functions (see Equation 1). The first function, with an α of 0.1, removed jitter and identified drumming and strumming motions. The second function, with an α of 0.5, removed jitter and identified short, sharp gestures such as violin strokes. The MIMI also used the Nunchuk's thumbstick to differentiate between the bass and guitar.

Gesture Recognition

Identifying user gestures is an important part of interaction with a Wiimote. We divide gesture identification into heuristic-based and machine-learning approaches.

Heuristic-based approaches. In these approaches, the designer manually identifies the heuristics that classify a particular gesture and differentiate it from other gestures that the system recognizes. Researchers have used this approach for handwriting recognition.[16] Although this manual approach can be time consuming and tedious, especially as the set of recognized gestures grows, it's also highly understandable. Understandability is useful when new gestures are added to the gesture set and must be differentiated from past gestures.

In the American-football application, we created multiple heuristics to understand locomotion and dodging tasks. The application performed locomotion, the forward-motion component of travel, by looking for impulse motions up and down as the user ran in place. When the application observed an impulse motion of a high-enough magnitude,

it moved the user down the field along his or her current orientation.

Dodging tasks, which the application identified using the Wiimote and an SBC, included jumping, spinning, and juking (a football move in which the runner moves sharply to the side to confuse and avoid a potential tackler). The application identified jumping by up-and-down impulse motions that had a greater magnitude than locomotion movements. A Wiimote by itself couldn't easily identify spinning without the SBC, but with the MotionPlus, the SBC was no longer necessary. Simultaneous sideways impulse motions from two Wiimotes identified juking.

Simple heuristics such as the ones we've discussed can be effective, especially when composed into more complex heuristics. The recognition of the American-football application has differentiation issues even though the individual gestures are readily classifiable. For example, a jump is difficult to differentiate from running in place, but this can be improved by composition. Up-and-down impulse motions followed by an impact event, when the user hits the ground, achieves this.

Machine-learning approaches. These approaches automate much of gesture recognition and are popularly used in sketching interfaces.[16] Wiimote gestures to be recognized consist of smaller features in the Wiimote data stream. Examples include the maximum and minimum acceleration vectors in a period of time, the acceleration vector's change over a time period, and the heuristics we've described.

These features define a feature vector, $\mathbf{f} = [f_1, \ldots, f_F]$. You then collect example gestures for training. Then, a machine-learning approach, such as those described in *Pattern Recognition*,[17] matches the feature vector to the set of gestures C, differentiating between gestures. If recognition fails, you can add new features to $\mathbf{f}$ that better identify or distinguish between gestures, and run the machine-learning approach again. Additionally, you can apply hidden Markov models to improve gesture recognition over time.

We used this approach in the recognition of one and two-handed military hand signals.[18] AdaBoost was used on a feature vector of 29 different Wiimote features.[16]

The Wiimote versus the InterSense IS-900

Given the issues with Wiimote data, why use a Wiimote and not a typical 3D tracking system? To address this question, we consider how using the InterSense IS-900 tracking system would have affected our projects. The IS-900 is a hybrid tracking system, using slower ultrasonic triangulation data to update faster accelerometer data. It is commonly used for 3D tracking, is robust, and is fairly accurate for most 3D tracking issues, making it a good model for comparison (see the sidebar).

The IS-900 is much more accurate and useful than the Wiimote because it gives 6-DOF information without any user-required computation. However, it has cables connecting the tracked receivers to the IS-900 box, which can become entangled when the user jumps, spins, and moves his or her arms and legs. The IS-900 has a wireless option, but this still requires a short cable connecting the tracked receiver to the belt-mounted wireless emitter.

The Wiimote's low cost means that you can use applications without substantially investing in hardware.

Also, the IS-900 uses ultrasonic triangulation to identify the receivers' position in the environment. In the applications we've described, poses and movements would have often occluded the receivers, allowing for drift (similarly to a Wiimote).

In addition, the IS-900 can't withstand the types of impacts a Wiimote can withstand. Stomping the feet, quick punches, and tapping the IS-900 receiver against something else would break the relatively fragile receiver.

Also, the Wiimote's low cost means that you can use applications without substantially investing in hardware.

Finally, the Wiimote requires little setup other than positioning the sensor bar (if needed) and connecting the Wiimote to the system. So, you can easily transport Wiimote applications.

Designing for Universal 3D Interaction

Although the Wiimote has limitations for universal 3D interaction tasks,[4] no single limitation completely prevents its use in a 3DUI. On the basis of this, we've developed the following guidelines.

Selection

This task—choosing one or more objects—is a useful starting point for manipulation tasks or for interacting with buttons or other widgets. Two basic egocentric metaphors exist for selection. *Virtual hand* techniques involve reaching to grab objects, with a correspondence between the virtual hand

and the user's hand. These techniques are useful for differentiating between occluding objects and near and far objects during selection. *Virtual pointer* techniques involve defining a vector along which the user selects intersecting or nearly intersecting objects. These techniques require less arm movement and can be faster than virtual-hand techniques.

Because virtual-hand techniques require depth data, the Wiimote's ability to specify depth relates directly to its effectiveness. This is critical because Wiimote depth data is error prone, even under optimal conditions when an SBC exists. Unfortunately, SBCs aren't always maintained because users often hold a Wiimote in a power grip pointing up (like a hammer). So, pointing at the sensor bar requires effort and effectively operates as a virtual-pointing technique anyway.

For controlling just orientation manipulation, a Wiimote is adequate, but most manipulation tasks pair position and orientation manipulation.

3DUI techniques can help you design around these limitations. Originally, virtual-pointer selection employed *reeling* to reel objects forward or backward along the selection vector.[4] Therefore, you can also effectively use reeling to reel the virtual hand along its orientation vector. Discrete buttons or incorporating the user's reaching (that is, extending and pulling in the arm control reeling) can achieve this. You could even steer the virtual hand around the scene by mapping the Wiimote's offset vector from a predefined position to the virtual hand's velocity vector. In this way, Wiimote depth errors only minimally impact the virtual hand's control. Both reeling and steering limit the jitter effect because velocity jitter is smoother for control than positional jitter.

For virtual-pointer techniques, the Wiimote alone can't define a pointing vector without an SBC or a gyroscope. This is because you can't determine yaw from accelerometer data alone. Fortunately, although typically it's difficult to assume an SBC exists, a selection task by definition has the user pointing at something on the screen. So, the SBC assumption holds in a natural way. An exception to this would be head-mounted displays (HMDs), in which the display is attached to the user and not the environment. However, most commodity setups don't use HMDs. You can use gyroscopes to generate a yaw vector, but their inherent drift requires constant realignment, which again requires an SBC.

Another option enables the Wiimote alone to specify a partial pointing vector. Yaw is vital to specifying the selection vector. However, if the user orients and holds the Wiimote such that roll and not yaw is lost (that is, pointing it up or down), virtual pointing becomes possible with yaw and pitch. This is because roll about the selection vector is lost because the selection vector is a ray (although manipulation after selection might be affected). To achieve this partial pointing vector, you can use story to affect the user's Wiimote grip.

In summary:

- Virtual-hand techniques can have difficulties because specifying depth is error-prone. Improving their depth-specifying capabilities can improve their effectiveness.
- Virtual pointing can be effective when the Wiimote remains pointed at the sensor bar or you can cajole the user into holding the Wiimote in a different orientation.
- Unless objects occlude or exist at many depths, you should use virtual pointing.

Manipulation

This task changes a selected object's position and orientation. It commonly occurs after a successful selection, implying that the object is being manipulated as if it were attached to a ray or virtual hand. Both attachment types rely on the specification of accurate depth information, which is difficult with an SBC. Doubly integrated position is a possibility, but orientation changes greatly impact its accuracy, unless assisted by an SBC or a gyroscope. In either case, problems will occur that require compensating design.

Manipulation control can be considered in terms of its position and orientation components. For position manipulation of objects attached to a ray, reeling is applicable. For position manipulation of objects attached to a virtual hand, the steerable virtual hand is applicable. Both manipulations are nonisomorphic and result in a loss of naturalness.

For controlling just orientation manipulation, a Wiimote is adequate, but most manipulation tasks pair position and orientation manipulation. A Wiimote with a gyroscope can sufficiently manipulate orientation. However, orienting the Wiimote can lose the SBC, which is vital for position manipulation. To avoid losing the SBC, you can use noni-

somorphic rotations. A rate-based approach can map Wiimote orientations to angular rotations. Or, you can use alternate rotational mappings—such as those for mapping 2D mouse movements to 3D orientation manipulation—to map smaller orientation changes to larger ones.

In summary:

- Isomorphic manipulation is problematic, especially for orientation specification.
- Many nonisomorphic techniques hold great potential for both position and orientation specification.

Travel

This task physically changes the user's position and orientation. You can achieve travel in two main ways. The first is by declarative means—that is, selecting a position to travel to. The second is by directional means—that is, indicating a velocity vector to travel along. Whereas you can achieve declarative travel by using a selection technique, directional travel requires the problematic specification of a vector to travel along. So, travel in 3D environments by Wiimote is limited in ways similar to selection and manipulation using a Wiimote. Both types of travel require

1. selecting position and/or orientation,
2. selecting velocity or acceleration, and
3. indicating when to start, continue, and end travel.[4]

Position and orientation selection can employ virtual-hand techniques. Velocity and acceleration selection can be a static value chosen by you or dynamically by the application or the user. Examples of the latter include having the user's reach or the Wiimote's pitch indicate velocity. Indicating when to start, continue, and end travel is typically achieved by discrete events such as pushing a button, but you can also use voice commands or clearly definable modal differentiations (for example, extending the Wiimote begins travel).

You can produce a travel vector in several ways. Although the Wiimote has enough buttons to control travel in a 3D space, this would be limiting and would feel unnatural. However, the discrete buttons could also produce precise, constrained movements if the task requires. Otherwise, the typical means of specifying a travel vector is to point in the direction of travel. With Wiimotes, this is hindered by the inability to determine yaw by accelerometers alone. Even with an SBC, there's a limitation of 45 degrees. However, as we mentioned before, a gyroscope enables the Wiimote to report yaw information. Pairing one with an SBC to deal with gyroscope drift will create a fairly robust travel vector.

Alternatively, a position offset of the Wiimote from a predefined point can indicate a 3D vector useful for travel. For example, the user can select an arbitrary point by pressing and holding a button. Any displacement from that point can be detected by the SBC or as impulse motions. This creates a vector that you can map to a velocity. This design leaves the Wiimote's orientation information available.

Whereas you can achieve declarative travel by using a selection technique, directional travel requires the problematic specification of a vector to travel along.

Several compensating design alternatives exist for when an SBC isn't available. For many 3D environments, travel is constrained to the environment's 2D surface, so only a 2D vector is required. In this case, you could easily map the Wiimote orientation's 2D roll and pitch to a 2D travel vector. Alternatively, you could use story to compel the user to hold the Wiimote upright for two possible uses:

- making roll and pitch appear as yaw and thrust and
- a virtual analog joystick.

You can extend these 2D vector specification designs to 3D travel. To travel off the 2D plane, you can map Wiimote buttons or raising and lowering the Wiimote to a magnitude vector. Raising and lowering are identifiable by impulse motions. However, this is limited by the Wiimote's inability to detect rotation and pitch while moving, unless the orientation is held constant, a gyroscope is attached, or an SBC is used. Orientations could easily break the SBC, but these cases tend to be rare owing to the constraining range of comfortable wrist movements. Although these compensating designs are sufficient for travel, they limit the ability to control orientation during travel.

In summary:

- With a gyroscope and an SBC, a well-configured travel technique can practically eliminate the Wiimote's limitations for travel tasks.

- If travel is required only along a 2D plane, the Wiimote alone can be sufficient.

System Control and Symbolic Input

These tasks inject command-based and discrete events into an application. In 2D interfaces, the keyboard and WIMP (windows, icons, menus, pointing devices) metaphor are the dominant means of generating this input. However, neither is optimally suited for the Wiimote. First, typing can be difficult while holding a Wiimote. Second, 3D interfaces let users move, so users aren't necessarily within reach of a keyboard. Third, mouse emulation with a Wiimote is possible, but even with filtering to remove jitter, the emulation lacks the precision users can achieve by manipulating a mouse with their fingertips. If you use WIMP interfaces with Wiimotes, larger on-screen elements should be used.

Several other companies are poised to introduce products exploiting alternate sensing technologies, which you could also classify as spatially convenient devices.

You can improve your Wiimote system control and symbolic input in four additional ways. First, nonisomorphic mapping of the Wiimote's orientation and position can result in more precise control of the Wiimote's cursor. Alternatively, using the Wiimote's position to control its cursor can improve accuracy but can quickly lead to fatigue. Second, identifying a small set of Wiimote gestures can lead to a simple vocabulary for system control. Keeping the gesture set small improves user recall and improves gesture recognition. Third, the Wiimote can provide feedback in the form of LEDs and sound. Incorporating better feedback with on-screen widgets could improve user interaction. Finally, low-frequency movements tend to indicate narrowing-in on a target during selection. Adaptively varying the Wiimote's sensitivity can improve the user's accuracy yet still allow large movements.

Designing a Wiimote keyboard is also a possibility. Symbolic input of an alphabet with the Wiimote has often relied on reproducing a virtual qwerty keyboard. Although these keyboards are familiar to users, they aren't optimally designed for all uses. Alternatives include using rotation to select symbols or devising keyboard layouts better suited to the Wiimote.

Chorded interaction can also improve symbolic input. Such interaction relies on the product of separate measures to create many usable symbols. For example, you could differentiate the Wiimote's orientation into four states: forward, back, left, and right. With just two buttons (1 and 2), which produce three button-press states (1, 2, and 1 & 2), chorded interaction can produce 12 distinct symbols (through four orientations and the three states). Users' ability to recall all these chorded combinations is another matter.

Overall, system-control and symbolic-input tasks for the Wiimote are similar to those for existing 3D hardware. They require special attention to the task and the equipment ergonomics. So, existing design guidelines are directly applicable, and a few become especially important:

- Understand the wrist's limitations for comfortable rotation.
- The Wiimote can be prone to jitter, especially as the distance from the screen increases.
- Chorded interaction can lead to many symbols, but without assistance, whether mnemonics or visual representations, recall quickly becomes the limiting factor.

Owing to the commercial success of devices such as the Wiimote, we can expect many new spatially convenient devices. This is validated by rumors of new 3D input devices from Sony (the Waggle) and Xbox 360 (the Newton) and Apple's patent applications. These devices continue to improve on the Wiimote, just as Nintendo improved the Wiimote by releasing the MotionPlus. With the addition of a gyroscope, the Wiimote becomes much more of a general-purpose tracker, more suited for 3D input. Additionally, Apple's iPhone 3GS includes magnetometers to detect magnetic fields, creating the ability to detect yaw in the Earth's FOR.

Several other companies are poised to introduce products exploiting alternate sensing technologies, which you could also classify as spatially convenient devices. Significantly, these devices differ from the Wiimote and overcome many of its limitations. The Gametrak Freedom uses more accelerometers than the Wiimote, enabling yaw in addition to pitch and roll, and uses ultrasonic sensors for position information. The Sixense TrueMotion avoids accelerometers, instead using magnetic tracking technology to allow true 6-DOF tracking within six feet of the emitter. Both the Freedom and TrueMotion allow up to four con-

trollers simultaneously as well as having multiple buttons and other functionality similar to the Wiimote. Finally, Microsoft has announced Project Natal, which uses video, IR depth sensors, and a microphone array to create a hands-free 3D motion gaming device.

Such spatially convenient hardware could provide a common input platform for 3D interfaces. Current commercial systems require expertise to install, maintain, and use. Their replacement and repair costs are in some cases matched by their frailty. Even so, their abilities to perform general 6-DOF tracking and to overcome occlusion, latency, and encumbrance issues are useful. Consequently, industrial, commercial, and academic applications have employed different input hardware to meet these varied requirements, thus limiting application and interface portability. If spatially convenient devices become a common input hardware platform, these limitations no longer exist, and 3DUIs can be brought to the masses.[5]

For some people, spatially convenient hardware's main contribution might seem to be its low cost. However, its impact could be more akin to that of commodity graphics hardware and software: providing a ubiquitous platform, in this case for 3DUIs, for a wide range of new applications.

Acknowledgments

We thank all the Interactive Systems and User Experience Lab members, especially Ross Byers, for their hard work and help in preparing this article.

References

1. A. Shirai, E. Geslin, and S. Richir, "WiiMedia: Motion Analysis Methods and Applications Using a Consumer Video Gamecontroller," *Proc. 2007 ACM Siggraph Symp. Video Games*, ACM Press, 2007, pp. 133–140.
2. J.C. Lee, "Hacking the Nintendo Wii Remote," *IEEE Pervasive Computing*, vol. 7, no. 3, 2008, pp. 39–45.
3. R. Azuma, "Predictive Tracking for Augmented Reality," PhD thesis, Dept. of Computer Science, Univ. North Carolina at Chapel Hill, 1995.
4. D. Bowman et al., *3D User Interfaces: Theory and Practice*, Addison-Wesley Professional, 2004.
5. J. LaViola, "Bringing VR and Spatial 3D Interaction to the Masses through Video Games," *IEEE Computer Graphics and Applications*, vol. 28, no. 5, 2008, pp. 10–15.
6. H. Luinge, P. Veltink, and C. Baten, "Estimating Orientation with Gyroscopes and Accelerometers," *Technology and Health Care*, vol. 7, no. 6, 1999, pp. 455–459.
7. S. Nasiri, "A Critical Review of MEMS Gyroscopes Technology and Commercialization Status," white paper, InvenSense, 2009; www.invensense.com/shared/pdf/MEMSGyroComp.pdf.
8. T. Shiratori and J.K. Hodgins, "Accelerometer-Based User Interfaces for the Control of a Physically Simulated Character," *ACM Trans. Graphics*, vol. 27, no. 5, 2008, article 123.
9. K. Ohta, *Image Processing Apparatus and Storage Medium Storing Image Processing Program*, US patent application 11/522,997, Patent and Trademark Office, 2006.
10. R. Williamson and B. Andrews, "Detecting Absolute Human Knee Angle and Angular Velocity Using Accelerometers and Rate Gyroscopes," *Medical and Biological Eng. and Computing*, vol. 39, no. 3, 2001, pp. 294–302.
11. B. Williamson, C. Wingrave, and J. LaViola, "REALNAV: Exploring Natural User Interfaces for Locomotion in Video Games," to be published in *Proc. IEEE Symp. 3D User Interfaces* (3DUI 10), IEEE Press, 2010.
12. D. Giansanti et al., "Is It Feasible to Reconstruct Body Segment 3-D Position and Orientation Using Accelerometric Data?" *IEEE Trans. Biomedical Eng.*, vol. 50, no. 4, 2003, pp. 476–483.
13. J. LaViola, "Double Exponential Smoothing: An Alternative to Kalman Filter-Based Predictive Tracking," *Proc. Workshop Virtual Environments 2003*, ACM Press, 2003, pp. 199–206.
14. J. Bott, J. Crowley, and J. LaViola Jr., "Exploring 3D Gestural Interfaces for Music Creation in Video Games," *Proc. Int'l Conf. Foundations of Digital Games*, ACM Press, 2009, pp. 18–25.
15. E. Charbonneau et al., "Understanding Visual Interfaces for the Next Generation of Dance Based Rhythm Video Games," *Proc. 2009 ACM Siggraph Symp. Video Games*, ACM Press, 2009, pp. 119–126.
16. D. Rubine, "Specifying Gestures by Example," *ACM Siggraph Computer Graphics*, vol. 25, no. 4, July 1991, pp. 329–337.
17. S. Theodoridis and K. Koutroumbas. *Pattern Recognition*, 3rd ed., Academic Press, 2006.
18. P. Varcholik and J. Merlo, "Gestural Communication with Accelerometer-Based Input Devices and Tactile Displays," *Proc. 26th Army Science Conf.*, US Army Science Conf., 2008; www.asc2008.com/manuscripts/B/BP-16.pdf.

Chadwick A. Wingrave *is a postdoc in the University of Central Florida's School of Electrical Engineering and Computer Science and a member of the university's Interactive Systems and User Experience Lab. His primary research interests involve 3D*

user interaction, human-computer interaction, and 3D user-interaction infrastructure for virtual, augmented, mobile, desktop, and entertainment environments. Chadwick has a PhD in computer science from Virginia Polytechnic Institute and State University. Contact him at cwingrav@eecs.ucf.edu.

Brian Williamson *has a master's in computer science from the University of Central Florida. His interests include computer graphics, video game design, user interfaces, military applications, and GPS/INS (Inertial Navigation System) navigation. His current project involves navigation techniques using the Wii Remote and MotionPlus technology. Contact him at brian.m.williamson@knights.ucf.edu.*

Paul D. Varcholik *is a PhD candidate in modeling and simulation at the University of Central Florida. He's also a faculty researcher at the university's Media Convergence Laboratory and an instructor at the Florida Interactive Entertainment Academy. He recently was a lead software engineer for Electronic Arts, where he worked on such titles as Madden NFL 06 and Superman Returns. Varcholik has a master's in modeling and simulation from the University of Central Florida. Contact him at pvarchol@bespokesoftware.org.*

Jeremy Rose *received his master's in computer science from the University of Central Florida. His areas of interest are computer graphics, interactive technologies, and embedded systems. Contact him at eklipsej@gmail.com.*

Andrew Miller *is a PhD student in computer science at the University of Central Florida. His current projects involve motion estimation and tracking applied to Wii Remotes for 3D interfaces. He has also been involved in research related to the Darpa Urban Challenge, unmanned aerial vehicles, video surveillance systems, and DSP/GPU implementations of computer vision algorithms. Miller has a bachelor in computer science from the University of Central Florida. Contact him at amiller@cs.ucf.edu.*

Emiko Charbonneau *is a PhD student in computer science at the University of Central Florida. Academically, her focus has been computer graphics, mixed reality, and 3D user interfaces. She's working on a dance aerobics system using Wiimotes and has worked in computer vision, GPU programming, and statistical simulation. Charbonneau has a master's in computer science from the University of Central Florida. Contact her at miko@cs.ucf.edu.*

Jared Bott *is pursuing a PhD in computer science at the University of Central Florida. His research interests are pen-based computing and 3D user interfaces. Bott has a master's in computer science from the University of Central Florida. Contact him at jbott@cs.ucf.edu.*

Joseph J. LaViola Jr. *is an assistant professor in the School of Electrical Engineering and Computer Science and directs the Interactive Systems and User Experience Lab at the University of Central Florida. His primary research interests include pen-based interactive computing, 3D interaction techniques, predictive motion tracking, multimodal interaction in virtual environments, and user interface evaluation. He coauthored 3D User Interfaces: Theory and Practice (Addison-Wesley Professional, 2004), the first comprehensive book on 3D user interfaces. LaViola has a PhD in computer science from Brown University. Contact him at jjl@eecs.ucf.edu.*