GAMASUTRA
The Art & Business of Making Games

PRINT

Taking Games Beyond Whack and Tilt

By Rob Kay,Ian Wright,Stuart Reynolds,Anupam Chakravorty

[The team at AiLive (co-developers of Nintendo's Wii MotionPlus peripheral) discuss the current state of motion control -- offering specific and relevant feedback about how to properly detect motions, as well as preparing you for the Move and Kinect-powered future.]

Whack!

Back in 2006 when the Nintendo Wii launched, few realized it would kick-start a new era of motion control in games. Fast forward four years and today interest in motion control in games is at an all time high. Motion controls have widened the demographic for games because they are intuitive, accessible, active and -- most important of all -- fun.

Sony and Microsoft have now unveiled their motion control systems to the world. Responding to current popularity and demand, hundreds of developers are actively engaged in developing motion-controlled games. Game designers are exploring a new degree of freedom in their designs, and inventing new kinds of gameplay experiences.

Creating a motion control scheme that goes beyond the generic "whack and tilt" approach and delivers a fun and engaging gameplay experience is a non-trivial problem. A button press remains the same no matter how it is done, but a "swipe" can be performed in a variety of ways. Motion control schemes are therefore trickier to design and develop compared to traditional button-oriented control schemes.

At AiLive, we have been fortunate to be early pioneers of motion control. During our development of the LiveMove product series and the Wii MotionPlus peripheral, we've been privileged to work with platform-holders, publishers, and game developers involved in creating motion controlled games.

In this article we will draw on our experience and discuss some of the challenges faced by developers. Then we will gaze into our crystal ball and look far beyond the motion control of today. We'll try to extrapolate current trends and predict what genre and mind-bending entertainment the future may hold.

We'll start motionless with our feet planted firmly on the ground and give a brief explanation of the properties of the major motion control systems available to developers today.

Nintendo's Wii Remote, Nunchuk, and MotionPlus

You know what this is.

The undisputed champion of accessible gaming for all and the main attraction of countless inebriated parties, the Nintendo Wii is an example of an "inertial sensor"-based motion control system. The Wii Remote and Nunchuk contain

accelerometers that measure force along three axes (X,Y,Z). Since force is measured relative to the three axes then, except for special cases, we cannot know the direction of the force in an absolute frame of reference.

In consequence, two moves that look visually identical will generate wildly different data if the player changes their grip, and therefore initial orientation, of the Wii Remote. Game developers therefore face the challenge of controlling for initial orientation when interpreting accelerometer data. In addition, the accelerometer doesn't measure rotations around its axes, so any twists and turns of the wrist during a move get indirectly measured as linear accelerations rather than changes of orientation.

To detect rotations directly you need a three-axis gyroscope, a device that measures angular velocity (speed of rotation) around its respective axes. Say hello to the Wii Motion Plus. In combination with the Motion Plus, the Wii Remote offers 6 DOF (degrees of freedom) motion processing. With a gyroscope, players can finely control their movements, whether flicking a delicate putt as Tiger Woods or slashing a back swing as Roger Federer.

In addition to providing more granular control over movement detection, with the aid of libraries like LiveMove 2, developers for the first time can use the Wii MotionPlus to include short-term 1:1 position and orientation tracking in their games.

Sony's PlayStation Move

Sony's PS3 Eye Camera and Move Controller.

Sony's Move system uses the PlayStation 3 and the PlayStation Eye camera to detect the controller's precise movement, angle, and absolute position in 3D space. The Move controller contains a three-axis accelerometer, three-axis gyroscope, and a magnetometer. Additionally, the RGB LED lit sphere perched on its top is tracked by the PlayStation Eye camera. The size of this glowing sphere, as seen from the camera, gives depth information.

To implement motion tracking without a camera, the position of a controller must be inferred via mathematical calculations on inaccurate data. The calculation errors blow-up over time and therefore motion tracking beyond a couple of seconds quickly loses accuracy. In contrast, obtaining 3D positional data from the PlayStation Eye camera is more straightforward. The motion tracking remains accurate and does not degrade over time.

Microsoft's Kinect

Microsoft's Xbox 360 Kinect system.

Microsoft has entered the arena with a high-end hardware bundle, called Kinect, which integrates a microphone and depth-sensing cameras. While Nintendo and Sony consider the tactile feedback of a hand-held controller to be central

to the game play experience, Microsoft has instead focused on a "controller free gaming and entertainment experience" featuring full body tracking, gesture, and voice recognition.

Kinect includes a RGB camera and an infrared time-of-flight depth sensor that together provide input to software that infers the 3D position of multiple points on the front of a player's body. So Kinect can in principle "see" and track the whole human body.

Tracking multiple body points from a sequence of images augmented with depth information is a more complex problem compared to tracking a single glowing controller held in the hand. In other words, Kinect gains more general tracking capabilities at the cost of more expensive inference.

However, both Kinect and PS3 Move are flexible and open-ended systems. Kinect could be used in conjunction with a held controller, and the PS3 Move could be combined with camera-based full body tracking.

Kinect also processes voice commands, which enables users to interact with the Xbox 360 console in a natural manner without the need to touch a game controller or TV remote.

Now that we've briefly described the primary motion-sensing hardware available today, let's move on to some of the challenges we spoke of earlier that designers face when creating motion-controlled games. We start by talking about a topic that is at the heart of motion control: motion recognition and tracking.

Motion Recognition, Tracking, and Hybrid Approaches

A necessary, high-level decision faced by designers is whether to use motion tracking or recognition, or a hybrid approach in their games.

Motion Recognition tells the game *what* move is being performed. Think of it as a more general form of gesture recognition, which outputs a high-level "semantic" summary (e.g., "the player is doing an underarm throw") of a large amount of "syntactic" low-level motion sensor data (e.g., numerical time-series data from a single controller, or joint positions on a 3D skeleton etc.).

Motion Tracking provides a 1:1 representation of *how* a move is being performed. Simply put, it tells the game the position and orientation of a point, or points, over time (e.g., the trajectory in space of your hand, the tip of a motion-sensitive controller, or joint positions of a 3D skeleton etc.). But tracking doesn't know what the move may signify, so it's like having the ability to see characters on the written page without being able to read them.

Sometimes it's important to know what kind of move is being performed (e.g., waving hello) while ignoring the precise details of how the move is performed (e.g., vigorous versus slow waving). Other times, it's important to know exactly how a move is performed (e.g., picking up, throwing and moving objects in a 3D physics simulation). And sometimes it's necessary to combine both kinds of information.

Motion recognition is essential when you want to funnel a large variety of human motions into a well-defined set of predetermined categories (which then get linked to in-game events). Motion recognition simply replicates our shared ability to quickly categorize human and animal movement. We can simply look and immediately know that someone is walking, running, or jumping, despite the huge variety of gaits, body sizes and viewing angles.

This comes so naturally to us we never stop to think how we do it. Many motion-controlled games need the same categorical powers in order to match the player's expectations about the meaning of the motions they perform.

For example, let's say you are developing a sword fighting game that requires the player to pull off different kinds of sword moves -- left slash, right slash, jab, block etc. -- that in addition to dealing varying damage to enemy units, fire off different animations. But your game design doesn't require knowledge of the precise way the player performed a "left slash". The best choice in this case is motion recognition.

But if you want the on-screen avatar to grip the sword exactly as the player grips the controller, matching the player's wrist twist to the on-screen avatar, then you need motion tracking capabilities. Motion tracking is the best choice if your game design requires knowledge of precisely how a "left slash" is performed (e.g., to impact a ragdoll physics model in order to mimic what would happen in real-life as closely as possible).

A hybrid approach consists of using a combination of motion recognition and tracking. Use motion tracking to give the player precise control over the sword. Use motion recognition to recognize what the player does with the sword, over and above its physical impact, in order to give qualitative feedback to the player, such as rewards for appropriate or hard-to-perform moves, or for inputs to NPC AI to react to what the player is doing.

A useful trick is that games can "cheat" a partial 1:1 visual feedback to the player without using motion tracking. This is sometimes helpful if you want to avoid a full-scale physics simulation of the interaction between a moving arm and the game world, or to avoid requiring players to move in an overly precise and controlled manner. For example, LiveMove keeps track of the motion's progress (e.g., "the player has completed 50 percent of the vanishing spell") and can synchronize animations with the player's movement, giving a realistic visual correlation.

In general, robust motion recognition requires less information than robust motion tracking. That being so, many devices that cannot support tracking nonetheless support recognition.

Motion tracking provides visual correlation, but most games also need to know what the player is doing, especially when integrating 1:1 control with higher-level game rules and events. For this reason hybrid approaches, which integrate motion recognition with tracking, are becoming increasingly common since they provide very complete motion control experiences. A recent example of a hybrid approach is Ubisoft's *Red Steel 2*.

Ubisoft's *Red Steel 2* for Nintendo Wii integrates pointing, 1:1 motion tracking, and higher-level motion recognition.

Feedback and Lag When Recognizing Motions

A simple button press immediately invokes an in-game event. But a motion, unlike a button press, has by definition a significant duration. Also, at times it's not possible to know the category of a motion until near the end of a motion (e.g., try guessing whether someone is writing a 2 or a 3 when they're just halfway through).

1:1 tracking has no lag because it does not categorize and relies on the game simulation to interpret the motion input. But motion recognition does categorize and therefore must deal with the problem. For example, if a sword-slash is recognized only when the player completes the move then the player will sense a distinct delay between starting their move and the on-screen animation.

In general, lag gets in the way of a satisfying gameplay experience. Our approach to this problem in LiveMove 2 is to break-out motion recognition into multiple stages and provide information to game developers that enables them to give feedback to the player from the start of the motion right to the end.

Stage 1: Start of an unknown motion

At this stage, we do not know which move the player is about to perform. It may be one among a valid set of moves, or it may be an invalid move. In this case the game should register to the player that a motion has been detected by generating "move-independent" feedback. This could be a sound, a rumble or a generic "get ready" animation.

Stage 2: Intermediate recognition

At this point we have more motion data and we therefore know if the move is within a valid set of moves, or if it's invalid. So we can transition from move-independent feedback to more specific feedback. This can be achieved with intermediate animations that correspond to the set of possible valid moves. For example in our sword fighting example if we think the player is performing one of ["overhead-attack", "spin-attack"] then an animation common to both can be played.

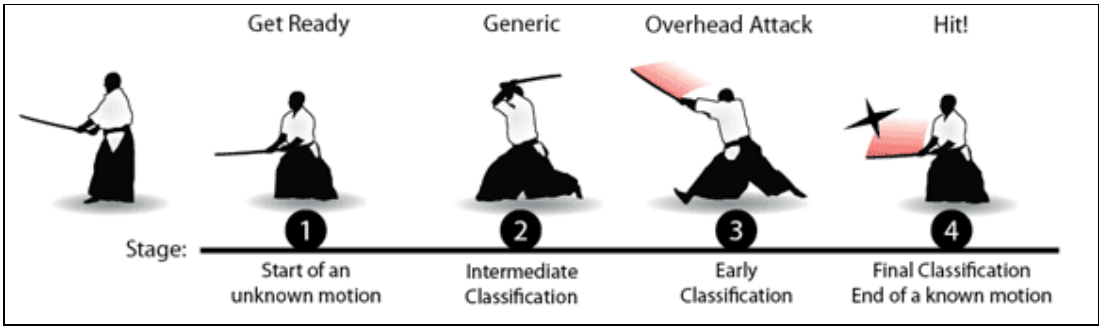
Stage 3: Early recognition

At this stage, we have sufficient motion data to know with a given probability what move the player is performing. Now we can transition to move dependent feedback such as playing finishing animations for the move. For example, if our best guess is "spin-attack", then we play the final segment of a spin-attack animation.

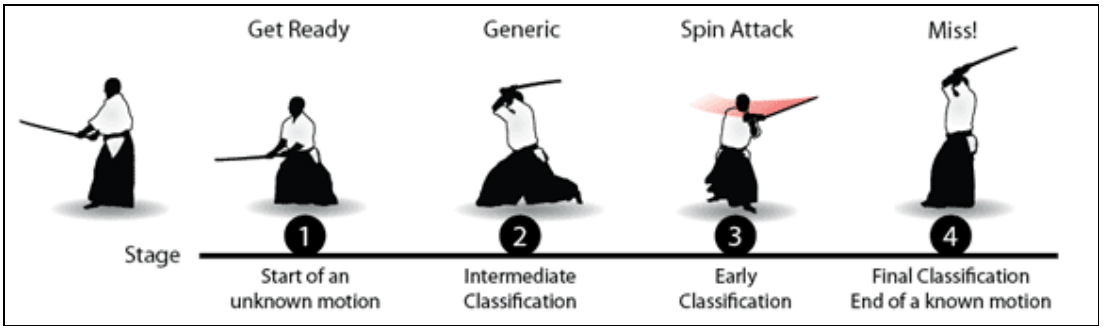
Stage 4: Final recognition, the end of a known motion

At this stage, the player has completed their move and it has been classified based on complete information. If final recognition does not match early recognition the motion can be aborted and the game can react accordingly. For example in stage 3 we committed to playing a spin-attack animation but final recognition results in something else. So although the spin-attack animation is played the move ends with a missed blow that glances off the enemy's shield.

It is important to have appropriate feedback to abort a motion at any stage if the player is not performing a valid move.



Continual motion feedback with successful final recognition.



Continual motion feedback with unsuccessful final recognition.

Design Moves That Diverge Early

Early feedback is improved if you design your moves so they diverge from each other as soon as possible.

For example, consider drawing a 2 and then a 3 in the air with your controller. During the first half of the motion they generate very similar motion sensor data which makes early recognition difficult. Now compare stabbing the controller forward with pulling it sharply backwards.

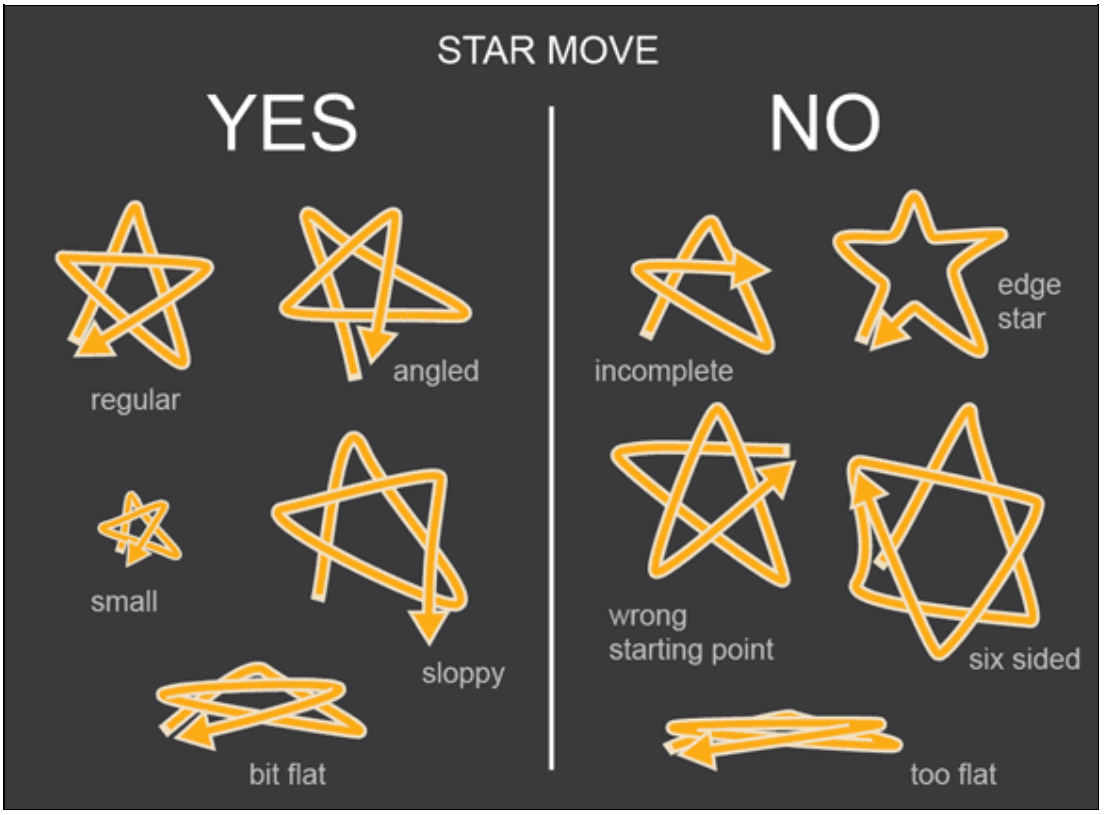
These motions immediately generate very different motion sensor data (+Z force versus --Z force). This makes early recognition easier. The general rule of thumb is: to recognize motions as early as possible design your moves to initially generate very different motion sensor data.

Be Explicit

The game design dictates the degree of precision players must achieve in order to match the motions required by a game.

For example, most card games might not care about the accuracy of player's motions (e.g., shuffling, dealing), but in action games that require realism, the fidelity of player motion may be pivotal to delivering a fatal blow.

Games span the spectrum from metaphorical moves (e.g., party games that merely require high jerk) to very concrete moves (e.g., a golfing game that requires real-life levels of control and performance).



Move design means getting specific about the boundaries of your move.

A central question in motion design is the first step: what does and does not count as an example of a move? Aspects of a motion that may be important include its shape, speed, direction and size. For example, you decide that your wizard-themed game will have a star-shaped spell.

Defining this motion means answering questions like: Will the star be 4 or 5 pointed, or either? Should the star be drawn slowly or quickly, or either? Should it be drawn big or small, or either? And so on. We adopted an example-based approach to move definition in LiveMove because we wanted developers to answer these questions according to what's right for their game without worrying about the technical feasibility of funneling hundreds of different kinds of "stars" into one kind of star spell.

You also need to consider the different play styles of different players. A single motion may be interpreted in various ways by different players. For example, a tennis serve might be performed overarm or underarm. Your game may want to distinguish each case, or allow either to behave as a "tennis serve".

It's imperative that the player be told how to perform the move either by in-game tutorials and instructions, or qualitative feedback to the player (e.g., "You need to hit the ball harder"). In our experience, developers often initially underestimate the production cost and difficulty of clearly communicating how a move should be performed and giving helpful feedback.

Player-Driven or Game-Driven Segmentation

Motion, of course, occurs in time. Motion recognition maps continual movement into discrete categories. Imagine watching a horse trot then break into a gallop and then rear, turn, and stop. We can easily recognize when the horse is trotting, when it is galloping and when it is rearing. But it is more ambiguous to identify exactly when the horse transitioned from trotting to galloping to rearing.

Any game that needs to recognize the kinds of moves players are performing faces the same requirement of *segmenting* continuous motion into discrete categories. The game needs to identify when a motion has begun and when it has completed. There are two primary ways to do this. Which is best ultimately depends on your game design.

You can ask the player to segment their own motion. Normally this takes the form of holding down a button while the player performs a move. The advantage of this approach is ambiguity is entirely removed in the sense that the player is responsible for telling the game when they start and stop a move. The disadvantage is that pressing a button to signal intent complicates the control scheme and may seem artificial.

The second approach is to get the game to segment the player's motion. In this approach the player simply concentrates on moving. A good approach, suitable for many games, is to tell the player when to perform a motion within a time window (e.g., "Ready, Go!") that naturally arises during gameplay and is communicated to the player. For example, Namco's *We Cheer 2* takes this approach.

Namco's *We Cheer 2* takes a "time window" approach to motion segmentation.

Alternatively, the game can segment motion using the outputs from motion tracking. The simplest, and often quite effective, method is to detect when a force threshold is crossed, for example high force signals the start of a move, and low force signals the end. Threshold values can be decided after a bit of trial and error before finally deciding on one that works best for the game.

Accessibility

Last but not least, one should not forget that people's physical abilities are very different. Left-handed people can generate wildly different motion sensor data compared to right-handed people performing the same move. Different body shapes, especially limb length, are also a source of data variability. Since we are creating entertainment experiences that require players to give physical performances it is important to consider the needs of players with different abilities, so that our games can be enjoyed by as many people as possible.

The fundamental problem is variance in the population of game players. Our approach in LiveMove has taken two tacks. First, we construct motion recognizers with very high "capacity" so that, if required, wildly different data can be recognized as a valid example of the very same move. Ultimately the QA process must verify that motion controls are robust over all shapes and sizes prior to shipping.

Second, as an insurance policy, we provide the ability for the game, after it has shipped, to "tune" a motion recognizer to a particular body type "under the hood" without distorting the definition of a valid motion.

Some Platform-Specific Issues

What we have described in the last few sections are essentially device-independent design choices. There are, of course, many device-dependent choices to consider, and we look forward to developers sharing their hard-won knowledge. Here we'll briefly mention some specifics that we think are important.

The Wii Remote is capable of capturing a wide range of human motions; however there are a couple of things to keep in mind when designing your moves. Accelerometers measure acceleration, so any motion with a constant velocity will not offer useful readings.

It is generally a good idea to design moves with inflection points (e.g., significant changes of direction) so that your motion recognition system has interesting accelerometer data to work with. Errors multiply quickly with full 6D motion tracking without a camera, so tracking moves longer than a second or two should generally be avoided.

Both the Wii Remote and the PS3 Move are devices based on inertial sensors which cannot sense beyond a certain cutoff range. If the player swings the controller too hard or rotates it too fast, you lose information.

This can really hurt tracking, but recognition is affected less. Simple technical tricks to try to infer the lost information (e.g., interpolation curves that stitch together the data) may not always be worth the effort.

Microsoft's Kinect provides information that can drive a full-body representation of the player, which includes the movement of head, arms, torso and legs. This capability opens up another space of exciting game design possibilities. But the physical feedback typically provided by the mere weight of a controller, button presses, rumble, in-controller audio etc. instead must be provided in other forms.

This is not a new problem. For example, Harmonix's *EyeToy: Antigrav* was an early camera-based game that allowed players to use their head to steer a futuristic hoverboard while performing gravity defying stunts using gestures. Players hit targets on rail sections that fired a number of audio and visual cues which made up for the lack of physical feedback. And the chosen activity of hoverboarding reduced the amount of physical feedback expected by the player (e.g., compare your feedback expectations when dancing compared to boxing).

Harmonix's *EyeToy: AntiGrav* for the PlayStation 2: An early controller-free motion controlled game.

The Future of Motion Control

Now that we've briefly sketched some of the major issues in designing motion control schemes given the technology available today, let's turn our eyes to the future.

Where is motion control heading? Predictions are always wrong in detail, and often laughable in retrospect. But despite the risk of appearing foolish we think it's important to look ahead and get excited about where our industry is heading. To try to predict we first need to consider the current trends.

Opening a door by reaching forward and grasping its handle is simply a more engaging and natural experience compared to pressing a small button. The industry trend is, we believe, toward a deepening of the medium of interactive entertainment by increasing a player's ability to naturally engage and participate in a virtual world. By "natural engagement" we mean modes of interaction identical to the modes we use in the real world.

A camera or a motion controller transmits a greater quantity of information that is both richer and more meaningful compared to a joystick. So motion control increases the interaction bandwidth from player to game. More of the player can now potentially be "in the game". This is the real meaning of motion control. Motion tracking and recognition are simply the first steps toward ubiquitous tracking and recognition of all human modalities.

A second, and important, technological trend for natural engagement is unit cost reduction and miniaturization. For example, today's smartphones feature increasingly impressive CPU and GPU horsepower and incorporate several motion sensing capabilities that include cameras, accelerometers, gyros and touch screens. Miniaturization means that soon it will be feasible to wear powerful but unobtrusive networked computers, coupled to meshes of motion sensors and cameras and direct-to-eye display technology.

Finally, techniques developed under the rubric of Artificial Intelligence (AI) will increasingly be deployed in consumer-level applications. The more general problem of recognizing all kinds of stuff with different sensory devices -- including recognizing not just motion but also everyday objects, scenes and locations, faces, social situations and natural language utterances -- encompasses much of AI. So we expect to see more "intelligent" algorithms running on game platforms that track, recognize and basically "glue together" the player, their environment and the game.

These trends obviously have important applications outside of entertainment. In the context of games, however, we expect players to "be fully present" in virtual worlds that are entirely artificial constructs or dramatic transformations of everyday reality, which they carry in their pockets and can access at any time. Players will naturally engage with new genres of interactive entertainment that seamlessly integrate with the natural and social worlds we inhabit.

In summary:

- Game worlds will be portable and therefore always ready-to-hand.
- Players will naturally engage with game worlds through more general kinds of recognition powered by ubiquitous sensors and intelligent algorithms.
- The barriers between game worlds and the real world will breakdown and begin to blur.

... and Tilt!

We hope that some of the observations and "rules of thumb" in this article may help you orient yourself in the space of motion control. There's a lot of unexplored territory. So we're very much looking forward to learning from developers as they push the boundaries further.

To say now is an exciting time for motion control in games in an understatement. The software tools available to developers are sufficiently mature to significantly reduce the technical problems of integrating sophisticated motion recognition and tracking in games. So designers can concentrate on being creative. Console manufacturers are committed to their new peripherals and game creators are busy delivering new content. Consumers can therefore look forward to fresh and entertaining games that allow players to participate in more natural and engaging ways.

We're excited about the future. How we deliver on the promise of motion control in video games depends on all of us. But one thing is certain -- there is more than a tilt toward a whacking bright future!

Return to the full version of this article with comments