Game Feel and Human Perception

Understanding exactly how humans perceive the video game worlds we create is key to designing good game feel. We'll begin by examining our feedback loop model of interactivity from Chapter 1 in greater detail. By deconstructing each piece of this system and incorporating the concept of the Model Human Processor, we'll be able to define real-time control at the level of specific, measurable properties of human perception. This will tell us exactly when and how real-time control can exist and what will cause it to break down. We'll also look at the computer's side of things: what, exactly, are the parameters of machine illusion? Finally, we'll look at some of the implications of perception for game feel.

When and How Does Real-Time Control Exist?

In Chapter 1, we defined real-time control as the uninterrupted flow of command from player to game resulting in precise, continuous control over a moving avatar. It's more like driving a car than having a conversation, as we said. The part of the definition that needs clarifying is "uninterrupted." What if the player can offer new input at any time, but the game can only receive it at set intervals? Or what if the player gets locked out for a certain amount of time, unable to add new input until an animation has finished playing? In other words, what is real-time control and how do we know when it's happening and when it's not?

Let's again look at interactivity, this time with more specificity. There are two halves to this process, the player and the computer (Figure 2.1). On the player's side of things, there are unchanging properties of human perception. For example, there is a minimum amount of time in which a player can perceive the state of the game, think about how to act and pass that impulse along to his or her muscles.

On the computer's side, this creates boundaries. To sustain real-time control, the computer must display images at a rate greater than 10 per second, the lower boundary for the illusion of motion. The computer must also respond to input
within 240 milliseconds (ms), the upper boundary for response time. There’s also a threshold for continuity; the game must be ready to accept input and provide response at a consistent, ongoing rate of 100ms or less. If the game responds to input sporadically, the flow of control is broken. The onus for maintaining real-time control, then, is on the computer. The computer’s half of the process is changeable. The player’s perception is not.

On the player’s side, the minimum amount of time it takes for a person to perceive the state of the world, think about what to do and act on that impulse is around 240 milliseconds. This is a very short amount of time.

This correction cycle is the increment at which people make the tiny adjustments necessary to assemble a sandwich, drive a car or exercise real-time control over objects in video games. The measure comes from Card, Moran and Newell’s “Model Human Processor,” the collected result of many different studies about human reaction and response time. The figure of 240ms is an amalgam of three different measurements, one for perceiving, one for thinking and one for acting. They break down as ranges, like so:

- Perceptual Processor: ~100 ms [50–200 ms]
- Cognitive Processor: ~70 ms [30–100 ms]
- Motor Processor: ~70 ms [25–170 ms]

What’s being measured here is the cycle time of each processor, the time it takes to accept one input and produce one output. The variation comes from physiology and circumstance. Some people have the capacity to process things more quickly than others and everyone tends to process things more quickly under intense circumstances, displaying a heightened sense of awareness. Likewise, processing speed goes down under relaxed or sub-optimal circumstances such as reading in the dark. In the model, each of these steps is defined as its own separate processor and has its own little cycle time (see Figure 2.2).

**Try It Yourself**

To appreciate the speed at which your processing functions, I highly recommend checking out humanbenchmark.com’s Reaction Time Test (http://www.humanbenchmark.com/tests/reactiontime/index.php). This will give you a clear sense of just how small the increments of time we’re talking about seem when you’re able to measure them against the computer. My best reaction time is around 170ms. If you’re like me, it will feel a bit weird to actually butt up against the limits of your own perception. But there it is: you can’t argue with the precision of the computer measuring your reaction time. It’s neat that we can measure this!

The idea is that perception, cognition and action are processed separately but feed into one another, enabling a person to see the state of things, think about how to change them and then act on that impulse. Note that this is an abstraction of human cognition—nowhere in the anatomy of the brain is there a structure called the perceptual processor—but it is a useful one because it lets us put hard numbers to components of our diagram.

The perceptual processor takes the input from the senses and makes sense of it, looking for patterns, relationships and generalizations. From all the sensory data, it creates recognizable state of the world for the cognitive processor.

The cognitive processor does the thinking. It compares intended result to the current state of things and decides what to do next.

The motor processor receives the intended action and instructs the muscles to execute it. After the impulses leave the body as muscle movements, they’re out into wild, wooly reality, and the process starts again with sensory perception.
Correction Cycles and Game Feel

When all three processors (perceptual, cognitive and motor) work together in a closed feedback loop, the result is an ongoing correction cycle. A correction cycle happens any time you do something requiring precise coordination of muscles over time, whether it's picking up a book, driving a car or controlling something in a video game. Robert Miller of MIT's User Interface Design Group describes the process: "There's an implicit feedback loop here: the effect of the action (either on the position of your body or on the state of the world) can be observed by your senses and used to correct the motion in a continuous process."

For example, imagine you want to reach out and grab a muffin that's sitting on your desk. You formulate intent: to grab the muffin (Figure 2.4). As soon as this intent is formulated, it is translated into action of your muscles—twist trunk in chair, activate arm muscles, open hand into "muffin claw" and so on. The moment this action starts, you perceive the position of your hand in space and see it start to move, responding to your impulses. The perceptual processor looks at where the hand is in space, passing that information on to the cognitive processor. The cognitive processor thinks about where the hand is relative to where it should be and formulates a new plan to correct that motion. The motor processor then takes the new plan and translates it into action. From the moment movement starts to when you have the muffin in hand, you run this continuous process of action, perception and thought, increasing precision each time as a factor of distance and size of target (Figure 2.5).
Because we know the cycle time of each processor (perceptual = 100 ms, cognitive = 70 ms, motor = 70 ms) we know the time for each one of these correction cycles, 240 ms (Figure 2.6).

Correction cycles are how people are able to track and hit targets with precision, steer things, point at things and navigate the physical world successfully. To experience this first hand, check out example CH02-1. Try putting your cursor directly over the dots as quickly as possible. You can see the correction cycle in action as you overshoot, undershoot and then home in, eventually coming to rest on your target.

Now imagine you’re hungry, and you’re out of muffins. You get in your car and begin to drive to the muffin store. The overall goal for this trip is to get a sweet, delicious banana muffin. This goal trickles down to different layers of intention, such as “Turn right on Elk Street.” At the lowest level, it’s about the moment-to-moment adjustments of the motion of the car to keep it in the lane, stop it at red lights and so on. As before, you perceive the state of the world, think about what corrections you need to make to the current motion and make the adjustments happen once every 240 ms (Figure 2.7).

The process is the same as when you were reaching across the desk except that this correction cycle goes on longer. The muffin on the desk was static and represented

FIGURE 2.7 The correction cycle of driving.

a single target, a single intent. Driving to the store might take 20 minutes, fulfilling 20 different sub-goals.

In a video game, real-time control is an ongoing correction cycle of this type. As with driving a car, control is an ongoing process where higher-order intentions trickle down and become individual, moment-to-moment actions. These actions are a part of an ongoing correction cycle, where the player perceives the state of the game world, contemplates it in some way, and formulates an action intended to bring the game state closer to an internalized ideal (Figure 2.8). This happens at the same cycle time of ~240 ms.

The difference is, at the point where action normally goes out to physical reality, a video game substitutes a game world for the real world. It hooks right in there. The inputs to the perceptual processor come from the screen, the speakers and the feel of the controller. The output, instead of acting on objects in the real world directly, acts on the controller, which translates to movements of objects in the game world.
Fitt's Law

There is a well-known formula, Fitt's Law, which can accurately predict how quickly you can move your hand to a target of a particular size at a certain distance. Fitt's Law is an unusually successful and well-studied HCI model that has been reproduced and verified across numerous studies. For reference, the formula is this:

\[ MT = a + b \log_2 \left( \frac{D}{W} + 1 \right) \]

where:
- \( MT \) = movement time
- \( a \) = start/stop time of the device
- \( b \) = speed of the device
- \( D \) = distance from starting point to target
- \( W \) = width of target measured along the axis of motion

The original formula predicted how long it would take to reach out and touch something of a certain size a certain distance away, as long as it was within arm's length. It was later discovered to be equally applicable to the time it takes to move a mouse cursor to an object of a particular size and shape on a computer screen, so it is applied and studied by user interface designers. For example, the menu bar in the Macintosh OS is always present and takes up the entire top edge of the screen. This means that the "size" of the menu bar is functionally infinite, enabling the user to get the cursor onto it quickly and easily, with very few correction cycles. Compare this to a tiny checkbox button or a hierarchical submenu.

The Computer Side of Things

Real-time control relies on the computer sustaining three thresholds over time:

1. The impression of motion (display above 10fps). The frames displayed on the screen must be above 10 per second to maintain the impression of motion. The impression will be better and smoother at 20 or 30 frames per second.
2. Instantaneous response (input to display happens in 240ms or less). The computer's half of the process must take less than correction cycle for the player. At 50ms, response feels instantaneous. Above 100ms, the lag is noticeable but ignorable. At 200ms, the response feels sluggish.
3. Continuity of response (cycle time for the computer's half of the process stays at a consistent 100ms or fewer).

The Impression of Motion

Similar to film and animation, the way that computers create and sustain the impression of motion is well understood. Think of each cycle of the player's perceptual processor as a snapshot of reality, incorporating visual, aural, tactile and proprioceptive sensations. Each 100ms cycle, the perceptual processor grabs a frame of all these stimuli. If two events happen in the same frame—Mario in one position, then Mario slightly to the left—they will appear fused, as a single object in motion rather than a series of static images (Figure 2.9). This is perceptual fusion.
From the computer's side of things, perceptual fusion explains how objects in a game appear to move. If the display is updated 10 times per second (100 ms cycle time = 10 frames per second) this is sufficient for the illusion of motion. This is right at the border, though, and won't feel very good—20 frames per second (fps) is better, and 30fps is where motion begins to be pleasingly smooth. Most games run at 30fps or higher for this reason. As game developers know, with frame rates that vary based on processing power used it's better to be safe than sorry. There's no such thing as a frame rate too high.

**Instantaneous Response**

Perceptual fusion also influences the impression of causality. If I flip a light switch and the light comes on within the same perceptual cycle, I will register this as a cause-and-effect relationship. My action caused the light to turn on. The same thing is true for computer response: if I move a mouse and the cursor seems to react immediately, I tend to assume that effect was caused by my action. An extension of this is the impression of responsiveness. Professor Miller describes the process: “Perceptual fusion also gives an upper bound on good computer response time. If a computer responds to a user’s action within [~100 ms], its response feels instantaneous with the action itself. Systems with that kind of response time tend to feel like extensions of the user’s body.”

With reality, there’s never a problem of lag. Response will always be instantaneous. In a game, response will never be instantaneous. Even a game running at 60 frames per second, a three-frame delay is all but inevitable. Three frames at 60 frames per second means 50 ms. (You can convert frames per second (fps) to milliseconds if you divide by 60 and multiply by 1000. So 3 frames at 60fps is 3/6 * 1000 = 50ms.)

Mick West, programmer-designer of the original Tony Hawk mechanic, defines this as response lag. “Response lag is the delay between the player triggering an event and the player getting feedback (usually visual) that the event has occurred. If the delay is too long, then the game will feel unresponsive.”

http://cowboyprogramming.com/2008/05/27/programming-responsiveness/

---

**Figure 2.10 Response time and player perception.**

Mick notes that games with a response time of 50 to 100 ms typically feel “tight” and “responsive” to players. This is because 50 to 100 ms is within one cycle of the human perceptual processor. Above this level, a game’s controls begin to feel sluggish. The progression, from responsive to unresponsive, is gradual (Figure 2.10).

**Mick West on Responsiveness**

Check out Mick’s articles about programming for responsiveness at www.cowboyprogramming.com. He offers an awesome technical grounding for avoiding response lag as well as an engagingly practical way to measure response time in any game using an inexpensive digital camera (recording both screen and controller at 60fps).

There’s no exact point at which a game’s response lag can be said definitively to have gone from tight to sluggish, because other factors, such as mapping and polish effects, can shape this impression of responsiveness. But there is a threshold above which the sensation of real-time control is broken: 240 ms. Past this response time, the player can perceive, think and act before the computer is ready to accept a new input.

**Continuity**

If it takes a player 240 ms to perceive, think and act, how is it that the computer has to finish its tasks and offer feedback within 100 ms for the response to seem instantaneous? This is because all the human processors run concurrently (Figure 2.11).

The perceptual processor passes information along to the cognitive processor, then starts cycling again. By the time the instructions from that original cycle have been sent out into the world as movements of the muscles, three perceptual frames have passed. In real life, this never matters because the response is always
Some Implications of Perception for Game Feel

To examine game feel is to see perception in a particular way. First, game feel includes many senses. Visual, tactile, aural, proprioceptive—experiencing game feel, these senses combine into one sensation. Second, the notion of proxied embodiment should be addressed. Game feel enables objects external to the body to be subsumed into body image and feel like extensions of the body. Third, game feel is an ongoing process of skill building and practice. It is part of the larger realm of human skill building and relates to skills like driving and playing tennis because it requires the same kind of repetitive practice to master. From nOOb to 133t, as it were. Lastly, a model for perception for game feel needs to encompass the physical nature of game
CHAPTER TWO • GAME FEEL AND HUMAN PERCEPTION

feel. Experiencing game feel is like interacting with a surrogate reality which obeys its own rules and which must be understood through interaction and observation.

Expanding on the experiences in Chapter 1, here are five interesting ideas about perception that support our definition of game feel:

1. Perception requires action.
2. Perception is skill.
3. Perception includes thoughts, dreams, generalizations and misconceptions.
4. Perception is a whole-body experience.
5. Tools become extensions of our bodies.

**Perception Requires Action**

In order to perceive something, you have to see it in action. This has been verified experimentally with kittens and blind people.

The kitten study (Held and Hein, 1963) involved two groups of kittens, each raised in the dark. The first group was allowed to roam freely; the second group was “kept passive” which we can only reasonably assume meant a Clockwork Orange style tie-down. The experimenters controlled the conditions such that both groups of kittens were exposed to the same limited stimulus: flickering lights, sounds and so on. Then, they released the kittens into a normal, lit environment. The ones who had been allowed to move around in the dark were able to function just fine, while the ones who were tied down helplessly staggered around as though blind. What this seems to indicate is that perception is an active rather than passive process. The lights and images used to stimulate the kittens didn’t make much difference; being able to explore their surroundings and perceive things in motion relative to their own bodies did. To perceive, you need to interact.

Another study (Bach y Rita, 1972) did something similar with a bunch of blind folks. The researchers created a special video-camera-driven matrix of stimulation points, shown in Figure 2.13. A TV camera (mounted on spectacle frames) sends signals through electronic circuitry to an array of small vibrators (left hand) strapped to the subject’s skin. The pattern of tactile stimulation corresponds roughly to an enlarged visual image.

Each vibrator was mapped to a particular pixel of the image that the video camera was receiving, giving a sort of tactile image of what the video camera saw. When the participants were allowed to move the camera themselves, they were able to learn to “see” in a limited way. If they were not allowed to control what they were “looking” at, the image stimulus device was just a gentle, if unskilled, massuese.

The concept that perception requires action has relevance to game feel because it accurately describes the sensation of exploring and learning your way around an unfamiliar game space. And it correlates physical reality with virtual reality in a meaningful way: the thing you’re controlling in the game becomes your surrogate body; your hands. Humans are adept at learning the physical properties of a new and unfamiliar object and do it very quickly. Noodle it around in your hands and you soak up a wealth of detail: weight, density, material, texture, color and so on. This same ability extends to virtual objects and, interestingly, to virtual worlds governed by different rules, laws and physics. For some reason, it’s immensely pleasurable to suss out a new and unfamiliar world by probing around in it using a virtual instrument. The thing being controlled becomes both expressive and perceptual: as you control it and move it around, feedback flows back through it to your eyes, ears and fingers.

**Perception Is Skill**

If perception requires action, that action must be learned. We don’t usually think of it this way, but perception is in some ways just a set of skills, honed across one’s lifetime. From the moment we’re born, we learn new distinctions, forge new neural pathways and generally undertake an ongoing process of becoming better at perceiving. Grow keys. Insert keys in mouth. Rinse, repeat and learn; so it goes until a fully functional adult emerges. “Perception is to a large extent an acquired bodily skill that is shaped by all our interactions with the world,” as Dag Svanaes says.

Part of this process is making generalizations and learning abstract concepts like justice, freedom and cheesecake. Another part is bringing with us all experiences that came before. There’s a concept from two venerable psychologists, Donald Snygg and Arthur Combs, that vividly portrays the role past experience, ideas, generalizations and fantasies play in perception: the perceptual field. People have a great deal

---

2Near as I can tell, the idea started with Snygg and Combs in 1951 as the “perceptual field.” Combs changed phenomenal field to perceptual field in a later work, so we’ll stick with that. I prefer that anyway, as it’s more descriptive of the concept as it applies to game feel.
of practice navigating the space around them and developing relationships with the things in that space. Snagg and Combs capture the phenomenon of memory, perception and skill building with their concept of the perceptual field.

The idea of the perceptual field is that perception is carried out against the backdrop of all previous experience, including our attitudes, thoughts, ideas, fantasies and even misconceptions. That is, we don't perceive things separately from what's come before. Rather, we experience everything through the filter, against the background, and within the structure of our own personal vision of the world. Another way to put it is this: "one's constructed representation of objective reality; the meaning given to the profusion of stimuli that bombard the brain and are organized and conceptualized on the basis of individual and personal prior experiences." And still one more way: "The perceptual field is our subjective reality, the world we are aware of, including physical objects and people, and our behaviors, thoughts, images, fantasies, feelings, and ideas like justice, freedom, equality, and so on."

So your perceptual field is your world; your structural understanding of everything you perceive around you and its meaning.

This is a cool idea because it goes a bit beyond the notion of a simple mental model, which is more about the dry, clinical details of how a person thinks a thing functions (usually compared to the system image, the way the thing actually functions).

In the case of a video game, the brain recognizes that the "world" of the game is a subdomain, a microcosm of the larger perceptual field of reality that it understands. It's apportioned, separate and obeys its own rules. At the same time, the mind brings all its past stored experiences to the table to help it understand this new place.

The difference is that a game world is not necessarily governed by the rules of physical reality or bound by them. This is a useful way to think about the creation of game feel—as the creation of a separate but related physical world. Simplified, but whole, cohesive and self-contained. Many times, creating and tweaking a game feel system means literally the construction of a set of generalized laws and rules that govern all action within the system. It's like writing your own universe from scratch: you have your own gravity, your own simple momentum and friction, your own simplified definition for what collision between two objects means and how it should be resolved.

The world around you is objective and immutable. You're not going to wake up one morning and discover that gravity has suddenly stopped working as expected. I throw a grapefruit at a wall, it's going to hit the wall with a thud and fall to the ground. This kind of consistency can be frustratingly difficult to achieve in a game world.

Because the way we cope with and understand events in a game world is so similar to the way we interact with the real world, we expect the same consistency. The trickiest thing can break this perceptual immersion.

http://phenomenalfield.blogspot.com/
Dr. C. George Boeree, http://webpage.ship.edu/cgboer/snagg&combs.html

Unlike a film, which presents one framed perspective on a world and has only to maintain visual and aural consistency within that frame, a game has to stand up to active perception. The player can freely explore every possible permutation of every action and response in the entire world. This is the skill of perception, the one every human being has been constantly honing since birth.

Because people are extremely skillful at perceiving the world around them, any tiny inconsistency becomes glaring and obvious. A character's foot clipping through a stair, an invisible wall—these things never happen in the world around us. So this process of discovery is much closer to our experience of being in the world than the experience of passively viewing a film or reading a book.

The same mechanisms we use to cope with the world around us, to understand it and function within it, come into play when exploring a game world. It is the same process of expanding the perceptual field into this new world and probing around to make the generalizations and distinctions needed to interact with and succeed in the world.

This is why a consistent abstraction is so much more important than a detailed one. It's fine to create a simplified version of reality for players to interact with; they'll figure out the parameters of the world—it's constraints, rules and laws of physics—within minutes of picking up the controller. Just don't violate the rules you yourself have set. If an object is portrayed as being large, heavy and massive, don't let it go flying off into space with the slightest touch, or pass through another object without interaction. Easier said than done, of course, but this is at some level a designer's decision. Better to have a simple, tight, cohesive world like Dig Dug than a weird, inconsistent world like Jurassic Park: Trespasser. People are going to figure out everything about your world either way—our physical reality is much more complex and nuanced than any game world, and we've got years of experience at perceiving it—better to make it simple and self-consistent than a bad inchoate mess.

Perception as skill is also correlated to the way people get better at things over time. You practice something, you get better at it. Your perceptual field includes an ever-growing body of experience about the task, which makes each new try at the task easier. It's not just a bank of information to draw on, but affects what happens at the moment-to-moment level of perception and action. Chris Crawford would say that the neural pathways are getting pushed farther and farther down, away from conscious processing and into subconscious, automatic processing.

Mereau-Ponty defines this distinction as abstract versus concrete movements. If an action is unpracticed and requires conscious thought and effort, it is an abstract movement. If it's so practiced that it happens automatically and without conscious thought—a pure translation of intent into action—then it's considered a concrete movement. By this process, people learn things incrementally over time, turning what was once difficult and requiring deep, conscious involvement, into an easy and subconscious action. Clearly, this is what happens with game feel, though the process is almost always much quicker when learning a game skill.

However we conceptualize it, it is obvious that humans get better at skills they practice. And if we consider all perception to be a skill, this handily explains why
game feel seems like such a skill-driven activity, why skill learning is the price of admission to experience it. Perception in a game world is just a simplified, modified version of perception in the real world. The rules are different, but the process is the same.

Perception Includes Thoughts, Dreams, Generalizations and Misconceptions

Another interesting offshoot of the concept of the perceptual field is that it encompasses not only physical reality, but attitudes and ideas. Perception of something is heavily influenced by biases, ideas, generalizations and worldview, all of which have been incorporated into the perceptual field through a lifetime of experiences. Of course, generalizations about a particular thing may not accurately reflect the objective reality of that thing.

For example, I have a central heating and air conditioning system in my apartment. One day I was cold and wanted to adjust the temperature. The wall mounted control unit displays the current temperature based on a thermometer which is... somewhere in the apartment. I hope. I think. The current temperature is compared to a sliding blue control representing the desired temperature. I assume that if I move the sliding control to a temperature different from the actual temperature, the air will turn on and will adjust the temperature. The current temperature reads 61°. I move the control from 63 to 75. The cold air turns on briefly and then turns off. A few minutes later, the air turns on for a longer period, blowing warmer air, then turns off. Eventually I get fed up and turn the temperature to 90°. Again, the air turns on for a few minutes, warm, and then turns off. I get a blanket. An hour and a half later, I walk into my office, the door of which has been closed and I'm suddenly sweltering hot and realize I've left the temperature on at 90°. I toss the blanket, strip down to a T-shirt, and check the thermostat, which now reads 77°.

I put the thermostat back down to 72. The air turns on for a few minutes, cold, and turns off. Gah!

What's going on here? In my conceptual model of the heating system, there is a certain threshold between the actual and desired temperature. If that threshold is exceeded—if the desired temperature is 3° above or below the actual, for example, the system turns on and applies an amount of heat or cold equal to the change in temperature that's necessary to bring actual back in line with desired.

The reality of the system is that it turns on and吹s air, either hot or cold, for a certain duration. Five minutes of hot or five minutes of cold, depending which way the temperature is supposed to go. The hot isn't particularly hot, and the cold isn't particularly cold, but applied across a long enough time scale, they get the job done. In addition, the timer that governs how long each on state should last functions independent of the controls. That is, it's always counting down five minutes at a time. If you switch the heat on with 30 seconds remaining on its five minute cycle, you get 30 seconds of cold air because it takes a while for the heater to warm up and there are only 30 seconds of "on" remaining before the system goes back to the off state. (Oh, and the thermostat, evidently, is in the office.)

Donald Norman would say that my mental model is out of sync with the "system image" of the heating system. I hold in my mind a logical construct, a picture of how the system works, but that picture is wrong. This construct frames my interaction with the system and dictates my expectations about what the result will be for a particular input. If my mental model differs from the system image, the way the system actually functions, errors occur. Norman would say that this is the designer's fault and go on to point out that the "design model"—the way the designer imagined the user would interpret the system—is out of sync with the user's mental model. This is definitely applicable to game feel as it helps greatly in designing what players refer to as intuitive controls. Norman suggests seeking out and exploiting "natural mapping" between input device and system.

For example, in Figure 2.14, stove C makes more sense than stoves A and B and is far easier to operate because there is a clear, spatial correlation between the position of the knobs and the position of the burners they operate. Finding natural mappings in games is a similar process, though not always an obvious one. I like the
example of the game Geometry Wars, which tends to be easy for people to pick up and play effectively—the movement of the thumbstick corresponds very obviously to the motion of the little ship in the game making this something of a natural mapping, albeit one in a virtual space (see Figure 2.15).

The only problem with Norman's concept of the mental model is its rigidity. To say I have a mental model of the world of The Legend of Zelda: The Wind Waker could be useful to isolate and tune certain mechanics, to eliminate ambiguities and make the controls more intuitive, but it ignores the fact that I have a relationship to that world. I feel a certain way about it and think a certain way about it, and breaking that down into a dry, clinical, logic diagram seems to ignore some of the most important parts of that relationship, some of the parts the designers understood and worked really hard to develop.

For example, it's extremely important to the success of the game that I feel a sense of freedom and adventure while sailing the open seas in the game. I could distill that to a system image of my possible actions while sailing, but that would miss much of the point; it's not as important to track exactly how far objects are spaced apart or how far I can sail in one direction before hitting something or how long it takes to get from object to object as it is to make the space of that world feel open, free, but full of possibility. I can sail aimlessly in a particular direction and be certain of two things: 1) I'll be free to sail as far as I want in that direction and 2) eventually I'll find something new and interesting. The system, speed of the ship, sharpness of turning, distance between islands and so on, would be interesting to track, as would a player's mental model of that system. But it definitely misses one of the essential experiential qualities of the game. A few small changes to the system, making the ship move 20 percent slower for example, could make the oceans of Wind Waker feel impossibly large, tedious or lonely. The idea of a perceptual field that incorporates not only the system image, but the thoughts, ideas, feelings and generalizations about the system that players have brought with them, and is constantly forming and reforming, is a much more effective for understanding that experience.

Another important thing that Norman's mental model concept ignores is paradigm shifts. "Aha!" moments, as master puzzle designer Scott Kim would say. For example, another Zelda game I've been playing recently is The Legend of Zelda: Phantom Hourglass for the Nintendo DS. In that game, there is a particular puzzle that requires players to expand their perceptual field. In one particular part of one temple, players are told to step up to an altar and "stamp" their "map" with the location of a new area to be explored later (see Figure 2.15). At first, this puzzle threw me for a loop.

To access new areas in The Legend of Zelda: Phantom Hourglass, you first need a "sea chart" for that area. The new location to be stamped is outside of your current available areas, but clearly corresponds to the map you've had access to and used throughout the game. I systematically exhausted each possible action I had ever used in the game, and then some I'd only ever used in other games. I pressed buttons, I scrubbed the stylus back and forth in every conceivable pattern. I drew X's and O's, I traced the pattern of the Triforce over and over again. Nothing in my perceptual field equipped me to deal with this puzzle. Clearly, my current understanding of this system was inadequate and flawed somehow. I had to take a step back and reflect on my perceptual field. So I started thinking about other possible interpretations of stamping, and other possible ways I might stamp this stupid map, short of putting the DS on the floor and stamping it with my foot. The solution eventually occurred to me: close and open the DS quickly. It's sort of a stamping motion, as you can see in Figure 2.16.

The problem was my perceptual field: in my combined understanding of the DS, how the DS functioned and the possible actions in the game, there was no reference point for using the functional opening and closing of the DS as an action in the game. Across all my experiences with playing DS games—which are numerous—I had never come across a game that used the functional closing and opening of the DS' lid as a button. New Super Mario Brothers says "Goodbye!" in Mario's voice if
you close the lid of the DS while the game is still on, but that was my only point of reference. My only hint in the entirety of my perceptual field. I solved the puzzle through a paradigm shift, by seeing the system in a new and different way. I felt a rush of pleasure and enjoyment as I incorporated this new information about the very nature of this game’s universe into my perceptual field. I had amended and changed my mental model not only of this particular game, but of all games I’ll ever play on my DS in the future. This was not an error or a breakdown in my ability to interact with this system, but the point of the puzzle and, in some sense, the whole game.

So what’s going on here? I was rewarded with pleasure by expanding my perceptual field. Norman’s mental model would call this an error, blaming the designer for misleading the player. Since this is one of the fundamental pleasures of playing this game, clearly something’s missing. The perceptual field gives us a way to understand that game feel is as much about how players feel about a particular space and their relationship with it, as it is about the dry clinical details of the mental model of that space they keep in their minds to help them deal with and understand events in that game world. The dry details are important—they represent the player’s understanding of the physics and rules of the game world and are a great way to find dissonance that causes player confusion—but they are not everything.

Perception Is a Whole-Body Experience

Eyes, ears, tactile sense, proprioception—there is no separation when a person perceives something.

For example, a fork. It’s shiny and it has a pointy end. It’s cold, hard and solid, but easy to pick up. I can eat food with it. It sinks in water. My ideas and perceptions about what that object is, how it behaves and what it means, combine into the concept of “fork.” This is my attitude toward forks and generalizations I can apply later to other fork-like objects. The subconscious nature of perception masks a complex process, one which involves all the senses and which constantly and rapidly brings us closer to the world we inhabit.

The takeaway here is not to think of each kind of stimuli as somehow separate, but as an integrated part of perception. This is how a combination of visuals, sounds, proprioceptive sensations (from the position of the fingers on the controller or whatever) and tactile sensations (from controller rumble or haptic feedback) become a single experience in a game. A game world substitutes its own stimuli for those normally created by interacting with the real world, but the experience of perception is much the same. This also indicates why we’re so sensitive to inconsistencies between stimuli. If a large hulking mass of a character steps through a staircase or has their arm pass through a wall, the brain says, “Hey, that’s not right!”

The experience of perception of real-life phenomenon never has inconsistencies across stimuli so the brain has a hard time ignoring them when they happen in a video game.

Tools Become Extensions of Our Bodies

As we said in Chapter 1, a tool, once picked up, is an extension of the senses. It is used for both action and perception. Intent and action can be expressed through the tool as though it were a part of the body, and feedback flows back through it.

Another way to visualize this is to think about a blind man’s cane. When the blind man first starts using a cane, it is unfamiliar and requires a lot of reflection. The tapping movements are unpracticed, abstract movements to him. As he gradually builds skill at perceiving the world through the cane, he is able to more accurately and effortlessly tap around, getting a clearer read on his surroundings. His intent begins to flow effortlessly from the cane, any barrier between himself and the cane is removed, and the cane becomes an integrated part of his perceptual field.

The cane now acts just as his hand would; it probes around, touching things, interacting with the world around him and returning him crucial orienting feedback. This helps him establish a much larger personal space (also sometimes called the “perceptual self”). Quite literally, his perceptual field has been extended to a much larger physical area around him. Effectively, he’s made his arm’s reach much greater. We might say that by integrating the tool of the cane, he’s changed his world.

So what does the blind man and his Cane of Reaching +3 teach us about game feel? One interesting distinction is between body space and external space. When we interact with the world, we perceive our bodies in two ways, as part of our self and as one object among the many objects of the external, objective world. As Dag Svendsen puts it: “The bodily space is different from the external space in that
it exists only as long as there are degrees of freedom and a skillful use of this freedom. The bodily space is mainly given by the subject’s specific potentials for action. For a totally paralyzed body with no kinesthetic experiences, there is no bodily space. Different bodily states give rise to different spaces, and so do external factors such as clothing, tool use and different kinds of prosthesis. It is important to notice that learning a new skill also changes the body space.

This idea, that bodily space is defined by the potential actions of one’s body in the world, correlates very clearly to the way players interact with game worlds. Players tend to think in terms of abilities and constraints within a game world. It is possible, and often desirable, to create a game where abilities change across time, where the avatar itself has different tools that change across time. For example, if I’m playing as Samus Aran my abilities, my “bodily space” as Samus within the game world of Metroid, are defined by the abilities currently available to me. I may or may not have the Morph Ball. If I do, I can transform into a small rolling sphere and explore tiny corridors. The very nature of the world has changed, as has my potential to interact with it. The objective world of Metroid is the same as it ever was—every block is still in the same position it was before. My abilities, my verbs, my virtual bodily space have changed the world.

The interesting thing about this thinking is that it does not consider Samus Aran a tool. To say that we can incorporate a tool into our bodies, that the tool can become an organ of expression and perception, and that our identity and perceptual field subsume it is useful, doesn’t quite define what happens when a player takes control of an avatar in a game. I wouldn’t call Samus Aran a tool. Not just because she has an “identity” as a freestanding character that I’m temporarily inhabiting and controlling, but also because she has her own bodily space. She has her own tools which can become a part of her body and extend her own perceptual space. In this way, a video game world is truly a microworld, and perception within this microworld is a surrogate for real-world perception. It’s an interesting idea—it seems like why if we can go from “being” Gordon Freeman one moment to cursing his vile clumsiness the next. This is because the constructed subdomain of a video game reality provides two other kinds of spaces: virtual bodily space and virtual external space.

A video game has its own model of reality, internal to itself and separate from the player’s external reality, the player’s bodily space and the avatar’s bodily space. The avatar’s bodily space, the potential actions of the avatar in the game world, is the only way in which the reality of the external reality of the game world can be perceived. As in the real world, perception requires action. The difference is that the action in the game world can only be explored through the virtual bodily space of the avatar. Players extend their perceptual field into the game, encompassing the available actions of the avatar. The feedback loop of perception and action that enables you to navigate the world around you is now one step removed: instead of perceiving primarily through interaction of your own body with the external world, you’re perceiving the game world through interaction of the avatar. The entire perceptual apparatus has been extended into the game world.

To wrap back to our earlier discussions of identity and game feel, how does this concept of avatar as perceptual substitute, rather than extending tool, relate to proxied embodiment? Because a game world represents its own reality external to its avatar’s bodily space in the same way that the physical world is external to our own bodily space, it seems much more like a substitution than an extension. The same might be said for identity. We said that objects outside ourselves—and objects in a game world—can become extensions of identity. Vessels for identity might be more accurate. The view of tool as extension of body defines the “self” in terms of perception. The perceptual self is the immediate surrounding environment and your ability to interact with it, your potential for action. To say “he hit me!” instead of “he hit my car” or “his car hit my car” is an artifact of the way we perceive the immediate environment around us and the fact that an inanimate object can become a part of the perceptual self, part of the perceptual field. You literally perceive the world through the car as you actively control it. Again, though, the way we perceive game feel seems to be much more of a substitution than an extension. I perceive the world of Hyrule as Link, via his virtual body space. My identity intermingles with Link’s as I take over and make my own his skills and abilities, his bodily space.

Summary

Where and when does real-time control exist? On the human side of the equation, we categorized three types of processors (perceptual, cognitive and motor), which work together in a closed feedback loop. This feedback loop results in an ongoing correction cycle. In a video game, at the point where action normally goes out to physical reality, the designer substitutes a game world for the real world. This reinforces the idea from Chapter 1 that game feel is an experience of a unique physical reality.

On the computer side, real-time control relies on sustaining three time thresholds: the impression of motion, perceived instantaneous response and continuity of response. Knowing the duration of the human correction cycle and the relationships between the three human information processors, we can say with certainty that a particular game has or does not have real-time control. The unknown variable here is the player’s perception. In the end, game feel is an impression in a player’s mind. Examining the frame rate, response time and continuity of response in a game and comparing this to the thresholds of 10 fps for motion, 240 ms for control and 100 ms for continuity gives us a baseline and is useful for classification. But motion at 10 fps feels stilted and a 200 ms response time feels sluggish. These impressions can be smoothed over by the use of gestural input or by playing back animations. In the end, the player’s perception is what’s important. The ultimate goal for game feel is to create an impression in the player’s mind.

Finally, we looked at some of the other implications of human perception:

1. Perception requires action.
2. Perception is a whole-body phenomenon.
3. Perception is an effortless fusion of visual, aural, tactile and proprioceptive stimulus.
4. Perception is an ongoing process of skill-building.
5. Perception can be extended to tools.

These explain, in terms of human perception, the experiences outlined in Chapter 1. Understanding how human perception works offers us insight about the imperfect apparatus of human perception that we're designing for. Knowing this will help us to develop a palette of game feel that's separate from the emulation of reality and doesn't borrow from film or animation except where applicable. If we understand how perception works, we can build games that feel good instead of trying to build games that feel like things that feel good.
FIGURE 3.1 The model of interactivity brings together all the elements of the gamer, the game and the world around him or her.
The flag in a capture the flag multi-player game is one example; for the player currently holding the flag, the game feels different.

At the lowest level, rules can further define the physical properties of objects. How much damage it takes an avatar to destroy an enemy changes the player's perception of how "tough" that enemy is. An enemy that takes one hit to destroy will feel fragile, while a boss monster that takes 20 hits feels much more solid.

Summary

The six pieces of the game feel system that are malleable for the game designer are:

- Input—The physical construction of the device through which player intent is expressed to the system and how this changes game feel.
- Response—How the system processes, modulates and responds to player input in real time.
- Context—The effect of simulated space on game feel. How collision code and level design give meaning to real-time control.
- Polish—Effects that artificially enhance impression of a unique physical reality in the game.
- Metaphor—How the game's representation and treatment change player expectations about the behavior, movement and interactions of game objects.
- Rules—How arbitrary relationships between abstracted variables in the game change player perception of game objects, define challenges and modify sensations of control.

For each of the six pieces of the game feel system, I've pointed out a few different things that are instructive to measure when examining a particular mechanic or a particular game feel system.

Each of these is discussed in more detail in Chapters 6 through 11. We'll be looking at what can be measured and what's useful to measure. We will be pursuing both soft and hard metrics. For each measurement, we'll go through why this is useful to know about a particular game and how it helps us compare the feel of two games in a meaningful way.

The point of measurement is to derive general principles about game feel which can be applied to future designs and to let us meaningfully compare the feel of two games. Instead of taking shots in the dark, emulating existing mechanics or trying to shoehorn someone else's tuning into your system, you want to be able to understand the tools at your disposal. If you want your game to feel like Sonic, MegaMan or Burnout: Revenge, you'll be able to do it with a deeper understanding. You might not have the exact recipe—it's probably secret—but at least you won't be staring at a finished cake wondering what kind of sugar was used.

Input Metrics

The pianos (think input devices) of Ludwig Van Beethoven's day were flimsy, cheap things. In the course of his exuberant performances, he often broke 10 to 15 piano strings, sometimes damaging the piano beyond repair. It was not just his playing that destroyed pianos, but the music itself: it was not written for the pianos of his day. Part of Beethoven's genius was his ability to look beyond the physical limitations of the piano and define a space of a greater virtuosity and musical expressivity than the one actually presented by the piano itself. When he looked at a rickety Viennese piano, he saw the robust grand pianos of today.

What Beethoven was able to see clearly—and exceed—were the physical and mental limitations imposed upon him. He understood that a tool or instrument inherently shapes and influences the activities that can be carried out employing it, both physically and mentally. Consider a screwdriver. A screwdriver is used to fasten things together. But it is a very specific kind of fastening, and the nature of that fastening is implied by the screwdriver itself. For one, it must employ a screw that matches the head of the screwdriver being used. If it's a Phillips head screwdriver, it needs a Phillips head screw, and the grooves in the screw head must be machined to a size comparable to the head of the screwdriver. A large Phillips head screwdriver cannot be used to screw in a tiny screw. A screwdriver, like all tools, contains within it a specific subset of possible uses, a possibility space for its use. As a tool, it defines what it can do, and, more importantly, what people will expect to do with it. If you buy a sports car, you're likely to get speeding tickets. If you have a hammer, everything looks like a nail.

This is an interesting notion and one which we rarely apply to the input devices used to control video games. Just how much does the design of a particular input device affect the feel of a virtual object controlled with it, and to what degree is game feel defined by the input device itself? In other words, to what degree is the possibility space of a virtual object defined by the physical object used to control it?

To answer this question, we need to be able to measure the input space represented by a particular input device. Next, we need to be able to compare the input space of one input device to another in a meaningful way. Finally, we need to examine
Micro Level: Individual Inputs

The first easily measurable thing about an input device is the number of separate, individual inputs it contains. My Xbox 360 controller, for example, has 15 separate inputs on it (Figure 6.1). This includes a couple thumbsticks; a directional pad; two “trigger” buttons; two “shoulder” buttons; four standard buttons; and some flimsy, seldom-used buttons for select, start, wireless resync and other miscellany. Culling out the inputs that are rarely used for game control, this leaves 4 usable inputs.

![Figure 6.1 The Xbox 360 controller has 15 input options.](image)

- X-, Y-, A- and B-buttons (standard buttons)
- Right and left “shoulder” buttons
- Right and left “trigger” buttons
- Directional pad
- Left thumbstick
- Right thumbstick

The common element every input possesses is the potential for motion. A button can be pressed down, a thumbstick pulled away from center and a mouse slid across a flat surface. In each of these cases, the input sends a specific type of signal to the computer. It is interpreted, responded to and fed back via the output devices (screen, speakers and so on). This potential for real-time manipulation and signal-sending is the fundamental property of an input. If you can’t move it in some way and have it send a corresponding signal to a computer, it’s not an input. The key to correlating seemingly unrelated types of input, then, is in this motion.

The first way to classify an input is as either discrete or continuous. That is, does it send signals continuously (joystick, mouse, steering wheel) or does it send individual, momentary signals (keyboard key, mouse button, controller button)?

Inputs that enable continuous input can also be categorized¹ like this:

- Type of Motion: linear vs. rotation. A mouse measures movement linearly (in two dimensions) while a steering wheel measures rotation.
- Type of Sensitivity: position vs. force. A mouse measures changes in position, while a joystick measures how much force is being applied against spring resistance.
- Dimensions of Motion: A mouse measures linear movement in two dimensions, as does a thumbstick. A trigger button measures linear movement in one dimension. A Wiimote measures rotational movement in three dimensions.
- Direct vs. Indirect Input: A mouse is indirect—you move the mouse on the desk and the cursor moves on the screen. The touch screen on the DS enables players to directly tap on or touch the thing they want to interact with.
- Boundaries on Motion: The thumbstick on an Xbox controller has a round casing enclosing it, while a mouse has no physical boundaries on its motion. The way that the motion of an input is bounded can change what it feels like to use it. For example, the slotted casing around the Nintendo 64’s thumbstick feels different from the smooth round casing of the Playstation 2’s.
- Sensitivity: Roughly, how many different states can the input exist in. A standard button is very low sensitivity; it has only two states (ON or OFF). A mouse

is highly sensitive by comparison; it has no physical boundaries making each tiny motion another possible state. Though inputs can be mapped to in-game motions that make them more or less sensitive, individual inputs have an inherent sensitivity.

- Signals Sent: What is the format of the signals each input sends to the game, and how do they change over time?

One way to visualize these properties for a particular input device is to hold out your hand as shown in Figure 6.2. This will also be useful for comparing movement of an input device to movement of the object being controlled, which can indicate whether a mapping is natural (in the Donald Norman sense).

Imagine that lines extend outward from your index finger, middle finger and thumb, as in Figure 6.2. Now imagine each finger as an axis. If you move your hand along any of these axes, you’re moving it linearly in a single dimension, X, Y or Z. If you rotate your hand around a particular finger you’re rotating in X, Y or Z. Mouse movement is unbounded, so if you slide your hand around the plane described by the X- and Z-axes (index and middle fingers) this is a good way to visualize the unbounded movement of the mouse (Figure 6.3).

A Playstation 2 thumbstick has the same kind of motion, but it has a boundary, so we can think about being able to move in that plane, but only so far in a particular direction (Figure 6.4).

Every button has two boundaries—fully pressed and fully released—between which it moves. Whether it’s a trigger button with a handful of states between the extremes of ON and OFF or a mouse button with only two states, there is a limit on movement. The same goes for thumbsticks and arcade joysticks with their circular plastic housings. (In contrast, Wiimotes and computer mice are two general-use input devices that have no built-in boundaries on their motion.) Boundaries are important to take note of because they reduce the overall sensitivity of the input to a particular range. Sometimes—as in the case of a thumbstick—the boundary can play an important role in defining the types of controlled in-game motions that are best suited to the input (Figure 6.5).
Now think about how many possible states there are between the boundaries as you move your hand around. For a thumbstick, it's a lot. For a mouse, even more. For a "trigger" button, it's more than 1, but less than with a thumbstick or mouse. This is a rough measure of the sensitivity of the input device.

The amount of sensitivity an input possesses is a soft metric. It is possible to calculate the actual, physical number of states an input can reside in—two for a standard button, something like 1,520,000 for a mouse on a 1,600 x 1,200 desktop—but that comparison doesn't accurately portray the sensation of using these inputs. It's more like the difference between an Etch A Sketch™ and a paintbrush. You can paint a picture with either, but the paintbrush offers a lot more versatility. The idea is that different inputs have different amounts of sensitivity inherent in their design. A standard button has the absolute minimum amount of sensitivity. It's either fully on or fully off. The dial on a classic Breakout paddle has a bit more; its one-axis rotation has a great deal of sensitivity in a limited way. More sensitive than either of those is an arcade joystick, which moves freely in X and Z, but which

is bounded on all sides by a circular plastic housing. Figure 6.6 shows a (rough) scale for input sensitivity.

More or less sensitivity in an input can be desirable depending on the intended feel, and how that feel is intended to fit in with the design as a whole. It's like the color blue—whether or not it's appropriate to use it depends on the context and the desired result.

The final thing to measure about an input is the types of signals it sends. This is the data you will be mapping to some response in your game, so it's important to keep track of the raw form in which it arrives in the computer. This is a hard metric—an input signal always ends up in a simple, numeric format a computer can easily understand—but it also supports our understanding of the soft metric sensitivity of the input device. A single button sends a binary signal, "up" or "down." Measure this over time and you get signals for "up," "pressed," "down," and "released." A mouse sends a pair of values, one for each axis, that get updated every frame. So a mouse might send 60 different pairs of values in a second, like Table 6.1.

The signals sent by the mouse are more complex than those sent by the button. Figure 6.7 shows the types of signals sent by various input devices.

FIGURE 6.5 The boundary of the Xbox thumbstick is important for the feel of Geometry Wars: Retro Evolved.

FIGURE 6.6 Sensitivity of input devices.

<table>
<thead>
<tr>
<th>Frame</th>
<th>Signal Sent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(0.52, 0.11)</td>
</tr>
<tr>
<td>2</td>
<td>(0.51, 0.21)</td>
</tr>
<tr>
<td>3</td>
<td>(0.50, 0.34)</td>
</tr>
<tr>
<td>4</td>
<td>(0.31, 0.42)</td>
</tr>
<tr>
<td>5</td>
<td>(-0.1, 0.61)</td>
</tr>
</tbody>
</table>
Input Measurement Examples

Measuring all these properties for each input is useful for understanding game feel because it is an immutable part of the interface to a particular game. Every game played on the Nintendo Entertainment System was designed to respond to that particular configuration of eight simple, two-state buttons. If we understand just how simple those buttons are in functionality, it becomes all the more remarkable that expressive mechanics like the swinging in Bionic Commando or the slippery movement of Super Mario Brothers were created. More importantly, if we look at the number, type and sensitivity of inputs used to create a mechanic, we can begin to get much closer to a meaningful comparison between the feel of games created with different input devices. Comparing the feel of Halo to the feel of Contra becomes a lot more viable if you understand that Xbox thumbsticks are inherently much more sensitive than the directional pad and two buttons of the NES controller.

Standard Button

The standard two-state button (Figure 6.8) is the most basic type of input in general use today. The button moves only in one axis, the Y-axis, and the motion is linear. A spring beneath the button pushes it upward constantly. A plastic housing or catch stops it at a certain point, which represents the fully released state. When the player presses the button, he or she overcomes the force of the spring and the button slides downward, in the Y-axis, into the controller. It's stopped at a certain point and then sits in the fully depressed or OFF state. There are no states in between these hard plastic boundaries for Y-axis movement. The button, then, has only two states: ON or OFF.

These characteristics also describe the essential functionality of a keyboard key, a mouse button or any other simple two-state button such as the "shoulder" buttons on a typical modern controller, though these are larger and can be depressed only a tiny amount. These buttons are remarkable for their lack of sensitivity. There's very little that can be expressed by a standard button alone. The feedback from the button is discrete rather than continuous, meaning that the signals it sends happen at one particular moment in time. The signals are binary; the button is either on or off at any given time. As brilliant one-button games like Ominous Development's Strange Attractors prove, however, it's possible to map a single button to a complex, nuanced, sensitive response from the game, but the button by itself is very limited as an input. It is not really possible to create functional input device with fewer states, with less expressive potential than a single two-state button.

The Y-axis movement of a standard button has hard boundaries at the fully released state (where the spring's pushing force is stopped by a plastic catch) and at the fully pressed state (where the player's pressing force is stopped by a plastic catch).
Trigger Button

Like the standard button, the “trigger” button (Figure 6.9) typically found on modern controllers moves in only one axis. In this case, I’d call it the X-axis, as it’s usually on the front of the controller and is usually operated by the index finger. Again, though, this is all relative to the controller’s position in space.

Trigger buttons are unlike standard buttons because they recognize many states between their boundaries. Between the fully pressed and fully released states, there is a zone of sensitivity inside which it’s possible to have many different positions of the trigger. Fiddling around with my Xbox 360 controller, I estimate that there are four or five discrete states including fully on and fully off. Like a standard button, the trigger is spring-loaded and defaults to a fully extended released state. The major difference is in the button’s range of motion. By carefully depressing the spring a certain amount, the player can keep the button a quarter, half or three-quarters of the way depressed without pushing it fully to one extreme or the other. This X-axis movement has hard boundaries at the fully released state (where the spring pushing force is stopped by a plastic catch) and at the fully pressed state (where the player’s pressing force is stopped by a plastic catch).

A trigger button typically returns a float value, a number between 0.00 and 1.00. For example, across three frames it might return 0.63, 0.81 and 0.97 as it is pulled from off to on.

Paddle

Though they’re not in common use anymore, it’s interesting to note that the paddle controllers sold with many of the first home consoles used a hard-boundary, one-axis rotation. There was one spinner input on the front of the controller (Figure 6.10). It was gripped between the thumb and forefinger and could be rotated left or right a certain amount before it reached a defined hard boundary point and the plastic would catch and stop it. Through a combination of factors, this input type fell out of vogue, but it was quite a sensitive input, with hundreds of possible states between fully left rotation and fully right.

A paddle controller returns a float value, in a range from -1.00 to 1.00. When the paddle knob is centered, it’s at 0.00. As it is rotated left of center, it goes negative (something like -0.26) and as it is rotated right it goes positive (something like 0.41).

Thumstick

A typical thumstick is movable in two axes simultaneously, left-right and up-down (Figure 6.11). It’s spring loaded in both directions, though, so it will always seek back to its straight-standing centered position. In most cases, the housing containing the thumstick provides the hard boundary against which it stops when pushed fully in any direction, and it is usually smooth and round. With a thumstick, it’s no longer meaningful to track the total number of possible states. When using a thumstick to control something in a game, there is no notion of discrete states, but a fluid, smooth sense of highly accurate positioning.

The thumstick is often used as a direct or semi-direct stand in for the intended in-game motion. Rolling the stick across the edge of its round casing creates a carve motion for a ship in Geometry Wars, a quick heel-turn for Mario or a quick Jab in Fight Night. With a thumstick, one can be said to “feather” or “flick” a
control; these are properties of highly sensitive input, one which vastly more sensitive than a standard or trigger button. From its centered position, the thumbstick can only be displaced until it comes into contact with the circular housing, which constrains its movement.

The thumbstick can be displaced from its centered position left to right and up to down, which creates a perceptually infinite number of possible states for the player.

The thumbstick returns two constantly changing float values at the same time, one for each axis of motion. Left to right motion (in the X-axis) returns one float value between -1.00 and 1.00, while up and down motion (in the Z-axis) returns another. The format of the signal could be: (-0.16, 0.93).

**Mouse**

A mouse—and here I’m talking about the input which detects positional movement, not the clickable buttons—is similar to thumbstick in that it enables movement in two axes. In the case of the mouse, however, there are no in-built boundaries (Figure 6.12). There’s a sort of a soft boundary in the sense that you need a small section of flat surface to rest the mouse on, and that potentially can lead to the mouse falling off a table or running into something. In practice, the boundary most often exists in software rather than hardware. The edge of the screen stops the cursor before the edge of the table.

Because there is no explicit boundary, the potential for different states is even higher than with a thumbstick. All position is relative, and there is no spring pushing the mouse back into a neutral position. These factors combine to make the mouse the most sensitive input device in common use today.

The boundaries for a mouse’s movement are in software, stopping the cursor (the meaning of further mouse movement) at the four edges of the screen (top, bottom, left and right). Technically, there’s also a boundary at the edge of whatever surface you’re mousing on, but this boundary is almost never reached due to the typical ratio of physical mouse movement to computer space movement. You get a lot of screen movement for a little mouse movement, so you don’t run your mouse off the table very often.

The mouse is a highly sensitive input device. On my 1,200 × 1,600 pixel desktop, the mouse cursor can potentially rest on any one of 1.9-something million pixels. In practice, a user can’t be expected to accurately hit targets smaller than a certain size (checkbox on a dialog), but the sensitivity is there.

Like the thumbstick, the mouse returns two separate float values in the form (0.18, -0.28). But in the mouse’s case, what’s being returned is a displacement. How far the mouse has moved in both the X and Z directions since the last frame, in other words. This is often mapped directly into screen space movement (as in the movement of a cursor) but the movement is not absolute. If it were, you would not be able to pick up the mouse, move it and put it back down to continuously move the cursor in one direction.

Table 6.2 compares all these input devices.
### Table 6.2 Input Metrics

<table>
<thead>
<tr>
<th>Type of Motion</th>
<th>Standard Button</th>
<th>Trigger Button</th>
<th>Paddle</th>
<th>Thumbstick</th>
<th>Mouse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>Linear. The button moves in only one axis, the vertical or Y-axis.</td>
<td>Linear. The button moves along one axis linearly.</td>
<td>Rotation. The paddle's motion is rotational around one axis.</td>
<td>Linear. The thumbstick moves linearly along the X- and Z-axes.</td>
<td>Linear. The mouse moves linearly along the X- and Z-axes.</td>
</tr>
<tr>
<td>Y-axis only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X-axis only</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X-axis only</td>
<td></td>
<td></td>
<td>Y-axis rotation only.</td>
<td>The thumbstick moves in the X and Z dimensions.</td>
<td>The button moves in the X and Z dimensions.</td>
</tr>
<tr>
<td>Indirect</td>
<td>Indirect; you press the trigger in your hand and something changes in the game. You don't directly touch the screen with the trigger.</td>
<td>Indirect; you don't directly touch the screen with the trigger.</td>
<td>Indirect input.</td>
<td>Indirect input.</td>
<td></td>
</tr>
<tr>
<td>Boundaries on Motion</td>
<td>Two hard boundaries, fully pressed or fully released.</td>
<td>Two hard boundaries, fully pressed or fully released.</td>
<td>Two hard boundaries, typically round (but can also be square or grooved, which changes the feel of using the joystick).</td>
<td>Four soft boundaries.</td>
<td></td>
</tr>
<tr>
<td>Sensitivity</td>
<td>The button has only two states, on and off.</td>
<td>The paddle is used to move in two directions.</td>
<td>Hundreds of possible states between the two extremes of rotation.</td>
<td>Thousands of possible states between up down, left/right move, and all the positions in between fully released and pressed against the housing.</td>
<td>Millions.</td>
</tr>
</tbody>
</table>

### Table 6.2 (Continued)

<table>
<thead>
<tr>
<th>Type of Sensitivity</th>
<th>Standard Button</th>
<th>Trigger Button</th>
<th>Paddle</th>
<th>Thumbstick</th>
<th>Mouse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Force</td>
<td>Force. The button is sensitive to how far it is pressed.</td>
<td>Force. The paddle knows how far it is moved.</td>
<td>Force. The thumbstick is sensitive to how far it is moved.</td>
<td>Force. The mouse is sensitive to changes in position.</td>
<td>Position.</td>
</tr>
<tr>
<td>Sensitivity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Float value between 0.00 and 1.00.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Two float values, each between -1.00 and 1.00.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Float value between -1.00 and 1.00.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Float value between -1.00 and 1.00.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Macro Level: The Input Device as a Whole

That takes care of the micro level of individual inputs. Now let's recompose the inputs into a complete input device and look at how inputs combine to create an expressive potential greater than the sum of their parts. To keep things simple, let's return to the NES controller, as shown in Figure 6.13.

Examining each input, we must concede that this is very low-sensitivity input device. There are six buttons for use in active gameplay, and each of them is a standard two-state button. More than that, certain buttons are mutually exclusive by design. You can't press up and down on the D-pad at the same time, nor can you press right and left simultaneously. But this controller has more sensitivity than its individual inputs would at first indicate. Even with six buttons and the limitations imposed by the D-pad, the possible combinations of buttons look something like Figure 6.14.

In order to truly come to terms with the input space of an input device as a whole, you have to consider it at both the macro and micro levels. How much sensitivity does each input have, and how do the layout and design of the controller reduce and/or increase sensitivity? In the case of the NES controller, the sensitivity is reduced by the mutually exclusive D-pad buttons and increased by the combined possibilities of the buttons, laid out as they are for use with both thumbs.
Again, this isn’t a hard metric. We can measure the total number of inputs and all the permutations of combining them as specific numbers, but that’s not especially useful. We’re interested in getting a rough idea of how the inherent sensitivity of an input device compares to the sensitivity of another device. It suffices to know that NES controller is much less sensitive than a computer mouse; from that point it is possible to make design decisions relative to our intended feel and to compare the feel of two games controlled with input devices of varying sensitivity.

**Tactile Level: The Importance of Physical Design**

It’s also useful to understand how the input feels physically. This is an overlooked aspect of game feel: the tactile feel of the input device. Games played with a good-feeling controller feel better. For example, the Xbox 360 controller feels good to hold; it’s solid, has the proper weight and is pleasingly smooth to the touch. By contrast, the first-run Playstation 3 controllers were lamented as being light and “cheap feeling”, like one of those third-party knockoffs.  

This difference in tactile feel of the input device has surprising implications for the feel of a given game. When I prototype something—platformer, racing game, whatever—it will feel noticeably better if I hook up the inputs to my wired Xbox 360 controller instead of using simple keyboard inputs. Of course, this is a very soft metric. It’s tempting to simply say “de gustibus non est disputandum” (there’s no accounting for taste) and leave it at that, but there are some noticeable, measurable qualities of various inputs that can be observed and taken into account.

**Weight**

The weight of a controller is an important quality for an input device. A heavier, more solid-feeling controller is perceived as being of higher quality. For a game’s feel, this can go a long way toward making actions feel weighty, powerful or satisfying. Of course, it’s also possible to push too far into the heavy direction, as the original Xbox controller seemed to, but in general input devices seem to trend toward being too light, flimsy and cheap-feeling. This significantly affects the feel of control of a virtual object.

**Materials**

The material used to construct the device has an impact on the way the user feels about the controller and, therefore, the game. The white plastic that houses my Xbox 360 controller has a smooth, pleasingly porous feel. It’s almost like skin. My Wiimote and Playstation controllers feel like plastic. It’s a subtle difference and measuring its impact on game feel is extremely difficult. All I can say is that

---

1By “Zeus” from http://forums.maxconsole.net/archive/index.php/t-19989.html
I prefer holding my Xbox controller. The Dell mouse I’m using currently has a similar porousness but is much coarser, making it less pleasurable to handle than my Xbox controller. This has a mostly subconscious effect on my interaction with these objects, and on how I perceive virtual objects controlled with them.

**Button Quality**

By “button quality,” I mean the feel of the spring resistance. Particular buttons on input devices are often described the same way the feel of a game is described: tight or loose, quick-responding or sluggish. This feel is contingent on the quality, construction, and type of springs that drive the motion of the input, whether they’re on a button or a joystick.

As James Goddard of Crunchtime Games says, “There is a huge difference in how input devices—even similar ones—can have on the feel of a game. Most people can tell the difference in the feel of control when a game is cross-platform but most do not know what exactly is causing it. Even a majority of developers really are not trained to know. The usual argument is that a specific platform’s controller is ‘just better.’ Assuming a game engine is truly ported equally across the platforms and button layout is close to the same, what people are perceiving as better is the actual mechanical differences in the sticks/buttons tension and mechanical ‘travel’ distance. This sensitivity can come down to millimeters.”

The design of controllers has a lot to do with industrial design and product design. Controllers are consumer products after all, as are game consoles, computers, mice, keyboards and handhelds. Every piece of hardware upon which game feel is built is a consumer product. The physical design of hardware can change the feel of control over virtual objects.

**Summary**

To summarize, we can categorize an input device according to individual inputs, the input space of the device as a whole, and tactile feel resulting from the materials and physical construction of the device itself.

Individual inputs can be measured according to their dimensions and types of movement, whether they track position or force, whether they directly or indirectly change things on the screen, the boundaries on their motion, and the signals they send to the computer (hard metrics). They can also be measured according to their sensitivity (soft metric).

The input space of the device as a whole can be measured by looking at how many different inputs there are on the device, and the ways in which they can be combined (hard metrics).

The tactile feel of the device can be measured by the feel of each input (the resistance to movement, springiness, etc.) and the feel of the input device as a whole (heavy and solid versus light and flimsy, physical properties of the materials). Both of these are soft metrics, affecting the feel of games in a mostly subconscious way.

**Response Metrics**

When I say response, I mean the game’s response to player input. The output, in other words. There are many different ways an input signal, once received, can be processed before returning to the player in the form of feedback, but the process has three essential steps:

1. Input signal comes in
2. Input signal is interpreted and filtered
3. Input signal modulates some parameter in the game

To measure the response of a particular game to input, we begin by looking at how each signal from the input device is mapped to a change in the game. What parameter does it modulate, and how does it change that parameter over time? And what are the relationships between those parameters?

There are many different ways an input signal can modulate a parameter in a game. While an input device is a physical object constructed out of plastic and springs, an avatar in a game has no such constraints. An input can be mapped to a change in the position of an avatar, as it is in Megaman. Inputs can also be mapped to rotation, as they are in Asteroids and Gran Turismo, where a forward thrust is steered via angular rotation. Or rotation and positional changes can be mapped to the same input, as they are in Jak and Daxter, Super Mario 64 and Geometry Wars, where pressing a direction with the thumbstick rotates and moves simultaneously.

An input can also be mapped to the creation of a new entity, as when Megaman fires his weapon and “spawns” a bullet or when Guile throws a sonic boom in Street Fighter II. In this case, an entirely new entity is spawned by an input, often from the position of another avatar. This entity often has its own properties of movement and its own velocity.

Another possible response to input is the playback of a linear animation, as in the games Soul Calibur, Samurai Showdown 4 and Street Fighter II. You press a button and a “move” happens. A move consists primarily of animation, created by a professional animator, and some game-relevant spatial movement. It occurs at the position in space and time of your choosing, but once triggered, the animation plays back as a linear sequence of frames, as it would if it were an animation playing...
back on a television. The duration of the animation may be short, but it’s interesting to note that what’s been mapped is still the linear playback of an animation to a particular input. This can be taken to ridiculous extreme, as it is in the original Prince of Persia, where the game can only be controlled via the playback of linear animations. Your input serves only to change which animation is playing at a given time, and you’re tasked with managing the movement of the character in that stiff, robotic way.

Still another thing that input can be mapped to is a change in one or more parameters in a simulation. For example, in Mario Kart DS, there’s a simulation running every frame which computes the game’s internal model of the karts—their relationships to one another, their weight, their mass, their velocity, their rotational force and of course, their friction values—how the forces that are acting on the kart are reconciled with the friction value of the surface they’re currently in contact with and how that affects their motion. You can map an input not only to moving the kart forward and rotating it, but to the friction value itself, altering the resulting motion of the entire system. When you enter the “powerslide” state by pressing the R-button, what you’re really doing is changing the friction value. It’s decreased, enabling you to corner better by carving less (sliding sideways instead of gripping the road, in other words).

Generalizing the possibilities, an input signal might:

- Set a new position for an object each frame.
- Set a new orientation for an object each frame.
- Add a force or torque to a simulated object, causing it to rotate or move.
- Modify a simulation variable, changing gravity or the friction of a car’s tires.
- Play back an animation from start to finish, like a single move in a fighting game.
- Change the speed at which a looping animation plays back.

The process of hooking up input signals to specific parameters and determining how they will modulate those parameters over time is known as mapping. For real-time control, however, there is a specific subset of mapping required: mapping to motion.

If there is real-time control, the input signals will be mapped, directly or indirectly, to the motion of an avatar. The movement of that avatar can, for the most part, be measured using criteria similar to the ones we used to measure input:

- Type of motion: Linear vs. rotation. Does the avatar move linearly or rotate?
- Dimensions of motion: In what dimensions, X, Y or Z, does the avatar move or rotate?
- Absolute or relative motion: What frame of reference does the motion use? Is the motion relative to the avatar, as in Asteroids or the camera as in Mario 64, or some other point in the world?

**Figure 7.1** The Mario Avatar moves in two dimensions, X and Y.

- Position versus rate/magnitude: Does the input modify a position, a rate or a magnitude? A mouse cursor is usually mapped to changes in position. Pushing the thumbstick to the left in Halo changes the rate at which the avatar turns (halfway causes a slow turn, while fully pressed turns very quickly).
- Direct or indirect control: Does the input modify the avatar directly or does it add forces to a simulation or cause another object to move or rotate? For example, in Zuma, movement of the mouse cursor determines which direction the frog will face.
- Integrated or separate dimensions: Does the input change one parameter in the game or many? For example, Geometry Wars maps both thrust and rotation of the ship to the left thumbstick. Jak and Daxter does this as well, changing both speed and rotation with one thumbstick.

The point of taking stock of the avatar’s movement in this way is to hone in on exactly which parameter each input is mapped to. We can look at the Sonic avatar and say he moves along in the XY plane. Or we can look at the movement of Crash Bandicoot and say, he moves in an XZ plane, but he can also jump or fall in the Y plane, as well as rotate in the Y-axis to change his direction. Kratos from God of War moves in a similar way. Knowing which dimensions an avatar moves in, we can identify which inputs control movement in which dimensions, and whether that movement is linear or rotational. For example, Mario’s horizontal movement is controlled by the left and right directional pad buttons, while his vertical movement is controlled by the A-button.

**Attack, Decay, Sustain and Release**

Regardless what parameter an input is mapped to—position, rotation, animation playback and so on—the modulation of a parameter over time will have some kind
of curve. One way to describe this curve is as an ADSR envelope. ADSR stands for attack, decay, sustain and release. An ADSR envelope describes the modulation of a parameter over time, in four distinct phases (Figure 7.2).

Such envelopes are used to describe the modulation of the sound of musical instruments. For example, when you play a note on a guitar, the resulting sound can be described in terms of attack, decay, sustain and response. The note is loudest just as the string is plucked, but it takes some time to go from silent to loud. This is the attack. From the loudest volume, the sound then drops down again before reaching a stasis point. This is the decay. The point at which the volume stabilizes is the beginning of the sustain part of the envelope. This lasts until the sound begins to fall off again, eventually returning to silence. This final period is the release. Graphed over time, it looks like Figure 7.3.

Contrast this with a pipe organ. The note starts at a constant volume, continues to play at that same volume and falls silent almost instantly when the button is released (Figure 7.4).

ADSR envelopes are often used to modulate the output of digital instruments to make them sound like their physical, real-world counterparts. They are also a good way to think about the modulation of parameters in a game relative to specific input. For example, look at the "left" input in Super Mario Brothers (Figure 7.5).

In this case the vertical axis of the envelope is movement. There is an attack phase as Mario ramps up to his maximum speed, no decay, a sustain as long as the button is held and a long release when the button is released. The result is that Mario speeds up gradually over time (Figure 7.6).

Now compare Mario's left motion to Donkey Kong's, shown in Figure 7.7. When Jumpman (the pre-Mario character from Donkey Kong) moves, he has no attack and no release. The moment the joystick is activated, he moves at a constant speed in the appropriate direction (Figure 7.8).

Once we know what parameter in a game is mapped to what input, we can measure the modulation of that parameter over time relative to the input signals coming in as an ADSR envelope. Assuming that the parameter being modulated feeds into the real-time motion of an avatar, we can make generalizations about how players are likely to experience the sensation of control based on this envelope.
A longer attack phase results in a floaty or loose feel. This is not necessarily a bad thing; the thruster mechanic in Asteroids has a long attack, and players generally seem to enjoy that feel. When a long attack phase begins to cause trouble, however, is when there seems to be no immediate response to input (Figure 7.9).

This is problematic because it starts to erode the impression of instantaneous response. There may be some small change happening immediately, but if the player can’t perceive it, the game feels unresponsive. An envelope like the one in Figure 7.10, which has a rapid initial attack but a long attack phase in general, will have both instantaneous-feeling response and a loose, organic feel. On the other end of the spectrum, a short attack phase will tend to feel tight and responsive (Figure 7.11).
**Playable Example**

You can experience the difference in example CH07-1. Press the “1” key for an unresponsive, the “2” key for a responsive but loose feel.

Even with tight, responsive controls, the attack phase usually has some nonlinear curving to it. In other words, the attack phase is a curve, not a straight line. This keeps the flowing, organic feel while enhancing the perception of instantaneous response. On the other hand, when the attack phase is short and when there is a more linear progression from off to on, most players describe the feel as twitchy (Figure 7.12). This too can be desirable depending on the intended effect. If the attack is totally linear and very short, the controls can feel stiff.

What’s interesting is that these sensations—floaty, twitchy, tight, loose, unresponsive—all exist on the same continuum. They’re just slightly different envelopes, slightly different modulations of motion over time. That motion could be direct or indirect, a force or a rotation; regardless, changes in attack will alter the feel of control.

Attack and release are often mirrored, as in the horizontal running of Super Mario Brothers. After you release the button, it takes Mario the same amount of time to slow back down to zero as it does to speed up from a standstill to his maximum speed. The soft release maintains the loose feel after the button is released. Having no release, as in Donkey Kong, feels more abrupt.

When decay is present in game control, it’s usually by accident. Sometimes a game designer will inadvertently make the speed of movement faster just after a change in input than during the eventual sustain period. This means that to maintain maximum speed, constant button pressing is necessary. This is almost never desirable for the simple, practical reason that it fatigues the player’s hands.

For example, in some of the earlier beta versions of Counter-Strike, it was possible for those in the know to “skate” by angling slightly sideways and pressing forward and sideways rapidly. When moving at a particular angle and switching between going sideways and forward, there was a decay phase—the attack took the maximum speed above the level of sustain (Figure 7.13).

This gave experienced players a huge advantage because they could move one and a half times more quickly. This was an exploit to be removed because it gave an overwhelming advantage to veteran players and enabled the player to move faster than intended.

The level of sustain can be thought of as a limit, such as the maximum speed of a car or character.

**Simulation**

So where do these envelopes come from? For any game, it’s relatively easy to track what parameter a particular input mapped to and how it modulates that parameter over time. But it is often difficult to discern exactly what sort of system gave rise to that modulation. Most often, envelopes are defined by relationships between variables in a simulation.

As a simple example, consider a cube that moves left and right. Both directions of movement have an ADSR envelope that looks like Figure 7.14.

**Playable Example**

To experience this, open example CH07-1. Move the cube left and right using the A and D keys. Enter new values by clicking on a parameter (such as “Max Speed”), typing in numbers and pressing enter to see how the envelope changes.
Currently, an acceleration value is added to the cube's velocity each frame, creating a smooth quarter-second attack phase. This speeds the cube up gradually, giving it a loose, organic feel. Mirroring this, a drag value is also applied in each frame, causing the cube to slow back to rest again when the button is released. Without this drag value, the cube would keep going indefinitely (switch "Drag" to zero to experience this). The Max Speed variable determines the level of sustain, the constant movement value the cube reaches after completing the attack phase.

This simple test demonstrates how simulations give rise to the different modulations of parameters, and how changes in that simulation modify the sensations of control. It gets a lot more complex than this (as we'll see in Chapters 12-17) but simulations like this are the building blocks for sensations of control. How the simulation is built determines the sensations of control possible. A particular tuning can change the feel of control drastically, but the construction of the simulation—which parameters are available to tune in the first place—determines what tunings are possible.

For example, consider the feel of the left-right movement in Ghosts and Goblins, Donkey Kong and the original Metroid. In all three of these games, horizontal movement has very little attack or release (Figure 7.15).

In the systems that create this kind of envelope, pushing the joystick or pressing the button directly overwrites the position of an avatar. Every frame in which the game detects the button as held, it adds some amount to the current position of the avatar in the appropriate direction and places the avatar in that new position. In the same way, the release value brings the player to a halt instantly when the button is released. The result is a system that feels stiff but responsive. It feels crisp and is good for dealing with challenges that require precise positioning and accurate jumping. This same kind of feel applies to other mechanics with little or no attack and release, such as the movement of a mouse cursor in response to mouse movement. Sometimes players categorize this feel as twitchy.

Compare this to the "thrust" mechanic in Asteroids. Pressing the thruster button has a long attack (Figure 7.16). In this case, it's because Asteroids keeps track of a separate velocity value for the ship. Instead of the position of the ship being set directly each frame, the ship keeps its own value for velocity and updates its own position based on that value. Pressing the thruster button adds to the velocity value in the direction the ship's currently facing. The result is that the ship speeds up gradually, a curved and gentle attack. It's a different kind of simulation and a different kind of feel.

The other way to define an envelope is by filtering input before it is plugged into the changes in the game system, as happens when rotating the Asteroids ship left and right. The envelope looks like Figure 7.17.

While the left button is held, the ship's orientation is changed by a certain amount each frame. There's a slight attack value, which is achieved by changing

---

**Figure 7.14** The cube has smooth, organic movement.

**Figure 7.15** A responsive, but stiff, feel.

**Figure 7.16** A loose, fluid feel.
The input slightly as it comes in. At the first millisecond the button is pressed, the amount by which the ship's orientation changes is less than it is a few milliseconds later. The value gets increased over time. In this way, the feel is responsive but slightly soft. The player can tap the button lightly to make small adjustments or hold it down to turn full speed. The rotation is just a filtering of the input signal over time. This is another way to modulate an envelope: just change the input signals as they come in.

State Changes

Another interesting, measurable feature of simulation is state changes. States are artificially constructed changes in circumstance that modify the meaning of incoming signals. In Super Mario Brothers, for example, there are ostensibly three controls: left, right, and jump. In Figure 7.18, we see different states that overlap and interact in different ways.

Mario has a "ground" state and an "air" state. As far as the simulation is concerned, Mario's potential for movement—his physical properties—change when he's on the ground or in the air. When in contact with the ground, the left and right buttons map to certain additive force. When Mario is not in contact with the ground, the strength of his left and right movement is greatly reduced, creating a different state. This is a very simple example of increasing sensitivity through state switching. Left and right movement means something different when Mario is in the air, meaning that one input is actually mapped to two separate actions that change depending on the state of the character. What's interesting is that this creates additional sensitivity in the system; there is greater expressivity when inputs are mapped to different responses across states which are altered and maintained by the simulation itself. You're getting two sets of responses mapped to one input, essentially.

This is used to great effect in the Tony Hawk games where there are as many as six separate states, each of which assigns a different value to each button on the controller. Every button means something different in each state. A relatively small number of inputs becomes an interface to a huge number of moves. The game has artificially created different physical states for the avatar to exist in. If these state changes are clear to the player, they can correspond to a huge number of possible responses. The same principle is applied to fighting games, where being in the ducking, blocking or jumping states changes the meaning of each input.

Examining the type and robustness of the simulation a game is running can yield useful fodder for comparison. Without getting too crazy deep into the mathematical intricacies of a given physics system, we can see that while Super Mario's vertical movement uses a simple simulation, the jumping in Metroid is a predetermined set of positions. This lends Metroid a crispier, more precise feel. It's also useful to catalog whether or not the avatars have different states they can exist in. If the avatars do have multiple states, how many states do they have and how do the different states cause different input signals to be interpreted by the simulation and responded to?

Filtering

Input signals come in from the input device in various forms, such as Booleans and changing float values. It is possible to map "raw" input either directly to a response or to a force or other modification of a simulation. This rarely happens, because to directly map raw signals is to forego the opportunity to tweak game feel. Most often, an input signal is received raw and passes through a layer of code where it
CHAPTER SEVEN • RESPONSE METRICS

is filtered in some way, into a different value range. Even if it’s just multiplied by 2 or 3 in order to create a greater force value to pass along to a simulation, most input signals are modified before being mapped to a response. Almost all input is transposed in this way; this is a large part of what “tuning” means as it is applied to mechanic design.

For example, the movement of a mouse-controlled cursor on any computer has a control-display ratio. The change in position of the mouse on the desk is mapped very closely to the movement of the cursor on the screen, but there is still a ratio between physical movement and virtual movement. If the cursor moves 2 inches on screen for every 1 inch the mouse slides across the desk, the control-display ratio is 1:2. In this case, the filtering of input is a simple multiplication. Inputs can also be divided, added to, multiplied by themselves and so on.

It is also possible to have complex, non-linear transformations applied to input signals as they come in. This is especially true when the signals represent a range, as with a thumbstick on a controller (which returns two float values, each in a range from -1.00 to 1.00). This is employed in the driving mechanics Grand Theft Auto 4 and other games featuring a driving metaphor. Instead of a constant car steering ratio (1 degree of steering wheel turn = 2 degrees of car turn) the amount of turning changes across the input space. The steering ratio increases the farther the thumbstick is pulled in a direction (Figure 7.19).

Pulling the stick about halfway to the right of center still yields a fairly small turn. This mitigates “twitchiness” that is present in some hardcore driving games like Vanishing Point or, to some extent, Grand Turismo. By making it much more difficult to oversteer, the mechanism is much more forgiving and creates a nice range between small avoidance adjustments and hairpin turns.

At this layer, there is the possibility not only for modifying input signals before passing them on, but for creating entirely new signals by further interpretation of the incoming signals. Think of the famous Konami code, which looks for a particular sequence of button presses over time. The game’s code examines the input

![Diagram](image)

**FIGURE 7.19 Turning changes as the thumbstick is pulled farther from center.**

signals it receives for specific, predetermined patterns and responds differently when it sees them. When a game is sensitive to patterns of inputs over time like this, the input space becomes larger. Moves in fighting games are fundamentally the same kind of interpretation. The Hadoken in Street Fighter II or “Dark Metamorphosis” in Castlevania: Symphony of the Night each require a certain sequence of inputs over time to trigger. Ditto the gestures used in many games controlled by the Wii Remote.

Fundamentally, what’s happening is that an additional process is running in the layer between input and response. The signal comes in and a piece of code checks to see if it recognizes the signal as the first part of a pattern. If it does, it moves ahead and waits to see if the next part of the pattern is going to follow. Usually this is time-based, enabling only a short window of time for the next input to occur before resetting the sequence. Essentially, it’s building additional sensitivity into the inputs coming through. After all, a sequence of inputs is not inherent in the signals coming from the inputs themselves. It’s not a response per se, just the game listening for additional patterns among the input signals it’s receiving. Once a pattern is identified, a special type of signal is generated by this interpretation layer and is passed along to the simulation, where the response is carried out. This response can be an animation, the unlocking of additional lives or the addition of a particular force into the game’s simulation. The same thing happens in most current Wii games; it just happens to be a much more complex and sophisticated pattern-seeking algorithm because it has to make mathematical sense of all the crazy data that flows when you move accelerometer and pointer data from a free-floating controller. Whether it’s a Hadoken or a Wii Remote sword slash, however, I would categorize any time a game listens for a pattern of inputs across time as a “gesture.”

Another way in which it’s possible to create additional sensitivity through interpretation is spatially, either across game space or input space. For example, while pressing the A-button may have one meaning and pressing the B-button may have a different meaning, pressing both simultaneously yields a third response. This is similar to a gesture, but instead of looking to correlate a sequence of inputs across time, it looks for combinations of input signals happening at the same time. In other words, it assigns a different meaning to a combination of inputs than it does to each of those inputs individually. This is commonly called “chording” and is used to great effect in games like Tony Hawk’s Underground, where every combination of a button and a direction maps to a different trick. Remarkably, chording is present even in early games such as Super Mario Brothers, where holding down the B-button modifies the meaning of pressing left or right (by adding more force).

The other type of spatial transposition happens across game space and is more commonly known as context sensitivity. For example, in Resident Evil 4, the position of the character in the game world can alter the meaning of a particular input. Standing by a window or a ladder changes the meaning of the A-button in a direct, one-for-one kind of way. It’s not necessary for context sensitivity to be this rigid, however, as proved by the game Strange Attractors. Strange Attractors has only one input, which activates a series of gravity wells placed around the level. The gravity from a well will affect the ship relative to its distance from that well (following
the inverse square law, I assume) meaning that Strange Attractors features a fluid, ever-changing sort of context sensitivity. The meaning of pressing the button is constantly changing as the ship moves around the game space, closer to some wells, farther from others.

Drawing a generalization from all of this, transposition is either spatial or time-based. Spatial transposition can mean turning a linear curve into an exponential one, or it can mean augmenting sensitivity by recognizing groups of input signals from different inputs as unique, and passing along corresponding (new) signals. Time-based transposition assigns different meanings to input signals across time, forming gestures which themselves create new and different signals. This offers us another way to compare one game to another: the types of transposition the input signals undergo and the resulting values that get passed along.

**Relationships**

Examining individual mappings and envelopes takes us most of the way to understanding how a game's feel is built. The final piece of the puzzle is the relationships between parameters in a system. This is where much of the tuning of game feel happens. For example, in the game Sonic the Hedgehog, there is a parameter for gravity. Gravity is the foil of jumping; the two work in concert to produce the feel of jumping in Sonic. In the same way, to create the feel of "carving" in a driving game requires friction. Without friction, the car's turning seems floaty, as if it's driving on ice. With sideways friction applied to the tires, they seem to carve and dig in, as a real car would. Individual mechanics—mappings of one input to one response—work in concert to produce an overall feeling of control.

The whole process looks something like Figure 7.20. As shown across the top, input enters the system when the player manipulates an input device. From the physical manipulations of various inputs, the input device generates and sends to the game corresponding signals. A raw input signal can be mapped directly to response, as with a mouse cursor, or it can feed directly into a simulation. Alternately, some kind of filtering happens, where the input signal coming in is altered in some way before being passed along to simulation and/or response. The simulation layer represents the game’s internal model of reality, the one which the player interacts with via input. Finally comes the game's actual response to the signals it received, whether transpose, raw or from a simulation.

To summarize, to measure response we want to know what inputs are hooked up to what parameters in the game. To do this, we want to know how many different things the player controls, how many avatars there are. We can then examine each avatar in the game relative to its type and dimensions of motion, the frame of reference for that motion, and whether the motion is direct or indirect. Knowing this, we can identify how each input stream modulates the parameter over time and can quantify this as an ADSR envelope. From this point, we can attempt to extrapolate the system that gave rise to this particular envelope. This could be a filtering...

**Figure 7.20 From input to response.**

of input signals directly, a change in a simulation or both. Ultimately, we want to understand what variables are being tweaked, the relationships between those variables and how they become the envelopes we've identified.

**Input and Response Sensitivity**

Out of the games discussed earlier, Donkey Kong is particularly interesting because it maps a relatively high-sensitivity input device (the joystick, which can return float values from -1.00 to 1.00 along its horizontal axis) to a very low-sensitivity response. Compare this to Super Mario Brothers, which has a very low-sensitivity input device but has a very sensitive response.

If we record the positions of these characters over time and include jumping as well, it's obvious just how much more expressive Mario's loose movement is (Figure 7.21).

From this comparison, it's apparent that Super Mario Brothers has a more expressive mechanic than Donkey Kong. The combination of input and response produces a fairly accurate picture of the overall "virtual sensitivity" of the system (Figure 7.22). This is a soft metric, of course, but useful for comparing the expressivity of two different games.
contemplative, less visceral feel. As in Pacman, all rotation and superfluous directions of movement have been stripped away for simplicity. The result, however, is not a very compelling virtual sensation when removed from its context.

Press "2" to experience low input sensitivity and high response sensitivity. This time, the cube moves organically, loosely and smoothly. The simulation is adding forces rather than overwriting position directly. This is a much better feel, no? The lines of motion are flowing, curved and organic.

Press "3" to experience high input sensitivity and low response sensitivity. With this combination, you have very high sensitivity with the input device, the mouse, but almost zero reaction from the game. The cube has become essentially a very large cursor. This is a natural mapping: the position of the mouse on the screen matches the position of the mouse sitting on the desk, so it's very easy to feel oriented and get a sense of mastery and control. It's boring, isn't it? Because the mapping is so internalized from years of computer use, there's nothing to learn, no motion translation to master. The motion is quick and snappy and leaves the cube with no feeling of mass, weight or presence.

Press "4" to experience high input sensitivity and high response sensitivity. There's a very interesting motion here, one that requires a bit of mastery. It feels nice to whip the block around again and again to hit the red dot and to experiment with trying to slow the block down again and reverse direction or to make little figure eight patterns. Even a game with high input sensitivity and low reaction sensitivity (a first-person shooter that ties mouse movement directly to looking around a 3D space, for example), smooths that snappy, jerky input with a little bit of reaction from the game.

This is a simple demonstration of some of the different ways input and response can be combined to create different sensations of control. This rough measurement can be applied to any game.

Summary

Our final metrics are as follows:

- **Hard**
  1. How many objects the player controls
  2. The dimensions, type and frame of reference for the movement of each avatar
  3. The ADSR envelope representing each modulation of a game parameter by an input over time

- **Soft**
  - The overall sensitivity of the system as a function of its input and response sensitivity

---

**Playable Example**

To experience this first hand, check out example CH07-2. There are four options for control that can be accessed by pressing keyboard keys 1-4.

To begin, press "1" and use the W, A, S and D keys to move the cube around. These controls have low input sensitivity and low response sensitivity. The input sensitivity is low because there are only four buttons, each of which only has two states, on or off. The reaction sensitivity is low because the game's reaction for each button has only two states, moving at full speed or not moving at all. This is not a very good virtual sensation, very stiff with very little fluidity or appeal. In some instances—the original Legend of Zelda, for example—this grid-like rigidity is desirable because it creates a more
Understanding the simulation and input filtering that give rise to particular sensations of control is craft knowledge. If you want to build real-time control that feels a certain way, it’s useful to know how simulations give rise to what sensations. For measuring the sensation of control across games, however, measuring the output envelope, is sufficient. For measurement, as for the player, the underlying simulation is mostly irrelevant. What is relevant is the output, the sensation of control and the overall sensitivity of the controls.