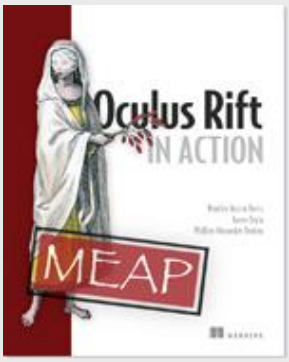

Oculus SDK: drift correction and prediction

By Bradley Austin Davis

In this article, I will talk about two bits of functionality in the Oculus SDK that you should be aware of: drift correction and prediction.

The SDK tracking system includes two additional bits of functionality that you should be aware of: drift correction and prediction.

Drift correction

The sensors in the Rift aren't infinitely precise. As messages are accumulated and the orientation updated, tiny errors will build up. We could be optimistic and hope that the average total of sensor errors would add up to zero and cancel out; but in practice, the various environmental factors that mislead the Rift are invariably asymmetric and errors grow instead of diminishing. Drift can occur on any of the three axes of rotation, so there needs to be to account for and correct it. The SDK does this in a few ways.

For drift on the X or Z axes (pitch and roll), the SDK can use a known vector to account for them. The known vector is gravity. Because the headset contains an accelerometer, the SDK can always tell which way is down, because it causes the accelerometer to constantly report an acceleration of ~9.8 meters per second squared in the opposite direction. Because the SDK knows which direction is up and which is down, the roll and pitch of the headset can always be found relative to those vectors.

Yaw drift is a little trickier. There’s no gravitational force acting on the headset that tells you which direction is forward, so you have to find something else that can provide your drift correction.

For the DK1 Rift, the magnetometer built into the headset was used, but only if it had been calibrated. This tended to be problematic since the Earth’s magnetic field might not actually be dominant when you’re surrounded by a bunch of computer equipment. In fact, even the DK1’s own screen produces a non-negligible magnetic field, which sometimes made performing the calibration difficult.

For the DK2 Rift, yaw correction is accomplished via the tracking camera. Since the camera can determine the full orientation of the Rift along with its position, this made the more temperamental magnetic correction superfluous, to the extent that the recent versions of the Oculus configuration software no longer support magnetic calibration for the DK2.

Prediction

One of the greatest impediments to presence is latency. If the view doesn't change in response to movement fast enough, then at best the feeling of immersion is lost, and at worst you can make yourself ill. A commonly cited upper limit for acceptable latency in VR applications is 60ms, but many people can still perceive latency as latency at that level. The lower limit appears to be around 20ms, but it's subjective. So one of the goals of development is to make sure that the rendered view at any given moment represents the orientation of the Rift at that moment, within (ideally) about 20ms.

The difficulty lies in the delays imposed by the various steps in the rendering process. Fetching sensor information, rendering the scene and distorting the image all take time. Double buffering and frame delays by the graphics driver may cost more time. On top of this, there are the physical limitations inherent in the actual display panel.

The end result (figure 1) is that if you're turning your head, the image appears to lag behind where you're looking. The faster you turn your head, the further behind the image is.

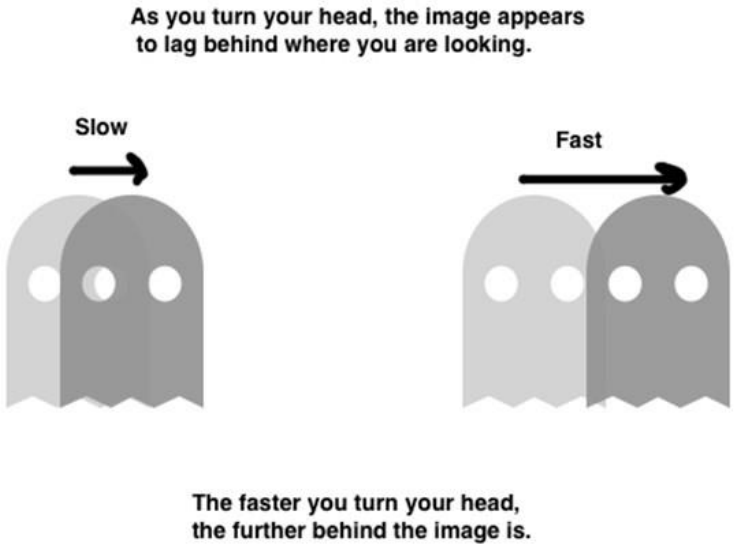

Figure 1: The view doesn't change in response to movement fast enough

50 degrees per second is a moderate rate of turn for the human head. At that rate, every 20 ms of latency causes 1 degree of error. A more rapid, but not uncommon rate of rotation is 250 degrees per second. Suddenly as we're moving our heads everything we see is about 5 degrees away from where it should be. It may not seem like much, but your brain is very accustomed to seeing things where it expects to see them when you move your head, and defying that expectation can cause distress and motion sickness.

In order to correct for these issues, the SDK allows you to use prediction. Specifically, instead of asking for the pose of the headset *now*, you can ask for the pose of the headset where it *will be* when the image you're rendering actually appears on the screen. The `ovrHmd_GetTrackingState()` in particular will let you query for the pose at any point in time between the present and 0.1 seconds in the future.

Prediction isn't a panacea, since clearly our heads tend to stop, start, change direction and change speed all the time. However it still provides better starting point for rendering than simply using whatever the current position is.

Using drift correction and prediction

The good news is that you don't have to exert any effort in order to take advantage of drift correction or prediction. Early versions of the SDK left it up to the developer to enable them and apply them correctly, but as the software has evolved it's been recognized that if their implementation is sufficiently bulletproof, there's no real reason to ever *not* use them.

Drift correction ends up being automatic and invisible. Prediction also ends up being automatic, to the extent that you don't even need to do any work calculating how far ahead you should be predicting, assuming you're using the `ovrHmd_GetEyePoses` function for interacting with the tracker. Chapter 5 of my book, Oculus Rift in Action, really ties the rendering and head tracking functionality together. You can use the code 15dzamia at manning.com to save 39% on the book.