Developing *Racing Thrill* with Gesture and Voice

By John Tyrrell

**Introduction**

Inspired by the gesture-control capabilities of the Intel® RealSense™ SDK, PlayBuff Studios developed the latest iteration of *Death Drive: Racing Thrill* for devices equipped with Intel® RealSense™ technology. The team at PlayBuff observed that few driving games offered the player the ability to drive and fire weapons simultaneously, and most also relied on traditional control inputs. They set out to create a game that would deliver a more innovative and visceral experience through the implementation of gesture and voice controls combined with the ability to fire weapons while driving.

Tarun Kumar, PlayBuff CEO, and his team were convinced that using gesture controls while driving would be a lot of fun and would significantly improve the overall game experience. Working with the Intel RealSense SDK for Microsoft Windows®, PlayBuff integrated gesture and voice controls into the heart of the game experience. The result is a game that delivers a uniquely intuitive combination of action and driving.

Through the development process, the team grew to understand the different approach and thought processes required for the successful implementation of gesture and voice controls in a game compared to using traditional inputs. They explored numerous voice and gesture control options, learned a great deal about performance optimization with the Intel RealSense SDK, and developed a vision of the impact the technology will have on the future of computing.
Decisions and Challenges

Intel® RealSense™ SDK Data

PlayBuff discovered that the Intel RealSense SDK gives developers a vast quantity of information that can be used in many different ways. They spent time early in the project evaluating interface options and determining which data was necessary and relevant to the development objectives. The team chose to implement three features of the SDK in Death Drive: Racing Thrill: gesture recognition, hand tracking, and voice recognition with command control mode.

Gesture and Hand Tracking

The team decided to deliver a completely touchless experience that makes full use of the Intel RealSense SDK gesture and hand-tracking capabilities. On an Intel RealSense technology-equipped device, all the gameplay sections of the game can be completed from start to finish using only gesture control if the player so chooses. Hand tracking is used to map the points of the hand and differentiate between an open and a closed fist, and gesture is used to identify a range of hand movements, including lateral movements and rapid shaking.

The following code sample shows how the team implemented the gesture tracking in-game.
// Get a hand instance here (or inside the AcquireFrame/ReleaseFrame loop) for querying features
// pp is PXCMSenseManager instance
PXCMHandModule hand = pp.QueryHand();
if (hand!=null) {
    // Get Hand Tracking Processed Data
    PXCMHandData hand_data = hand.CreateOutput();
    hand_data.Update();

    PXCMHandData.IHand mainHand;
    // retrieve the hand data
    pxcmsStatus

    sts=hand_data.QueryHandData(PXCMHandData.AccessOrderType.ACCESS_ORDER_NEAR_TO_FAR,0,out mainHand);
    if (sts>= pxcmsStatus.PXCM_STATUS_NO_ERROR)
     {
        if(startTimer<4)
            startTimer += Time.realtimeSinceStartup-startRTime;

        Openness = mainHand.QueryOpenness();
        handDetected = true;

        // openness is zero when hand detected in starting, so use startTimer for some delay
        if(Openness < 20 && startTimer>3)
            fist = true;
        else
            fist = false;
    
    PXCMHandData.JointData jointdata;
// get world coordinates of detected hand.

pxcmStatus xy1 = mainHand.QueryTrackedJoint(PXCMHandData.JointType.JOINT_CENTER, out jointdata);
PXCMPoint3DF32 xy2 = jointdata.positionWorld;
PointOfCamera = new Vector3(-xy2.x, xy2.y, xy2.z);
// Normalise vector and make screen centre point to origin of world coordinates
    PointOfCamera = new Vector3(Mathf.Clamp(PointOfCamera.x/0.20f, -1, 1), Mathf.Clamp(-
PointOfCamera.y/0.12f, -1, 1), 0);
    PointOfCamera = new Vector3(PointOfCamera.x/2 +0.5f, PointOfCamera.y/2 +0.5f, 0);
    } // clean up
    hand_data.Dispose();
else
    startTimer = 0;
// Swipe code using hand tracking
// _velocities is vector3 Queue for storing coordinates of hand tracking
    timer += Time.realtimeSinceStartup - startRTime;
original = new Vector3(-xy2.x, xy2.y, xy2.z);
if(!FirstTime && timer>2)
    {
    StartingPoint =original ;
    FirstTime = true;
    swipeVector = Vector2.zero;
    _velocities.Clear();
    }
else if(timer>3)
    {
if(_velocities.Count>2)// check for three frames, we need at least three frames to get directions
    {
    _velocities.Dequeue();
    _velocities.Enqueue(original - StartingPoint);
    StartingPoint =original ;
    Vector3 total = Vector3.zero;
    varenuma = _velocities.GetEnumerator();
    while(enuma.MoveNext())
    {
    total += enuma.Current;
    }
Gesture-Based Steering

Implementing the steering controls using gesture was a critical part of the development process. To control the steering wheel of the car, the game recognizes the closed fist gestures of the left and right hand to steer in the respective direction (see Figure 1). When the fist is subsequently opened, the steering wheel automatically returns to the central neutral position. To brake or move the car in the reverse direction, the player makes fists with both hands.

```c
//check direction vector coordinates for different gestures.
//MOVEMENT_THRESHOLD is minimum value for swipe detection.

if(total.x > MOVEMENT_THRESHOLD) // for swipe right
{
    FirstTime = false;
timer = 0;
}
else if(total.x < -MOVEMENT_THRESHOLD) // swipe left
{
    FirstTime = false;
timer = 0;
}
else if(total.y > MOVEMENT_THRESHOLD) // swipe down
{
    FirstTime = false;
timer = 0;
}
else if(total.y < -MOVEMENT_THRESHOLD) // swipe up
{
    FirstTime = false;
timer = 0;
}
else if(total.z > MOVEMENT_THRESHOLD) // move hand away from camera
{
    FirstTime = false;
timer = 0;
}
else if(total.z < -MOVEMENT_THRESHOLD) // move hand towards camera
{
    FirstTime = false;
timer = 0;
}
else
{
    _velocities.Enqueue (original - StartingPoint);
    StartingPoint = original;
}
```
PlayBuff went through a number of iterations before arriving at a satisfactory implementation of the gesture controls for steering. Initially, the team employed a simulation of the actual steering wheel drawn on the screen, which the player controlled using both hands. There were two practical problems with this implementation. First, it required having both hands in front of the screen at all times. With this kind of “realistic” simulation of the steering wheel, the player’s view of the screen was impaired—a significant hindrance that the team didn’t foresee in the design phase.

The second problem was that in order to control the car properly, the steering wheel had to automatically come into the neutral position. The initial implementation put the user in control of returning the wheel to the neutral position, but through testing, the team found that this was non-intuitive and resulted in poor steering control and a frustrated player.

Making the steering position automatically return to neutral required the team to rethink how the gesture controls were used for steering. After extensive experimentation, they arrived at the solution of using a single closed fist gesture of the left or right hand to steer, and once released, the wheel automatically returns to a central position.

In addition to steering, gesture tracking is used to control other in-game actions. A number of different weapons can be mounted on the car, and these are fired with a handshake gesture. In the
normal gameplay mode, there is no action to shift gears, but PlayBuff also incorporated a Drag Racing mode in which players can shift gears using closed-fist or swipe-out gestures.

*Gesture Findings*

PlayBuff found that driving the game with gesture controls is an altogether different experience for the player. Because gesture control is a relatively new technology and its implementation can vary significantly between applications, there is a learning curve associated with controlling the vehicle in-game.

The game features an interactive tutorial that helps players familiarize themselves with the various gestures needed to play the game. To avoid overwhelming the player with too much information, the on-screen instructions are delivered when needed according to specific in-game situations (see Figure 2).

![Figure 2: The gesture controls are explained to the player during gameplay.](image)

PlayBuff has found that once the player becomes used to the controls, the feedback regarding the experience is positive. Figure 3 illustrates the flow of tutorial instructions in the game.
Another challenge with gesture controls is the need to calibrate the camera to ensure that the game responds accurately. PlayBuff found that one-time calibration at the start of a play session was not always effective because users tended to move their hands while playing out of the camera’s frustum—the cone area in front of the camera where hand movements can be detected. To counteract this problem, continuous detection of the position of the hands was implemented. If the game cannot detect an appropriate input, it pauses and prompts the player to bring the hands back into position, allowing for recalibration.

As a result of the testing process, the team also discovered that the gesture controls were frequently causing players to over-steer the car on turns, which was spoiling the overall user experience. As a workaround, PlayBuff implemented steering assist where the game adjusts the speed of the wheels’ rotation according to the desired steering angle, which is calculated as the angle between the orientation of the car and the wave points on the track.
For example, damping comes into play if the player’s steering angle is in the wrong direction or over-steer reaches a certain threshold. The team used the results of player testing to calculate the optimum degree of damping to apply. To avoid the sense that the car is steering on its own at any point, they also ensured that steering assist would only come into play in response to an input from the user.

Testing showed that the time taken to complete levels with gesture is largely comparable to a traditional interface, hence there was no noticeable negative impact for players who are more comfortable playing racing games using traditional inputs.

**Voice Recognition**

At the start of the project, Kumar had some reservations about the voice-recognition capabilities of the Intel RealSense SDK, but he soon discovered that the performance in command control mode was good. When calculating the time for the player to execute each menu input, he found that not only was the SDK’s command recognition acceptable, but it also took less than 2 seconds with voice commands versus 3 to 4 seconds for gesture. This clearly demonstrated the efficiency of using voice, making it significantly faster for the player to traverse the menu and start the actual game, which improved the user experience by saving as much as 2 to 3 seconds for each input.

The primary use of voice recognition is in the game’s menu. For example, the player can select the mission by simply saying, “mission 3” or “mission 4.” Voice can also be used to select the car model and to change the car’s color (see Figure 4) by saying the color choice, such as “yellow” or “black.” If the player needs to select a specific level or task at a particular stage of the game, other voice commands such as “task one” or “task two” can be used. Voice commands were also implemented in the actual gameplay for firing the weapons as an alternative to the handshake gesture control.

![Figure 4: Players use voice commands to choose vehicle customization options.](image-url)
Performance

The most challenging aspect of development was the CPU-intensive nature of the physics involved in a racing game. Simulating the real physics of a car takes a lot of CPU, and most of the development time was devoted to packaging the car physics with the gesture control and optimizing CPU usage. Visuals are important in any game, and initially Death Drive: Racing Thrill had considerably more graphical detail, such as particle effects for storms and fog. While the game still retains its real-time shadows, once the team implemented the Intel RealSense SDK beta to integrate gesture controls alongside the car physics, they found that the CPU performance was lagging below the target frame rate of 30 frames per second (FPS) with the beta version of the SDK. This observation was made based on the target minimum spec of an Intel® Core™ i5-4250 processor @ 1.3 GHz with 4 GB RAM and Intel® HD Graphics 5000.

Since implementation of the Gold version of the Intel RealSense SDK, however, the team has found a noticeable increase in performance, allowing the game to capture both hands in a single frame and still meet the target frame rate. This increase has ultimately improved the game’s responsiveness and the overall user experience.

Input Choices

PlayBuff strongly believes that players should have a choice in how they play the game when choosing input modalities. Although the player will never be obliged to use traditional mouse and keyboard inputs or touch controls to complete the game, they can do so if desired. For example, to fire weapons, the player can use the F keyboard input in addition to the handshake and voice commands.

Testing

Testing was instrumental in the development process since the start of the project. In the early stages, and to facilitate the in-house testing process, PlayBuff and the Intel QA team discussed establishing test cases that were specific to the Intel RealSense SDK. Subsequent user testing at PlayBuff helped to greatly improve the overall user experience, and throughout the development cycle the team worked with a test group comprising players with varied skills in racing and action games.

The game’s menu uses hand tracking for selecting different options, for which there are different ways of collecting the coordinates. Early in development, the team used the MassCentreImage parameter to get the coordinates of the hand on the screen, but after testing different options they discovered that the JointCenter parameter provided more accurate and stable results.

Other major learnings that were direct results from testing included the problem of habitual over-steer by players, leading to the implementation of steering assist, and the fact that navigation through the menu is much faster with voice commands. Testing also helped with designing the flow of the in-game tutorial. The tutorial is need-based, with instructions appearing when the specific
related action is required in-game. The team carried out comprehensive testing to calculate average response time for tasks to accurately calculate the appropriate timing for various instructions.

Conclusion

Kumar believes that designing and implementing an application with Intel RealSense technology compared to using traditional inputs requires a different way of thinking. In his experience, a key aspect to taking the right approach is making time to review the detailed design guidelines that Intel provides with the Intel RealSense SDK at the ideation stage of the project.

Kumar advises developers to carefully consider the specific strengths of the Intel RealSense SDK and potential use scenarios where a gesture-control interface can add real value to the experience. Whether the application is gaming, edutainment, or something more serious such as healthcare, Kumar believes the real strength of Intel RealSense technology is in the creation of new user experiences rather than in simply replacing the traditional inputs of existing applications with gesture. With the right approach to the technology, an open mind, and some blue-sky thinking, Kumar is sure developers can find success.

For Kumar and the team at PlayBuff, the most rewarding part of the project was the opportunity to learn about the technology itself. At the start, they had only theoretical experience of touchless interfaces, but now they are much better positioned to appreciate the challenges and the advantages associated with Intel RealSense technology. Their experience also allows them to approach the design of their future games from the perspective of gesture controls, voice recognition, and the resulting user experience.

Potential Applications of Intel® RealSense™ Technology

Through his experiences in working with the Intel RealSense SDK, Kumar developed a clear vision of the technology’s great future potential. He foresees a number of use cases where Intel RealSense technology could bring a real and unique value to users. For example, in the healthcare sector gesture-control interfaces would allow medical practitioners and their patients to interact with a greatly reduced risk of spreading infection when making rounds and updating case progress in a hospital.

Intel RealSense technology also has numerous potential applications in education. In Kumar’s native India, he identifies the challenge of English pronunciation as one that Intel RealSense technology could help with. An app equipped with the technology’s voice-recognition capability could analyze the accuracy of a speaker’s pronunciation and help users improve their English-speaking skills.

In the casual gaming sector—for example, an exercise app for home use—Intel RealSense technology could be used to give feedback on exercise form and whether the user performed it correctly with the full range of movement, correct intensity, and speed of motion. For Kumar, a key factor in making this kind of application a reality is increasing the Intel® RealSense™ 3D camera’s frustum from its current
1.3 meters to a distance that can accommodate the entire body and function accurately when the subject is farther from the camera. Following such an increase in the future, other game applications could include tennis or golf, where the camera accurately measures the power, direction, and quality of the swing. Additionally, success in the casual gaming sector depends on establishing a broad user base with the technology available in everyday laptops and portable devices.

Another functionality that Kumar thinks can help grow the footprint of Intel RealSense technology in the marketplace is facial recognition—still a fledgling technology but showing great potential. Kumar doesn’t foresee gesture, voice, and facial recognition replacing traditional interfaces entirely, but rather views them as complementary with great value for specific use cases. Overall, Kumar sees exciting potential for Intel RealSense technology in a broad range of case-specific applications, bringing real benefits to users and great opportunities for developers.

About the Developer

Based in Chandigarh, India, PlayBuff Studios was established in 2010 with the goal of developing world-class games. The company started with 2D games targeting primarily the Nokia Store and ranked in the top three of companies from India during its first year. To date, PlayBuff has clocked more than 65 million downloads on the Nokia Store, placing it among the top 25 developers in terms of downloads.

In parallel, the team started working on games in three genres where it plans to focus its efforts over the coming two to three years: racing, multiplayer third-person shooting, and mobile social games. The studio’s third-person shooting game, Battle It Out*, was recently soft launched on the Apple App Store, and its multiplayer social game, Bands of Woods*, is in an advanced stage of development.

PlayBuff has so far released two titles based on its Racing Thrill IP. Death Drive: Racing Thrill was first released in April 2014 as an app for iOS* devices, and subsequently selected by Microsoft for porting to Windows* 8 and Windows Phone* products.

Death Drive: Racing Thrill was reconfigured to feature intuitive gesture controls and was named one of the phase I winners of the Intel® Perceptual Computing Challenge 2013. Since that initial win, Kumar and his team have spent five months working with the Intel RealSense SDK to add a full complement of gesture and voice controls in the run up to the public release of the Intel RealSense SDK-enabled version.

PlayBuff’s idea of a gesture-enabled Electronic Health Record (EHR) solution was selected as one of the top ideas in the Ideation Phase of the Intel® RealSense™ App Challenge 2014, a global contest that received hundreds of submissions from around the world. PlayBuff has entered the Development Phase of the challenge and is now working on its prototype of a gesture- and voice-command enabled EHR solution that will allow doctors to access patient records without the need to touch anything, thereby reducing the risk of spreading infection through contact.
Death Drive: Racing Thrill will launch soon for Windows 8 devices with the integrated Intel RealSense 3D camera. The Intel RealSense SDK is available for download [here](#).

For more information about PlayBuff Studios and its games, including *Death Drive: Racing Thrill*, visit their site [here](#).

**Helpful Resources**

*Documentation*

Kumar and his team found the Intel RealSense SDK documentation helpful, and they were able to quickly integrate the SDK without any major hurdles. Kumar strongly recommends that all developers devote time to thoroughly reviewing the design guidelines at the ideation stage of development, believing that this will significantly benefit the overall development process and the insightful application of Intel RealSense SDK functionality.

The Intel RealSense SDK documentation is available for download [here](#).

*Intel Support*

Kumar was positive about the support he and the PlayBuff team received from Intel throughout the Intel RealSense SDK integration process, citing particularly the support team’s responsiveness and transparency. Support requests for both hardware and technical issues, such as problems with storing calibrated data and mirroring, were always responded to within one or two days.

*Intel RealSense Technology*

**Intel Developer Zone for RealSense**

**Intel RealSense SDK**

**Intel® RealSense™ Developer Kit**

**Intel RealSense Technology Tutorials**

Intel, the Intel logo, and Intel Core are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2015. Intel Corporation. All rights reserved.