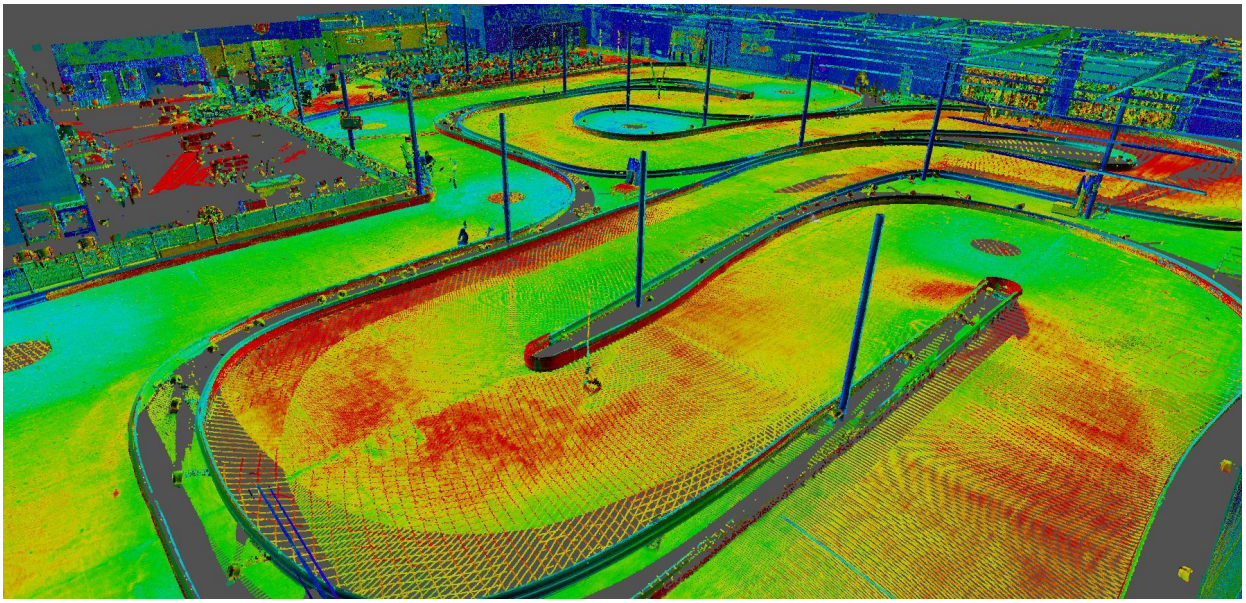

Race Real Go-Karts in Virtual Reality with the Intel® NUC

By Adam Amaral, founder/CEO/Creative Technologist at Master of Shapes

Master of Shapes is a design-driven interactive studio, made up of creatives, gamers, directors and technicians. We partnered with Intel to create a unique, game-changing virtual reality (VR) experience, using a prototype Intel® NUC powered by an AMD* Radeon* RX Vega GPU and an Intel® Core™ i7 processor.

In a partnership with another client, we tested the capabilities of our technology by creating a first-of-its-kind virtual race course on which two professional drivers raced on adjacent tracks, but were unified in the virtual world. It made for an exhilarating experience, as the drivers had to dodge flames, boulders, cliffs, falls—and avoid crashing into their opponent—to make it to the finish. Then the new Intel NUC inspired our team to create something even bigger.

We used a large PC-server to power our previous project. For this one, we wanted to make our technology more efficient, to scale the project across smaller and easier-to-drive vehicles. Many of us at Master of Shapes grew up playing *Mario Kart** and were keen to create a grown-up version of our favorite childhood video game. Equipped with the small-form Intel NUC, and a powerful GPU, we created *VR Karts*, in which the user races against the clock in a virtual world—while in a real kart.

Figure 1. Driving in VR on the K1 Speed indoor go-karting track.

On the Right Track

We reached out to K1 Speed, the indoor-karting specialists, to ask if we could purchase one of their high-performance electric go-karts. Instead, they invited us to build and test our creation at their 40,000 square foot facility in Torrance, California.

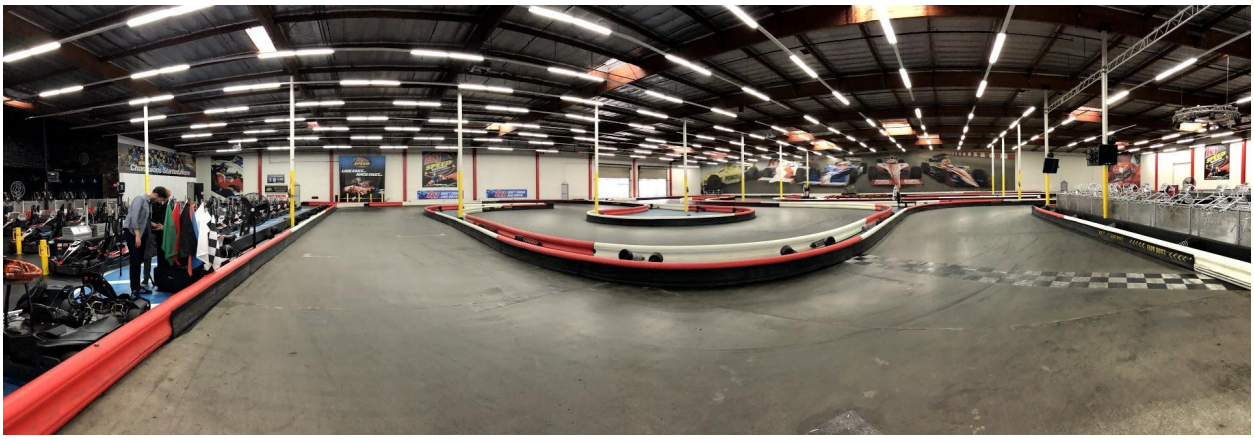

Figure 2. The track we replicated using lidar.

An accurate representation of a real-world track is vital for driving in VR. To achieve this, we used lidar scanning. This is a remote-sensing technology, similar to radar, that uses pulsing light waves from a laser to collect measurements. These can be used to create 3D models and maps of objects and environments. We used a Faro* Focus3D X 130 lidar scanner.

At the K1 Speed venue, we scanned the 190 x 160-feet track from eight vantage points to ensure complete coverage. Initially this took the form of a point-cloud representation of the space.

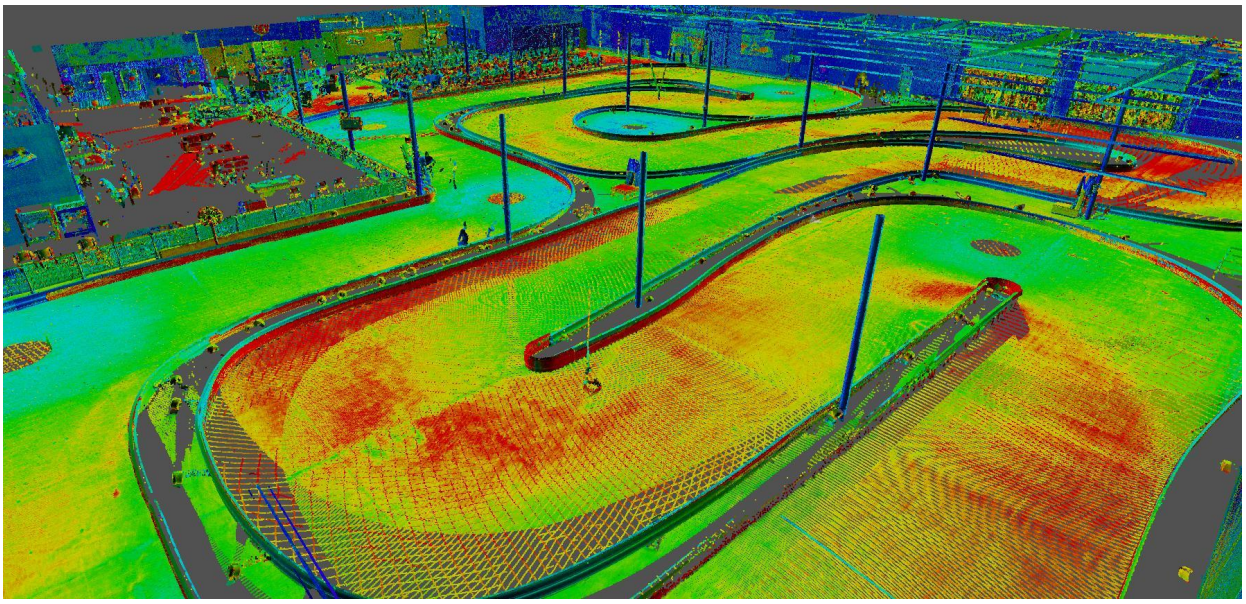

Figure 3. A set of data points known as a point cloud represents the track.

We used Faro Scene to export the point cloud into Unreal* Engine.

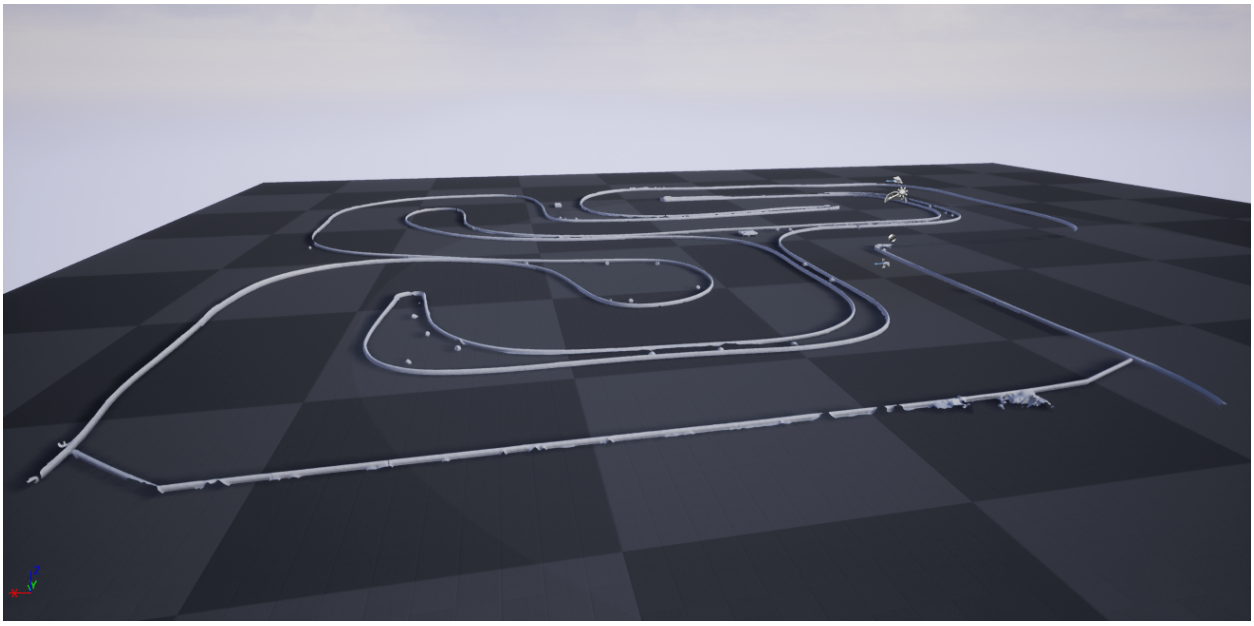

Figure 4. Meshed lidar scan in Unreal Engine.

Motion Capture

Once we had an accurate scan of the racing environment, we needed to map the location of the kart in real-time, to translate its location into the virtual world.

We used an infrared (IR) motion-capture solution from the Toronto-based BlackTrax. Cameras were carefully placed and calibrated around the track. Information from the cameras would be streamed from the BlackTrax server to the computer aboard each kart, rendering the virtual environment for the racer.

The next step was to attach IR beacons to the kart. BlackTrax beacons are long cords with a sensor at the tip. Our goal was to make them highly visible to the cameras. (For seamless coverage, the beacons mounted on the kart needed to be seen by two cameras at any given time.) Using a computer-aided design (CAD) model and 3D-printed custom mounts, we rigged them high on the kart. This provided unobstructed views and made them rigid, to reduce vibration and error.

The cameras and beacons enabled us to upload the position of the kart into Unreal Engine. Positional data was sent to the headset instantaneously, so the movement of the kart in VR mirrored real life. We tracked the kart's motion at 100 frames-per-second (fps) for the most accurate representation of the real track's curves and dimensions.

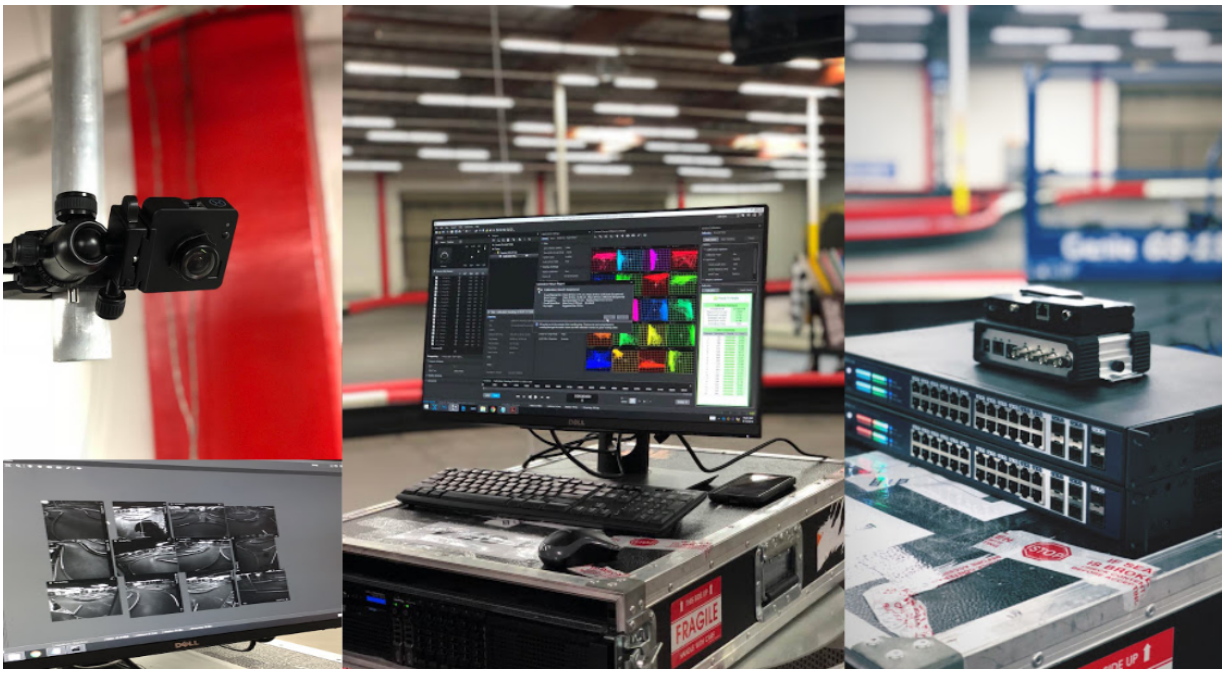

Figure 5. The BlackTrax motion-capture system.

Figure 6. BlackTrax transponder (top) and beacon (bottom).

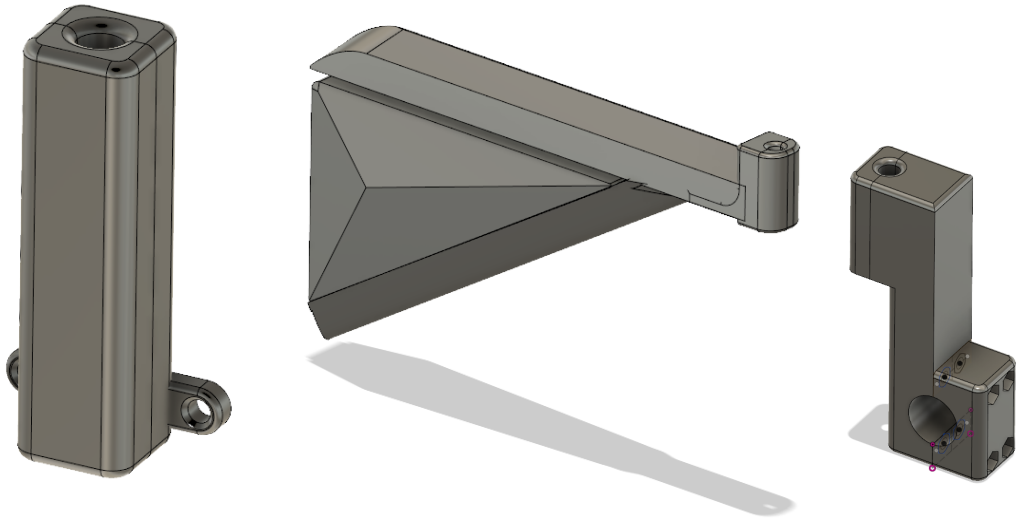

Figure 7. Custom mounts for the BlackTrax beacons. These are attached to the frame of the kart, and to the electronics assembly module behind the driver.

Equip the Kart with the Intel NUC

We fashioned a wooden enclosure and metal brackets to mount the Intel NUC to the back of the kart. We also created a custom sensor solution to detect the kart's steering angle and velocity. The sensors fed into a microcontroller that processed and sent this data to Unreal Engine for gameplay and onscreen display. We designed and 3D-printed a bracket and pulley system to track the steering column's rotation.

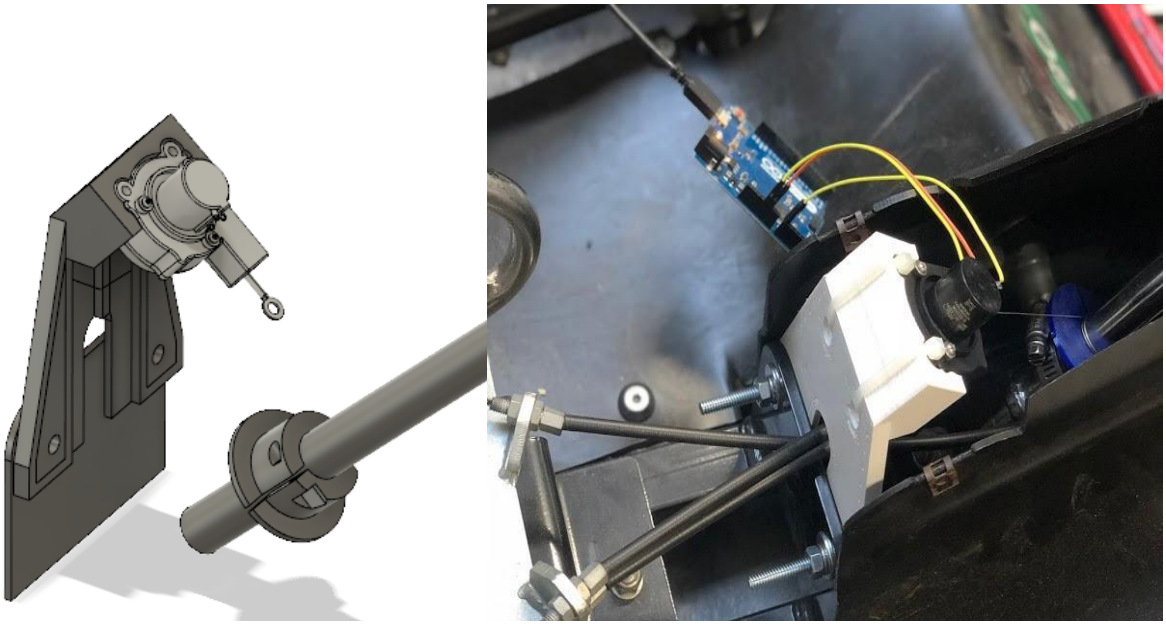

Figure 8. The bracket-and-pulley system used to detect steering angle and velocity, in its CAD (left) and finished (right) form.

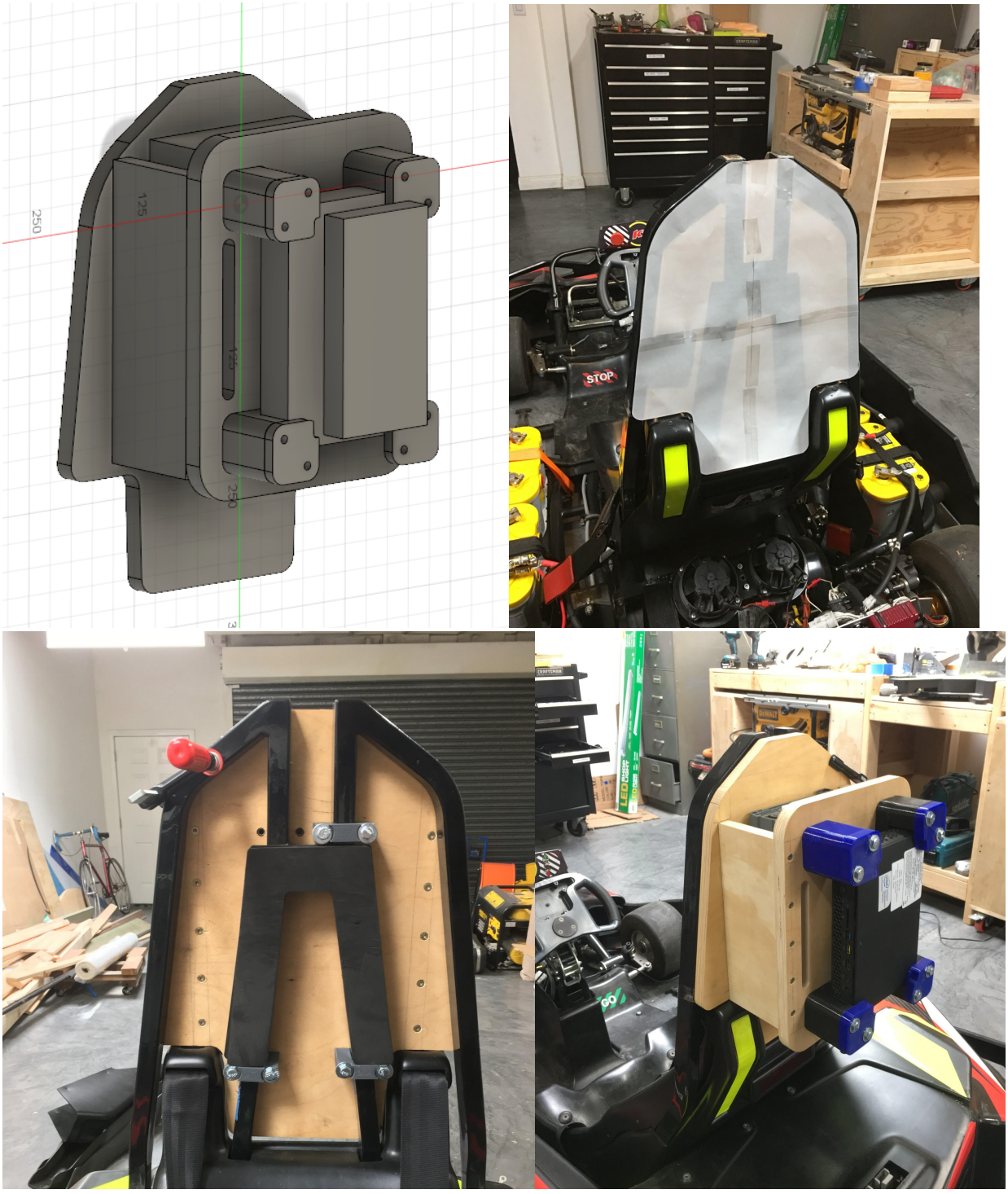

Figure 9. From drawing board to prototype. The enclosure for the Intel NUC in CAD form (top left). Mapping onto the back of the go-kart seat (top right). The frame onto which the enclosure will be mounted (bottom left). The Intel NUC, enclosed, and held in place by 3D-printed shock mounts (bottom right).

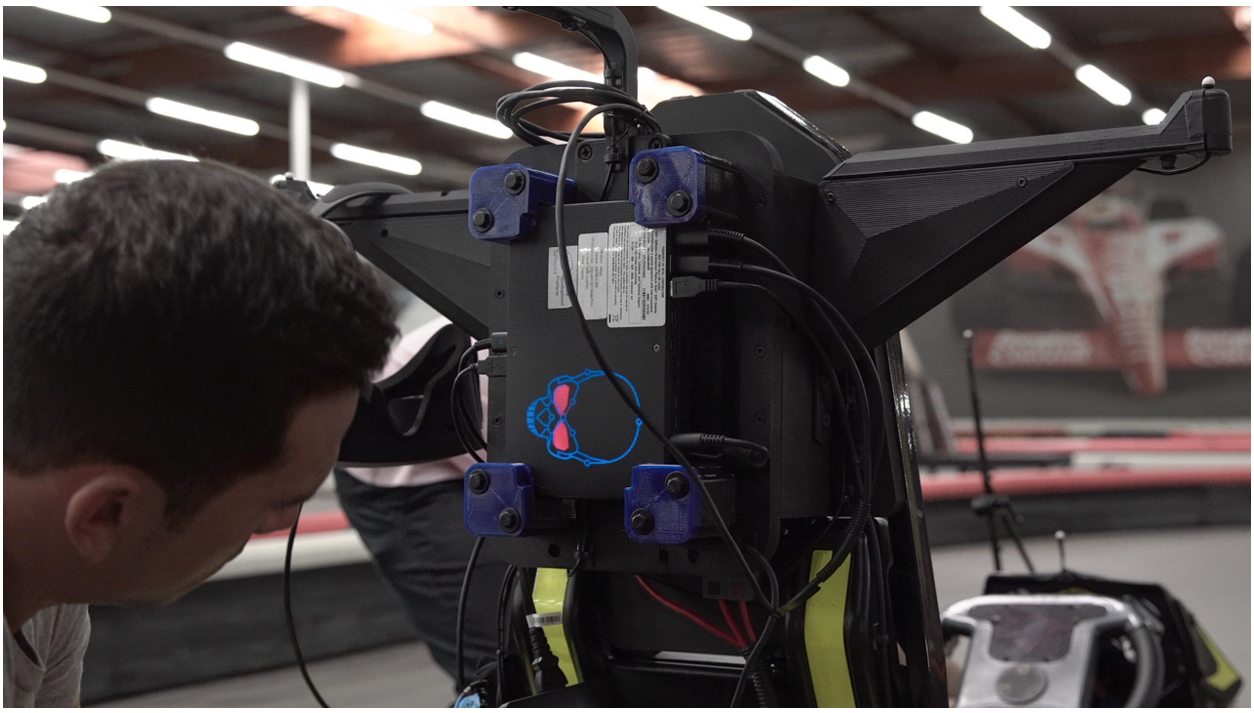

Figure 10. The Intel NUC mounted on the kart. Its wooden enclosure has been painted black.

Power the Platforms

We needed to power an Oculus* Rift VR headset, a custom Arduino* sensor suite, and the Intel NUC from an electric go-kart running at 48 volts (V). We divided the 48V into sections—to power each component—and equipped the Intel NUC with an in-line fuse to protect it from spikes in current.

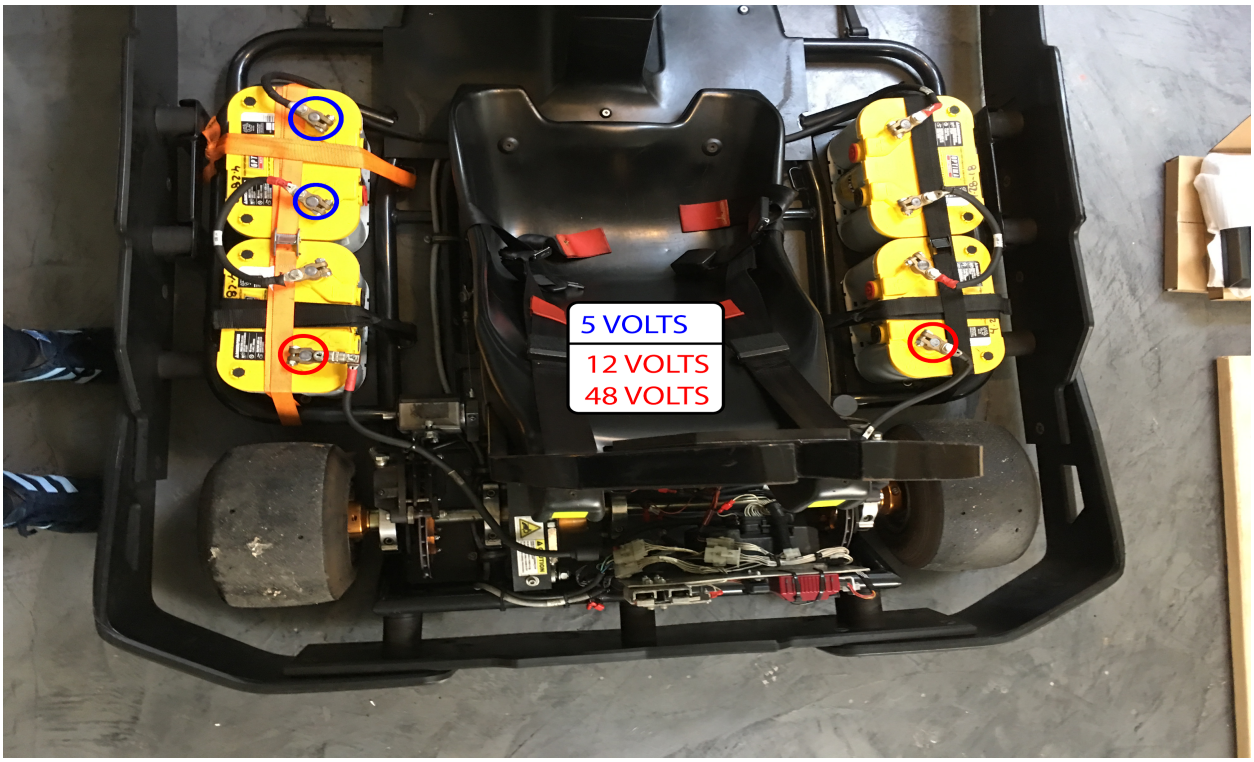

Figure 11:.48-volt system.

Create the Game

With the hardware established, our next step was to create the game itself. Inspired by *Blade Runner*, we used a 1980s synth soundtrack, scattered skyscrapers, flaming barriers, lasers and wreckages, all along a track that wound through a futuristic take on Tokyo. To transport the user, we used bold red lighting, with blue street lights guiding a path through the darkness, also punctuated by illuminated windows in the high-rises. Particle effects, fire, and smoke added to the atmosphere.

Figure 12. In-game footage of optimized buildings and particle effects.

We used the Unreal Engine plugin Houdini Engine* (a 3D animation and special effects application) to create neon signs. Houdini Engine converts typed text into 3D ultraviolet lightmaps. Typing in various Japanese words to populate signs gave us a quick variation and saved having to go back and forth between Houdini Engine and Unreal Engine.

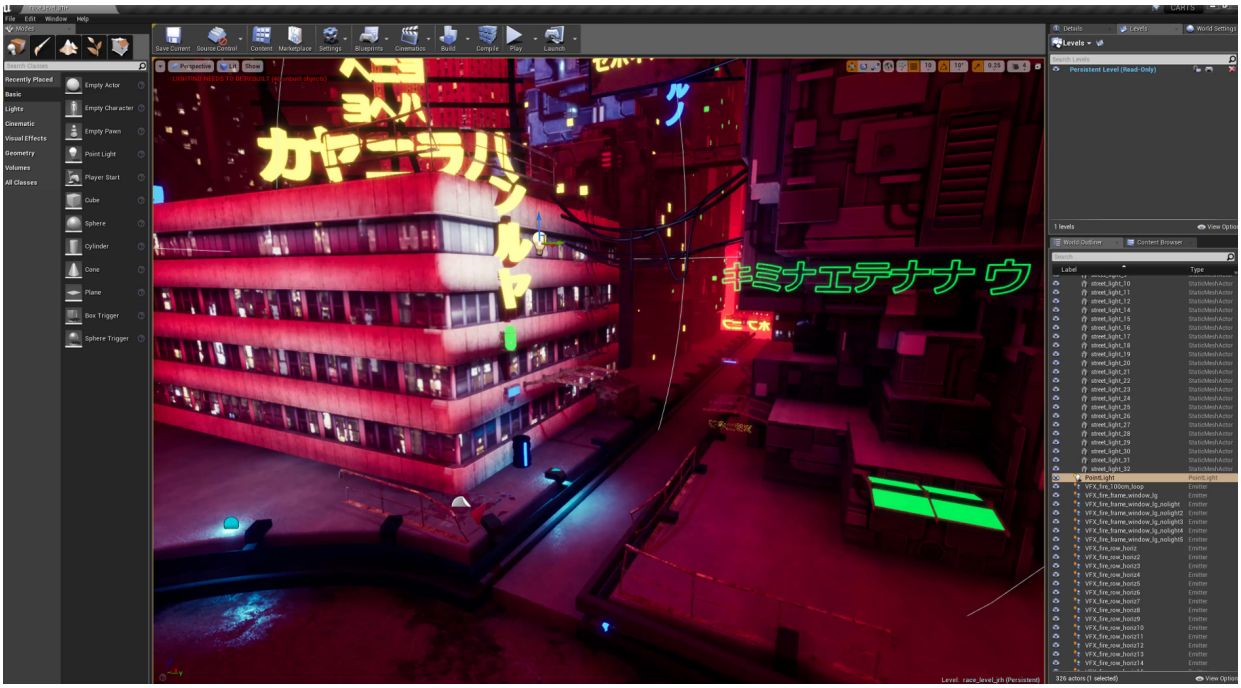

Figure 13. Houdini Engine sign-creator tool inside Unreal Engine.

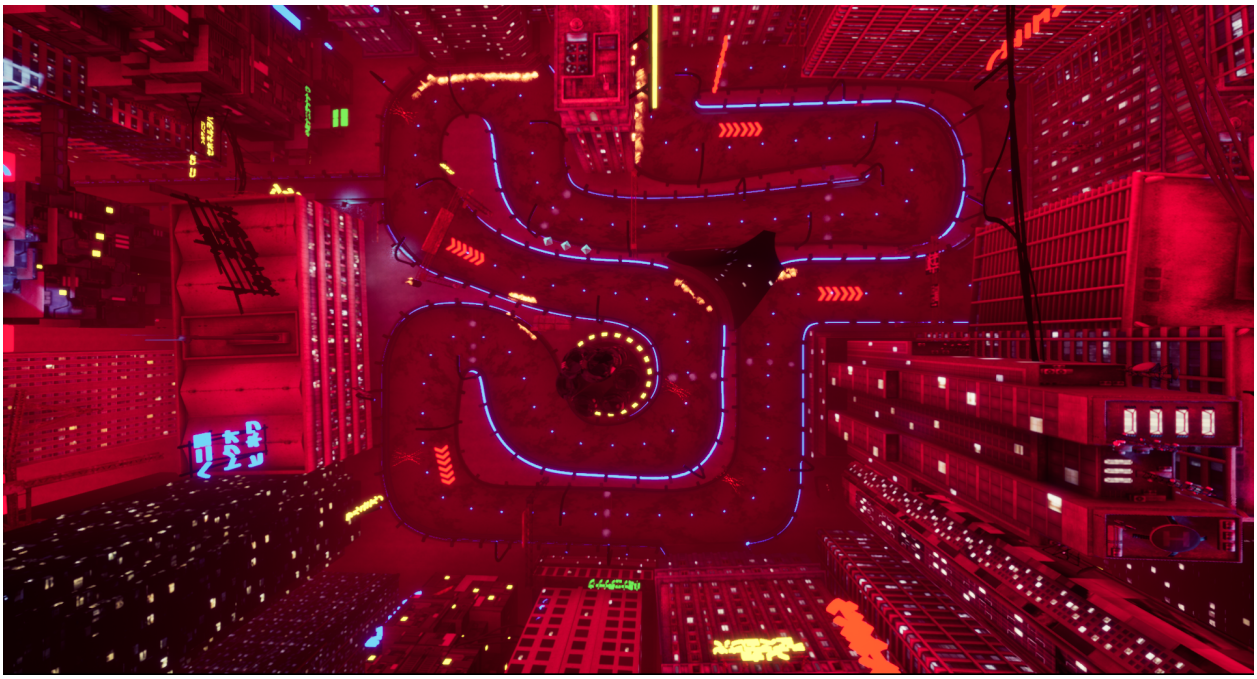

Figure 14. Aerial view showing the real and digital track merging.

Optimizing the polygon count was a priority during the build. VR makes demands on the most powerful of PCs, and maintaining 90 fps is crucial, especially for a driving experience.

The built-in Unreal Engine CPU status console commands meant we could easily debug and optimize. The multicore processor ensured we were never CPU-bound, and the AMD GPU performed well.

We needed the application programming interface (API) Vulkan* to prevent artifacts (visual anomalies) and glitches in VR. Vulkan also provides a performance gain of up to 40% in Unreal Engine, so it was a great tool to use. Learn more about optimization with Vulkan with this slide presentation.

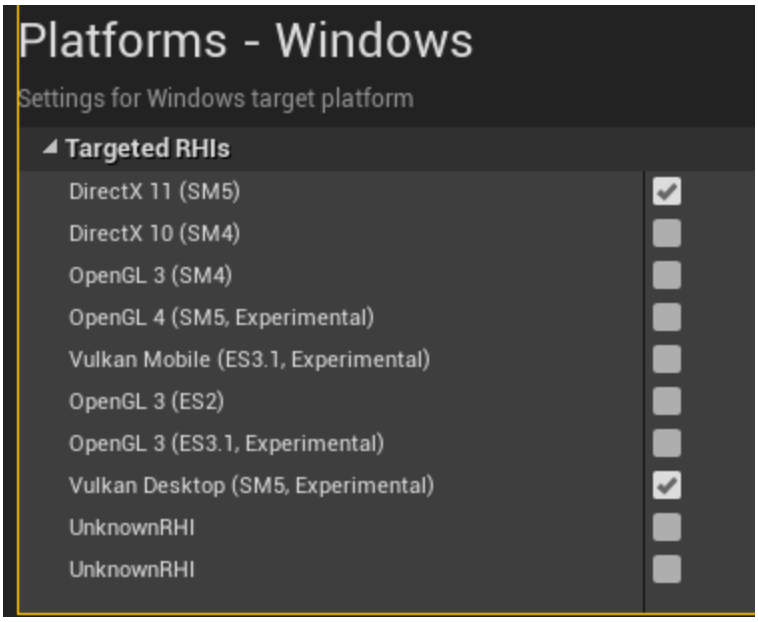

Figure 15. Project settings window, with Vulkan Desktop checked.

We wanted to pay tribute to *Mario Kart* by including digital power-ups in the game and a key feature of the karts enabled this. All K1 karts are equipped with alternative controls for manual take-over of the kart functions—in case a driver is out of control for some reason, or breaking speedway rules. Specifically, each kart has an independent remote that communicates with a built-in speed control, and braking system, on the kart. We reverse engineered the alternative control system by decoding and capturing the control signals from the remote to rebroadcast over our own custom transmitters that communicated with the built-in system. Through this process we were able to provide players with real-time reactions to the power-ups, instantaneously speeding up the kart or causing a momentary loss of power, just as one would experience when they hit a boost, or got damaged, or slowed down by an obstacle or opponent in *Mario Kart*.

Real-time features such as these were the most innovative and important aspects of *VR Karts*. When you turned the steering wheel in the real world, you saw it turning in the virtual world. We wanted everything to sync. This was important for us, as our brand is centered around making memorable and immersive experiences.

Figure 16. *Mario Kart*-style power-ups in the game interface.

Keep on Track

After designing the environment and creating assets, we began programming the logic that handled tracking and sensor data from the kart. The main server broadcast all the kart's possible positions, and communicated with the optical tracking solution system. The main server has the power to start games and stop karts.

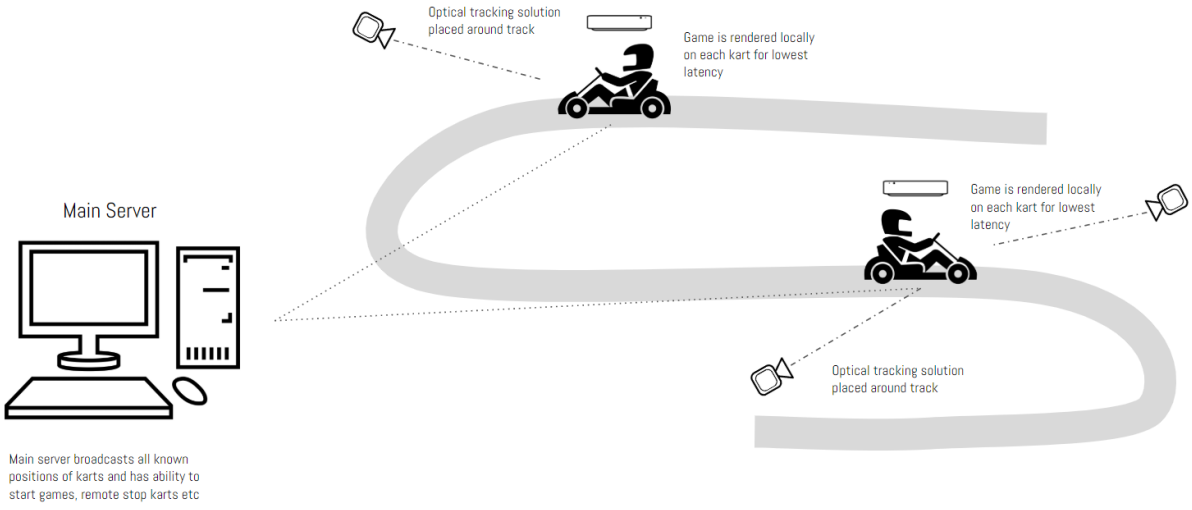

Figure 17. The *VR Karts* server-client structure.

To set up the server-client structure illustrated above, we created a multiplayer game in Unreal Engine.

(If you have never created a multiplayer game, the Blueprint Multiplayer Shootout Game tutorial is highly recommended. A good understanding of blueprint replication, and how it works inside Unreal Engine 4, is also useful.)

Our first steps included sending position data to the player's headset, so they knew where they were on the track. We also had to ensure that the driver and the server started at the same time.

We created and hosted a session on our server computer:

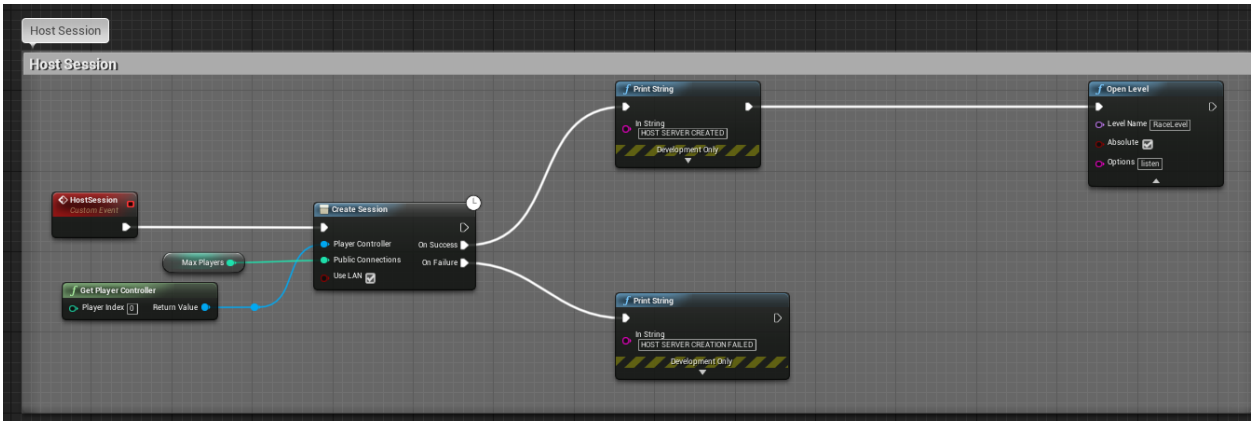

Figure 18. Hosting session on the server.

Our clients join like this:

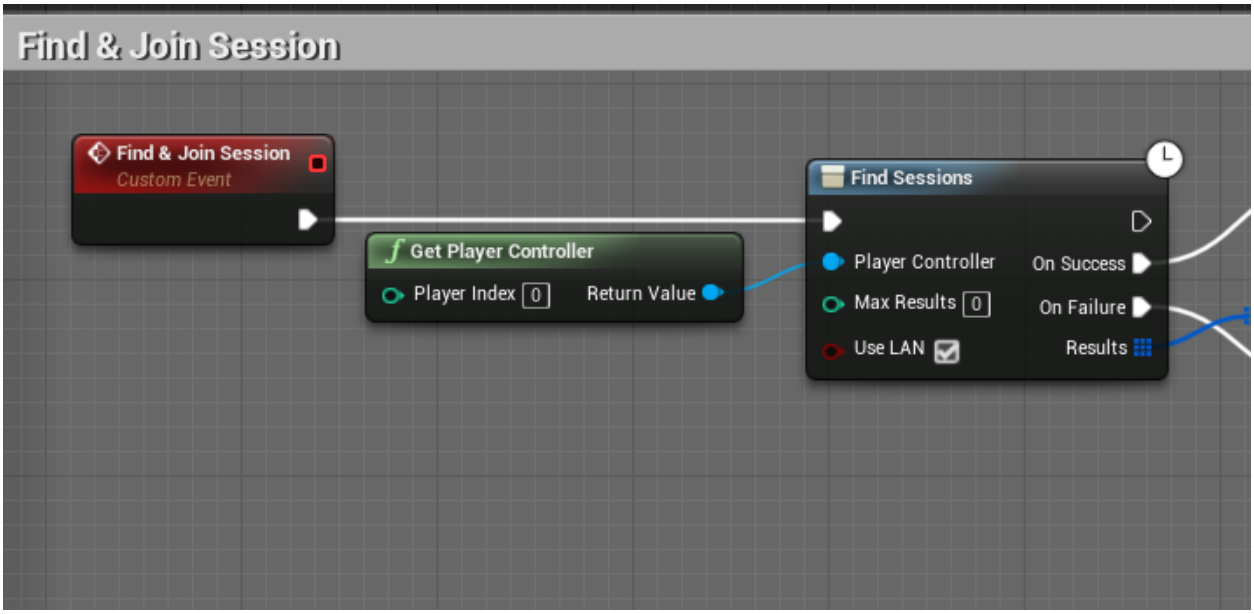

Figure 19. Joining a session on the client.

The admin (server) hosts the game. The player is assigned vrPawn and ID numbers when they spawn in the server.

We set up a race timer that replicates from server to clients. The timer begins when the admin and other players are in the game. Each race can be monitored from a central location that can control the kart and gameplay, as well as monitor and maintain the system.

The start call happens in the GameStateBase. Time updates happen in GameModeBase.

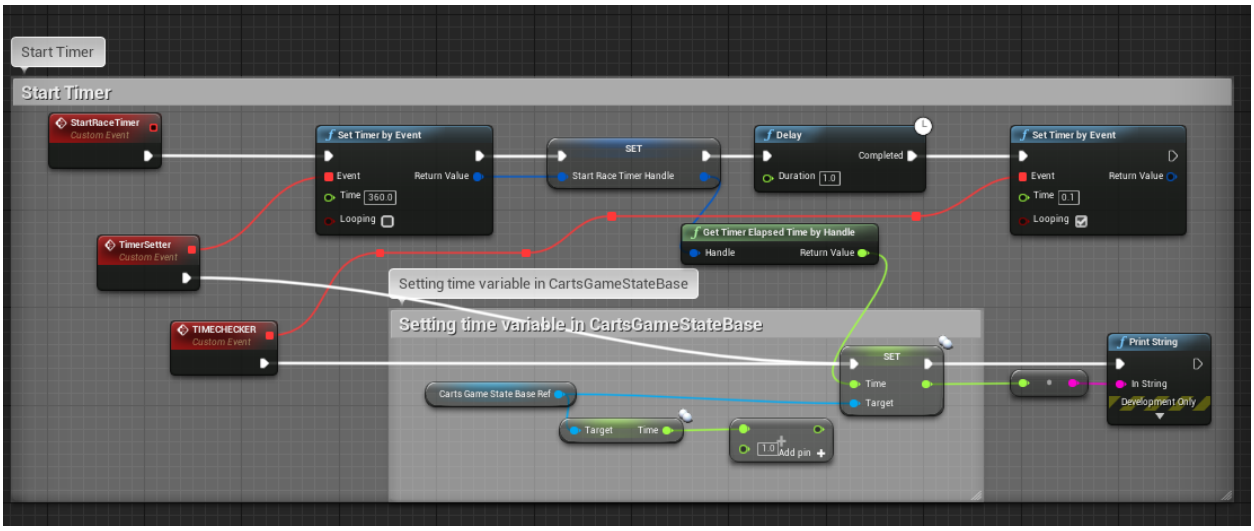

Figure 20. Timer synced across server and client.

We have an override in the player controller that saves the pawn if a player accidentally disconnects from the server.

To accomplish this, we made a C++ player controller.

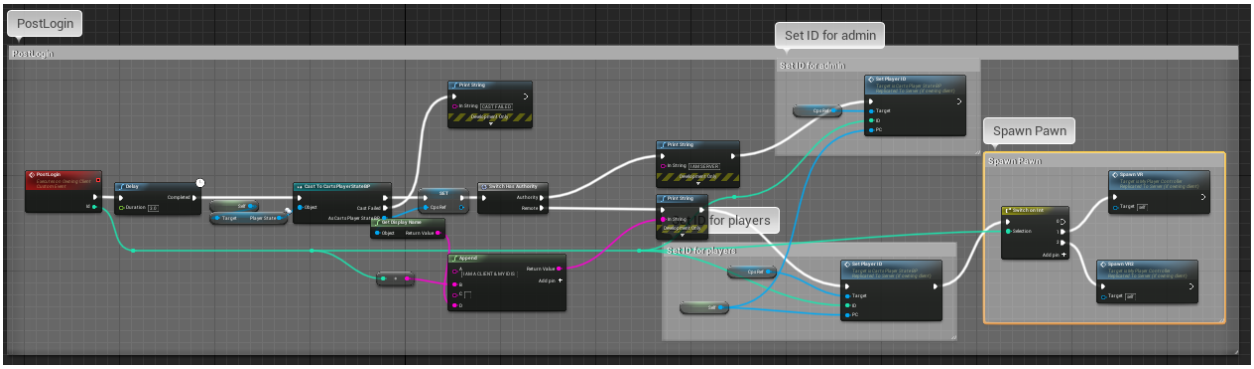

Figure 21. Post-login setting player ID.

```
#include "PlayerControllerCpp.h"
#include "Runtime/Engine/Classes/GameFramework/Pawn.h"

void APlayerControllerCpp::PawnLeavingGame_Implementation()
{
    if (GetPawn()!= NULL && DestroyActorThenUnpossess)
    {
        GetPawn()->Destroy();
        SetPawn(NULL);
    }
}
```

Figure 22. Overriding the player controller class so a pawn is saved if a player accidentally disconnects from the server.

Kart Sensor Data

Sensors attached to the kart were read by Arduino, which is an open-source prototyping platform that enables users to create interactive electronic objects. The Arduino commands were translated by Python* and sent via user datagram protocol (UDP) to Unreal Engine. We used this socket plugin.

We initialized the UDP connection on the Intel NUC on event “Begin Play”.

The messages are parsed into values that drive data in Unreal Engine. We attached a potentiometer—an instrument for measuring electromotive forces—to the steering wheel. When the user turns the real wheel, we mirror that in the virtual world, adding to the immersion of the experience, and making the player feel in control of both the digital and physical kart. The server processes all the kart’s locations along the track, using information from the tracking cameras. The server also communicates with the kart about in-game activity in real time.

The Challenges

One of our main challenges was the time frame. Usually, a project of this size and complexity would take up to four months. We had only half that time. This made us become more creative in the way that we programmed the virtual world.

Normally, when building an environment in a game engine, you place items manually along the user’s path. We did not have time to build the world piece by piece. Instead, we developed a spine system along the track using Houdini Engine. We could quickly populate the system with buildings, walls, flames, and so on. And we could easily change the interface to create an entirely different track by populating the spine with different 3D meshes, rather than coding each

piece of the environment separately. This was a completely new method for us that has become a tool for the future.

The project solved one of the main problems with VR experiences. With stationary VR, users are prone to motion sickness. If you see yourself moving in VR, but don't feel yourself moving in real life, it's easy to get sick. Now, as users accelerate and drive, they see and feel themselves moving.

Another success is that we created an experience that can differ every time. The user can choose different go-kart styles or even drive through a custom world of their choosing. Currently, the project supports one player racing against the clock, but it's fully scalable to multiplayer, and we are excited about future implementations.

The final experience can be seen in this video.

Figure 23. VR on the move. A player enjoying *VR Karts*.

Conclusion

Intel technology continues to enable leading-edge developers as they dream big, and execute that vision. The Intel NUC at the heart of this project is a prime example of how faster and more powerful processors can lead to innovations in unexpected fields. It's a powerful 4x4-inch mini PC with entertainment, gaming, and productivity features, including a customizable board that is ready to accept the memory, storage, and operating systems that you want. Whether adapting fully configured, ready-to-use Mini PCs to kits, or building around boards in true do-it-yourself mode, users are pushing boundaries and exploring new ideas every day. To learn more about

the Intel NUC and how it can help you with your next challenge, check out the specs, videos, and related documentation here.

Author Bio

Adam Amaral is the founder, CEO and Creative Technologist of Master of Shapes. He strives to be pushed out of his comfort zone and, consequently, has developed a wide array of expertise while creating award-winning work. If Adam is not taming buffaloes or playing with robots, he keeps busy designing tours for Miley Cyrus, or producing award-winning commercial content and interactive installations. Most recently, he has been creating innovative VR/AR experiences and working with new technologies—all while attempting to get to outer space.