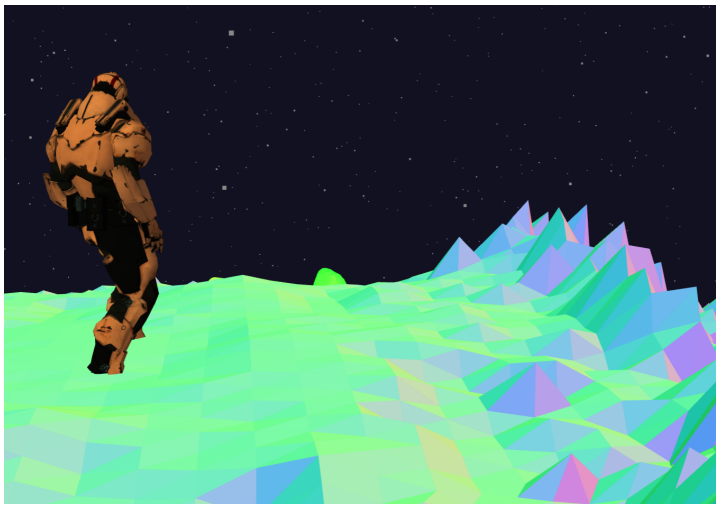
WORLDSCAPE: A Three.js Game

Louis Pang and Stephenie Chen
COS426 Final Project, Fall 2023



The avatar idle on the generated landscape

Abstract

This write-up describes WORLDSCAPE, a Three.js game that we created for our COS426 final project. Inspired by games such as Super Mario Galaxy but wanting to incorporate more of an exploratory element, we created this project. Our game is an open-world game with spherical mechanics, where the player traverses our built world using WASD controls. Our main crux was figuring out the mechanics of player movement on the sphere (i.e., how to make it seem like the character was navigating the sphere, as well as how to create a complementary camera view), which we successfully accomplished. We also created imagery using a combination of Blender and texture mapping, which we incorporated into the scene to add more of a dream-like element. Overall, we have created a strong basis for a spherical world game, writing functions and mechanics that enable the character to traverse the world, and for the developer to easily populate the world with props such as flowers, trees, etc, which hopefully could be used in the future for more interactive elements.

1. Goal

Our goal was to create an open-world game on a spherical world with a controllable character that could traverse said world. We were set on incorporating a spherical world and spherical physics — one of our team members grew up playing Super Mario Galaxy, and creating a spherical world seemed to be a unique endeavor

with sufficient difficulty and of substantial enough ambition to be suitable for the COS426 Final Project.

Our initial plan described a minimum viable product (MVP), a target product, and a reach product.

Our MVP was to create a spherical world that was traversal by the player using keypresses, and for the camera to be suitably positioned. We also wanted there to be some basic decorations, such as trees and flowers, and a night sky. We wanted to create a simple puzzle of sorts as well (e.g. making a tree in the center of the world grow), but eventually decided that it was unfeasible within the constraints of our time and ability.

Our target product was to embellish the spherical world with more advanced decorations, to incorporate running, and to handle collision detection with the world.

Our reach product was to add in mathematically complex embellishment for the world, such as procedurally generated landscapes, a sun (to simulate sunset and sunrise on the world), interactions with objects in the world (such as the trees and flowers) and an animated setting (like clouds, meteors orbiting the world).

2. Previous Work

Images from Super Mario Galaxy gameplay (Source 1, Source 2)

Our game mechanics were heavily inspired by Super Mario Galaxy, a 2007 platform game developed by Nintendo for the Wii.[1] The gameplay consists of Mario circumnavigating various spherical objects / planetoids, with the goal of accumulating stars, killing enemies, and completing other missions. Each planetoid has its own gravity mechanics, allowing the player to jump, walk, and run around the object, in a variety of different orientations. For example, Mario is able to walk seemingly (to the viewer) upside down, but that is because the gravity is holding Mario's feet up to the

[1] https://en.wikipedia.org/wiki/Super_Mario_Galaxy

bottom side of the planet. The player can also jump between different planetoids, if one planetoid's gravity overcomes the one he is currently on.

The storyline of the game is very well-developed, but it was the fun spherical mechanics that drew us in as players. We found a webgame built in WebGL that foregoes the Super Mario Galaxy storyline for the basic mechanics of movement around a sphere and gravity.[2] We took some inspiration from this when building our character movement.

These two games are extremely fun to play due to the whimsical ways the player can orient themselves, so we wanted to take these game mechanics and apply them to something like an open world exploration or puzzle game.

3. Approach

We initially wanted to create a puzzle or action game, but implementing the spherical mechanics were significantly more challenging than we'd initially thought, and so we decided to change our MVP to accommodate that, creating an open-world traversal game instead. Our approach then became more of a study of Super Mario Galaxy and other spherical games, as well as a project in building an aesthetically engaging spherical world.

4. Methodology

4.1 Character

4.1.1 Character Design

To create our character, Amy, we decided to follow the instructor's advice and used a premade texture and animations from Mixamo. Although this was the easier path, we still had to apply our animations to the custom armature, doing this using Blender.

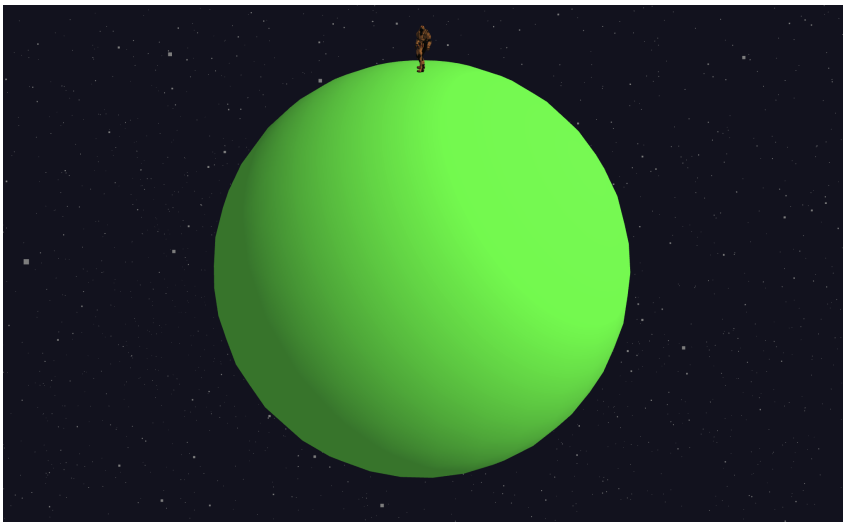
4.1.2 Character Movement

In order to simulate movement of the character, we explored a number of options. In Super Mario Galaxy, Mario has gravitational attraction to the nearest large object. This gravity is applied in the direction opposite to the normal of the closest plane. So if the surface he is walking on is round, such as a sphere, then Mario can walk around the entire circumference of the sphere without floating into space. Alternatively, if Mario is walking on a cube, then the gravity is applied evenly across the plane. For our implementation, we planned on mimicking this behavior by tracking Amy's closest surface, and applying acceleration to Amy in the negative normal of the direction of

[2] https://github.com/hlorenzi/galaxy

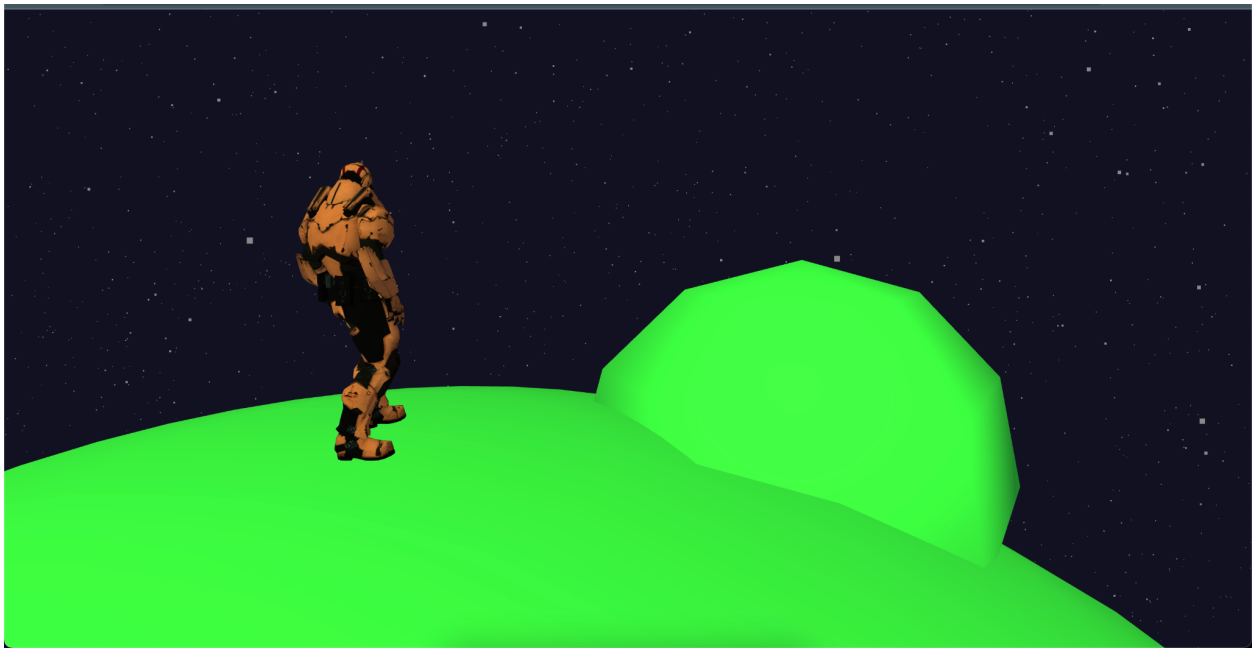
the closest surface. We had a class which tracked Amy's last contact with a plane and detected when the closest surface changed, which provided animations and changes in orientation as they approached the new surface.

However, a number of factors (outdated web-gl version, THREE.js and web-gl incompatibility) meant that the original boilerplate we had planned on using had to be rewritten line-by-line and so extending it with our own features was not possible in the given time constraints.



Amy on the sphere

Instead, we implemented a simpler physics system, which rooted Amy to a singular surface – a sphere – and provided a more cinematic experience than the one we originally planned on creating. In our new movement system, the character remains at the center of the world, and the world moves around them. Vertical changes are applied to account for terrain, but even when climbing or stepping animations are applied, our character continues to look in the direction perpendicular to the ground. This is because the main movement mechanic in our game is the movement of the sphere underneath the character themselves. As we detect input from the user in a given direction, we apply a slight angular acceleration to the sphere in the opposite direction. At each step, we rotate the sphere around the axis which is defined by the cross of the character's looking direction and their up direction. Doing this makes the character appear as though they are walking around the surface of the sphere, without leaving the ground.



Amy's perspective on the sphere (we added the bump temporarily to track how the sphere was rotating)

Computationally, this is significantly quicker than incrementally moving the character model along the sphere, calculating the normal of the closest section of the sphere below, and finally setting the character's up direction to that normal. Given our camera setup, which is tied to Amy's shoulder, although Amy themself isn't moving, there is no visual difference to the user.

In order to animate Amy, we created a finite state machine to track her current state, and allow us to transition between walk, run, and idle. In theory this was quite simple. Similar to Assignment 5, we had an event handler which tracked the keys being pressed. But in practice, animating the character smoothly was quite difficult and required tuning. For example, notice that if you break out into a run (hold *Shift*) and then slow down, there is no animation clipping. In order to accomplish this, as we fade from the run action to the walk action, the walking animation is resumed at the same duration as the run animation is ended.

4.2 Camera

Once we'd settled on the movement of the sphere instead of the player character, we decided that the camera had to change as well. Before, we had the player move relative to the camera's perspective, but we decided that with the rotation of the world, this would become too confusing. So instead, we locked the camera to a fixed displacement from the character. The camera was placed over the right shoulder and looking into the distance, at a point directly in front of the character. In addition to hiding our neat trick of moving the world under the character, this also made the game

much more immersive. The fixed third-person camera provides a much better sense of scale and place than the Orbiting camera.

We accomplished this consistent rotating displacement by passing our scene camera into a custom class, which took in an initial offset and initial looking point. These were transformed by applying the quaternion of the character model (which was somehow inverted, so we inverted it again), and the offset of the actual player coordinates (which remained the same). This allowed us to keep the camera over the player's shoulder. By applying a suggested dampening factor, which we found online, we were able to achieve smoother camera motion and make the third-person perspective feel more natural.

4.3 Environment

4.3.1 Night Sky — Parallax Stars

An example of the background — some stars are closer to the camera than others.

To create the background, we begin with a dark blue color and then instantiated an array of random coordinates, which had to be a fixed distance away from the origin – the rough location of our sphere and player. We added all of these coordinates to a buffer point geometry and set their color and material to a very bright yellow. When viewed from the player's perspective, they appear to look like stars. However, we placed them close enough to the character that when the camera pans, there is a parallax effect between stars, creating an illusion of smaller scale, which is on-theme with our traversable planet.

A better example of the parallax is included in our Blooper video in the demos folder of the repository.

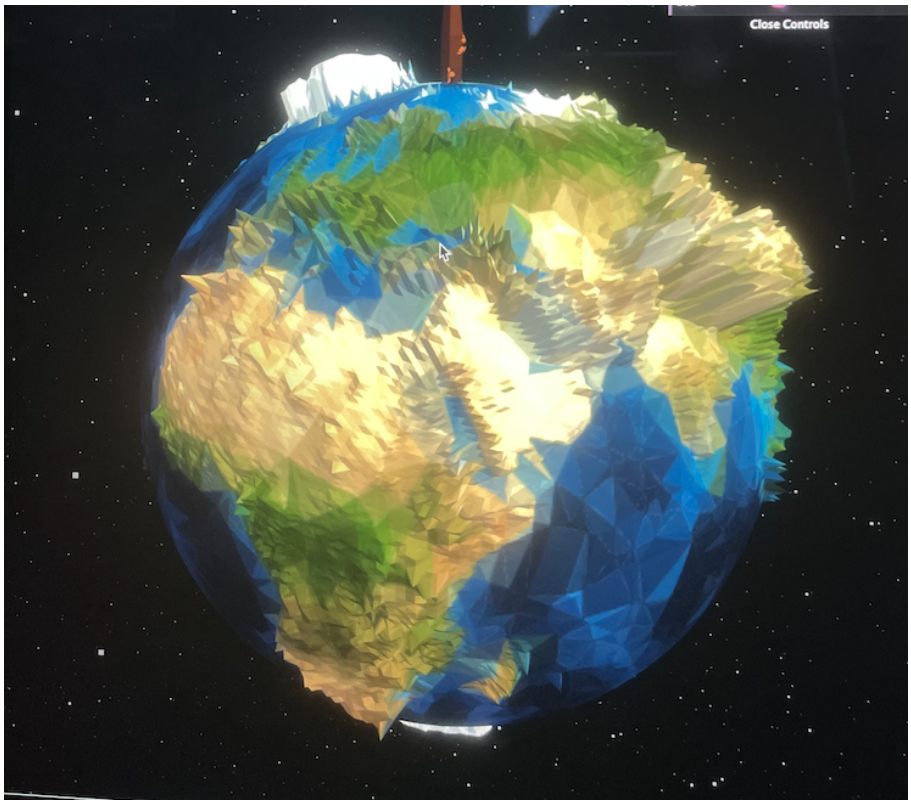
4.3.2 Sunrise and Sunset

The sunrise and sunset were created by tracking the timestamp (from when the game had first started), and using a scaled sinusoidal function with the time as the parameter to linearly interpolate (using the lerp function) from nighttime to daytime. When it is daytime (function < 0.5), the cloud objects are present in the sky.

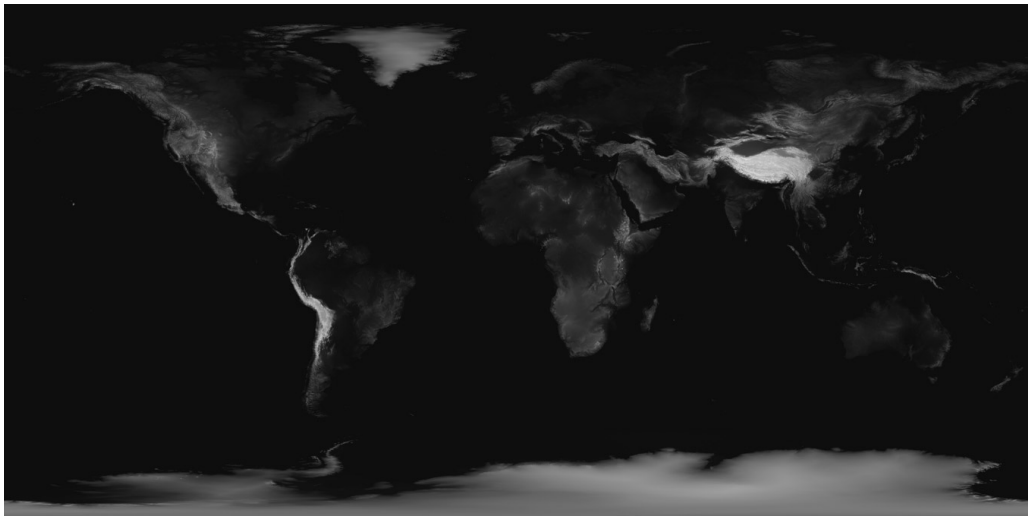
4.3.3 Terrain

When looking at terrain, we wanted to have a low-poly environment, with sufficient texture to make the terrain interesting to walk through.

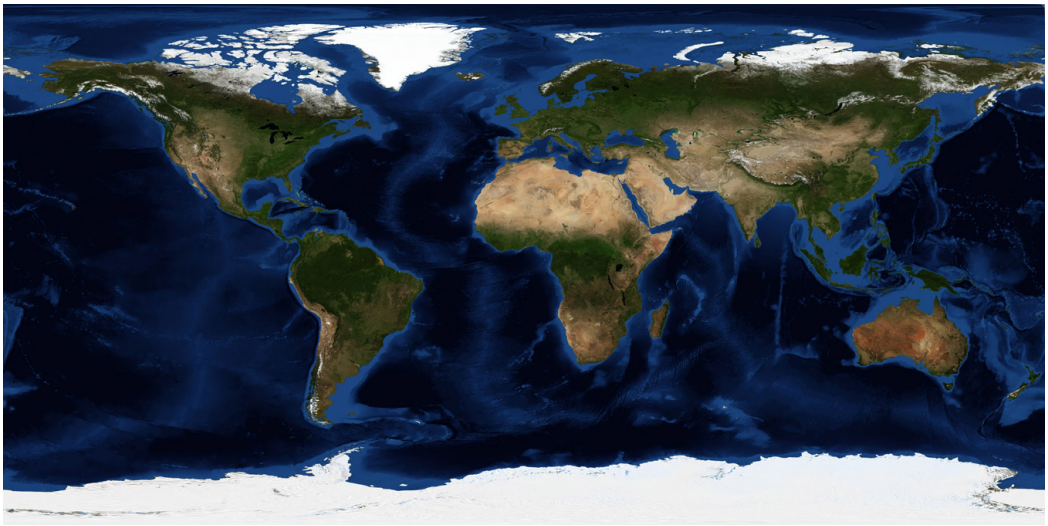
Initially, our approach was to wrap a material around the world created from an image. This worked, but it lacked the actual 3D texture that we wanted the player to be able to traverse through. Our next step was to find heightmaps we could use to complicate the material of the sphere. We used a SRTM heightmap, as well as the corresponding image, to create a globe. We triangulated the image in order to make it fit the art style.



The low-poly sphere with the SRTM height map and triangulated material applied



SRTM Heightmap (https://sbcode.net/topoearth/srtm-heightmap/)



Corresponding world material (https://sbcode.net/topoearth/blue-marble-texture-5400x2700/)

Although this was cool and we were happy with the heightmap textures, we wanted to be able to create our own terrain colors for our world to make it more dreamlike and space-y. What we wanted to do was to follow the low-poly terrain generation process described here, where we colored the triangular faces of the terrain based on its displaced height. Our approach was to calculate the distance from each vertex on the mesh to the center (0,0,0), which simplifies to being the position of the vertex. However, we ran into an issue — since the texture mapping is only applied to the vertices after the render process, there was no way to access the displaced (texture-mapped) position of the vertices intermediately, hence we could not color the vertices appropriately. To remedy this, we tried to map UV coordinates from the sphere to the heightmap, and manually take the grayscale values and the normal vector of the vertex to displace the vertices in the mesh ourselves. However, we found this difficult

with sphere mechanics as it was hard to find the spherical UV coordinates. Instead, we used MeshNormalMaterial as the material, which maps the normal vectors of the faces to RGB colors, creating a dream-like pattern. We also turned on flat shading for the low-poly effect.

In the future, we'd like to explore 3D Perlin noise for procedural terrain generation in order to create a slightly different world for every player.

4.4 Scene Objects

We incorporated a variety of objects in the game, with the hope that in the future we can add animations to them, as well as create an action key in order for the player to interact with them. We'd also like to incorporate non-playable characters (NPCs) as objects in the same way for the player to interact with, so they can reveal more information about this world. The objects in the game were mostly manually created in Blender, and we often used the decimate tool in order to triangulate the objects and create the low-poly style. We imported them into the game as .gltf objects. When we are looking towards procedurally generated terrain, we would also like to look at generating scene objects that are part of the environment and non-interactable, saving them from being computationally expensive.

5. Results

Our MVP of a traversable world was a lot more difficult than anticipated, so we are proud that we managed to figure this out. From idea generation to the beginnings of making the game, we found that our focus shifted from creating a playable game to focusing more on the player experience when walking around the globe — we found that being able to walk in a straight line and return to the starting point was almost meditative. Thus, we ended up focusing a lot on perspective, in order to highlight the fact that we had a spherical world. This includes this like the parallax error and the textured terrain, which made the scene much more immersive and engaging.

We ended up focusing mainly on the mechanics of traversing a sphere, as this is a good foundation for future development on top of this world base, as well as wrapping textures to a spherical base shape. We hope that this is a good base for creating more interactive and gameplay elements in the future for this world, as they build on top of the existing functionality that we've created.

Our game is definitely not as polished as it could be — there are issues we need to revisit in the game. Our two biggest issues are the collision detection with the world so the character is walking on top of the terrain instead of through it, as well as the rotating. Once we combined the terrain and the character mechanics, the rotation of the world updated less accurately, and thus the direction of the rotation and the character's movement direction are a little out of sync. This happens only when the

character is turning; as when the character is walking in a straight line it appears to be normal. In our demo videos, we included a video of the character walking on the bare spherical world, and the character walking on the textured terrain.

6. Discussion

Although we learned a lot doing this project, using THREE.js libraries, incorporating math and physics into our code in order to make the movement look realistic, as well as building an aesthetically interesting environment, we are unsure if this payoff was worth it. The main question is whether there is a benefit of a spherical open-world game as opposed to a flat open-world game, regardless whether the flat game is infinite or finite. We decided that although the spherical world has interesting mechanics, it would be substantially more interesting if we took it further and created gravity on different types of 3D shapes, where the shape is integral to the gameplay, instead of just being a traversable environment.

Our next steps are developing the game further based on the existing framework that we have, fine-tuning our elements, adding a story to the world, incorporating more interactive elements, adding shadows, and working with Blender to enhance the art style. We also need to fix major issues like collision detection. Although we had many failures, we also learned a lot from them — we now know in theory how to procedurally generate terrain using Perlin noise, move a character around a 3D space, and many things that could not be accomplished due to the constraints of the spherical world, but are all lessons we'd like to incorporate into future work.

7. Conclusion

Overall, although we ran into a few roadblocks in the construction of our game, we managed to attain our altered goals successfully, creating a good base spherical world that the player is able to traverse using their character. However, we definitely did have some goals left unfinished — namely, being able to create a puzzle, which was our original goal, as well as being able to interact with the objects in the scene which we created in Blender.

There are still issues that we need to revisit, such as the walking issue, which we need to fine-tune in order to ensure that the direction of the sphere rotation matches up perfectly with the direction of the character walking. We also look towards implementing collision detection with the planet, which would dramatically improve the polished look of the game, as currently the character is able to walk through objects in the world.

After that, some ideas for the future are to create a puzzle game where the player has to figure out how to make the tree in the center of the planet grow — this

would bring together both our aim of creating a puzzle game and being able to interact with the objects.

We learned a lot in this project, and are proud of the work that we accomplished. We believe that this is a good framework for future development, should we decide to continue with the project of creating traversable spherical worlds!

Appendix

Contributions

Louis focused more on the character movement and the camera direction, as well as creating the star background. Steph created the environmental objects in the world and the terrain.

References

- Super Mario Galaxy (Nintendo, 2007)
- Galaxy (https://github.com/hlorenzi/galaxy)
- Terrain Heightmaps
- Low Poly Terrain Generation
- Stack Overflow
- Three.js API

Acknowledgements

Thank you to Professor Adam Finkelstein, Nintendo, and Three.js for making it happen.