Hello, World!

Ishani Kulkarni, Abhinaya Raghunathan, Al Liang, & Monique Legaspi

COS 426: Computer Graphics, Spring 2020

Abstract

"Hello, World!" is a web-based, open-world game developed primarily in ThreeJS. The player inhabits the role of a nature photographer exploring a foggy forest, taking pictures of various wildlife over the course of several rounds. To earn points, the player must take a picture of an animal in frame, keeping it as close to the center of the viewport as possible. There is also no time limit, so they may freely explore the world around them, observe the animals, and listen to the peaceful background music. The creation of this game involved implementing such computer graphics concepts as the procedural generation of 3D objects (plants and animals in our case), mesh rendering, event handling, and user-controlled camera movement, as well as a scoring system based around measuring animal position in relation to the viewport center.

1 Introduction

1.1 Goal

Our goal for this project is to create a realistic, procedurally-generated environment which the player can explore, as well as to implement a functional picture-taking game mechanic that contributes to an accurate scoring system. In a time where so many of us are stuck at home, we want our game to give the player an opportunity to venture "outside", so to speak, and interact with flora and fauna in a relaxing way. Our gameplay model of simply walking around and taking pictures affords the player a set of stress-free, enjoyable tasks with no time limit, much like engaging in a favorite hobby. There is no danger in this game; animals walk, hop, or fly by slowly, posing no physical or scorewise threat to the player, and the charming background music can be toggled on and off as desired. By playing "Hello, World!", anyone can escape into a peaceful, natural world without ever having to leave their computer.

1.2 Previous work

We looked at work involving randomly placed objects, the ThreeJS examples, and different examples of user controls to provide an immersive experience. We also looked to previous projects for inspiration for our own game. We used this research in hopes of creating an immersive and relaxing experience for the user while they explore a forest on a foggy day.

1.3 Approach

To tackle this project, we started by dividing our tasks into subsections that each person was assigned to. These tasks were animals, scene, and game. We felt this would be a good way to divide the work since these aspect could be developed in parallel.

2 Methodology

2.1 Scene

2.1.1 Atmosphere To create the general atmosphere in which all of our objects reside, we initially used the setup code from Assignment 3 (Raytracer), removed the items inside the Cornell box, and extended the walls to create a sort of false open-sky terrain. However, upon trying this, we found that the atmospheric lighting made it such that, even if we colored the "sky" and "ground" panels to be very bright, and placed spotlights intermittently throughout the scene, it would always eventually extend into darkness. Next, we used the setup code from Assignment 5 (Simulator) and removed everything besides the sky, ground, which is a PlaneBufferGeometry, and fog, which ended up being much better, as it more closely resembled the sunny, "outdoors-y" look we wanted.

2.1.2 Trees To add trees, we started by finding different forest trees from Google Poly. We decided on including elm trees, oak trees, and pine trees to simulate a forest environment. At first, some of the trees we found had lighting integrated into the .gltf file, so we spent some time on finding other trees that did not have this so that we did not have to go into the .gltf file to remove the lighting. We then used a .glb packer to convert the .gltf and .bin file into one .glb file, which we then added to the scene. To create the effect that the player is walking through the forest, we made the trees really tall by increasing their y-scale. However, we found that making them tall resulted in less sky visibility. To combat this, we made the width of the trees smaller by shrinking their x- and z-scales accordingly. To place the trees, we generated a random x- and z-coordinate for each tree and placed them on the ground. The ranges of z-values were different for each type of tree. We chose to keep the x-coordinate range the same to simplify the edge of the forest. We varied the ranges to produce a less uniform-looking forest. The x-coordinate for the pine trees was a randomly generated number between -50 and 50, and the z-coordinate was a random number between -60 and 90. We used Math.random() to generate the random numbers, and arrived at the following ranges through trial and error to see which would produce the most realistic looking forest scene. For oak trees, the x-coordinate was between -50 and 50, and the z-coordinate was between -50 and 150. For elm trees, the x-coordinate was between -50 and 50, and the z-coordinate was between -30 and 70.

2.1.3 Grass To find the grass mesh, we looked on Google Poly. At first, we placed a lot of randomly generated grass on the ground, but the number of objects that we were rendering ended up slowing down the game. So to combat this, we loaded the mesh onto Blender to create a larger grass patch using the smaller one. Using the larger patch, we were able to add fewer grass patches, which lightened our rendering load and sped up our game. Additionally, much like the trees, many of the models we found had lights built-in, which meant that adding a lot of grass, regardless of whether they were patched together or not, would cause the environment to look unnaturally bright and yellow. We were unable to find a grass model without lights, so we spent a while researching how to remove the lights from the .glb file whose shape we liked the best. Eventually, we figured out that we needed to zero out a few light attributes directly inside the .gltf file before converting it to a .glb file. To place the grass randomly in the scene, we varied the x, z, and rotation of the mesh using Math.random() as for the trees. The x-coordinate was between -50 and 50, the z-coordinate was between -30 and 70, and the rotation calculated by multiplying the Math.random() value by 2π to give an angle between 0 and 2π in radians.

2.1.4 Mushrooms Adding mushrooms was done in a similar manner to trees and grass. We found a suitable model on Google Poly and downloaded its .gltf and associated .bin file, then used the same .glb packer to convert these two files into a .glb file. As with grass, we had to manually zero out the light attributes within the .gltf file before converting it to a .glb file, as the only mushroom model we found had lights built-in and would have altered the colors of the scene.

We distributed these mushrooms across the landscape by generating random x- and z-coordinates for each mushroom and placing them at that position on the ground. We also randomly varied the rotation and scale of the mushroom. The range of values for the x- and z-coordinate was -20 to 30. The range of values for the rotation was 0 to 2π radians, and the scale ranged from 0.05 to 0.08.

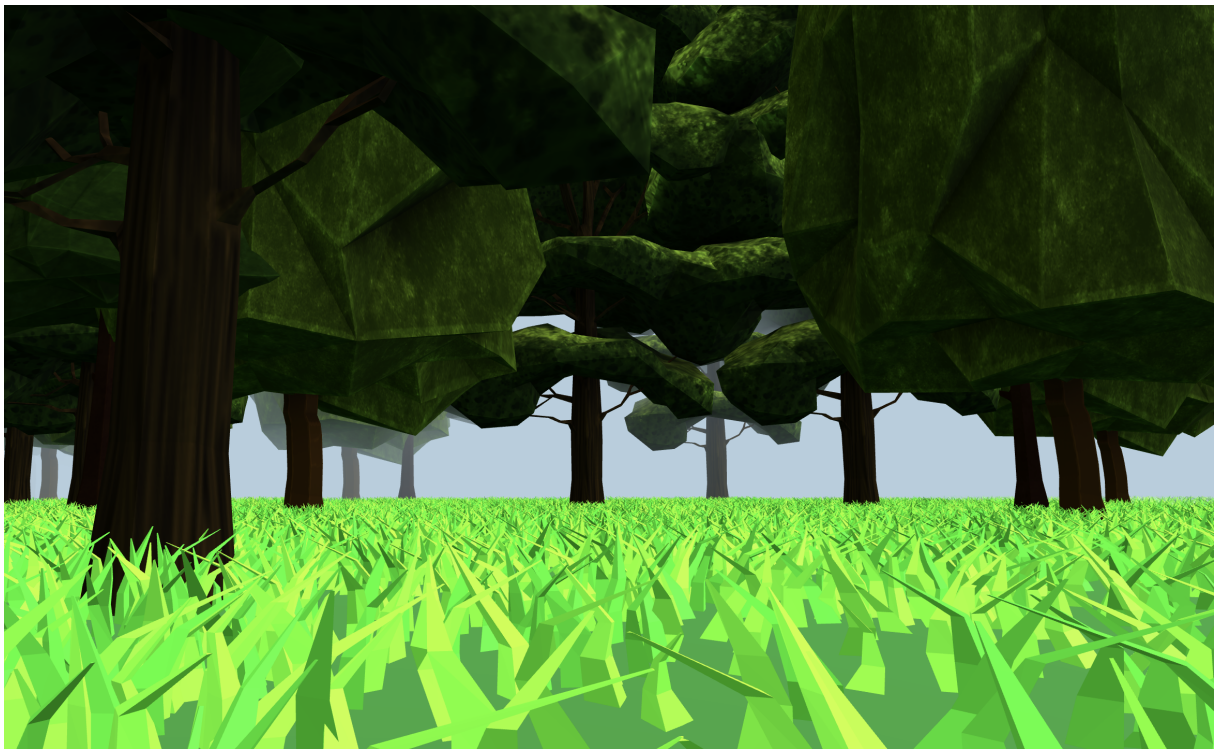

Figure 1: The forest scene, including elm, oak, and pine trees with grass and fog.

2.2 Animals

2.2.1 Making animals appear on screen Adding meshes to the scene was our first challenge. By adding the .glb file type to the webpack.config.js file, this allowed the animal to show up using the animal's .glb file from Google Poly models. This meant creating a constructor for each animal, adding an index.js file for each animal, and then importing it into our SeedScene.js file. We also exported each animal it to the objects' index.js file. For each animal, we added fields for direction and speed. Finally, we created a loop in SeedScene.js to create multiple instances of each animal and make them appear on the screen.

The stork presented some challenges, as it was an animation instead of a static model, but by referencing a ThreeJS tutorial, we were able to get the animation to appear, so that we could see the bird flapping its wings. When we loaded the .gltf file, we used the ThreeJS Animation Mixer's clipAction to create an action to play.

2.2.2 Randomizing animals on frame In order to make the animals appear on the screen in a way that makes the scene look like a forest, we had a few options to randomize the movement of each animal. The first method we tried, which we did not ultimately use, was to choose a random time for each animal. The animal starts at a position of Infinity, so that it does not show up initially. Then, when the timestamp reaches the given time, we change the position to a random position near the camera. This implementation seemed reasonable, and was working when there were a small number of objects on the screen. However, when we added the trees and grass to the scene,

the animals stopped showing up, presumably due to lag. So, in response to this, we changed our implementation to not depend on time.

So, the implementation we used was to create many instances of each animal and position them to start in a random location within a large bounding box, so that most of them will start out of frame, and then appear in frame at random times. We did this by using Math.random() for each coordinate's x-, y-, and z-value, and then scaling it. For instance, this could look like x = Math.random() * 1000 - 500. For the deer and bear, we did this scaling for the x- and z-value, and kept the y-value constant so that they stay on the ground. For the stork, we did this for the y-value as well so the bird could fly in the vertical direction as well. Then, we wanted to make the animals come towards the camera, so that they come into frame. To do this, we found a random point within a smaller bounding box, that is close to the camera. Then, we made the direction vector the vector from their starting position to this random point. This allowed the animals to move towards the camera in a random direction, to a random point, and from a random starting point. This way, it looks like they are all coming at random times during the game.

2.2.3 Movement To make the animals move, at each timestamp, we updated the animals position. At each update step, we found the animal's direction vector, which was in a field of the animal object. Then, we normalized this direction vector, and scaled it by the randomized speed. Then, for the deer and bear, we added this value to the x- and z-coordinates, because we want the y-value to remain on the ground. However, for deer, we also varied the y-value by using a scaled sinusoidal function with absolute value: Math.abs(0.5 * Math.sin(timeStamp / 300)) + 1.35. This made the deer look like it was hopping across the scene.

2.2.4 Rotation One issue that we encountered with this part was that the animal would not be facing the same direction it was moving. To fix this, we eventually found the .lookAt() function, which rotates the object in a certain direction. We set this direction to be the same direction the animal is moving that we calculated earlier.

2.2.5 Speed Finally, we varied the speeds of the animals so that they would look more natural. This required playing with numbers, because a deer should travel at a different speed than a bear. We used the Javascript Math.random() function to randomize the speeds, and then multiplied this by a small scale factor to give each animal a unique range for speed.

2.2.6 Keeping track of animals As we created the animals, we added them to an array of animals so that we can keep track of them to calculate the score.

Figure 2: Bear, stork, and deer travelling in random directions at random speeds

2.3 Camera

2.3.1 Movement In order to implement the camera movement within the scene, we used event handlers with the arrow keys using the same technique as Assignment 5. We used the ThreeJS PointerLockControls as controls for the camera. Then, we used those functions to move forward and move right in the positive and negative directions, depending on which key was pressed.
2.3.2 Rotation To implement the camera rotation, we followed the ThreeJS example misc_controls_pointerlock. We added an event handler with the spacebar to begin the camera rotation and lock the pointer. Then, we follow the pointer and move the camera with the pointer with another event listener.

2.4 Music

To implement music, we used ThreeJS's Audio API and placed it inside an extended Group object so that it could be called in a similar manner to the animal/plant meshes above. We bound an event listener to the "M" key on the keyboard, which allows the player to play and pause the music at will. We originally wanted the audio to play automatically upon opening the webpage, but Google Chrome unfortunately has an issue where it will not autoplay audio unless the player interacts with the canvas first, leading to some curious double-muting scenarios if the player does not interact fast enough (i.e., the player has to press "M" two or more times to get the audio to play for the first time). To combat this, we altered the code so that it does not autoplay the audio, allowing the player to choose when they want to start the music. This ends up being better than before, because in our experience, it can be annoying when a website autoplays music (especially if you're playing in a quiet space, like a library!).

Our soundtrack of choice is the Animal Crossing: Wild World 1AM background music. It was originally a placeholder, but we liked it so much we decided to keep it. :)

2.5 Gameplay Elements

2.5.1 Photography After moving and rotating the camera's position to their liking, the player can press the "I" key to capture an image. Taking a photo triggers the scoring function to cycle through the animals that have been generated, see if they are onscreen, and ascribe point values accordingly. There are a set number of photos the player is able to take before the game ends, and the amount of remaining photos available is tracked with a variable. Once the number of photos left reaches zero, the game ends.
2.5.2 Bonus Animal In each "round" of our game, the player is assigned a type of animal to optionally seek out and photograph, which we will call the "bonus animal". The bonus animal is randomly chosen amongst all possible species of animals that will spawn and are subsequently displayed onscreen. Taking a photo with the bonus animal in it will advance the player to a "new round", i.e. give the player another bonus animal they can capture. This is achieved through a conditional within the scoring function: should the bonus animal's species be the same as an animal registered to be inside the photo, call a function that generates and displays a new bonus animal.
2.5.3 Scoring Each animal in our game has a base score, which is awarded if that animal is within the picture taken. Specifically, for each animal within frame, its species is noted through the animal's "name" attribute, and the score is added accordingly. Additional bonus points are awarded based on the animal's proximity to the dot in the center of the photo, as well as the animal's closeness to the player. Lastly, if the photographed animal is the bonus animal, a multiplier is

applied to the points awarded for that particular animal. If multiple animals in the photo were of the bonus animal's species, the point values are boosted for each of those animals.

2.5.4 Display By default, the display shows a dot in the center of the screen to assist the player in aligning their photos, the score tracker in the upper left, and the bonus animal in the upper right. For each of these elements, a type was created in HTML using style CSS, and then a div of that type was made. These elements were then introduced into our main app.js file by appending this HTML to the body of the document. The display can be toggled on and off using the "D" key, if the user would prefer to play without viewing the score, bonus animal, and camera dot. The color yellow was chosen for these elements because it was the color that felt the most natural in the game environment, while still being visible.

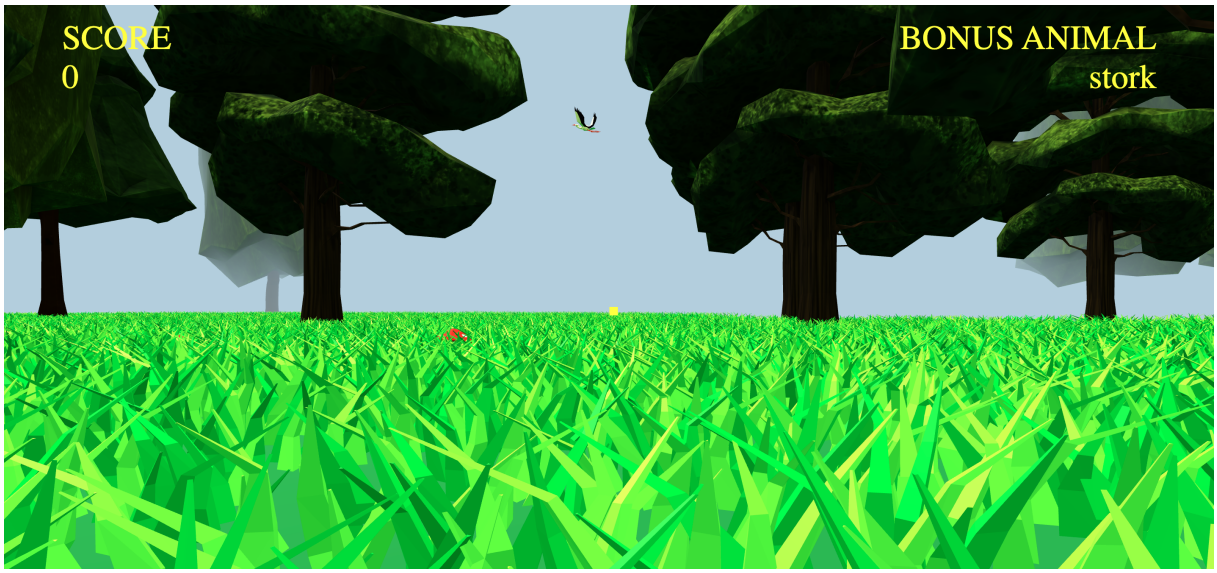

Figure 3: The game display, including the score, the camera dot, and the bonus animal.

2.6 Menu

To implement the opening menu, we went into our app.js file and added HTML elements such as divs and buttons that would sit on top of the ThreeJS canvas. Our menu background consists of a full-screen, semitransparent div, so that the player gets a sneak peek at the animals moving around in the forest before they even begin playing. The menu panel is a solid green div with vertical flex enabled inside so that the text lines up. The text is simply made up of center-aligned header and paragraph elements, and the Start button is bound to an event listener so that, when clicked, it will hide the menu and display the canvas in full. We also placed a small Menu button at the bottom-right of the screen which will un-hide the menu when clicked, allowing the player to access the menu at any point mid-game, in case they need a refresher on the controls.

3 Results

Overall, we wanted to create an immersive experience for the user, so that they feel like they are in that environment. We would consider ourselves mostly successful in this endeavor. Adding and adjusting the trees and grass made the scene feel much more realistic. Furthermore, adding a first-person camera movement made it feel more immersive for the user. One goal that we were

Figure 4: The opening menu.

unable to reach due to resource limitations was using animals that have a .gltf file of them moving. We found a ThreeJS example of a stork flapping, and adding this to the scene made it seem more realistic. If there were a similar gif of a deer or bear moving, that would elevate the project to make it seem much more realistic. To test the project, we played through the game and tried to take pictures of different animals and of empty scenes to see how the score function reacts. We tested different event handlers and moving the camera in all directions.

4 Discussion and Conclusion

Overall, the approach we took looks promising. We were able to achieve an immersive and relaxing experience of wandering through a forest and coming across a variety of animals. We used the ThreeJS methods to allow the user to walk through our randomly generated scene and take pictures of animals. We learned a lot about optimization techniques and how to speed up the game so that it provides the fastest and smoothest experience for the players. We also spent a lot of time on the design of the scene to make it feel realistic and calm, using darker greens and making the trees tall and large. We tried to modularize the codebase as much as possible, creating classes to represent objects and make the code more readable.

Some issues we would like to revisit are the displaying of images when someone takes a screenshot. If we could save the images locally and then display them in a photo album, that would be ideal. Unfortunately, due to the security issues that would ensue should we be able to save the images to a specific folder on the player's computer, which would be necessary in order to display the images from local files, this approach for image displaying was not possible. We were able to store the canvases from each photo taken into an array, but have not yet been able to successfully convert them to images and display them. Fixing this issue would also make this project more enjoyable and interactive. Lastly, while we tried our best to modularize, we feel that utilizing more global variables would help clean up our dependencies.

For follow-up work, we would like to make the game even more immersive in the future, using animated meshes and procedurally generated terrains to vary the ground levels. Furthermore, future work that we would want to add are collisions with trees and animals, so that animals do not go through each other or through trees, and so that the camera does not go through animals or trees. Another stretch goal was to display the screenshots to accumulate an album of photographs that the player could access in-game. Finally, another goal would be to add weather, such as rain and lightning, and to vary the sky so that it starts as daytime and then as time passes, it turns to night.

We learned a great deal through this project. Primarily, managing a full Graphics project with a group of four helped us understand how to split up a large project into smaller, manageable tasks, and manage our time efficiently so that we stay on track. Also, since this project was so abstract and free to interpretation, communication among the group was imperative, so that we each had a similar idea of our goals for the project. Finally, we learned many technical skills, such as using ThreeJS, Blender, and Google Poly images. We also got practice using Github and Javascript, and improved on our debugging and problem-solving techniques.

5 Contributions

To split up this project, we split up the game into the scene, animals, and backend game tasks. Abhinaya and Monique worked on the scene, adding grass, trees, fog, and other elements to make the scene look realistic. Ishani worked on animals, and randomized all of the movements. Abhinaya deployed the project to Github pages. Ishani and Abhinaya worked on moving the camera around the scene and rotating the camera. Al worked on the photography functionality, scoring functionality, and in-game display features. Monique added background music and the pre-/mid-game menu screens.

6 Acknowledgements

We would like to thank our Point TA Joanna Kuo for meeting with us and guiding us throughout the process of making this game. We would also like to thank Darby Haller and Ethan Tseng for offering advice and assistance through office hours and email.

7 Honor Code

We pledge our honor that we have not violated the Honor Code during this project.

/s/ Ishani Kulkarni
/s/ Abhinaya Raghunathan
/s/ Al Liang
/s/ Monique Legaspi

8 References

1. [Online]. Available: https://discoverthreejs.com/book/first-steps/animation-system/
2. [Online]. Available: https://threejs.org/examples/misc_controls_pointerlock.html
3. [Online]. Available: https://www.youtube.com/watch?v=EKiDg9xWYcg

4. [Online]. Available: https://poly.google.com/

5. [Online]. Available: https://threejs.org/

6. [Online]. Available: https://en.wikipedia.org/wiki/Rotation_matrixAxis_and_angle