Bird-like Flight in Virtual Reality

RORY SOIFFER and EVERETT CHENG, University of Washington

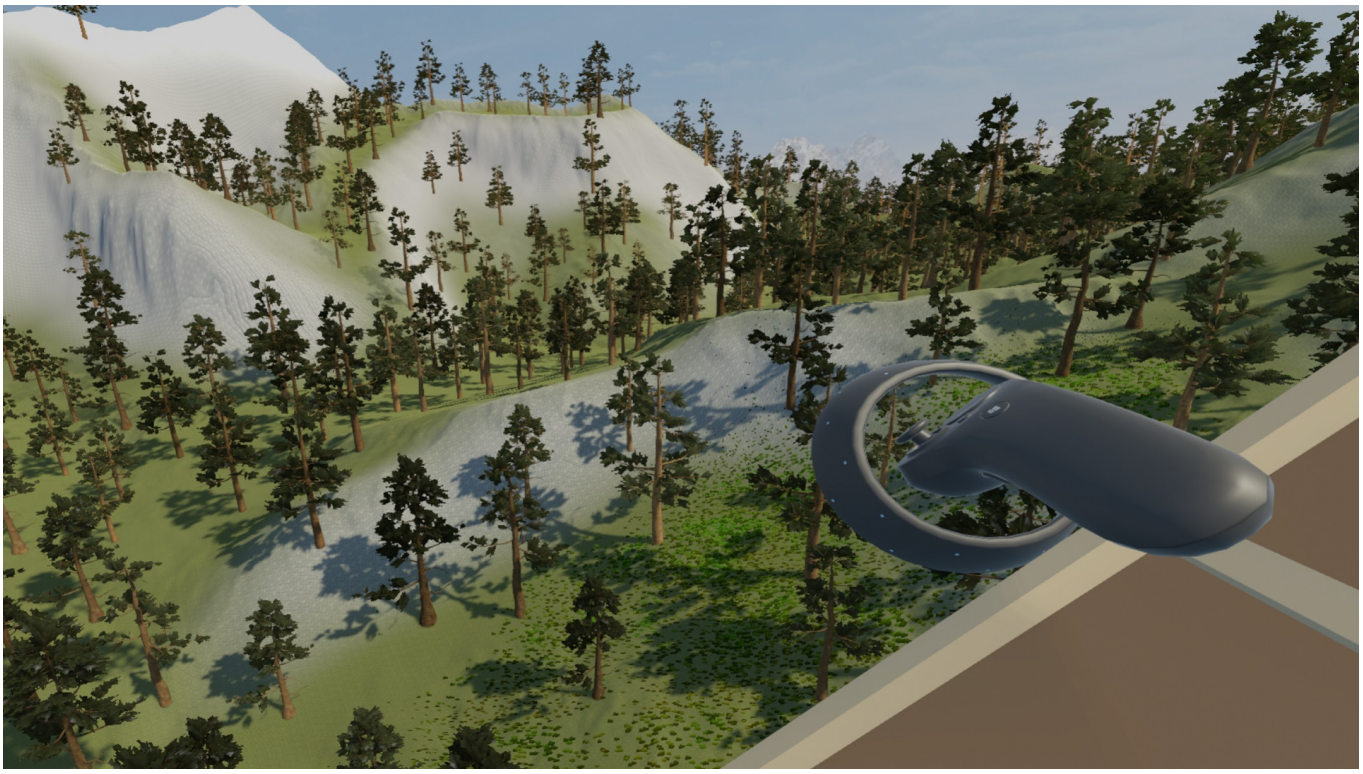

Fig. 1. Inside the game, the player stares at a forested mountainside. The player's virtual wing is visible on the right.

We created a game about flying as a bird through a massive virtual world. The terrain is procedurally generated and extends for over 4 km in every direction. The game includes a physical wing attached onto the player's arm. The game tracks the position of this physical wing, using it to simulate air flow and calculate realistic flight physics. This creates a unique experience where the player has a feeling of true flight.

1 INTRODUCTION

Humans have always dreamed of flight. Compared to walking, flying is literally a whole new dimension of freedom. Despite this, very few virtual reality games allow the player to fly through the skies like a bird. Our project fills this gap, creating a compelling and unique VR experience.

There do exist some VR games that allow players to fly like a bird. The most popular is Birdly VR (http://birdlyvr.com/), a game that has players lie down on a large, complicated machine that has movable wings below the player's arms. This experience is designed for large conventions, but not for personal use.

Authors' address: Rory Soiffer, rorys4@cs.washington.edu; Everett Cheng, eccheng@cs.washington.edu, University of Washington.

Our project is similar to a small-scale version of Birdly VR. Instead of a large, complicated machine, we have a single wing built of inexpensive and commonly available materials. We hypothesize that having even a simple physical wing that matches the player's virtual wing makes the experience both more intuitive and more engaging.

1.1 Contributions

- We created a massive procedurally-generated virtual world for players to fly around in, with basic gameplay and interactive elements.
- We built a physical wing, tracked its position in the game, and implemented a physical simulation of the wing's interaction with the air.

2 RELATED WORK

We could find no existing work on creating custom VR controllers that model bird wings. The closest existing product is Birdly VR. It replaces the controllers entirely, instead using a large custom machine with handles for the player to hold. It also includes physical rotation of the player and uses a fan to simulate wind. This approach

leads to a very high quality experience, but at the cost of setup difficulty, no portability, unusual controls, custom hardware, and an extremely high cost.

3 METHOD

Wing controller

In building a physical wing that attaches to the player's arm, we followed the common technique of building the custom hardware around an existing VR controller, which handles both positional tracking and user input. We ensure the wing is extremely lightweight, so as to not place a strain on the player, while also being extremely sturdy, so that it doesn't break if the player accidentally hits an object in the real world. For the details on how we built the wing, see section 4.

We implement flight physics in the game based on a real-world model of air resistance, with a number of "cheats" to help the player. While the game is running, the wing continually applies a drag force to the player, using the high-velocity drag equation $F_D = \frac{1}{2}CAv^2$, where C is a constant, A is the cross-sectional area, and v is the velocity of the object relative to the air. To account for the fact that different points on the wing may have different velocities (as the wing may be rotating), we divide the wing into numerous small points along its surface and compute the force for each point separately. After tuning parameters, we set $C = 5\ \mathrm{kg/m^3}$.

The equation above is a realistic model of flight. However, humans are realistically completely incapable of flight. We implement several deviations from the above model, or "cheats", to make flight practically possible:

- We disable torque and rotation. This both makes the game easier to control and helps avoid VR sickness. It also makes it possible to fly with only one wing.
- We enhance the motion of the player's arm in real life by a factor of 15. This increases v in the equation above by a similar factor, which increases the force by a factor of 225. This makes the player incredibly strong in game, capable of moving dozens of meters in a single flap.
- We constantly apply a small forwards thrust force while the player is in the air. This makes it much easier for the player to glide long distances without losing altitude.

Terrain generation

In order to create a large world for the player to fly around in, we combined several procedural terrain generation techniques. To get a heightmap with realistic-looking mountains and valleys, we began by using the approach from [Argudo et al. 2019]. Their synthesis pipeline first generates a *divide tree*: this includes the locations and elevations of all peaks, as well as which peaks are connected by ridges, and the locations and elevations of the saddles lying on each ridge. The divide tree is generated so as to match measured real-world distributions of several orometric characteristics. Given a divide tree, the pipeline proceeds by generating a river (or valley) network: the Voronoi diagram on the set of peak and saddle positions is calculated, and rivers are placed along Voronoi edges which do not run across ridges. The elevations of river source nodes are set to values close those of nearby saddles, and then the elevations of the rest of the river nodes are propagated from the sources by following the direction of flow. Next, the ridge and river networks are refined by splitting segments and randomly adjusting the positions of vertices, so as to give a natural appearance. Finally, the heightmap is built by sampling many points and interpolating the heights of the nearest ridge and river.

We used the code provided by the authors of [Argudo et al. 2019]. However, we had to make some changes in order to get good results. This is because there was an issue where rivers would often be generated very close to ridges, and then during the refinement step parts of them would be moved through the ridges, causing large discontinuities in the final heightmap (see Figure 2). In order to fix this, we added a step after the refinement, where any river nodes within a certain distance of ridges would be removed. Any river nodes orphaned by this removal also need to be removed, so that each river node still flows to another. Making this addition prevents the discontinuities in the final heightmaps, as shown in Figure 2.

After generating a heightmap in this way, we further refined it in two ways, as suggested in [Argudo et al. 2019]. First, we added fractal noise to provide more variation and reduce its smooth, flat appearance. Then, we ran a simulation of erosion to get more realistic landforms and mountain shapes.

From here, the next step was to texture and decorate the terrain. We used three main textures: grass, rock, and snow. We applied these three textures in different proportions to each location on the terrain, depending on its height and steepness: snow at high elevations, rock for steep slopes, and grass otherwise. Specifically, we first compute

$$s = \texttt{Clamp01}\left(0.5 + \frac{\text{TerrainHeight} - \text{SnowLine}}{\text{SnowTransition}}\right),$$

$$g = \texttt{Clamp01}\left(0.5 + \frac{\text{TerrainAngle} - \text{GrassThreshold}}{\text{GrassTransition}}\right),$$

and then derive the proportions of the three materials by

$$\text{SnowAmount} = s,$$

$$\text{GrassAmount} = (1 - s) \cdot g,$$

$$\text{RockAmount} = (1 - s) \cdot (1 - g).$$

(Here SnowLine, SnowTransition, GrassThreshold, and GrassTransition are tunable parameters, and `Clamp01` is the function $x \mapsto \max(0, \min(1, x))$.)

For trees, we placed many instances of a few tree models randomly using Poisson disc sampling across the entire terrain. At each Poisson sample location, a tree was placed with probability equal to GrassAmount; thus trees will not appear on high, snowy peaks, or on the sides of steep rocky slopes.

Finally, we placed long grass objects across the terrain using Perlin noise. The density of grass at each location (as a fraction of the maximum possible) is

$$\text{LongGrassDensity} = \text{GrassAmount} \cdot \max(0, 1 - 4 \cdot \text{NoiseSample}^2).$$

Here NoiseSample is between 0 and 1. The use of a quadratic here gives a bias where areas are likely to either have lots of grass or no grass, not somewhere in between.

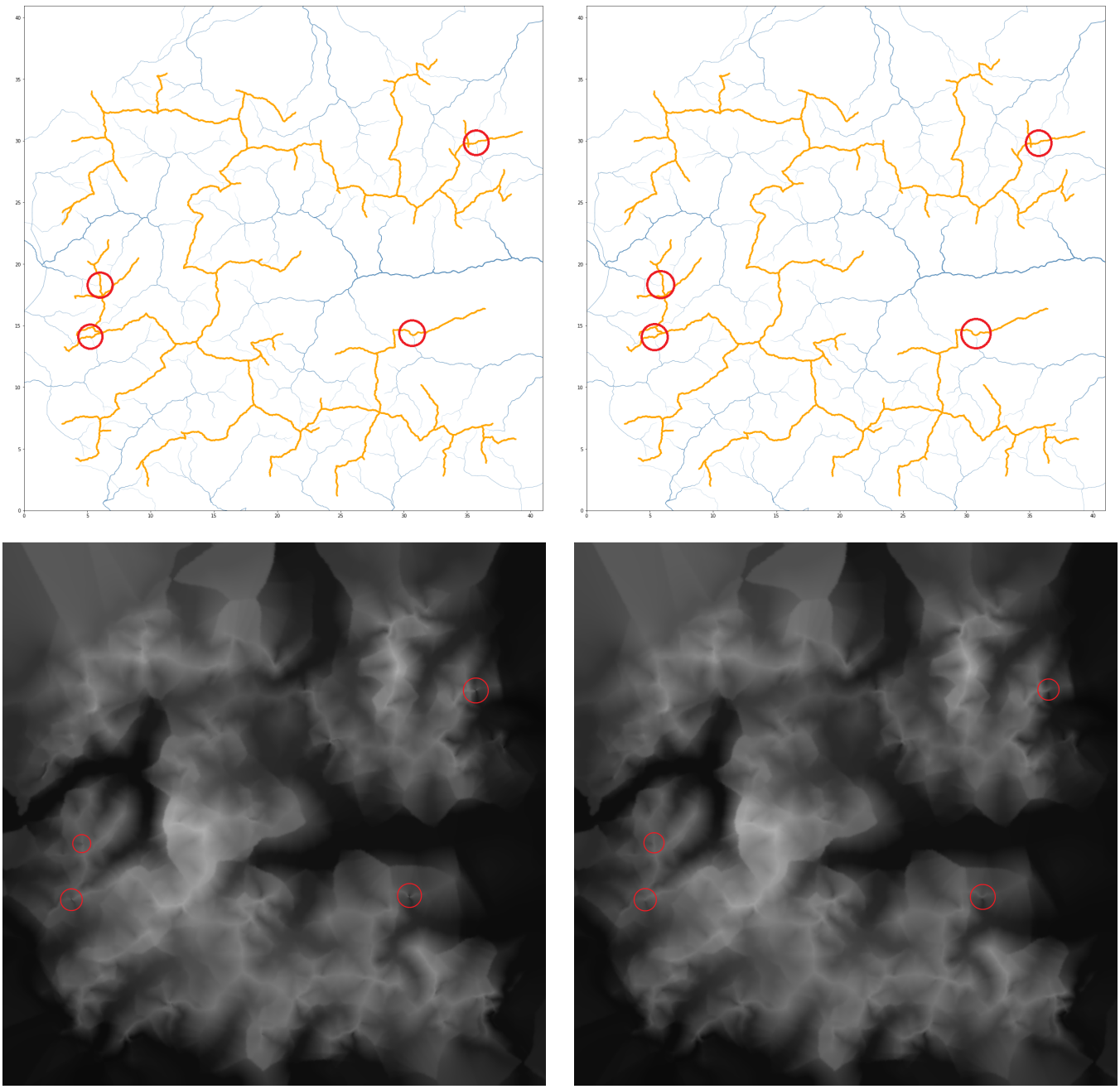

Fig. 2. Left: generated ridge/river networks and final heightmap before our changes. Right: networks and heightmap after our changes.

Since we have two different long grass sprites, we obtain NoiseSample using slightly different noise patterns for each. In particular, we set

$$\text{NoiseSample} = \alpha \cdot \text{CommonSample} + (1-\alpha) \cdot \text{TypeSpecificSample},$$

where we sample CommonSample and TypeSpecificSample from different Perlin noise patterns (optionally using multiple octaves). For CommonSample we use the same noise pattern for both grass types, but for TypeSpecificSample we use different patterns. Further, the CommonSample is weighted more heavily (eg. $\alpha = 2/3$) and uses lower-frequency noise: this ensures that both types of grass generally spawn in the same areas, with only small, high-frequency variations between them.

Fig. 3. This is the wing controller. The player holds the Vive controller, rests their forearm against the pad to the upper-left, and secures the wing with the straps. The wing extends 2 feet forward from the player's hand, and 1 foot to the side.

4 IMPLEMENTATION DETAILS

Wing controller

We built the wing controller out of wood, 3d-printed plastic, felt, and velcro. We use balsa wood for the skeleton of the wing because of its light weight, sturdiness, and flexibility. The primary bone of the wing is a single 1/2" x 1/2" x 36" piece. The 3 secondary bones are 1/2" x 1/2" x 12" pieces. The bones are connected together with custom-designed 3d-printed joints. The joints have 1/4" of plastic around the outside of the wooden bones to provide strength. Each joint is made of two pieces, for the top and bottom, as 3d printers can't effectively print parts with overhangs. The very low tolerance of 3d printers let us print joints so precisely that they can hold the bones through friction alone. We glued the joints and bones together to be even more sure of the structural integrity.

We glued other custom 3d-printed parts to the forearm part of the primary bone, with holes to allow velcro through. The velcro winds through the plastic parts and around the player's arm, where they can adjust it to fit securely. We use three straps to provide redundancy even if one or two give out. We then cover the straps and the entire forearm piece with felt to increase comfort. A similar custom 3d-printed part attaches to the middle secondary bone, where it holds the Vive controller. While the controller is secure through friction alone, we also have space to tie the controller strap as a failsafe in case the controller gets loose.

We then covered the entire bottom side of the wing with pieces of felt, which we attached together with more velcro (as we didn't have the tools to sew the felt together properly, and glue proved ineffective). The felt is then attached to the wooden bones with thumbtacks. The result is very sturdy. It holds up to vigorous flapping without issue, easily survives being dropped, and can hit or push against other objects safely. It is also very light: the entire wing weighs about as much as the Vive controller.

We ran into one major challenge while building the wing: due to the coronavirus, all the makerspaces at UW suddenly closed, leaving us unable to 3d print parts. Luckily, we had just barely printed enough parts to construct one wing. Our original plan of constructing two wings had to be scrapped.

Terrain generation

First, we used the Python code from [Argudo et al. 2019] (with our modifications) to generate a heightmap. The divide tree was generated using orometric data from the Swiss alps as reference, and covers a 40.97 km × 40.97 km region. At one pixel per 10 m, this gave a 4097 × 4097 heightmap (outputs shown in Figure 6).

For the augmentation with fractal noise and erosion simulation, we used Houdini [1]. After this, we imported our finished heightmap into Unity using the built-in terrain system, and scaled it down by

[1] https://www.sidefx.com/products/houdini/

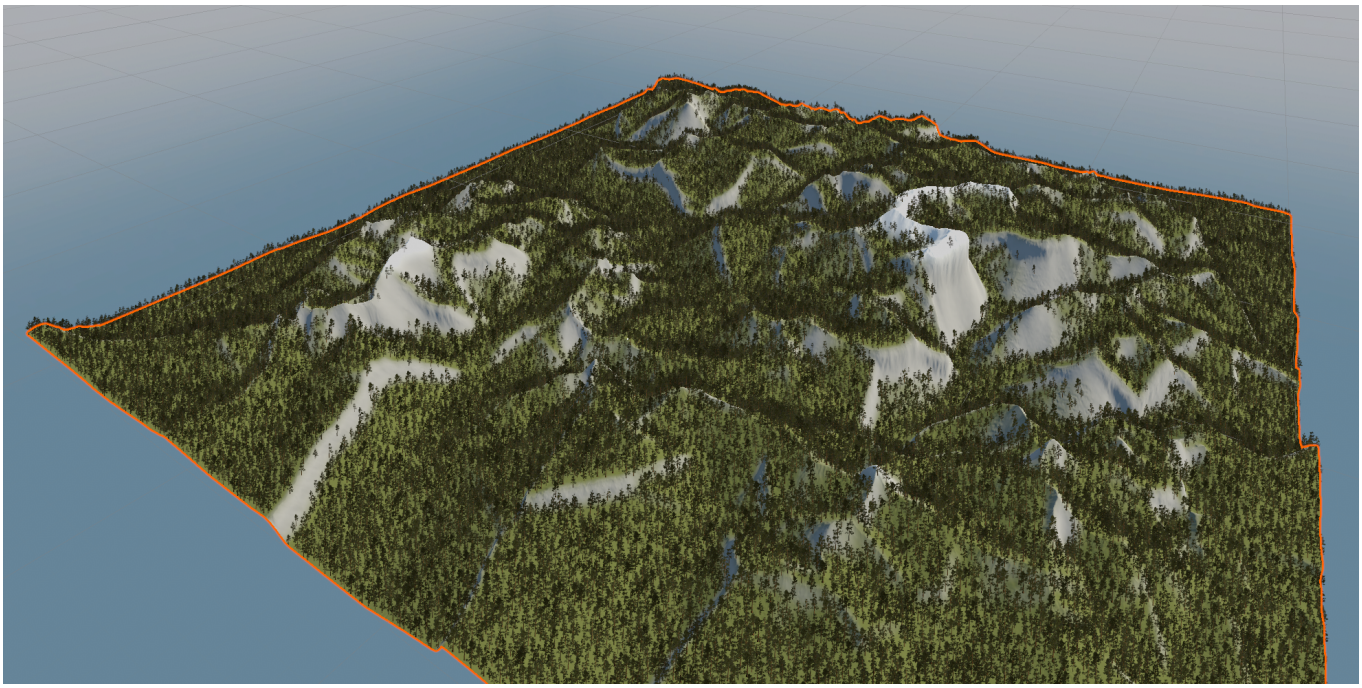

Fig. 4. This is the game world, seen from far away. The terrain is over 4 km wide in each direction. The player has freedom to fly anywhere.

a factor of 10 (otherwise it would simply be too large, and would take players far too long to fly across).

We then added textures, trees, and grass detail objects using Unity editor scripts. For the texturing, we set SnowLine = 3800 m, SnowTransition = 400 m, GrassThreshold = 45 degrees, and GrassTransition = 10 degrees (these values use the height of the terrain before it was scaled down). We also blurred the texture splatmaps to give smooth transitions between grass, rock and snow, by applying a box filter of size 10 m. For tree placement, we used Poisson samples with a radius of 15 m or 25 m (after scaling down).

5 EVALUATION OF RESULTS

The finished wing controller is both functional and stylish. In fact, 100 percent of sampled users thought the resulting design looked cool ($n = 4$). The wing is very lightweight, weighing about 1 pound. When attached to a user's arm, it gives them an extra 2 feet of reach. For a person with a normal armspan of 6 feet, a pair of these wings would increase their total armspan to 10 feet.

As for the terrain, the aesthetic results are mixed. While it is realistic enough to provide the experience of flying over a natural landscape, it is still very simple and lacking detail. We believe the mountain and valley shapes look more realistic than what we could have achieved by manual sculpting. However the terrain could use a wider variety of textures and decorations (currently the same small textures are tiled across the terrain, which causes cliffs to look plain and uniform).

The results for the flight physics are mostly positive. Flying in the game is relatively easy, quite versatile, and involves a moderate amount of physical exercise. We found that experienced users were able to easily and quickly fly to any location they chose. One potential concern is the difficulty of getting used to the flight physics for someone new to VR. Unfortunately, the coronavirus prevented us from running a user study.

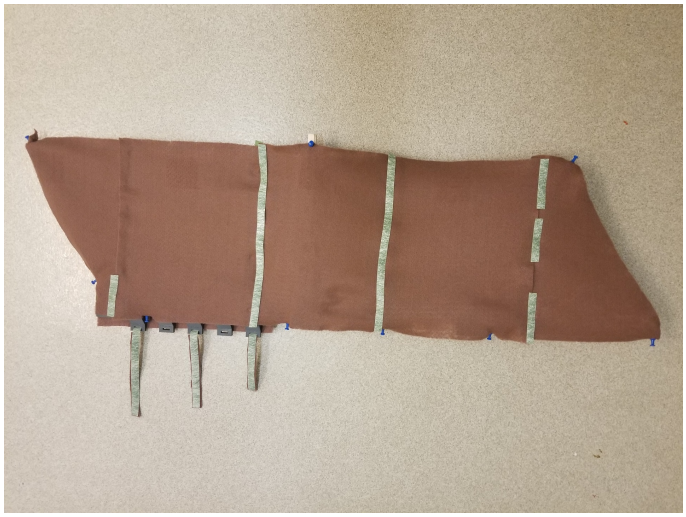

Fig. 5. This is the back of the wing controller.

6 FUTURE WORK

In the future, we would try to create a second wing. Our original plan was to create a pair of wings. However, the makerspaces at UW shut down in the middle of our project due to the coronavirus, making it impossible to 3d-print parts. We barely had enough parts to finish one wing. Building the second wing would be easy now that we have the design.

Future wings could be improved in both size and quality. The wing currently extends 2 feet past the player's arm - a significant distance, but still far too small for an actual wing for a human-sized creature. We aimed low with this project to ensure that we wouldn't have any issues with the wing's sturdiness. This turned out to not be a problem at all, so in the future we would attempt to build an even larger wing. In addition, we could try adding hinges to the wing to allow it to fold. Birds do fold their wings as part of flapping in real life, so this change would greatly increase the realism of the flap motion.

We would like to improve the gameplay of the demo. There are currently no game objectives, it's just a sandbox environment where the player can freely fly around and interact with drones. Adding a more advanced combat system where you fight drones, or a time trial system where you race a set course, would vastly improve the game.

For the terrain, we would like to improve the texturing, as well as add a wider variety of tree, plant and rock models. We would also like to add rivers and streams with flowing water, as well as lakes.

7 CONCLUSION

We created a VR game where the player flies around a massive landscape by flapping like a bird. We built a physical wing controller to complement the game, so the players flap a wing in real life and in the game simultaneously. We developed an enormous procedurally-generated mountain forest landscape for the player to explore in the game.

ACKNOWLEDGMENTS

We thank John Akers and the UW Reality Lab for technical advice and access to the Incubator space.

We thank Kirit Narain and the UW XRA for loaning us the Vive hardware we used for the project.

REFERENCES

Oscar Argudo, Eric Galin, Adrien Peytavie, Axel Paris, James Gain, and Eric Guérin. 2019. Orometry-Based Terrain Analysis and Synthesis. *ACM Trans. Graph.* 38, 6, Article Article 199 (Nov. 2019), 12 pages. https://doi.org/10.1145/3355089.3356535

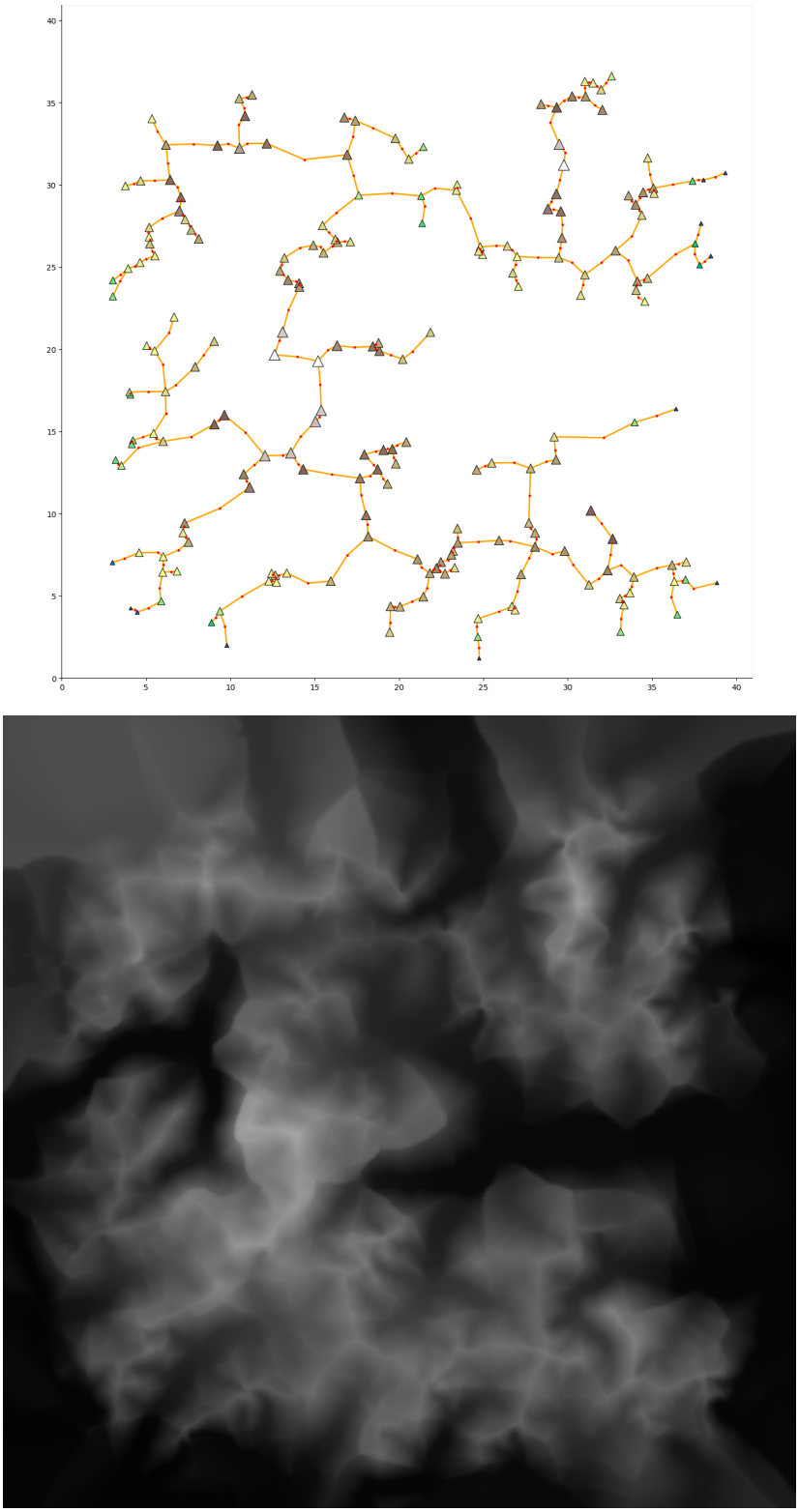

Fig. 6. Top: coarse elevation map and peak probability map provided to the divide tree synthesis code (see [Argudo et al. 2019]). Middle: the generated divide tree. Bottom: the final heightmap after adding noise and simulating erosion.