New Technologies

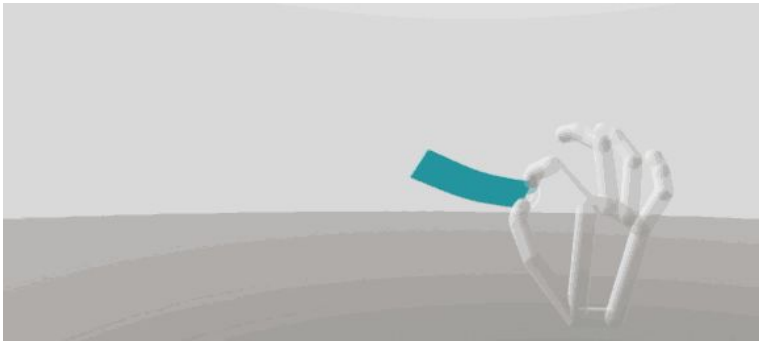
Rick Ruiten 3013670

Index

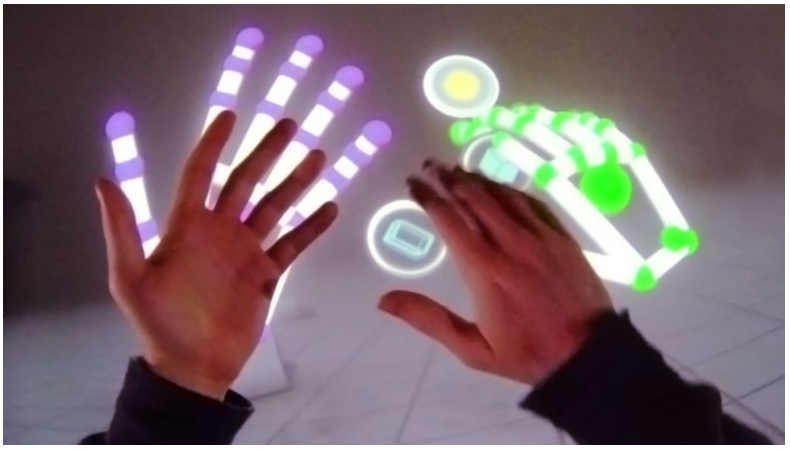
- Technology
- Goal
    - Experiment 1
    - Experiment 2
- Design
- Workflow
- Results
    - Experiment 1
    - Experiment 2

Technology :

The Leap Motion is a tracking device that is capable of tracking your hands while using VR or just as a controller. To use the Leap Motion in VR you will need to stick it onto your VR headset as shown on the image. The old Leap Motion was connected to your PC and was able to recognize your hands from below. You would hold your hands above the Leap Motion and through infrared sensors it would detect your hands their position and rotation. Leap Motion figured out that controllers in VR are awful to use and decided to focus more on the VR aspect of motion control.



Right now they are using the Orion SDK which is, according to their website, more ***Natural, Accurate and Robust*.** Not only do they own a solid hand tracking device, they also write that the next generation of VR / AR headsets will have a built-in Leap Motion



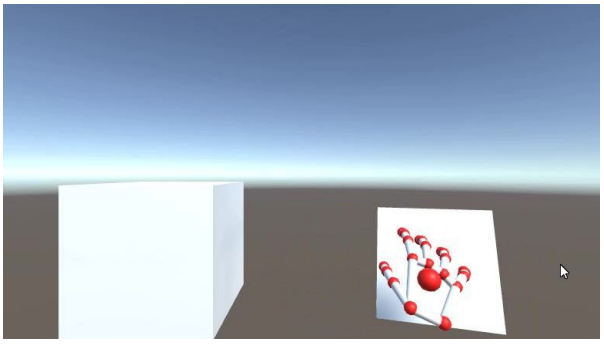
Goal :
My goal is to create a User Interface that is easy to use and helps the user to visualize the values he is adjusting. The user will create his own planet by adjusting the values with his left hand and selecting the values with his right hand. There are already some User Interfaces available for the Leap motion, but I want to check if the Leap Motion can be used as a pointer / controller and not as an object which is used to display the interface.

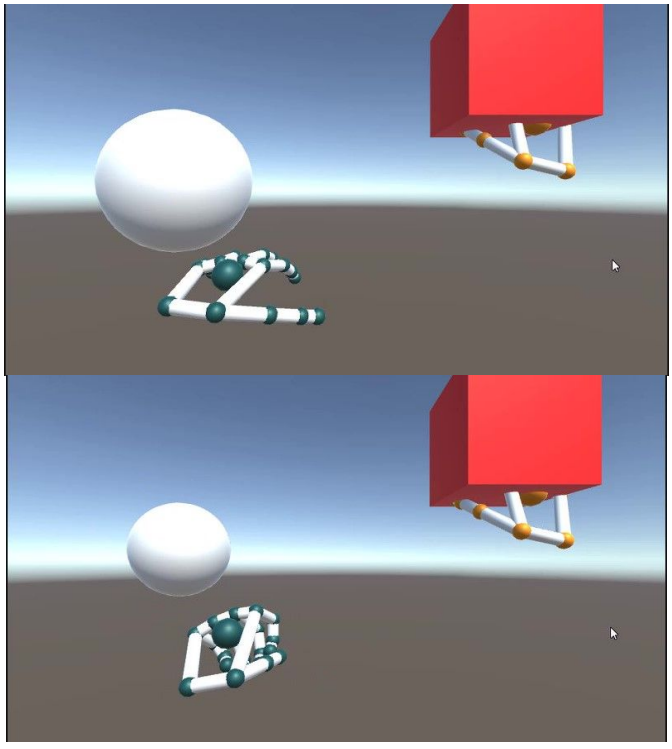
Experiment 1:
Create a UI that is easy to use.
Experiment 2:
Adjusting values with few interactions and no hand gestures.

Design :
The design of my project started as a project where you would create a whole universe by spawning planets, asteroids and other celestial objects. However, after some small prototypes I figured out that a whole universe might be a bit too much and so I reduced it to only the planets. Not only did I want to let the users play a little bit with the Leap Motion and the system, but they should also be able to save their planet and export it to use it in their own projects or just for fun as a screensaver.

Workflow:
I figured out that I still had an old version of a procedural planet package which we used at our internship. This asset might've been a little bit outdated, but it still did its job. To use this asset you'll need a planet manager, an object that resembles the sun and a planet which you want to edit. To adjust the values you will have to use the inspector because that is the only place where the values are visible. I found this quite annoying as there is no information about what the variable does, except for the name, and thought that this could be done way easier. I came up with the idea to visualize all the, useful, variables and wanted to use the Leap Motion as a controller.

First I had to figure out how the Leap Motion worked by testing the basic interactions I needed in a test scene. I figured out that by testing how the physics work of the Leap Motion and basic cubes it would be the easiest way to create an Interface by using 3D cubes.

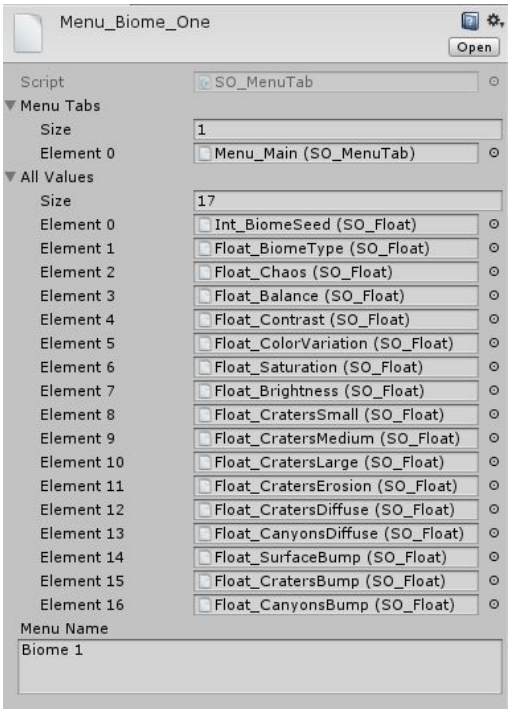


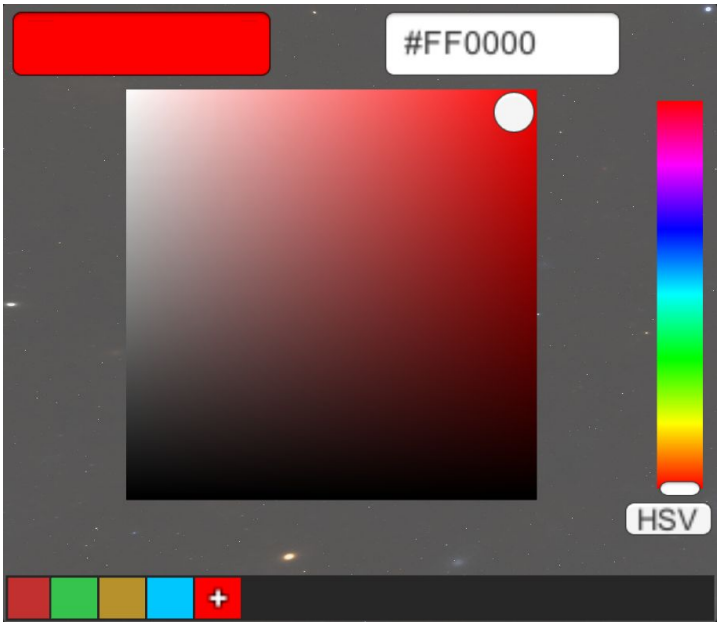


After the basic interactions I thought about how to adjust the values. I didn't want any sliders in it and it had to be simple basic interactions that everybody was able to use without using a big manual.

The Leap Motion's hand models exist out of small bone models creating the hand and has no hand gestures registered. These were the main reasons why I decided to not use any hand gestures in adjusting the values. Instead of any gestures I decided to use the average distance of the distance from your fingertips to your hand palm. The more the user would clench their fists the lower the value would be. This would only be in effect when the user would enter the trigger with his other hand.

I created Scriptable Objects for all the values of the planet, SO_Value, so that I could add them to objects in a menu and get the values form those object as a scriptable object. By creating a Scriptable Object for a menu, SO_MenuTab, I could easily rearrange the menus or go from menu to menu. In these Scriptable Menus I added a list of the SO_Values where I can add all sorts of values (eg. Colors, Floats, Integers) in a specific order. To create a Float I created a script that derives from the SO_Values script and added some values (eg. floatValue, minValue, maxValue, isInteger). The colors derive from the SO_Value script and only yields the color value. Not only can you add values but you can also add new SO_MenuTabs to it to create a folder structure. Using these scriptable objects in a Menu Creator script results in a dynamic menu that exists out of 3D triggers that pass their current value to the game manager and applies it to the planet.

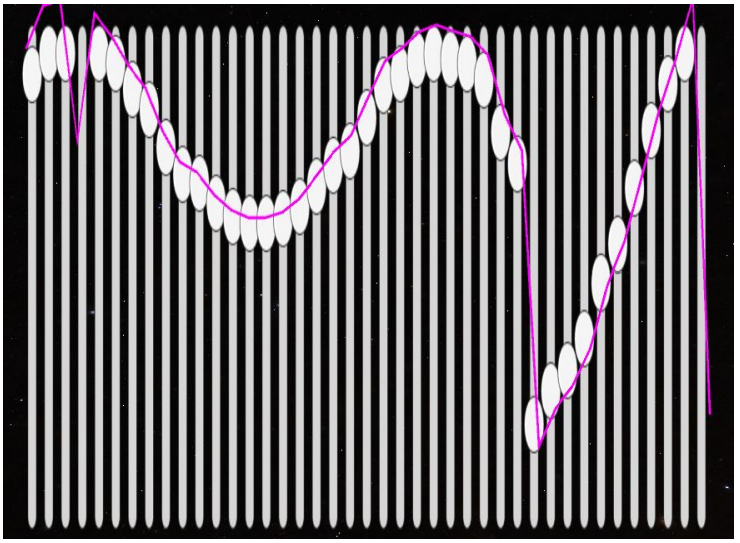




Now that the float values are all set up I had to think of how to edit color values. It was quite hard to find a solution, because I didn't want to use any gestures and wanted to keep the layout as simple as possible. I asked around if there was a possibility to create a 2D color picker that exists out of only one slider / color picker. Unfortunately, I couldn't think of any layout that would use all the colors so I decided to stick with the common color picker.

I decided to make the system procedural generated so that the users don't get overwhelmed by all the values at once and have to adjust complicated values they don't understand yet. After a short time of thinking I thought it might be interesting to create a graph of all the values which the user can edit. Maybe in the future it would also be possible to read values from a given graph or even from pictures so they can customize the planets using a picture they really like and creating a sort of bond with the planet.The user simply touches the graph at a specific point and the graph will set the value of that float.

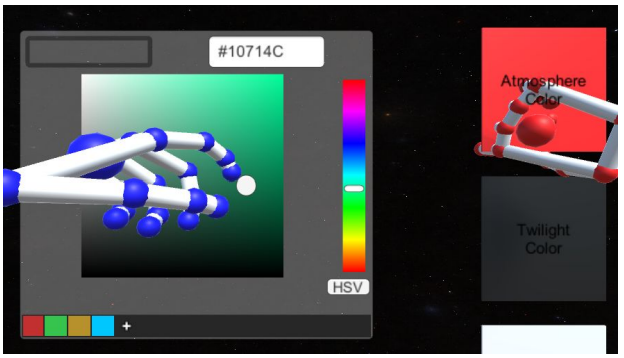
I couldn't find a way how to adjust a lot of values with a single graph using the Leap Motion, so I decided to create a big collider where I can read the position of the user's hand using the center, width and height of the collider. This value is clamped between 0 and 1 to know whether you are at the bottom left or top right. The values got adjusted by creating a slider for each value and setting the value of that slider equal to the height of the user's hand within the collider. I could easily figure out what value I had to edit by dividing the user's position value by the amount of SO_Values in the graph. Basically the user is editing multiple sliders, as you can see on the picture, and creates a lineRenderer based on the values of the sliders.

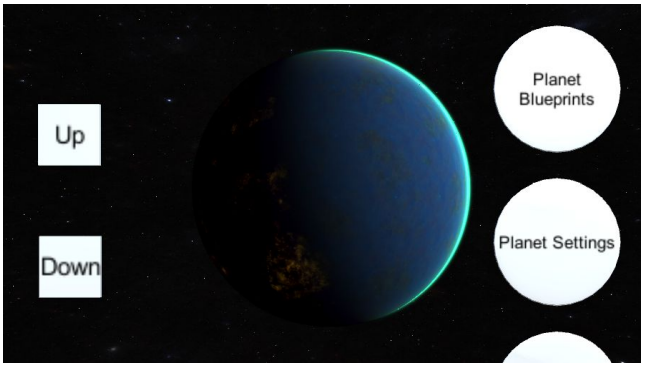


This is the script I've used for the graph.

```
void Start() {
    lowerLeftCorner = new Vector2(transform.position.x - (transform.localScale.x / 2),
                                  transform.position.y - (transform.localScale.y / 2));
    upperRightCorner = new Vector2(transform.position.x + (transform.localScale.x / 2),
                                   transform.position.y + (transform.localScale.y / 2));
    sliderDistance = 1.0f / sliders.Count;
    lineRenderer.positionCount = sliders.Count;
    for (int i = 0; i < sliders.Count; i++) {
        float xPos = lowerLeftCorner.x +
                     (sliderDistance / 2 + i * sliderDistance) * transform.localScale.x;
        float yPos = lowerLeftCorner.y;
        float zPos = transform.position.z - transform.localScale.z / 2;
        lineRenderer.SetPosition(i, new Vector3(xPos, yPos, zPos));
    }
}
void Update() {
    if (colliding) {
        collisionCoordinate = new Vector2(collidingObject.transform.position.x - lowerLeftCorner.x,
                                          collidingObject.transform.position.y - lowerLeftCorner.y);
        collisionInput = new Vector2(collisionCoordinate.x / transform.localScale.x,
                                     collisionCoordinate.y / transform.localScale.y);
        int currentSlider = (int)Mathf.Floor(collisionInput.x / sliderDistance);
        sliders[currentSlider].value = collisionInput.y;
        float xPos = lowerLeftCorner.x +
                     (sliderDistance / 2 + currentSlider * sliderDistance) * transform.localScale.x;
        float yPos = lowerLeftCorner.y + collisionInput.y * transform.localScale.y;
        float zPos = transform.position.z - transform.localScale.z / 2;
        Vector3 graphPointPosition = new Vector3(xPos, yPos, zPos);
        lineRenderer.SetPosition(currentSlider, graphPointPosition);
    }
}
```

After you've edited the graph you would continue to the next part which is the color picking. There would pop up a menu that shows multiple colors that have to be picked by the user. By default they are all black and when the users touch the color box the color picker would pop up. The users can pick their favorite colours for the given values and once they are done with this part you would proceed to the generation. From here on there are some random values that are being set, such as the seed of the continents which is between 0 - 255. The users can edit this later on, but to keep the generation relatively easy I decided to keep these values at random.





After the generation has completed the user is still able to edit the values with instant feedback, but from here on they've set the major part and should be more comfortable using all the variables than at the start.

When the users are finished editing their planet they can export the settings to a JSON file which will be copied to their clipboard. They have to manually save this in a file on their computer, but when they purchased the asset pack they can instantly copy their planet into their project.

Experiment 1:
Test:
Create a UI that is easy to use.

Results:
The Leap Motion itself isn't hard to use, but the way you interact with the environments is a tricky one. It could be the Leap Motion that I've used, but I noticed a lot of jittering in the hand models which resulted in frustration while trying to create a UI for the users. Besides that, I've had the feeling that I wasn't able to hit the sides of the screen with my hands because the Leap Motion wouldn't be able to track my hands anymore. I wasn't able to utilize the full screen and with all the jittering I couldn't put everything close to each other.

Experiment 2:
Test:
Adjusting values with few interactions and no hand gestures.

Results:
By giving the users the idea that they are adjusting values with their hands is very pleasing for them. Even though you use invisible sliders and visualize it with a line makes them feel in control. The toughest part is when you want to adjust a variable just a little bit more than it was before while tracking how much the user has clenched his hand. When the value of your hand is 1 and the value itself is 0, it will jump immediately from 0 to 1. This is pretty annoying but I couldn't think of a way to fix this for the time being. However, the value adjustment seems to work fine and needs little to no practice to use.