Planets N Stuff

Sarah Brown (7765915)
Noah Palansky (7771772)
Max Swatek (7764152)

Overview

The initial idea for our project was to create a realistic looking solar system comprised of a sun, multiple planets and moons, all within a space setting. Along the way we also added in an asteroid cluster, a spaceship, music, and sound effects. Our final project plays like a game allowing users to select one of two modes to explore the solar system with; Flight mode and Planet mode. Flight mode allows the user to pilot a TIE Fighter around the solar system, shooting at moons, diving between gas giants and their rings (Image 2), barrel-rolling, etc. Whereas Planet Mode puts users in a geosynchronous orbit of any planet in the system allowing them to zoom in on a world and observe its rotation, geography, storms, rings, moons with more control and precision than flight mode offers.

Download Project at: https://github.com/Max-Swatek/GROUP_PROJECT

Components

- **Planets**
  - **How they work**

    Early in beginning of this project we stumbled across a blog post (appendix 11) which detailed a good strategy for building planets in Threejs which we used as a foundation for our planet design. Each planet consists of 5 different textures on 2 spheres (a terrestrial sphere surrounded by a 1.5% larger atmosphere). Every texture was eventually resized to be in powers of two in order to eliminate webGL warnings. To give the appearance of an active planet the atmosphere and terrestrial base are both rotated at different rates to simulate days and weather.

    3 textures make up the planets terrestrial sphere;

    1. Surface projection (just a normal map). We found images online that we thought would fit our vision and decided to put our own creative adjustments to them on Photoshop CC. It was important to us that as the images were being wrapped onto the objects there would not be an obvious seam visible. This entailed multiple reworkings until they fit our requirements. One technique that proved useful for this particular issue was mirroring one half of the image onto the other. Additional creative choices we made on the textures were things like changing the colour palette, adding storms, and rivers/lakes. Surface storms were achieved by using the distort tool found inside of Photoshop's filters menu. The edges of the storms were smoothed out further using the smudge and blur tools.

2. Bump map which is based off of the surface projection but colour coded with information that gets passed to a shader (brighter areas on the map will appear more raised than darker areas) allowing us to give the appearance of very complex shaped surfaces with no additional polygons. Attempting to make these from scratch was quite challenging as we didnt want it to appear too unnatural. Initially the bump maps we made didn't really match the textures we had imagined for the normal maps. The terrain would either appear too flat or would have a really awkward terrain that looked very cartooney. It took many reworkings playing with the grayscale colours in order to achieve the look we were aiming for.

3. Specular map which is also based off the surface projection, it works very similar to the bump map in that it is colour coded with information that a shader will interpret (brighter areas on the map will reflect more specular light than darker areas). This effectively meant that land masses would be darker than the oceans and lakes.

Along with 2 textures based from (Appendix 11) make up the planets atmosphere;

4. Cloud projection map (similar to the surface projection except with clouds instead of continents).

5. Cloud transparency map (similar to the bump or specular map except that darker shaded areas are treated with higher transparency)

- **Features**

1. Moons
   We make the moons recursively in our planet making function, as our moons are just simple planets (without atmosphere, rings or moons of their own). Most of our planets have moons, the gas giants have many moons. In total our solar system has around 20 moons. With so many moons it became important to prevent all the moons from looking the same while at the same time we didn't really want to make 20 different moon textures. So we made 5 different copies of the same texture based off of Nasa's surface projection of Mimas (one of Saturn's cooler moons) that we edited with different colours and axis flips. Each moons size is also pseudo randomly generated such that no two moons are the same.

2. Orbits
   All of our planets and moons orbit a parent in a circular motion. We decided on circular orbits over elliptical (more realistic) orbits for 2 reasons;
   Reason 1: Circular movement is so much easier and looks almost as good (a user wouldn't be able to tell whether or not it's circular or slightly elliptical)
   Reason 2: If moons were on elliptical orbit they could potentially intersect with each other (we never made any collision detection so they would pass through each other and that would look really bad)

   So we came up with a very simple way to make circular orbits, every planet is the child of a tether point that is the child of/centered on the Parent. For planets the parent is the sun (or just the origin), for moons their parent is a host planet. Now all we have to do to make an orbit is rotate that tether point. Also the rate of the rotation is determined by an inverse square of the distance between the planet/moon and its parent (that's how it works in real life!) for example if you're looking at a gas giant you will notice that the moons closer to it will orbit quicker than moons farther away.

3. Rings
   We kind of lucked out on this one. Another person online happened to make a Threejs geometry called "THREE.XRingGeometry" specifically made for creating flat rings with holes in the middle. The geometry let us create a small ring segment for each planet and apply it uniformly around a planet.

- **Skybox**
  We spent a very, very long time trying to find a high-res skybox that we could borrow for this project. Unfortunately, there weren't any out there that fit our needs. There was, however, a convenient program specifically built for generating outer-space sky boxes. The program, Spacescape, was targeted towards advanced users looking for highly customizable, multi-layered procedurally generated landscapes - needless to say we didn't fit the target demographic. However, after spending a few hours of playing around with it, we were able to generate a skybox that we think is absolutely breathtaking.

- **Lens Flare**
  This was one of those things that we thought would be incredibly difficult to put together, but in reality, it was incredibly easy. We just followed the guide on the Threejs docs (appendix 10) and were able to do it pretty quickly.

- **Point mesh / space rocks**
  The space rocks are actually a point mesh where each point is wrapped with a rock texture. The space rocks aren't actually spheres - they're cubes. The rock texture just gives them transparent corners. Unfortunately, this hack is noticeable when a rock intersects with the sun or the rings of a planet. We could fix this by swapping them with actual sphere geometries, but we decided that it wasn't worth it because of the performance hit (⅔ of us have trouble running it as is). Each rock moves independently of each other rock along a randomly determined linear path. The rocks were placed in a plane to simulate an asteroid belt - about 50,000 x 50,000, x 3750 units. When a rock reaches the bounds of the plane, it teleports (overflows) to the other bounds in the same axis so that it can continue on its path without ruining the illusion of an asteroid belt.

- **Laser beams**
  We decided to use "threex.laser" (appendix 7) extension for our lasers, but had some small issues setting it up. The creator(s) set up their examples in a different way than any other Threejs code we'd seen, which made them very difficult to read. Once we deciphered how their code worked, we were able to setup the lasers fairly easily! Rather than shooting one continuous beam (boring) we opted to have our TIE fighters shoot a series of small lasers (fun!). The lasers are a child of the ship, which made it easy to make them shoot straight. Unfortunately, this came with a few bugs. The first one, was that laser beams move with the ship after firing. We fixed this by deciding that our ship shoots remote controlled laser beams. Another issue was that the beams would go on indefinitely, even after they were too far away to see (causing some obvious inefficiencies). When we put in some code to stop them after a set amount of time, we realized that their position would still stay relative to the ship. This meant that if you shot a bunch of times and then zoomed out, you'd see a big field of laser beams (image 1). We couldn't think of a clever excuse to get out of fixing this, so we instead decided to remove them from the scene after a set amount of time.

- **Implementing Blender models into Threejs**
  Ever try to do something that should be super simple but instead makes cry and question your life choices? That was our experience with Blender! It attempted to melt my computer, and took 25 minutes to export a model. After exporting, we realized that the only way to get Threejs to recognize our model was to get all 20ish export settings exactly right. It took about 6 tries multiplied by 25 minutes. It was so bad that we had a serious conversation about making our ship a flying cube. Needless to say, we don't like Blender - however, it was instrumental in allowing use of an actual spaceship model in our project. It was also a pretty good learning experience, and taught us a lot about why HCI matters.

- **HUD and main menu**
  We wanted to be able to draw on top of the scene in 2D. Unfortunately, Threejs had one too many dimensions for our tastes. Instead, we implemented something at the recommendation of Stack Exchange user “Taio” (Appendix 1). The basic idea on this is that we create a second scene that renders an orthographic camera on top of the original. We then create a HTML 5 canvas and use it as a texture map for a material. Finally, we use the material to make a PlanGeometry that takes up the entire viewport. The end result is that we can draw fixed content to a canvas that is rendered on top of our scene. This is used by the HUD to display speed bars in flight mode, and to display a main menu screen while the rest of our scene loads. We also learnt a lot about how to use a canvas in HTML / JS. This came with a lot of challenges - for example, we had a lot of trouble loading a picture in to use as a background. As it turns out, loading an HTML element into JS is an asynch operation, and we had to use a callback to wait for loading to complete before it could be used. There were a lot of other similar challenges along the way, but at the end of the day, implementing this was incredibly education.

- **Music and Sound effects**
  While we were working on our project we realized it would feel more complete with a space themed soundtrack. We decided that a song from the video game Stellaris would match the simulation we had created. We ultimately decided that the track called Faster Than Light was a good fit and we took the youtube video link (listed in the appendix) and placed it into a program that converted it into an mp3 file. Once the file was placed into our project folders we were then able to add in the code in the HTML file allowing the song to play upon launch. We also included a blaser sound effect in flight mode so that the shooting would be more engaging to the user.

Controls:

- Main Menu:
    - Select a mode using the up and down arrow keys and the enter button
- Flight mode:
    - Shift: Increase speed
    - Control: decrease speed
    - S: pitch up
    - W: pitch down
    - A: Yaw left
    - D: Yaw right
    - Q: Roll left
    - E: Roll right
    - Click and drag: adjust camera view
    - Scroll: adjust camera zoom
    - Escape: return to main menu
- Planet mode:
    - P: switch between planets
    - Click and drag: adjust camera view
    - Scroll: adjust camera zoom
    - Escape: return to main menu
    - Sources:

Notes:

- You may not be able to see the ship in flight Mode if the camera is zoomed out too far. If you lose sight of the ship just use scroll to zoom in.

- In Flight mode will experience some lag as it first renders the gas giants with their many moons, however by the time you fly to them performance should be nominal.

- In the system all planets begin aligned with each other, this is for ease of navigation, as you explore the system, planets will slowly disperse as their orbital period depends on their distance from the sun.

- The moons of the gas giants also appear close together in the beginning, but much like the planets they will disperse as you explore the system.

- In Flight mode you will not be able to see the lasers if you adjust the camera to be directly behind the ship. The default for the camera is slightly raised to prevent this.

Running Instructions:

- To run this project open the directory with a local host (just like assignment 2) and open file PLANETSnSTUFF.html

Appendix 1: HUD implementation idea:
https://stackoverflow.com/questions/12667507/drawing-ui-elements-directly-to-the-webgl-area-with-three-js

Appendix 2: xRingGeometry for giving planets rings
http://cdn.rawgit.com/bubblin/The-Solar-System/master/js/shared/xRingGeometry.js

Appendix 3: Moon1 texture (was colourized and altered further in photoshop for Moon2-4)
https://photojournal.jpl.nasa.gov/jpeg/PIA11673.jpg

Appendix 4: Gas Giant texture (altered further in photoshop)
https://img00.deviantart.net/d6f5/i/2010/233/0/4/upsilon_andromedae_c_texture_by_enderion.png

Appendix 5: used understand how to make a heads up display to overlay ship speed
https://codepen.io/jaamo/pen/MaOGZV

Appendix 6: Tatooine texture (altered further in Photoshop)
http://frederickhiggins.com/celestia/index_htm_files/8505.jpg

Appendix 7: Three JS laser plug in
https://github.com/jeromeetienne/threex.laser

Appendix 8: Tie fighter model
https://www.blendswap.com/blends/view/86192

Appendix 9: Pokemon World Map
https://orig00.deviantart.net/efdb/f/2011/234/5/f/a_c_x____world_map_by_project_phoenix-d46lkql.png

Appendix 10: Lens Flare
https://threejs.org/docs/#api/objects/LensFlare

Appendix 11: how to make earth in webgl (blog post)
http://learningthreejs.com/blog/2013/09/16/how-to-make-the-earth-in-webgl/

Software Used:
Photoshop CC
Spacescape
Github
Atom
Web server for chrome
Google Chrome

Image 1:

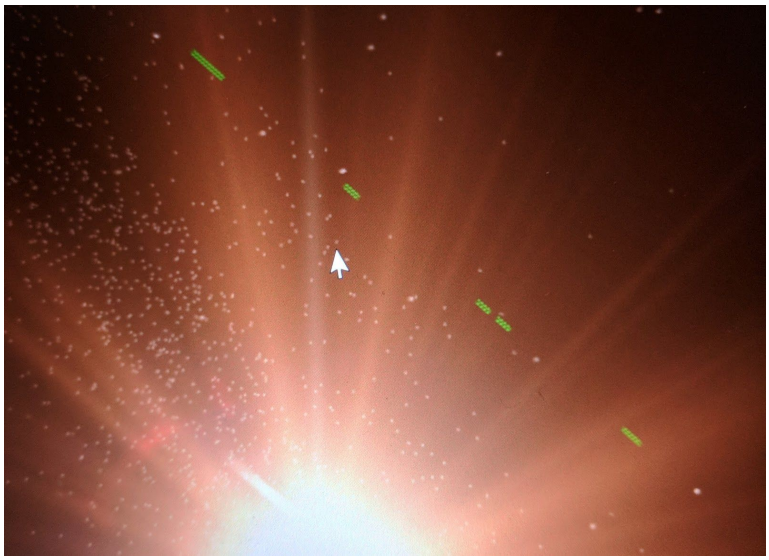

Image 2: