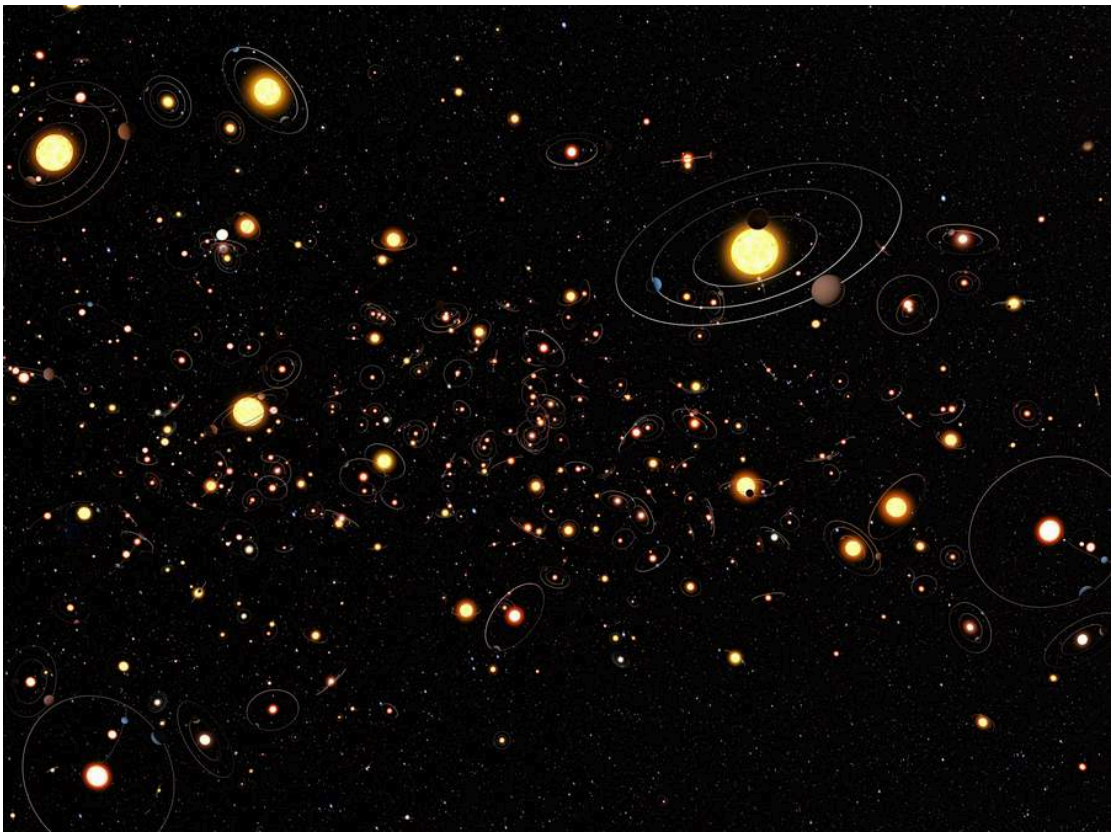

A study into the procedural generation of multiplanetary extrasolar systems

George Batchelor
Level H i7907585
Innovations Project Report

How do you organize a party in space?
You planet

Table of Contents

"So far, astronomers have found more than 500 solar systems and are discovering new ones every year. Given how many they have found in our own neighbourhood of the Milky Way galaxy, scientists estimate that there may be tens of billions of solar systems in our galaxy, perhaps even as many as 100 billion."
(Rayman, M. 2013)

1.0 Abstract

The aim of this project is to produce a program that will procedurally generate interactive solar systems. The solar systems will contain stylised planets that are visually representative of their probable atmosphere based on various deciding factors.

This paper contains research into extrasolar systems and exoplanets, this includes information about the more common types of planets that have been discovered, as well as methods of calculating planetary orbits. The paper also contains details about how the program was made; the procedural generation methods and shaders.

2.0 Introduction

There have been a number of attempts into procedural planet generation, many with more realistic goals. An example of a more realistic approach can be found in figure 2.0.1. Here the creator has focused on textures for the planet surfaces, this gives a very interesting, realistic effect, but is also computationally very expensive and not appropriate for a constantly updating program.

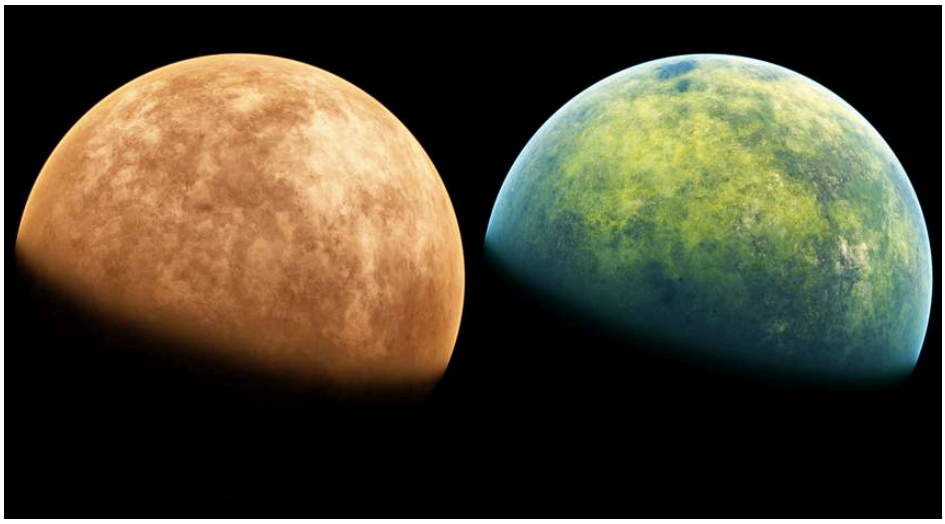

Fig. 2.0.1 A realistic approach to procedural planet creation *(nvseal, 2009)*

For this project, my goal is to generate planets procedurally at run-time in a game engine, so that users can easily run the program repeatedly without waiting and study the different types of planetary systems that could exist. Doing this requires a different technique, as the focus is on efficiency and speed. In terms of similar projects, the 2008 video game, *Spore* is the most interesting example. Spore extensively uses procedural generation throughout the game, including generating full planetary bodies (Fig. 2.0.2). The game has numerous 'types' of planets that are generated; rocky planets, lava planets, ice planets, etc. And there are tools that allow artists to further sculpt details to achieve desired effects. Where Spore has more creative motivations, my objective is to create something that is both appealing from a visual standpoint, but also interesting from a

scientific perspective. With this in mind, I will be researching exoplanets currently known to be in orbit of other stars and using this data to inform an average probable type of planets. The program could be used to test the likelihood of finding a terrestrial Earth-type planet, or to visually represent data in a way that is more accessible to people with limited knowledge of the subject. I believe the innovative aspect of this project is in the marriage of both the visual side and the scientific, in that real world facts and statistics are used to influence what appears to be a more visual piece of software.

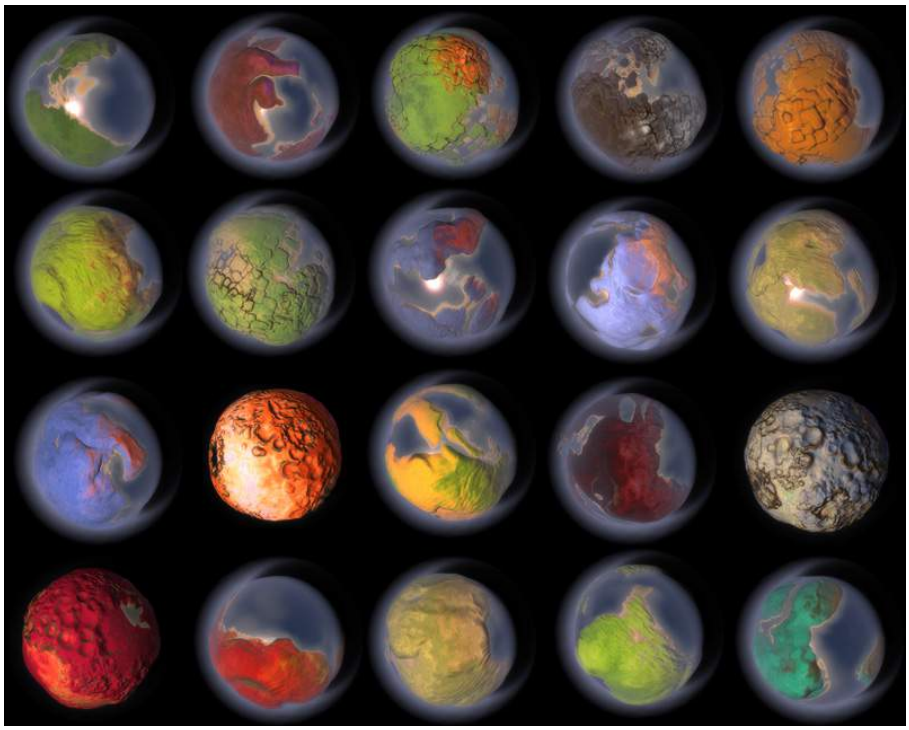

Fig. 2.0.2. Examples of procedural planet generation in *Spore (Quigley, 2009)*

Personally, I begin this project with limited knowledge of astronomy, but mainly with the intention of learning about procedural generation. Procedural art is something I've never explored before, and it will involve the use of the Unity3D game engine and the C# programming language, both of which are relatively new to me, as well as learning about shader writing, another area I have no experience with. I will be tackling the problem in two parts, firstly the procedural generation of planet, and secondly placing these within a solar system.

The first half of this report will describe the astronomy research that has been done for the project. This will include information about the types of planets that have been discovered and their typical atmospheres and orbits. The second half will be about how I procedurally generate planets to be used in the system. Finally I will describe how these elements come together and give a summary of the project.

3.0 Astronomy Research

3.1 The Methods of Detecting Extrasolar Planets

Detecting exoplanets is extremely difficult. Because of their distance from us, planets must be detected indirectly. There are two main ways this is done: using Radial Velocity Methods and Transit Methods. These involve searching for tiny wobbles in the motion of stars by detecting periodic redshift and blueshift, and detecting periodic drops in the brightness of a star, respectively. Periodic changes indicate a planet has passed in front of a star. The amount of change and the time between changes can be used to calculate the minimum mass of a planet as well as the radius, speed and shape of its orbit. These factors combined can give an approximation of the atmosphere of a planet.

3.2 Types of Exoplanets

"There is now clear evidence for substantial numbers of three types of exoplanets; gas giants, hot-super-Earths in short period orbits, and ice giants." (NASA, 2013)

There are many types of exoplanets that have been discovered, but the majority can be categorized into the following:

- Gas Giant (41%)
- Hot Jupiter (45%)
- Super Earth (9%)
- Terrestrial (2%)
- Other (3%) (Planets of unknown composition)
  (PlanetQuest, 2014)

These are the types of planets that will be generated in the program, different planets depending on their distance from the star, mass and other factors. I will describe these types of planets in further detail in section 5.0.

3.3 Stars

There are several types of stars in the galaxy that have been studied with different masses, surface temperatures, even multiple stars in orbit of each other. For the purpose of this project, the stars will remain similar, with properties that can be likened to the Sun, as this is a common average (Fig. 3.3.1).

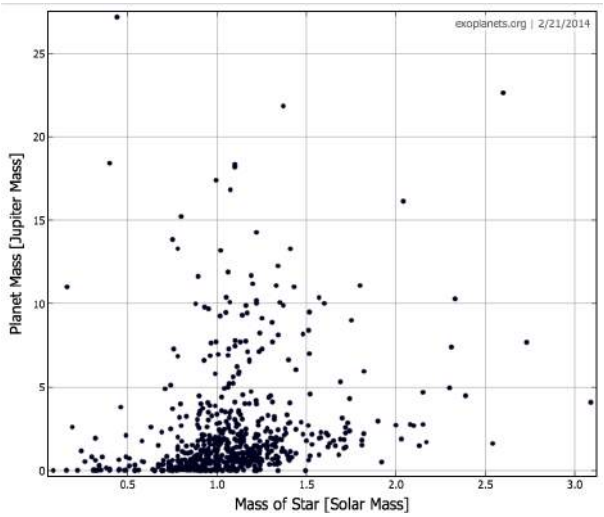

Fig 3.3.1. Masses of stars that have been documented compared with the planets that orbit them (exoplanets.org, 2014)

3.4 Planetary Orbits

Kepler's first law states that *"All planets move in elliptical orbits" (Nave, 2000)*. This is defined by a semi-major axis, a semi-minor axis and an orbital eccentricity (Fig. 3.4.1). The eccentricity is related to how circular an orbit is, a perfect circle having an eccentricity of zero and an ellipse following: 0 < eccentricity < 1.

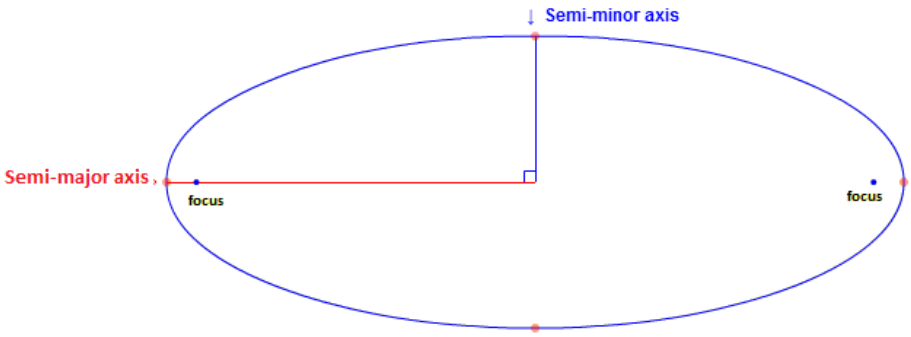

Fig 3.4.1. Semi-major and semi-minor axes of an elliptical orbit. (Wikipedia, 2014)

Information about exoplanetary semi-major axes and eccentricity can be found from *The Extrasolar Planets Encyclopaedia*, and given these two pieces of information we can calculate the semi-minor axis.

In order to get a likely result for procedural generation of orbits, we can observe the distribution of exoplanets semi-major axes. Figure 3.4.2 is a histogram presenting this. The planets of our Solar System are marked for comparison.

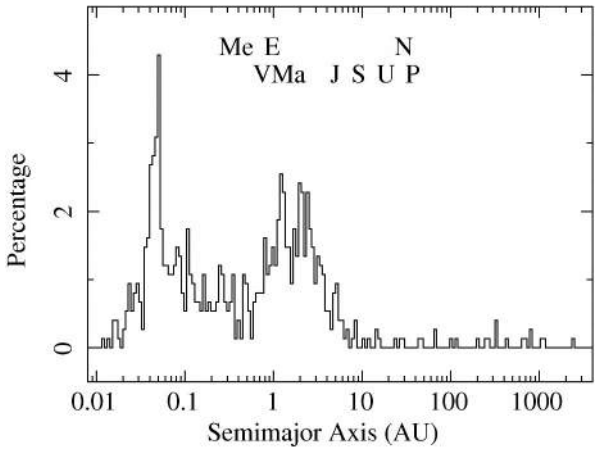

Fig. 3.4.2 Histogram representation of the semi-major axes distribution of known exoplanets (Yaqoob 2011)

We can see from this diagram that there is a strong peak in planets with a smaller semi-major axis between around 0.03 and 0.06 AU. This corresponds to many close-in planets such as Hot Jupiters that have been discovered. There is a second noticeable peak around the 1 AU distance (the distance of Earth). The reason behind this form of distribution remains largely unknown by astronomers.

Data can be found for both the semi-major axis and eccentricity of exoplanets. Comparing the two tells us that there isn't a strong relationship between a planets semi-major axis and its eccentricity (Fig 3.4.3). What it does give us is a weighted average of planetary eccentricities though; planets are slightly more weighted to lower values of both eccentricity and semi-major axis.

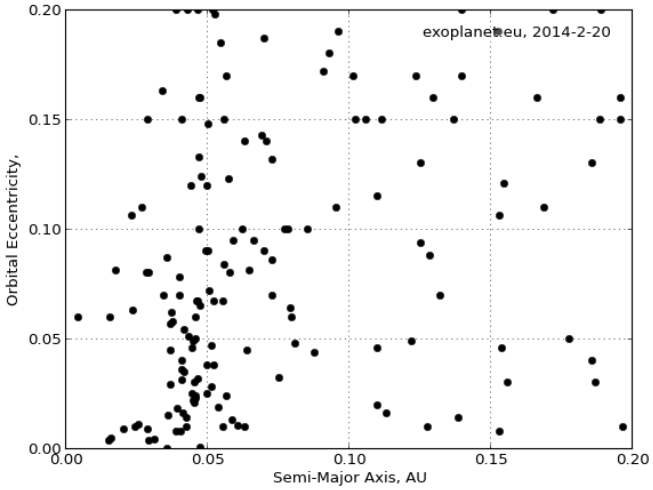

Fig. 3.4.3 Scatter graph comparison of semi-major axes and orbital eccentricity of known exoplanets (exoplanets.eu, 2014)

I wrote the following function to calculate a float value with a weighted average, which multiplies the result to the power of a value gamma. This is effectively applying a gamma curve to the result. Higher values of gamma will produce a greater likelihood of number closer to the minimum boundary, and lower values the opposite. Using this I can generate a random number that is still quite varied but slightly favored in one range.

```
float weightedRand(float min, float max, float gamma)
{
    float offset = max - min;
    return (min + Mathf.Pow(Random.Range(0f, 1f), gamma) * offset);
}
```

The weighted random function is used to calculate a likely eccentricity. Now we can calculate the semi-minor axis using the following formula:

$$b = a\sqrt{1 - e^2},$$

Where b is the semi-minor axis, a is the semi-major axis and e is the eccentricity.

Finally, to turn these results into an elliptical orbit, I follow the formula for calculating an ellipse:

```
float x = origin.x + (semiMajor * Mathf.Cos(alpha));
float y = origin.y + (semiMinor * Mathf.Sin(alpha));
```

Where x and y are the positions of the planet, origin is the position of the star, and alpha is a value which is incremented every frame in order to move the planet.

To test this out, keeping the semi-major axis the same, I ran the program several times to see how the orbit would differ. The resulting orbits are super-imposed on top of one another in figure 3.4.4.

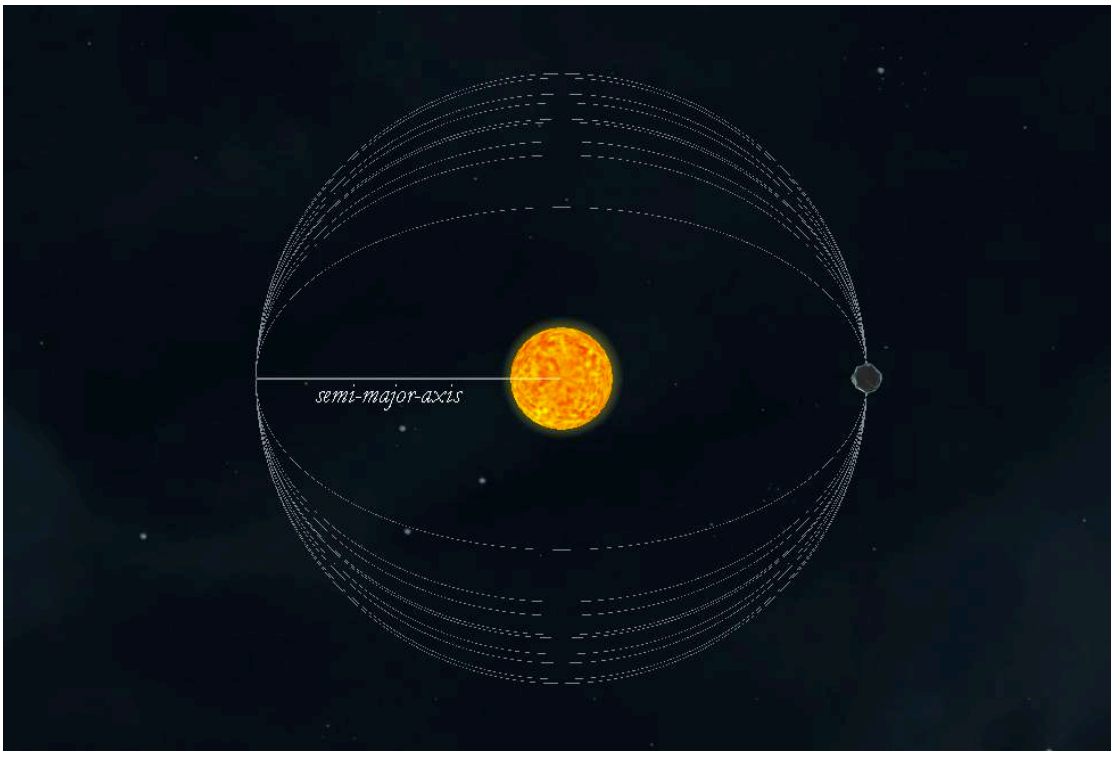

Fig. 3.4.4 Combined results of gamma-weighted planetary orbits

We can see the weighted function producing orbits that are more circular, and therefore have eccentricities closer to 0. This is in keeping with the more common eccentricities found in exoplanets. These results were produced with a gamma value of 0.7, which still generates reasonably varied outcomes. With a greater gamma value of 0.95 for example, the orbits would usually be almost circular.

In the program, planets will orbit along the same 'orbital plane', which is the geometric plane in which the orbit lies. This could vary slightly in reality, but for simplicity has been kept the same here.

3.5 Orbital Periods

Kepler's Third Law states:

"The square of the orbital period of a planet is proportional to the cube of the semi-major axis of its orbit." (Nave, 2000).

Which means that we can calculate a planet's orbital period, or the time it takes to orbit a star, based on its semi-major axis. Figure 3.5.1 is drawn from data found on The Extrasolar Planets Encyclopaedia, and it demonstrates Kepler's law. We can see that generally as a planets semi-major axis increases so does its orbital period. This makes sense as both the distance it has to travel becomes greater and the gravitational attraction from the sun becomes weaker.

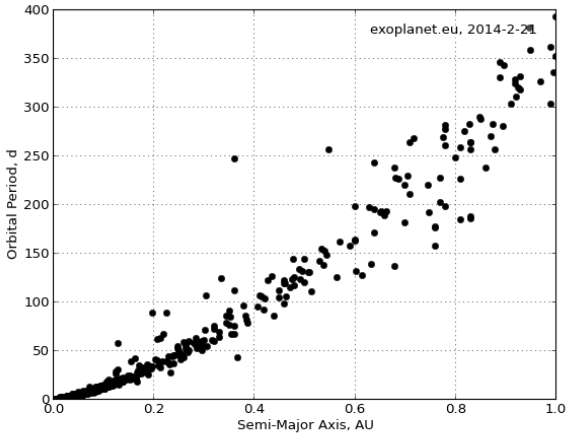

Fig 3.5.1 Scatter graph comparing semi-major axis and orbital periods of known exoplanets (exoplanets.eu, 2014)

The equation used for calculating a planet's orbital speed is as follows:

$$v_o \approx \sqrt{\frac{GM}{r}}$$

(wikipedia.org, 2014)

Where G is the gravitational constant, M is the mass of the star and r is the distance between the planet and the star. This formula is for circular orbits, but as the formula for

elliptical orbits is much longer and more complex it will be too expensive computationally so the circular orbit function is used.

The next thing to calculate is planetary mass. Fig 3.5.2 is drawn using data about known exoplanets. We can see a notable peak of planets that are similar in mass to Jupiter, or just larger. The mean value is 2.2758 Jupiter masses. This is the figure we can weight the average when generating planetary masses. There isn't any strong relationship between planet mass and semi-major axis, so these will be calculated irrespective of one another.

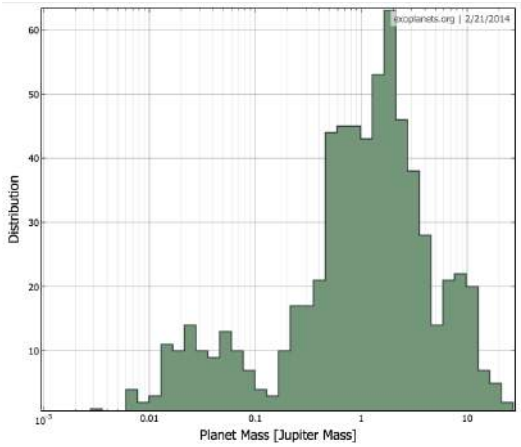

Fig 3.5.2. Histogram of distribution of exoplanets masses (in Jupiter Masses) (exoplanets.org, 2014)

Next we must convert Jupiter masses into solar masses, as this is the unit used in the function. 1 solar mass = 1047.92612 Jupiter masses. A mass is calculated using the weighted average function that will be between 1 and 10, with a weighted probability of being around 2 Jupiter masses. This is then divided by 1047 to roughly convert into solar masses. We can now have planets moving in orbit at speeds relatively accurate.

4.0 Procedural Generation of Planets

4.1 Starting Point

The planets in the program are generated procedurally at run-time. For a planet, the obvious starting point is with a sphere. Using the primitive sphere available in Maya, there tends to be pinching of vertices at the poles; this leads to problems when editing geometry procedurally. There are several different techniques for dividing up a spherical surface. Figure 4.1.1 shows a comparison between four different tessellations, with the tessellation marked 'ECP' being the default primitive sphere tessellation.

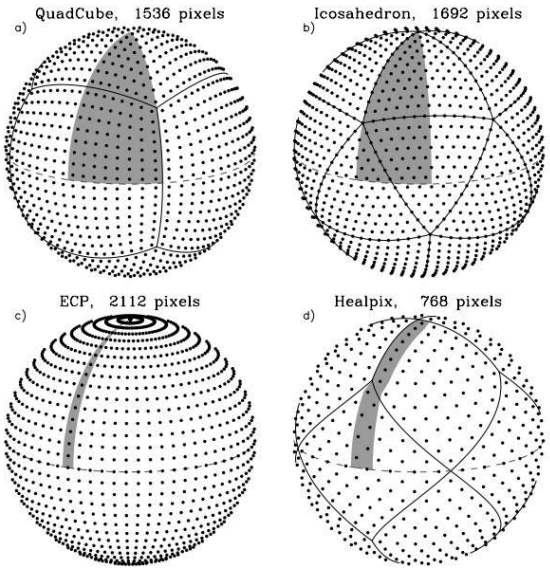

Fig. 4.1.1 Comparison of different types of spherical tessellation (Górski et al, 2004)

My aims for the project are speed and simplicity, for this reason I chose the 'quadcube' method. To implement this a cube is divided into a certain number of subdivisions. I found 64 subdivisions across the x, y and z-axes yields efficient results with an appealing level of detail. By normalizing each surface point it is projected outwards and turns the cube into a sphere. Figure 4.1.2 demonstrates the result.

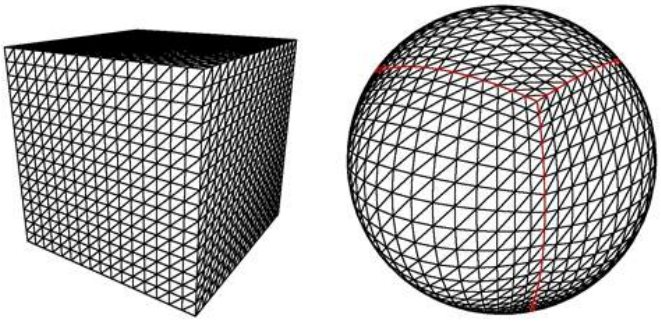

Fig 4.1.2. Regular cube and 'quadcube', its spherical projection. (Wittens, 2009)

With this as the basic framework, we can then apply functions to affect the shape of the mesh.

4.2 Perlin Noise

Noise is basically a seeded random number generator. Random numbers are generated and by smoothly interpolating between them we can create a less harsh kind of noise. By assigning these numbers to something visual, for example the value of a pixel, we can use

it practically. Adding numerous variations of noise together gives Perlin noise, an example of this in 2D can be found in figure 4.2.3.

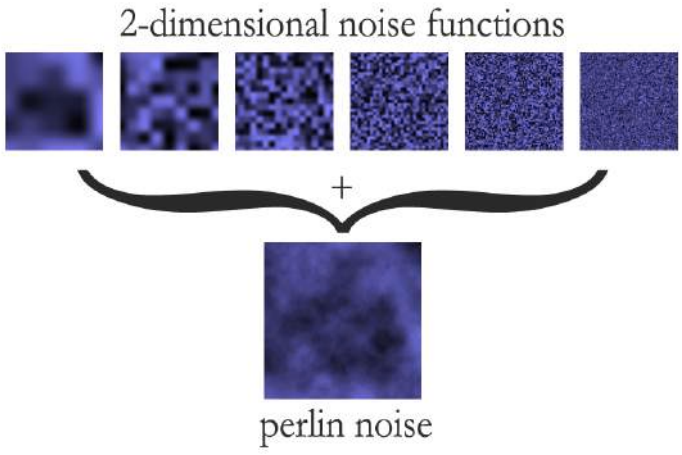

Fig. 4.2.3. Two-dimensional noise

To see how this would translate into 3D in a computer, I used the value generated by the noise to move the vertices on a subdivided plane along the y-axis to create a height-map (Fig. 4.2.4). The Perlin noise functions used are adapted from Ken Perlin's example available online *(Perlin, 1997)*.

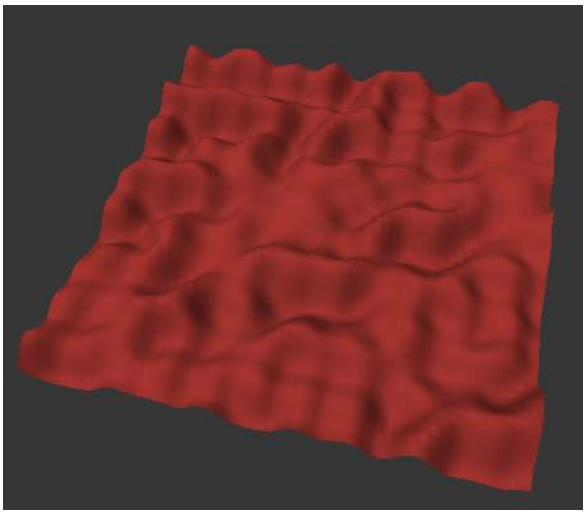

Fig. 4.2.4 3D height map terrain generating with 2D Perlin noise

Perlin noise can be generated for as many axes as desired. For a spherical, planet type object, three-dimensional noise is used, requiring a trilinear interpolation of the input values, in this case; the x, y and z co-ordinates of each vertex.

In pseudo code, the formula for planet generation is essentially the following:

FOR every vertex in the mesh:
- Create a float value using 3d noise, reading in the x, y, z position of the vertex
- Multiply the current vertex position by this value

A simplified version of this from the code:

```
for(int i = 0; i<vertices.Length; i++)
{
    Vector3 vertex = vertices[i];

    noise = perlin.Noise(vertex.x, vertex.y, vertex.z);

    vertex.x *= noise;
    vertex.y *= noise;
    vertex.z *= noise;

    vertices[i] = vertex;
}
```

Which generates a very blob-like deformed sphere (Fig. 4.2.5). Interesting, but not very planet-like.

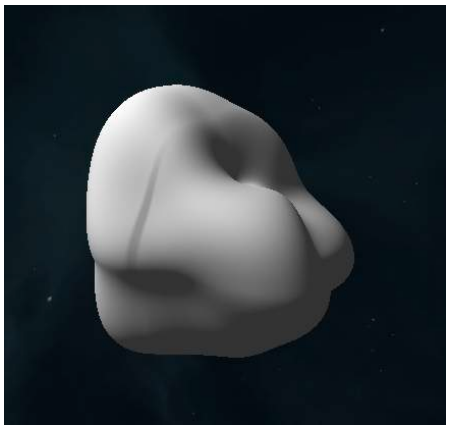

Fig. 4.2.5 Simple Perlin Noise

4.3 Fractals

There are many ways we can combine noise-functions to produce different results. One of these ways is known as *Fractional Brownian Motion (fBm)*. Each successive noise function added in called an octave. The idea of fBm is to sum together numerous octaves of noise, each with increased frequency (the gap between numbers generated) and decreased amplitude (the range of numbers generated). An example can be seen in figure 4.3.1.

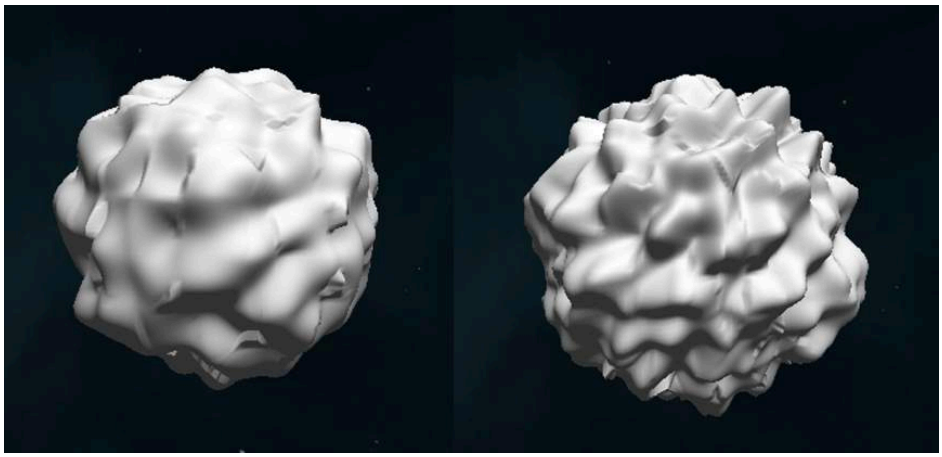

Fig. 4.3.1. Fractional Brownian Motion with 4 octaves (left) and 5 octaves (right)

One characteristic of fBm is, as described in *Texturing and Modelling: A Procedural Approach;* it is *"statistically homogenous and isotropic." (Musgrave, 2003)* meaning it is the same everywhere and the same in all directions: quite uniform and uninteresting. For something more visually interesting, we require *"heterogeneous fractal functions"*.

This is where *multifractals* are required. Their key feature is that they are not the same everywhere, so can represent things like mountains and valleys as their dimension varies with locations. Where fBm could be described as using an *additive cascade*, multifractals use a *multiplicative cascade*, so instead of adding noise of higher frequencies together, we multiply. See figure 4.3.2 for an example of this.

Fig. 4.3.2 Multifractal Noise

By initializing the noise functions with a set seed value, the results will be the same every time which means the features of the planet can be edited in real-time and tweaked easily. These different noise functions I will use as the base for each type of planet, changing values for different atmosphere types.

4.4 Shaders

I researched different methods for adding colour to the planets. Generating a texture for the planet proved to be too costly in terms of computation time, because it has to calculate the colour value for each pixel of the texture and if the planet is being edited in real-time it can't be updated without having an extremely low resolution texture. The best method I found for real-time updates of colour values is to use a shader.

Using a surface shader in Unity, the vertex data can be accessed. By calculating the distance each surface point is from the center of the object using the following code, we can check certain heights and interpolate between colours based on this.

```
float distanceFromCenter = distance(_CentrePoint.xyz, IN.worldPos);
```

There are several float value parameters that correspond to different levels, each level then has a colour assigned to it. There is also a 'water level' parameter, which creates the appearance of water at the lowest level with a solid colour. The interface for this can be seen in figure 4.4.1.

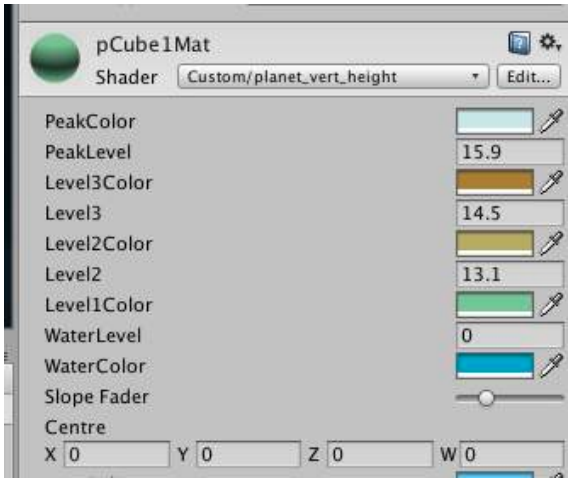

Fig. 4.4.1 the parameters for the planet shader

Now the program can generate a wide variety of stylised planets, which can be used to illustratively represent different types and atmospheres of real exoplanets. Below (Fig. 4.4.2 and Fig. 4.4.3) are some early tests using vertex height shader.

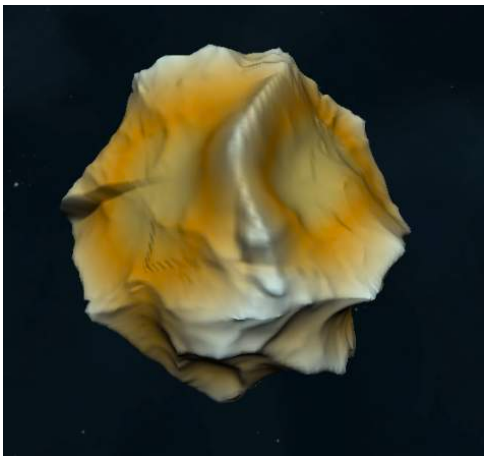

Fig. 4.4.2 Ridged multifractal planet with no water and the appearance of snow-capped 'peaks'

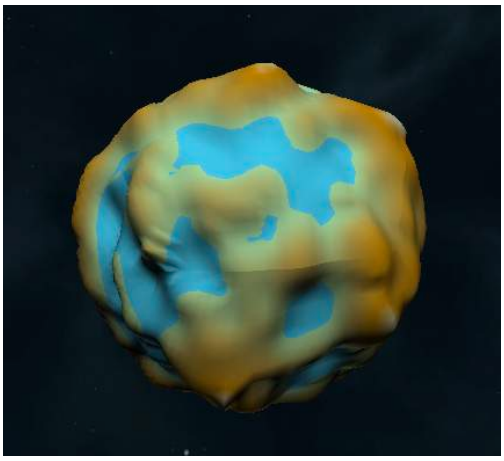

Fig. 4.4.3 fBm planet with a water level introduced

5.0 Types of Planets

With the framework for generating planets in place, I defined some parameter ranges that will create a certain 'type' of planet. I created a type of planet to represent each atmosphere by tweaking the parameters of different noise functions. The interface for this can be seen in figure 5.0.1.

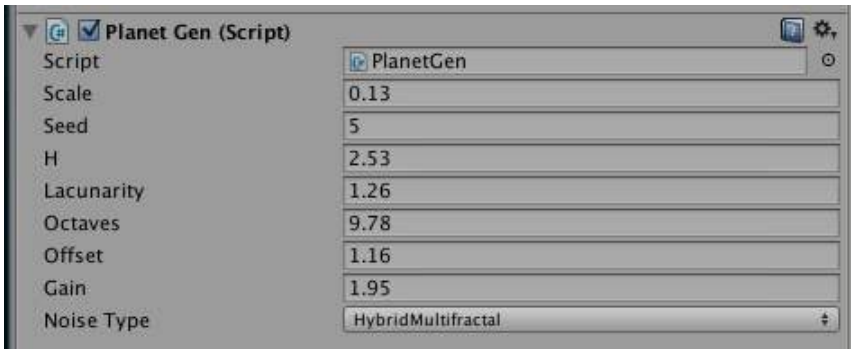

Fig. 5.0.1 The parameters for the noise functions in Unity

Each planet will be a stylised version of the confirmed types of exoplanets in existence. So for instance, an 'ice giant' will be represented by a planet with an icy appearance, and so forth.

The types of planet in the program are:

- Gas giant
- Ice giant
- Hot Jupiter
- Terrestrial
- Super Earth

The seed value is then randomly generated for each planet, which is what changes the appearance every time.

5.1 Gas Giants

Gas giants are large planets that are not primarily composed of rock or other solid matter. Examples in our solar system are Jupiter and Saturn. They typically have distinct bands of cloud layers stretching across their surface, caused by strong winds on the surface.

To achieve the effect of cloud layers, the functionality of the program has to be extended. Creating a sine function of the y co-ordinate of the vertices and applying this to a multifractal noise function gives a striped effect. The amount of 'stripes' is adjustable by changed the number of noise octaves. By using this in conjunction with tweaking various other noise parameters, we can create Gas Giants (Fig. 5.1.1).

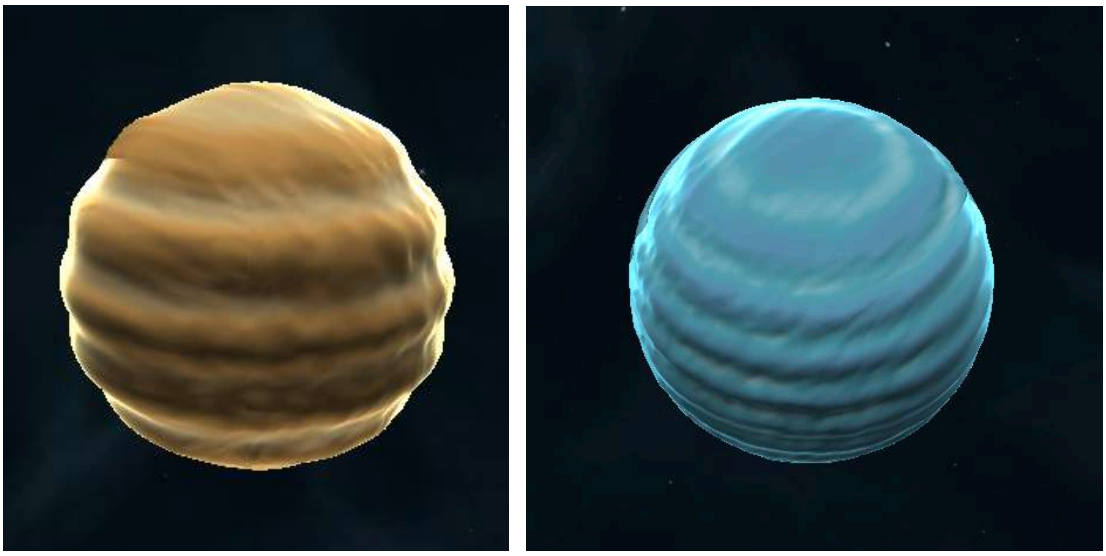

Fig. 5.1.1 Gas Giants with different colour variations

5.2 Ice Giants

These planets are technically a Gas Giant composed of less volatile materials, the term 'ice' doesn't actually refer to the surface being made of ice, but is more a reference to the gases they consist of being of a lighter consistency than other gas giants. Uranus and Neptune are considered Ice Giants. These are represented as being made of ice in the program, but this physical representation is purely metaphorical. Some examples can be seen in figure 5.2.1.

The Ice Giants are generated using ridged-multifractal noise and blue/white colour values for different heights. The shader has also been modified to include rim lighting with a blue colour to highlight the edges. The main addition to the shader code to include rim lighting is:

```
half rim = 1.0 - saturate(dot (normalize(IN.viewDir), o.Normal));
o.Emission = _RimColor.rgb * pow (rim, _RimPower);
```

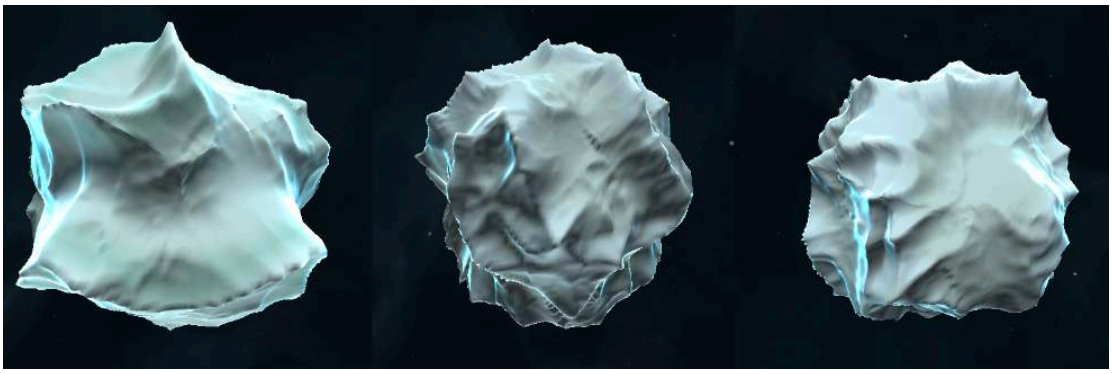

Fig. 5.2.1 Ice Giants generated using multifractal noise

5.3 Hot Jupiters

The difference between Hot Jupiters and regular Gas Giants is that Hot Jupiters have very high surface temperatures because they orbit very close to their parent stars (between 0.015 and 0.5 AUs). Gas Giants orbit further away causing lower surface temperatures. In the program, the parameters for the model generated are similar to that for Gas Giants, but the colours are altered to give the impression of a hotter surface (Fig. 5.3.1).

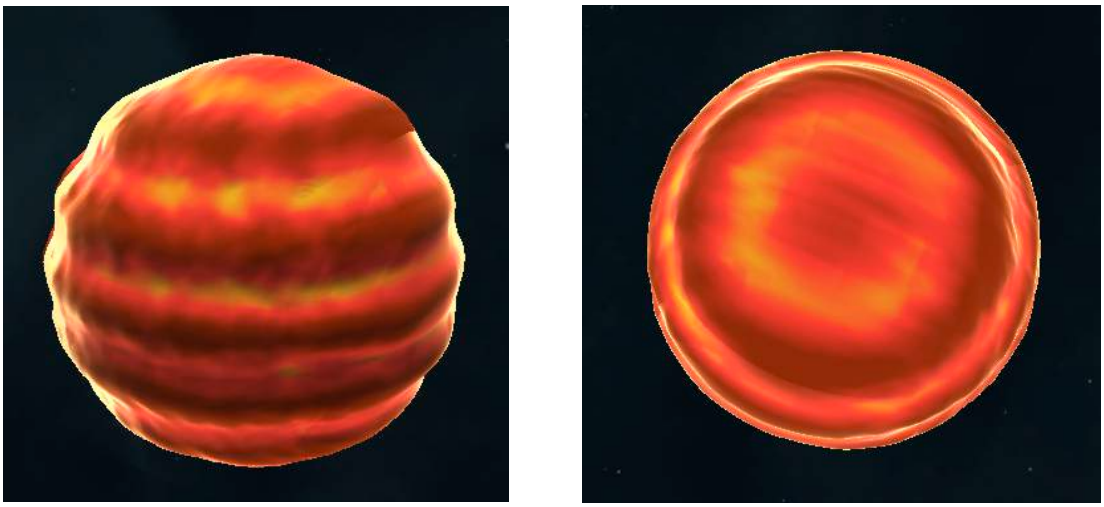

Fig. 5.3.1 Hot Jupiter Gas Giant, side view (left) and top view (right)

5.4 Terrestrial

The term 'Terrestrial' refers to a planet composed primarily of rocks or metals; examples in our solar system are Mercury, Venus, Earth and Mars. The terrestrial planets in this program will be represented with a rocky appearance. Planets nearer the sun will be barren because of the heat (Fig. 5.4.1). Earth-like planets are possible in this category, but only if the conditions are right (Fig. 5.4.2). For the conditions to be right the mass and distance from the sun must be balanced, which makes it aptly unlikely for one to appear. In the program, water levels can be increased or decreased (Fig. 5.4.3).

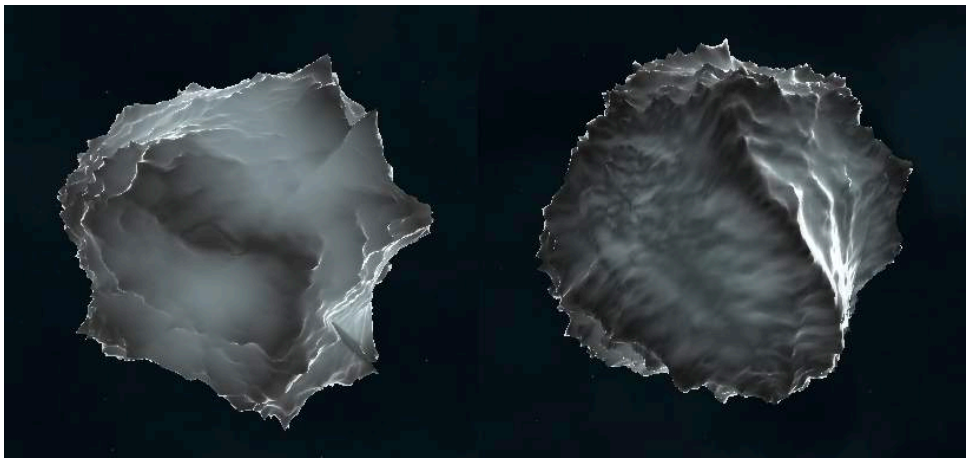

Fig. 5.4.1 barren rocky Terrestrial planets

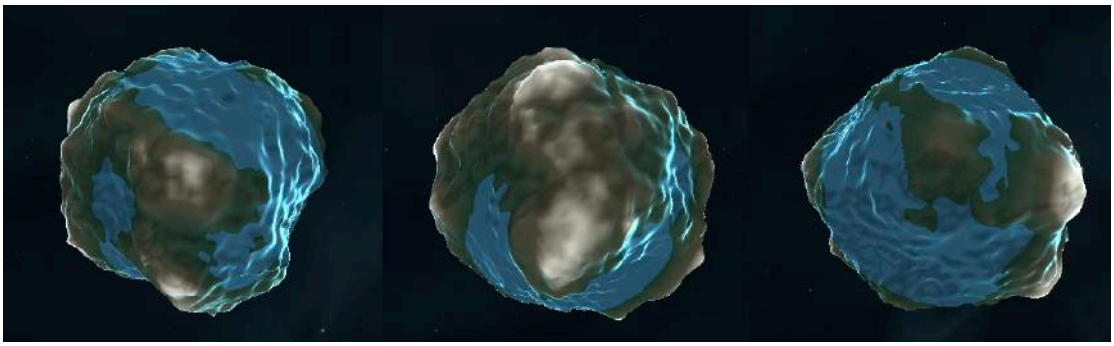

Fig. 5.4.2. Getting a balance between terrain and water for an Earth-like planet

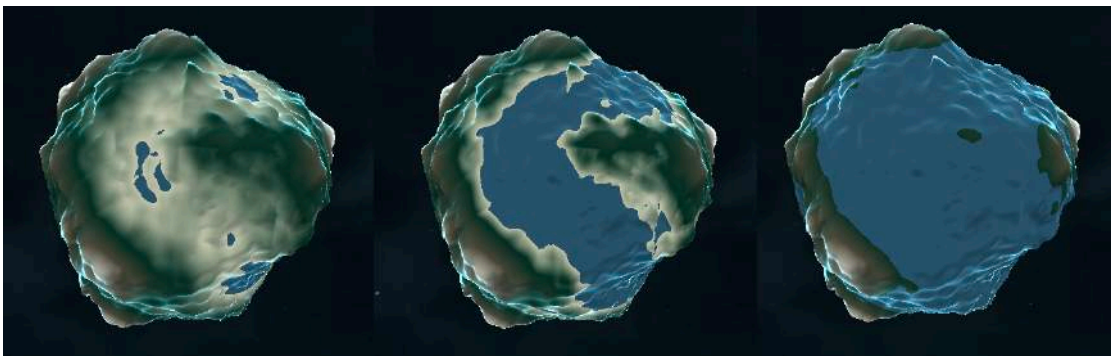

Fig. 5.4.3 Water levels can be changed to affect the atmosphere of a planet

5.5 Super Earth

Super Earth refers to the mass of a planet, and includes planets that are a minimum of two Earth masses, but generally less than 10 Earth masses. The term super Earth does not imply temperatures, compositions or environments similar to Earth, therefore they can differ greatly. This being the case, planets meeting these criteria in the program will be a combination of more random elements. Some examples can be seen in figure 5.5.1.

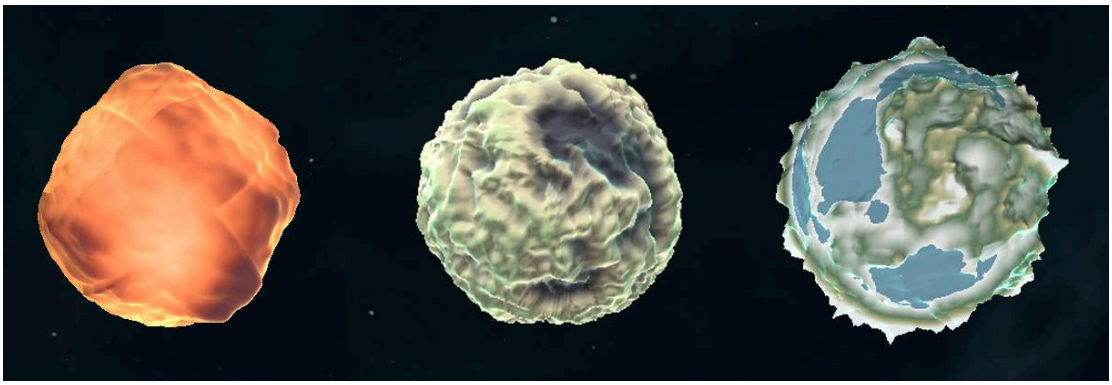

Fig. 5.5.1 Three types of Super Earth generated in the program

5.6 Stars

Stars are made with the same process. Using fBm noise with a low scale value and high number of octaves gives a subtle irregular surface to the star and these changes in levels are used to affect the colour of the star. There are two colour variations used, one to represent a star similar to our Sun, and another red variation to represent a red dwarf star (Fig. 5.6.1).

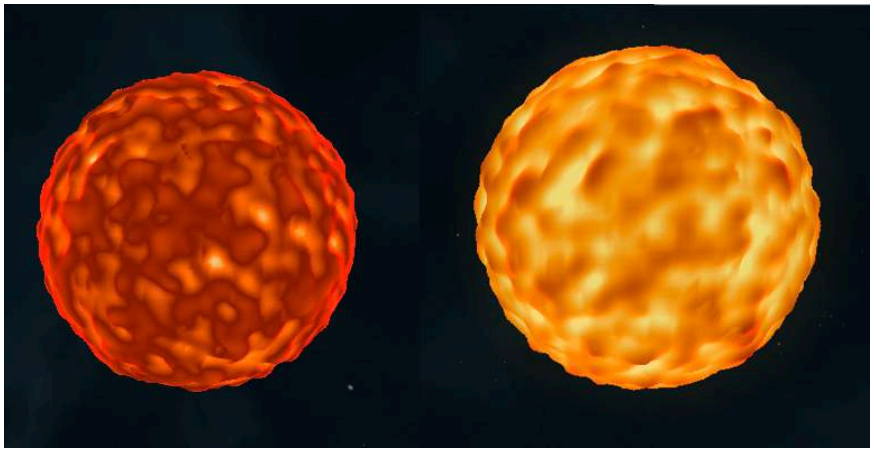

Fig 5.6.1 Two colour variations of fBm stars

6.0 Combining Everything

To decide what planet will be created where, there is a set of rules in place. Firstly, to establish *how many* planets are created, the solar system is split into five zones (Fig. 6.0.1).

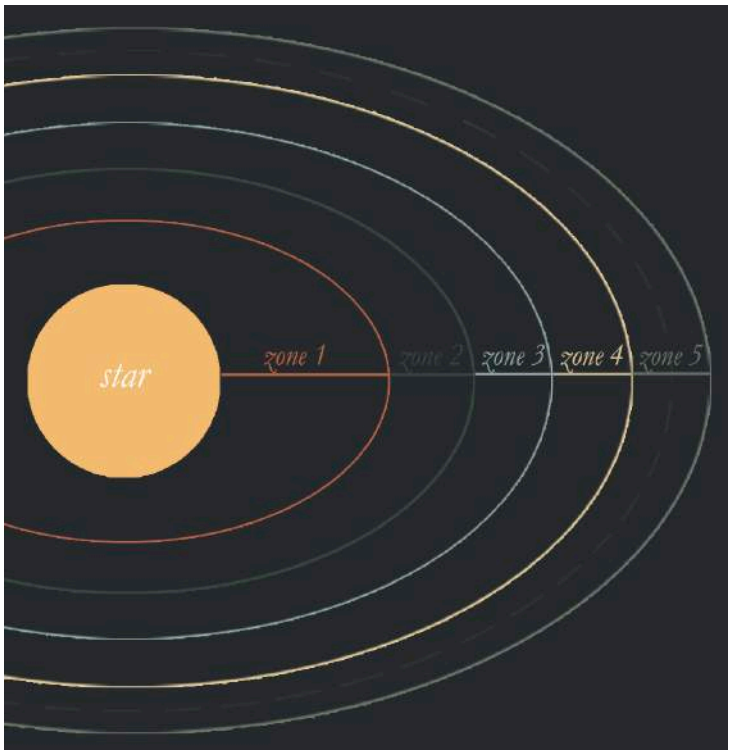

Fig 6.0.1 The possible zones for planets to orbit within

In this version, there will be a maximum of five planets created; this amount could be modified to account for more if desired. The semi-major axis is used to specify the orbit shape and speed. Subsequently, a mass is generated. The mass, together with the semi-major axis, will determine what type of planet is created.

When the program is loaded, the view is from a distance observing the whole solar system. Each planet is labeled with its type, and its orbit is traced with a line. The user can then move around the solar system to get a closer look at any of these planets. The orbits can be paused for a better look, and the labels can be switched on or off. See figure 6.0.2 for the controls given in the program. The background is comprised of textures I painted to resemble stars. There are two layers of stars, one closer to the camera to give a sense of depth when moving around. The solar system can be reset at any time to see a different combination. Some screenshots of the program can be seen in figures 6.0.3, 6.0.4, 6.0.5, 6.0.6 and 6.0.7.

Fig. 6.0.2 The controls the user is given, as shown in the program

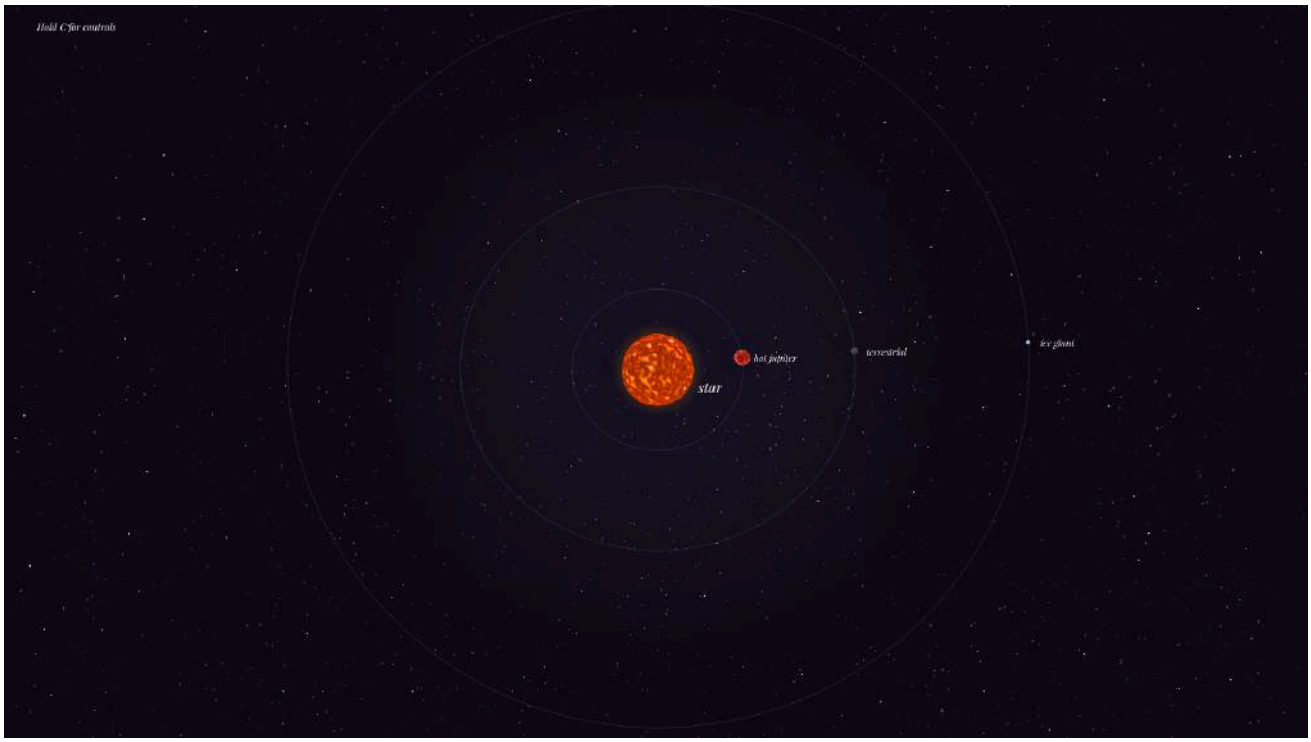

Fig. 6.0.3 The view when loading a new Solar System. Each planet and orbit is labeled

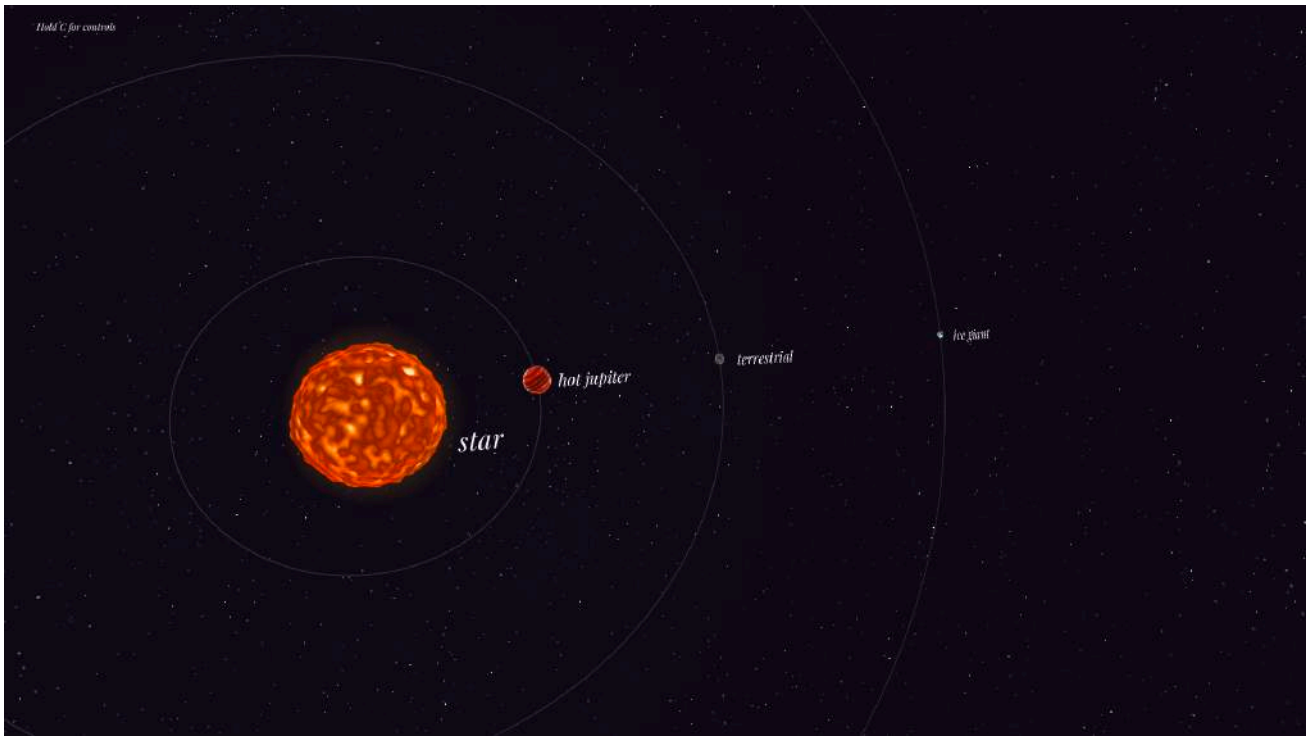

Fig. 6.0.4 The camera can be moved around to view the planets from any angle

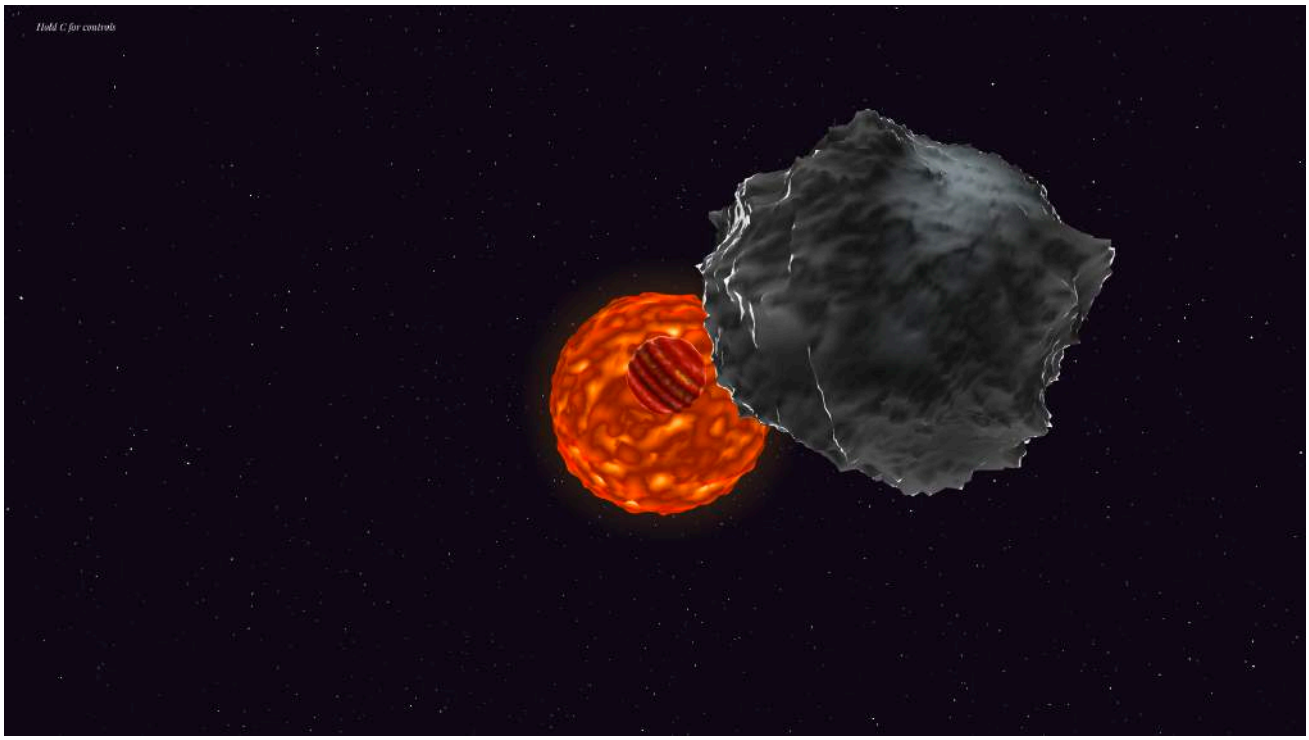

Fig. 6.0.5 Here the labels have been switched off, with a terrestrial planet is in the foreground

Fig. 6.0.6 A different Solar System, with all five planets in view

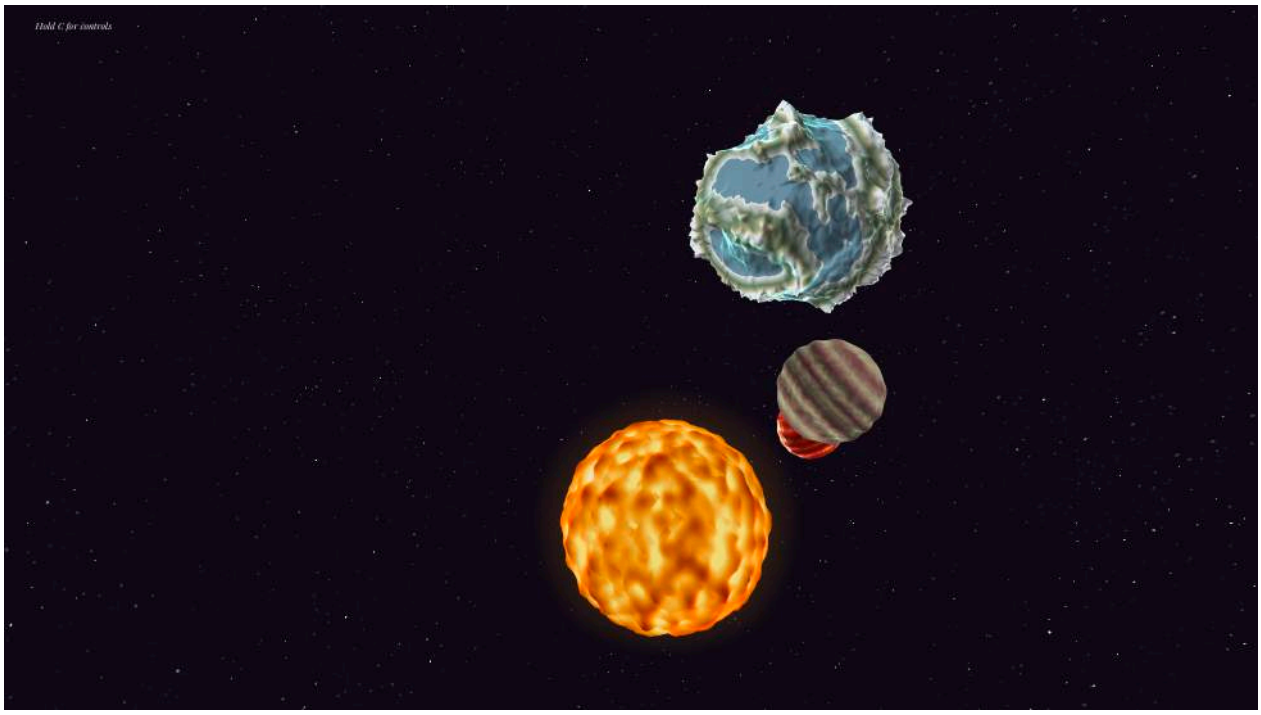

Fig. 6.0.7 Another view of a Solar System. When the orbits are paused, the planets continue to slowly rotate about a local axis to see them from all angles.

7.0 Analysis

In the end, the project exceeded what I originally hoped to do. I set out with the intention of making a procedural planet generator, extending this into a solar system if I had time. As the project progressed, the project evolved into something much more scientific as I discovered the vast amount of data about planets available and could use this in the program. From the beginning I wanted the planets to be visually appealing and this was something I struggled with throughout. In the end it was writing shaders that pulled everything together and allowed me to experiment with the tools in place to create a wide range of planets.

In terms of the program, I am pleased with the results. The project presented a great deal of difficulties as I was working in areas completely alien to me, but I've learned about procedural generation, about using the Unity game engine, about programming in C# and about shaders.

I tried to make the program user friendly so that anyone can use it. I would liked to have facts included so that users could learn about what type of planet each one was, perhaps see some characteristics about them and how many similar planets have been discovered in the real world.

Of course, there is a lot more that could be done to extend the software. Moons orbiting planets is something I would like to come back to. The framework is in place to add these; they could be generated as rocky planets and the orbit logic is the same as the planets orbiting the star but for simplicity I kept the program as just planets. The process of creating the planets in the Unity editor is quite a satisfying one, and I would like to extend the software at a later date to include a separate mode that would allow the user to create their own planet and perhaps see where it would appear in a solar system.

8.0 Conclusion

The outcome of this project is a program in which solar systems are procedurally generated. The solar systems will consist of a star being orbited by up to five planets, the atmosphere of each being dependent on it's distance from the star and mass, both of which are generated with weighted random factors, using data from real exoplanets to influence the random weight. The secondary outcome of this project is this paper, in which research is presented into the astronomy behind the solar systems, including information about the current planets that have been discovered. There is also information regarding procedural generation and the methods in which the program was constructed.


Types of Exoplanets
all planets and stars represented here have been procedurally generated in engine.
star
red dwarf star
hot jupiter
terrestrial
super earth
habitable
super earth
gas giant
rocky terrestrial
ice giant
(not to scale)

9.0 References

9.1 Astronomy Research

NASA, 2014. *Kepler Overview* [online]. NASA. Available from: http://www.nasa.gov/mission_pages/kepler/overview/index.html#.UwYzkUJ_ura [Accessed 20 February 2014].

NASA, 2013. *About the Mission* [online]. NASA. Available from: http://kepler.nasa.gov/Mission/QuickGuide/ [Accessed 21 February 2014].

Gammon, K., 2013. *Exoplanets: Worlds Beyond Our Solar System* [online]. SPACE.com. Available from: http://www.space.com/17738-exoplanets.html [Accessed 20 February 2014].

Exoplanet.eu, 2014. *The Extrasolar Planets Encyclopaedia* [online]. Exoplanet TEAM. Available from: http://www.exoplanet.eu [Accessed 20 February 2014].

Exoplanets.org, 2014. *The Exoplanet Data Explorer* [online]. Available from: http://exoplanets.org/ [Accessed 21 February 2014].

PlanetQuest, 2014. *New World's Atlas* [online]. Available from: http://planetquest.jpl.nasa.gov/newworldsatlas [Accessed 22 February 2014].

Rayman, M., 2013. *How many solar systems are in our galaxy?* [online]. NASA. Available from: http://spaceplace.nasa.gov/review/dr-marc-space/#/review/dr-marc-space/solar-systems-in-galaxy.html [Accessed 20 February 2014].

Smith, R., 2000. *Encyclopedia of the Solar System.* London: Fitzroy Dearborn Publishers.

NASA Ames Research Center, 2012. *Destination Innovations – Episode 1 – Kepler* [video, online]. Available from: http://www.youtube.com/watch?v=ArLEfh8GbEY [Accessed 20 February 2014].

Jet Propulsion Laboratory, 2012. *Kepler's Planetary Systems in Motion(Artist Concept)* [online]. NASA. Available from: http://www.jpl.nasa.gov/spaceimages/details.php?id=PIA15264 [Accessed 20 February 2014].

Wakeford, H., 2011. *It's About Time!* [online]. Stellar Planet (blogspot). Available from: http://stellarplanet.blogspot.co.uk/2011/11/its-about-time.html [Accessed 20 February 2014].

Kornmesser, M., 2012. *Planets everywhere (artist's impression)* [digital art]. European Southern Observatory. Available from: http://www.eso.org/public/images/eso1204a/ [Accessed 20 February 2014]

Seager, S., 2008. *Exoplanet Mass, Radius, and the Search for Habitable Worlds* [online]. MIT Physics Annual 2008.

Atkinson, N., 2014. *The Most Common Exoplanets Might be "Mini Neptunes"* [online]. Universe today. Available from: http://www.universetoday.com/107816/the-most-common-exoplanets-might-be-mini-neptunes/ [Accessed 21 February 2014].

Wikipedia, 2014. *Semi-Major Axis* [online]. Wikipedia. Available from: http://en.wikipedia.org/wiki/Semi-major_axis [Accessed 21 February 2014].

Wikipedia, 2014. *Orbital Speed* [online]. Wikipedia. Available from: http://en.wikipedia.org/wiki/Orbital_speed [Accessed 21 February 2014].

Yaqoob, T., 2011. *Exoplanets Semimajor Axis Distribution* [online]. Available from: http://exoplanets.co/exoplanet-correlations/exoplanets-semimajor-axis-distribution.html [Accessed 21 February 2014].

Nave, R., 2000. *Kepler's Laws* [online]. Hyperphysics. Available from: http://hyperphysics.phy-astr.gsu.edu/hbase/kepler.html [Accessed 21 February 2014].

Astrophysicsformulas.com. *Orbital Period Equation According to Kepler's Third Law* [online]. Available from: http://astrophysicsformulas.com/astronomy-formulas-astrophysics-formulas/orbital-period-equation-according-to-kepler-s-third-law/ [Accessed 21 February 2014].

Musgrave, I., 2011. *Evolution News and Views Fails at Exoplanets Too* [online]. Astroblog. Available from: http://astroblogger.blogspot.co.uk/2011/09/evolution-news-and-veiws-fails-at.html#links [Accessed 21 February 2014].

UNL: University of Nebraska-Lincoln Astronomy Education, 2013. *Detecting ExtraSolar Planets* [online]. Available from: http://astro.unl.edu/naap/esp/detection.html [Accessed 22 February 2014].

Nvseal, 2009. *Procedural Planet Package Pre-Release Images* [online]. Available from: http://www.planetside.co.uk/forums/index.php?topic=8319.0 [Accessed 2 March 2014].

9.2 Procedural Research

Ebert, D., et al., 2003. *Texturing and Modeling: A Procedural Approach*. Third Edition. San Francisco: Morgan Kaufmann Publishers.

Nguyen, H., 2008. *GPU Gems 3*. Boston: Addison-Wesley Professional.

Perlin, K., 1997. *Noise and Turbulence* [online]. Available from: http://mrl.nyu.edu/~perlin/doc/oscar.html [Accessed 24 February 2014].

O'Neil, S., 2001. *A Real-Time Procedural Universe, Part One: Generating Planetary Bodies* [online]. Gamasutra. Available from: http://www.gamasutra.com/view/feature/131507/a_realtime_procedural_universe_.php [Accessed 20 February 2014].

Willmott, A et al, 2007. *Siggraph 2007 Maxis Sketches* [online]. San Diego: Siggraph 2007.

Wittens, S., 2009. Making Worlds 1 – Of Spheres and Cubes. *Acko.net* [online].

Górski, K. et al, 2004. HEALPix: A Framework for High-Resolution Discretization and Fast Analysis of Data Distributed on the Sphere. *The Astrophysical Journal.* 662 (2), 759-771.

Spore, 2008. [game, cross platform]. California: Electronic Arts.

Quigley, O., 2009. *Spore's procedural terrain texturing* [online]. Ocean Quigley's Projects. Available from: http://oceanquigley.blogspot.co.uk/2009/04/spores-procedural-terrain-texturing.html [Accessed 22 February 2014].

Elias, H., 2003. *Perlin Noise* [online].