LUND
UNIVERSITY

Procedurally Generating an Artificial Galaxy

Department of Statistics
Bachelor Thesis, 15 ECTS
Supervisor: Björn Holmquist
March 2016

Author: Olof Elfwering

Abstract

The idea of procedurally generating artificial worlds has been around for a long time. It is used both for CGI effects in movies and, more prominently, for video games. It is done by big companies and small teams; professionals as well as beginners and hobbyists. It is an instrumental tool that enables even the smallest of development teams to create large worlds. While there is a lot of inspiration to draw from, the undertaking may be daunting for the beginner. The goal of this paper is to give an introduction to some of the concepts by taking us through the steps to procedurally generate a virtual galaxy full of stars and planets. Something simple that can be taken much further. We analyse statistical data and observations of space to create about 30 equations that superficially mimics a spiral galaxy with 100 billion solar systems, all with the help of some basic probability theory.

Sammanfattning

Idén om att processuellt generera artificiella världar är gammal. Metoden används både för CGI effekter i filmer och, framförallt, för tevespel. Det görs av stora företag och mindre studios; av professionella programmerare såväl som nybörjare och hobbyister. Det är ett kraftfullt verktyg som möjliggör även små team att skapa enorma världar. Men även om det finns mycket inspiration att hämta kan det framstå som en svår uppgift för nybörjaren. Målet med denna uppsats är att ge en introduktion till några av de grundläggande koncepten genom att gå igenom stegen för att processuellt generera en virtuell galax full med stjärnor och planeter. Något enkelt som kan utvecklas vidare. Vi analyserar statistiska observationer av rymden för att skapa ungefär 30 ekvationer som ytligt efterliknar en spiralgalax med 100 miljarder solsystem, allt med hjälp av grundläggande sannolikhetslära.

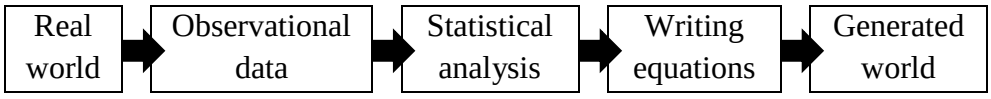
1. Introduction

Instead of having artists design everything in the virtual world of a videogame or a CGI sequence in a movie, certain aspects can be handed over to algorithms for procedural generation. This can either be done beforehand or as the scene plays out. It can be as simple as generating numbers that determines random positions for objects designed by artists. Like making a forest out of trees for example. Or it can be more complex, like procedurally generating the trees themselves and place them in natural formations.

The main benefit of procedurally generating something rather than making it yourself is quantity. For an artist, every tree takes time to create and place in a scene. With procedural generation, the time investment is restricted to making the algorithm. As soon it's done you can generate as many trees as you like. The same is true for procedural generation in general.

Procedural generation can also have an effect on the form of interaction in a videogame. This became clear when Rogue came out in 1980. The game generated a new world every time you started it, which made every play-through unique and removed the possibility of memorizing the layout, forcing you to think on your feet. Without the ability to study specific locations the player is led to decipher the rules that generate the world. Rather than memorizing what's inside a particular room, one might start to associate certain kinds of rooms with certain things (Wichman 1997 & Parish 2007).

In this paper, our goal is to define a set of algorithms for generating an artificial galaxy with stars and planets, something that might be suitable for a simple game. Space is a popular environment for procedural generation, partly because of the scope, and partly because of the relative ease of mimicking the superficial arrangement of celestial bodies. What follows is just a rudimentary example, but a rather powerful one at that. We will place stars in the formation of a spiral galaxy, base their characteristics on real data and give them their own planetary system. The general idea can be described in the following way:



The focus of this paper is on the statistical analysis and formulating the equations which are, with a couple of clearly stated exceptions, of my own design. These equations are part of a game that I am working on. The game is made with Leadwerks Game Engine (2016) and the code is written in Lua. With the purpose of presenting a generalizable approach I give the math, but not the specific code since its syntax is specific to Leadwerks and my game.

Before we start the analysis and creation of the equations, though, we get an introduction to how computers generate random numbers: a crucial element of what we are about to do.

2. Pseudorandom Number Generation

While you have the ability to roll a die, spin a wheel or toss a coin to generate random numbers, computers are restricted to so-called pseudorandom number generators (PRNG's). Their basic principal is simple: an input number, called a seed, is transformed into a seemingly random output number by an algorithm. An example of such an algorithm is the outdated middle-square method. It will take a seed of up to four digits, square it into an eight digit number, adding zeroes in front if necessary, and output the four middle digits of this new number (Introduction to Random Number Generators 2007). Like all PRNG's, this algorithm will always generate the same output from the same seed. It is a deterministic process that generate numbers that appear to be random, i.e. pseudorandom numbers.

The middle-square method was designed with simplicity in mind, which makes it a good example of the general principle behind a PRNG. As it would happen, it also makes it a good example of potential problems with a PRNG. Imagine the seeds between 0 and 9. They will all output 0. Furthermore, if we run a sequence of seeds through the algorithm, subsequent outputs will be higher with intermittent drops. To get rid of this pattern you need to run the output numbers through the algorithm a few more times. The problem is that doing this will increase the number of outputs that are 0, after six times the pattern is gone, but 316 out of the 10 000 possible seeds will generate the output 0. In general, we do not want any specific outputs to be more common than others, nor do we want there to be any pattern between the outputs of subsequent seeds.

There is a wide range of PRNG's with fewer problems, ones that are actually used. Among them, the linear congruential generator is the most common. In the same way that you can choose how many times to run the seeds through the middle-square algorithm, the linear congruential generator can be configured in different ways with different results, some of which are better than others. It is slightly more complicated than the middle-square method, but understanding an algorithm is not a prerequisite for using it. Taking note of its properties, however, may be.

Most programming languages have some integrated PRNG-function that output numbers from a uniform probability distribution, meaning that every possible output number has the same probability. This is what basic PRNG-functions generally do. It’s the same principle as rolling a die. And just like there are crooked dice, there are PRNG-functions claiming to have a uniform distribution that actually don’t. The rand() function in C++ is a good example of this. It’s a linear congruential generator with a specific implementation that makes lower output numbers more common than higher ones (C++ Resource Network 2015, Sourceware 2015). If you are relying on this number to be from a uniform distribution in your calculations, using this function might alter your predictions. So, while you may not need to understand the

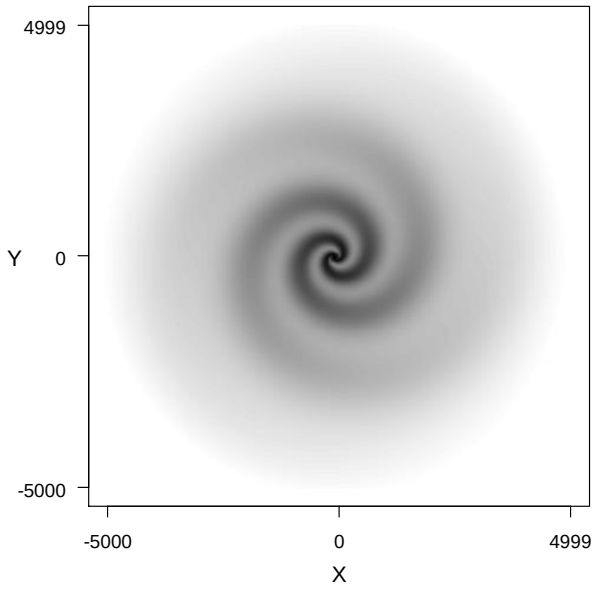
algorithm on a deep level, knowing about its deficiencies is important. Such knowledge could come from a deep comprehension of the algorithm, a statistical analysis of it, or from simply reading the documentation. I use the random() function in the Lua Mathematical library (Ierusalimschy 2003), it's got its limitations, but is suitable for our needs.

3. The Positioning of Stars

Galaxies take on a multitude of different shapes, but we regard the spiral as the epitome: it's what we think of when we think of a galaxy. Therefore, we want to generate a reasonably sized spiral galaxy. We give it a diameter of 100 000 light-years (ly), a thickness of 1000 ly and about 100 billion solar systems. Any way you look at it, that's an awful lot of information to handle all at once, which means we need to break it down into smaller pieces. This can be done in different ways, we elect to go with the convention and split it into cubes stacked in a three-dimensional grid. The size of all cubes are 10×10×10 ly. Next, we need to distribute our stars into these cubes in a way that mimics the shape of a spiral galaxy. To do that, we start by thinking in two dimensions, looking down at a spiral-shaped galaxy below, like what we have in Figure 1.

The two dimensions in Figure 1, X and Y, are integers ranging from -5000 to +4999. They mark the coordinates of our cubes, or squares rather, as we momentarily disregard the third dimension. The only information we have on these squares are their positions in the grid. We cannot do a whole lot with this information directly, but we can transform it into something more useful.

Figure 1. Top down-view of a spiral



For computational reasons the resolution is set to 800×800 so only 0.64 % of all the squares are shown. Darker colour signifies more stars, see Appendix 1 for the code written in R.

At the centre (X,Y) = (0,0) is the centre of the galaxy. In equation (1) we use the Pythagorean theorem to calculate the distance between a given square with coordinates (X,Y) and the centre. In equation (2) we use trigonometry to get the angle between the X-axis and the shortest line from the centre to that given square.

$$\text{Distance}(X,Y) = D_{X,Y} = \sqrt{X^2 + Y^2} \quad (1)$$

$$\text{Angle}(X,Y) = A_{X,Y} = \arctan(Y/X) \quad (2)$$

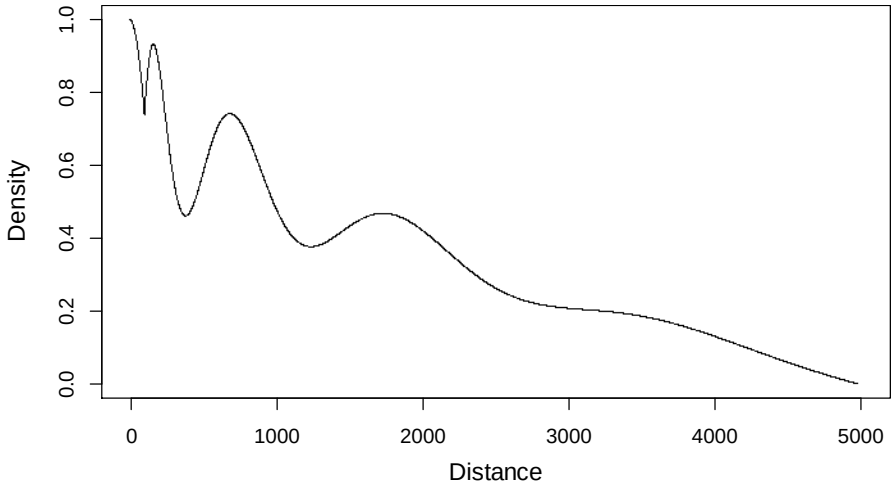
These two statistics can be put into an equation to determine the comparative density of stars in a given square in Figure 1. Imagine a slice of the spiral and make a graph out of it. Put distance to the core on the horizontal axis and density of stars on the vertical axis. The shape of the graph will depend on the angle of the slice, but in general we expect to see a spike at the core, the density of stars will be at its highest here. The highest density is set to 1, the meaning of which is determined later, and the minimum is 0, meaning no stars in that square, this gives us equation (3). Dividing by 200 sets the radius of the core at 2000 light years (200 × width of square), raising to the power of 2 determines the development in the rate that the density decreases as the distance increases, slowly at first and faster as the distance increases. The equations in this paper will have many constants like these: numbers that may be exchanged to get differing effects, some of these numbers will be explained while some, for the sake of brevity, will not.

$$\text{Core}(X,Y) = 1 - \left(D_{X,Y}/200\right)^2 \quad (3)$$

After the spike at the core, we also expect a spike every time we intersect an arm, these spikes should widen and lessen in magnitude the further we get from the core. To get a spiral shape, the spikes must move as the angle changes. This can be done with a correctly configured sinewave multiplied with a diminishing exponential function of the distance. Adding a constant and subtracting a simple function of the distance will raise the floor from 0 near the core, so that there are some stars in-between the arms. All this is achieved by equation (4). Dividing the distance in the exponential function with 1500 sets the length of the galaxy's arms and the transformation of the distance within the sine function sets the width of the arms. If we were to remove the angle variable we would get a galaxy core with rings around it, like the Sombrero galaxy. Multiplying the angle variable by two would double the number of arms. Removing the sine function would give a simple elliptical shape like the Andromeda galaxy. With some minor tinkering we could get a more general way of generating different kinds of galaxies, but for a spiral galaxy this specific equation is a decent start:

$$\text{Arms}(X,Y) = e^{-\frac{D_{X,Y}}{1500}} \times 0.5 \times \sin\left(\left(0.5 \times D_{X,Y}\right)^{0.35} - A_{X,Y}\right)^2 + 0.5 - \frac{D_{X,Y}}{10000} \quad (4)$$

Figure 2. Stellar density and distance at a given angle



The final two-dimensional density is then determined by equation (5), which is a simple maximum of equation (3), equation (4) and 0. This density varies between 1 and 0, denoting how many stars one square has in relation to the others. The density varies based on the coordinates of a square in a way that will generate a spiral shape. With $A = 0$, this generates the curve in Figure 2. And when we go through all the combinations of X and Y (all the squares), give a darker shade for a higher density and arrange them in the grid we get a spiral shape like the one in Figure 1 (Figure 1 only shows a subset of all combinations, the resolution is set to 800 × 800, see Appendix 1 for the specific code written in R).

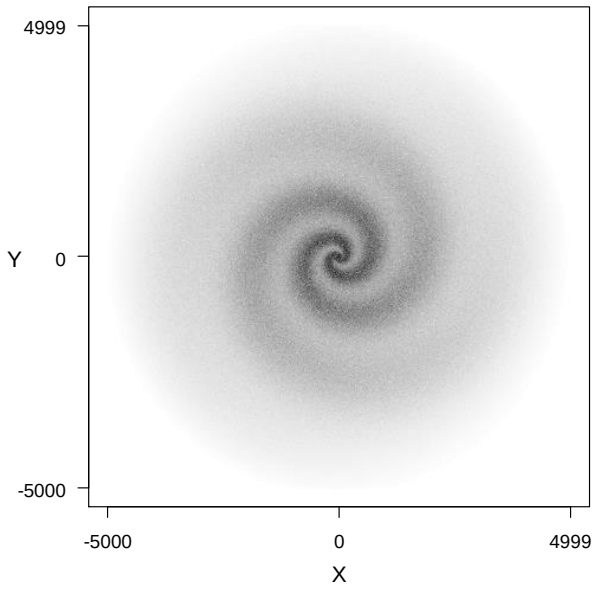
$$\text{Density}(X, Y) = \max(\text{Core}(X, Y), \text{Arms}(X, Y), 0) \quad (5)$$

We now have a model for creating a two-dimensional galaxy, but we obviously want three dimensions. So we bring in (Z), an integer ranging from -50 to +49. This gives us a three-dimensional grid (X,Y,Z). With one cube at every combination of X, Y and Z, we have 10 000 × 10 000 × 100 = 10^{10} cubes.

The density of stars in every cube is set by its position in the grid, this has already been done for X and Y, so it's time for Z. The density should be higher near Z = 0, which is the central plane of the galaxy, and fall to zero as it gets far away. Roughly corresponding to what a spiral galaxy looks like in profile. To do this we can modify the density in equation (5) by multiplying it with an exponential equation of Z that reaches 1 at Z = 0 and falls to 0 at Z = ± 50. This gives us equation (6).

$$\text{Density}(X, Y, Z) = \text{Density}(X, Y) \times \left(1 - \left(\frac{Z}{50}\right)^4\right) \quad (6)$$

Figure 3. Stellar density with noise at central plane



For computational reasons the resolution is set to 800×800 so only 0.64 % of all the cubes at Z = 0 is shown. Darker colour signifies more stars in a cube, 0 = white and 99 = black.

Any cube will have only a slight difference in density compared to the adjacent ones, we can see this by the smoothness of the spiral in Figure 1. This means that going from one cube in the grid to the next will mean little change in the number of stars. To make the structure more chaotic we will add some stochastic noise. This can be done in multitude of different ways: what follows is just an example.

We generate a random number (U_0) from a continuous uniform distribution between 0 and 1, U(0,1) for short. In theory this number has an infinite number of decimals, but in practice our PRNG sets a limit. We subtract 0.5 from the generated number which gives a 50 percent chance of a negative difference. The difference is then raised to the power of three to make outcomes closer to 0 more probable. It is then multiplied with 2.2 to increase the spread somewhat. Raising this number would increase the spread further. This transformation is then multiplied with Density(X,Y,Z) in equation (7). By doing this we get a higher variance for denser regions.

We also want to get the total number of stars to 100 billion. To do this we need the mean number of stars per cube to be about 10 (with 10^{10} cubes: $10 \times 10^{10} = 100$ billion stars). In other words: when we calculate the number of stars in a cube with coordinates (X,Y,Z) in equation 7, the mean needs to be about 10.

To get it there, we need to know the mean value of Density(X,Y,Z). This mean can be retrieved by calculating the density for all combinations of X, Y and Z. But since there are so many combinations (10^{10}) we may opt to estimate the mean by taking a random sample of 100 000 cubes, i.e. generating random numbers from uniform distributions that cover the ranges

of X,Y and Z, insert them into our equations and calculate the density of the randomly selected cubes. Doing this gives us the estimated mean 0.129. If we multiply Density(X,Y,Z) with 78 we get a new (estimated) mean of just over 10 stars per cube, which brings us to over 100 billion in the entire galaxy.

Equation (7) gives us the number of stars in a cube(X,Y,Z). The result is rounded to the nearest integer. As illustrated by Figure 3 we maintain the shape but loose the smoothness.

$$\mathrm{Stars(X,Y,Z)} = \mathrm{round}\left(78 \times \mathrm{Density(X,Y,Z)} \times (1 + 2.2 \times (\mathrm{U_0} - 0.5)^3)\right) \quad (7)$$

Equation (1) through (6) determines the relative density of stars in the cubes in a way that creates a three-dimensional, albeit rather flat, spiral shape, only one percent as thick as it is wide. Equation (7) then takes us away from that perfect pattern and adds some stochastic noise, giving us a probability distribution dependent on spatial position. The minimum number of stars in a cube will be 0 and the maximum will be 99, this many stars appear when both Density(X,Y,Z) and U_0 are close to 1. Restricting the maximum number of stars means a restriction in the amount of information the computer will have to handle at any one time. We chose to multiply the stochastic component with 2.2 partly in order to keep the maximum number of stars per cube under 100.

The last step in determining the spatial distribution of stars is to generate the location of the stars within each cube. We will generate three random numbers (U_1, U_2, and U_3) from U(0,1) for each star. These numbers determine the stars position in the three dimensions where 0 is at one edge of the cube and 1 is at the other. The likelihood of any stars getting too close to one another is miniscule, we could add further restrictions to prohibit it, but elect not to do so in this model.

It is a simple model, but it works. We can move through the galaxy, generate stars as we enter new cubes and drop the stars in distant cubes as we leave them behind. The number of cubes and stars in memory at the same time can be set as we see fit, but the minimum ought to be 27 (the cube we are in plus one extra in every direction i.e. a 3×3×3 grid). If we were to only have the one, stars would pop up right in front of us as we move from one cube to another.

In order for a cube to look the same as we return and generate stars in it a second time, we must always select the same set of seeds for the same cube. This can be done by making the seed selection a function of the cubes position in the grid (X,Y,Z), but we need to make sure that no two cubes get the same seed as that would make them identical. Equation (8) gives an example of how the seed for U_0 can be set, guaranteeing that no two cubes get the same seed.

$$Seed = 5 \times (10^9 + 10^5 + 10) + X \times 10^6 + Y \times 10^2 + Z \quad (8)$$

Since U_1, U_2, and U_3 should be unique for every star they need to get unique seeds. This can be done by setting the seed of U_1 to that of U_0 plus 10^{11} multiplied with the stars number (going by the order in which it was generated among the stars in its cube). The seeds for U_2, and U_3 can be set similarly but with 10^{13} and 10^{15} respectively instead of 10^{11}, the exponent is increased by two for every new variable since the maximum number of stars in a cube (99) is a two-digit number. Equation (8) works on a similar principle, basically reserving certain positions in the seed for certain variables. This is one of many ways to guarantee that every generated number gets a unique seed.

4. Stellar Characteristics

We have a way of generating about 100 billion stars in the superficial arrangement of a spiral galaxy, but the stars are all the same: simple dots in space. We need to differentiate them by determining their luminosity, size and colour. We will do this based on observations of real stars.

In order to limit ourselves, we focus solely on the so-called main sequence stars. They account for about 90 % of all stars and there is a correlation between heat, size, luminosity and frequency among them. Like all stars, they are categorized by their surface temperature, from warm to cold, in the categories O, B, A F, G, K and M. Warmer main sequence stars are larger, more luminous and less frequent than colder ones. There is also a continuous range of colours, from the blue O to the orange M.

We want to generate one number that can determine all of these aspects. It will be a number from U(0,1) called U_4 and similar to U_1 through U_3, its seed will be that of U_0 (from equation (8)) + 10^{17} multiplied with the stars number.

Based on observations (LeDrew, 2001, p.33), all the stellar classes can be assigned an estimated share of the total number of main sequence stars. These shares are seen in the penultimate column of Table 1. To its right is a column of the cumulative share, which is what our generated number corresponds to. This means that a generated number between 0 and 0.7645629 should get characteristics corresponding to the M-class, one between 0.7645629 and 0.8859221 should get ones corresponding to the K-class and so on. As the table shows, these classifications separate continuous scales for radius and luminosity. These continuous scales are what we want to mimic.

Table 1. Observational based distribution of stellar characteristics

Class	Radius	Luminosity	Share	Cum. Share
M	0.08 – 0.7	0.000158 – 0.086	0.7645629	0.7645629
K	0.7 – 0.96	0.086 – 0.58	0.1213592	0.8859221
G	0.96 – 1.15	0.58 – 1.54	0.0764563	0.9623784
F	1.15 – 1.4	1.54 – 4.42	0.0303398	0.9927182
A	1.4 – 1.8	4.42 – 21.2	0.0060679	0.9987861
B	1.8 – 6.6	21.2 – 26800	0.0012136	0.9999997
O	6.6 – 12	26800 – 78100000	0.0000003	1.0000000

Characteristics and shares are for main-sequence stars, classed by the Yerkes classification system. Data from LeDrew (2001), Kaltenegger & Wesley (2009) and Stellar Luminosity Calculator (2014).

4.1. Luminosity

We start with luminosity. It ranges from less than one thousandth to 78.1 million times as bright as the sun. We need a transformation that can turn 0.7645629 into 0.086, 0.8859221 into 0.58, 0.9623784 into 1.54, etc. If we had more data this could be done more elegantly, but we will simply find some transformation that puts us in the ballpark. There seems to be a logarithmical relation between luminosity and cumulative share, but it might not be that simple. The range of luminosity is rather extreme.

We could think of the process as fitting a line to the eight data points (the class-borders), we would then transform the data and find some linear regression. The problem with doing this is to find a way that fits the data well. We are not interested in absolute deviations: a deviation of 2.5 would be catastrophic for the borders of the M-class but highly negligible for the O-class. Instead, we are interested in the relative deviations.

To solve this problem, we construct an algorithm that cycles through different combinations of values for the parameters of a transformation. Then we try different transformations, optimize their parameters with the algorithm and compare them with one another. This process lands us with the transformation in equation (9). To get the best combination of values for the parameters we run it through the algorithm and single out the more interesting ranges of parameter values.

$$\text{Luminosity} = A + B \times \left(-\frac{\log(1-U_4)}{C}\right)^D \tag{9}$$

The algorithm is rather inefficient as it is set to cycle through all possible combinations, including the obviously bad ones. Therefore we want to limit the sets to cycle through. The algorithm calculates the overall relative deviance from the data points for every combination of values for the parameters and saves the set with the best fit. It is written in R and the code for this specific transformation can be found in Appendix 2. When we run the algorithm we get the specific values on the parameters seen in equation (10).

Since the transformation approaches infinity as our generated number approaches 1 we also cap it at 78 100 000 (the upper limit for the O-class).

$$\text{Luminosity} = \min\left(0.00016 + 45 \times \left(-\frac{\log(1-U_4)}{4.6}\right)^{5.4}, 78100000\right) \quad (10)$$

4.2.Radius

After luminosity, we want to generate the stars radius measured in solar radii. The range is from 8 percent to 12 times that of the sun, this range is rather reasonable and we end up with the transformation in equation (11).

$$\text{Radius} = \text{SR} = \min(0.08 - 0.43912 \times \log(1 - U_4), 12) \quad (11)$$

The transformations in equations (10) and (11) maintain the general trend but create aberrations from the data in Table 1. If this was a problem they might be amended, but we would never get spot-on, and we are already rather close as it is.

4.3.Colour

Generating the colour can be done in a similar way to luminosity and size. If we break it down to the RGB components we can make three different transformations. The desired numbers can be seen in Table 2 where the cumulative shares have been amended to reflect that the colours represent the middle of each class. The RGB-values are expressed in the range [0,1], but could simply be multiplied with 255 if the [0,255] range is to be used.

This time, we don't just want transformations that gets us close to the data. We want transformations that never diverge in the wrong way. If we were to get just a little less of blue and red for the F-class for example, we would end up with a greenish star. If we want something that looks somewhat realistic we need to steer clear of generating colours that stars don't have.

We start with red and blue. Red is strictly decreasing and blue is strictly increasing. Fitting a line as closely as we can to the data we end up with the transformations in equations (12) and (13).

Table 2. Observational based distribution of stellar colours in RGB

Class	Red	Green	Blue	Cum. share
M	1	0.662745098	0.435294118	0.3822815
K	1	0.866666667	0.705882353	0.8250998
G	1	0.956862745	0.909803922	0.9246997
F	0.984313725	0.97254902	1	0.9773497
A	0.792156863	0.847058824	1	0.9956247
B	0.666666667	0.749019608	1	0.9993747
O	0.607843137	0.690196078	1	0.9999998

Shares are for main-sequence stars, colours represent the mean in the Yerkes classification system. Data from LeDrew (2001) and Kaltenegger & Wesley (2009).

$$\text{Red} = \min(0.62 + (-\log(U_4))^{0.2}, 1) \quad (12)$$

$$\text{Blue} = \min\left(0.25 + 0.9 \times \left(-\frac{\log(1-U_4)}{4.4}\right)^{0.7}, 1\right) \quad (13)$$

To prevent unwanted deviations, we make the beginning of green into a function of blue and U_4 (equation (14)), and the decline at the end into a separate function of U_4 (equation (15)) that takes over when the first one exceeds 0.99.

$$\text{Green} = \text{Blue} + \frac{0.25}{(1+2\times U_4)^2} \quad (14)$$

$$\text{Green} = 1 - \max\left((U_4 - 0.95)^2, \frac{U_4^{1000}}{3.5}\right) \quad (15)$$

At this point we have taken our initial U(0,1) value from the PRNG and turned it into five. We have transformed a uniform probability distribution into five values that mimics the superficial characteristics of stars on a continuous scale. The number of different stars we can generate is limited only by the number of different outputs we get from our PRNG.

What we did was to go from one uniform probability distribution to a set of cumulative distribution functions. A cumulative distribution function gives the probability of getting a certain value or anything lower than it. This is the general method for generating pseudorandom numbers from non-uniform distributions. Since all five values are derived from the same U(0,1) value they have a strict deterministic bond with one another, quite like the main-sequence stars in the night sky.

There's a myriad of different probability distributions with different characteristics that are more or less suitable in different situations. Some programming languages come with functions that do these transformations for you, but finding the code to do it yourself is simple. If we had better data, we might have been able to select one of these well-known distribution by calculating some of the moments and compare them with moment generating functions. In the end, we would probably end up with something similar though.

Our approach gave us slightly altered versions of common distributions. For example: the transformation in equation (11) gives the radius of the stars an exponential distribution with a lower bound which makes it a two-parameter exponential distribution, but then we impose a higher bound which makes it something else.

Our approach manages to mimic the superficial characteristics of main-sequence stars from the smallest to the largest and everything in between. Not spot on, but close enough for superficial resemblance.

5. Stargazing

The first characteristic we generated was luminosity. Among other things, this variable determines the distance from which a star is visible. The fact that M-class stars on the main sequence are so dim means that none are visible from earth without a telescope (Croswell 2002). All the stars we see in the night sky are of the brighter and less common variety. This needs to be reflected in our generated galaxy.

We do this by calculating the apparent brightness as a function of the distance to the star and its luminosity, which we generated in equation (10) (the distance can be calculated by using the Pythagorean theorem twice). The luminosity we generated is applicable right at the star itself and is emanating in every direction in the shape of a sphere. The further we are from the star, the larger the sphere. The same amount of light gets spread out over a larger surface area. To calculate the apparent brightness we simply divide the luminosity with the surface of a sphere that has the star at its centre and our position on its surface, as in equation (16) which is a well-known equation (Palma 2014).

$$\text{Apparent brightness} = \frac{\text{Luminosity}}{4\times\pi\times\text{Distance}^2} \tag{16}$$

At a certain distance, a stars apparent brightness gets so low that it becomes invisible to the naked eye. For the sun (*Luminosity* = 1), this happens at about 72 light years going by the calculations in Celestia (2013). We could lower this to limit the number of visible stars if necessary. It might also be prudent to segment the generation of stars depending on how far away they can be seen, creating stellar classes of our own, but we'll leave that be in this model.

Planets reflecting the light of their star can also be said to have a certain amount of luminosity, but it's incredibly small. This means that we don't need to generate individual bodies in a solar system until we get close.

6. Major Bodies of a Solar System

When we get close to a solar system (within 0.1 ly or so), the first thing we need to do is determine the number of stars in it. As much as a third of all solar systems are believed to have more than one star (Lada 2006). If it turns out to be a binary system, it could either be a substantially smaller star orbiting a large one, which means the smaller star could be treated like a planet, or two somewhat similar stars orbiting a common barycentre. If you are far away either of these cases will appear as a single dot. In the first case the dimmer star would get visible as you get closer. In the second, the single dot would separate into two. Smaller stars may be part of a larger stars system, but are less likely to have a similar or smaller star in a system of their own (Lada 2006). To roughly approximate this relationship we turn to equation (17) for determining the probability of a generated star having another star of similar

size in its system. Since we don't have all that much to go on there is some room for artistic license. We set the probability to 1 percent for smallest and largest stars let it peak at about 52 percent in the G-class with an overall mean of about 24 percent.

$$\text{Probability of multiple stars} = \max\left(\exp(U_4 - 1) - 0.37 - {U_4}^{40}, 0.01\right) \quad (17)$$

To determine the outcome we then generate a new number from U[0,1] that we call U_5. If U_5 is smaller than the probability determined in equation (17) there will be two similarly large stars in the system, we disregard the possibility of there being more than that. We can set the seed for U_5 to the generated number U_4, make sure that the seed isn't truncated. Then, if we have a binary system, we need to determine the size of the second star, which will be slightly smaller than the first one. To do this we generate yet another number (U_6) which can get the same seed as U_5 minus one. U_6 is used to transform the number we generated for the first star (U_4) into something slightly smaller in equation (18). This transformed number is then used to determine the characteristics of the second star in the same way that U_4 determined those of the first one, which we'll call the primary.

$$\text{Second star statistic} = U_4 \times \left(U_4 + U_6 \times (1 - U_4)\right) \quad (18)$$

The orbit of the two stars can take on a number of different forms but we will limit ourselves to them being on opposite sides of the same orbit. We will also limit these orbits, and all others, to perfect circles on the same plane. The orbital period can be determined based on their masses and the distance between them. Their masses, in turn, can be calculated from their luminosity thanks to a known relation among main-sequence stars (Duric 2004, p.19-20). A generalized transformation derived from this relation is seen in equation (19) where a=0.23 and b=2.3 for luminosity<0.03; a=1 and b=4 for luminosity<16; a=1.5 and b=3.5 for luminosity<54 666; a=3200 and b=1 for the rest.

$$\text{Stellar Mass for star i} = SM_i = \left(\frac{\text{Luminosity}}{a}\right)^b \quad (19)$$

The distance between the stars may vary. By generating yet another number from U(0,1) called U_7, with the same seed as U_5 minus two, equation (20) will determine this distance to somewhere between 10 and 100 000 times the radius of the larger star calculated in equation (11) (called *SR* and measured in solar radii), shorter distances are more common. The masses and the distance can then be put into equation (21) to determine the orbital period measured in earth years. Equation (21) is a transformation of the general formula for determining orbital velocity and G is the gravitational constant which is equal to 6.674×10^{-11}. The other constant is there to give us the orbital

period in Earth years and was retrieved by inputting the mass of the Sun and Earth as well as the distance between the Sun and the Earth and transforming that to 1. Inputting the distance between the Sun and Jupiter and the masses of the Sun and Jupiter in equation (21) gives us 11.86 which is the length of a Jovian year measured in Earth years.

$$\text{Star Distance} = \text{SD} = \text{SR} \times \left((1 - \text{U}_7) \times 21 + \text{U}_7{}^{20} \times 100000\right) \tag{20}$$

$$\text{Orbital period} = \frac{\pi \times \text{SD}^{1.5}}{\sqrt{\text{G} \times (\text{SM}_1 + \text{SM}_2)} \times 1087679925} \tag{21}$$

On to the creation of planets and smaller stars. For binary systems, this will be done once for each star and one time for the barycentre. We need to determine the orbit, mass, size, rotation period and axial tilt for each object.

To determine the orbit we start with the distance to the star. Going by some very general observations of our own solar system we limit this distance to somewhere between 10 and 10 000 times the radius of the star and make closer orbits more likely. In reality it is obviously the mass of the star and planet in question that may set the upper limit of an orbit, but this limit is also hampered by the proximity of other stars pulling a faraway planet out of its orbit. Rather than taking all that into account we choose the easy route and make a reckless approximation that still maintains the illusion of accuracy.

For orbits around one of the stars in a binary system, we set the upper limit at a quarter of the distance between the two stars. For orbits around the barycentre in such a system, we set the lower limit at 1.5 times the distance between the stars and the upper one at 10 000 times the radius of the primary star.

To prevent any two planets from having too similar orbits we start inward and work our way toward the edge, making a planets distance into a function of the previous planets distance (planet i-1). As soon as a planet is generated too far away, it is removed and no more planets are generated. Each potential planet gets its own random number generated from U(0,1) called U_8 which has the same seed as U_5 plus the planets number (i). The planets numbers are determined by the order in which they were generated.

Equation (22) determines the planets distance to the star. $D_{i\text{-}1}$ is the generated distance for the previous planet, so it does not exist for the first one, it is also erroneous when switching from generating one kind of orbit to another in a binary system. If the planet is orbiting a star, this fist $D_{i\text{-}1}$ is therefore set to $10 \times SR$, where SR is the stellar radius measured in solar radii, generated in equation (11), if it is orbiting a binary systems barycentre it is set to 1.5 times the distance between the stars.

The distance in equation (22) is measured in solar radii. For orbits around a lonely star or the barycentre in a binary system, planets getting a distance of more than $10\,000 \times SR$ will be dropped. For orbits around one of the two stars in a binary system, that limit is set to half the distance between the stars,

as stated previously. Once a planet is dropped no more planets are generated for that star/barycentre. In a binary system, orbits will be generated sequentially, e.g. first for the barycentre, then the primary star, and lastly the secondary star, but the planets id-numbers continue counting up as no system should have multiple planets with the same id.

$$\text{Planet i Distance} = D_i = {D_{i-1}}^{1.1} - \left(\log\left({U_8}^{10}\right) - \frac{1}{1600}\right) \times 10 \quad (22)$$

After determining the distance we move on to mass and size. Planets that are too close to their star will lose much of their atmosphere (Tian & Toon 2005), this rules out gas giants close to the star. To mimic this, any planet less than 800 × SR (Jupiter is about 1040) from its star will not be a gas giant. If it is further away than that however, it is likely to be one. So for planets closer than 800 × SR to their star, equation (23) is used to determine mass and for ones further away, equation (24) is used. Both equations express the mass in relation to our sun and require a new number generated from U(0,1) called U_9, the seed may be set to that of U_5 + 100 multiplied with the planet number. To determine the radius expressed in solar radii we use equation (25) which makes smaller planets denser. In order to get some small stars we use equation (26) to replace planets that get a mass over 0.0012 with small stars, but only if the primary star has a radius of more than 0.5 solar radii. The range of possible sizes for the star in orbit is then dependent on the size of the primary star as equation (26) is a function of the primary's generated number (U_4).

This additional generation of stars helps bring up the overall number of binary systems from the 24 percent determined in equation (17) toward one third, which is the real world estimate (Lada 2006).

$$\text{Planet i mass (inner)} = \left(0.13 + 20 \times (U_9 - 0.5)^3 + 2 \times {U_9}^2\right) \times 10^{-6} \quad (23)$$

$$\text{Planet i mass (outer)} = \left(622 + 5 \times (10 \times (U_9 - 0.5))^3\right) \times 10^{-6} \quad (24)$$

$$\text{Planet i radius} = 0.058 \times \exp\left(\left(\frac{\text{mass}}{600}\right)^{0.2}\right) - 0.0665 \quad (25)$$

$$\text{Minor star statistic} = \frac{U_4}{3+10\times U_9} \quad (26)$$

The output from equation (26) is used to determine the characteristics of the minor star in the same way that U_4 was used for the primary. This smaller star could get planets of its own, and the other planets could get moons, but we leave it out of this model lest we get into too many details and repetitious equations.

Knowing the mass and the distance of the planets, we can determine their orbital period with equation (21), we only need to exchange SD with the distance to the planet or minor star derived in equation (22), and SM_1 or SM_2 (whichever one it is not orbiting) with the mass of the planet or minor star. If we have two similar stars and are determining an orbit around the barycentre we keep both SM_1 and SM_2 and add the planets mass at their position (inside the brackets) in the equation.

There are only two things left to generate now: the rotation period and the axial tilt. The rotation period has a relation to the planets mass and the proximity to the star it orbits. More massive planets generally rotate faster than smaller ones. Planets closer to the sun generally rotate slower, up to the point where they become tidally locked, meaning that the same side is always facing the star, just like the same side of the moon is always facing the earth. I.e. the maximum rotation period of a planet is equal to its orbital period (which we call OP_i for planet i). As a matter of fact, the suns gravity is forcing the rotation period of all planets toward their orbital period, it just happens faster with closer objects since the gravitational force is stronger there. Massive planets rotating faster may be due to a historical concentration of mass that sped up their initial rotation speed.

This initial rotation period is a stochastic component, to capture this fact we generate yet another number from U(0,1) called U_{10} which gets the seed of U_5 plus 10^4 multiplied with the planet number. Since a planet may have a negative rotation in relation to its orbit (giving it a negative rotation period), but absolutely not a rotation period equal, or too close, to zero (if a planet were to make a revolution in the blink of an eye it would seize to be a planet) we may transform U_{10} in equation (27) or (28). To determine which transformation should be used we generate yet another number from U(0,1) called U_{10} which gets the seed of U_5 plus 10^6 multiplied with the planet number. If U_{10} is less than 0.1 we use equation (27), otherwise we use equation (28), i.e. we guess that the probability of a planet rotating in the same direction it orbits is 90 percent. This initial rotation period is then altered in equation (29) to account for the distance to the star and the mass of the planet, all rotational periods are measured in Earth years.

$$\text{Initial Rotation Period (negative)} = \text{IRP} = 90 \times \text{U}_{10}{}^{5} - 100 \tag{27}$$

$$\text{Initial Rotation Period (positive)} = \text{IRP} = 0.01 + 100 \times \text{U}_{10}{}^{10} \tag{28}$$

$$\text{Rotation Period} = \text{IRP} \times \left(\frac{\text{SR}^{0.9}}{\text{distance} \times \text{mass}}\right)^{0.5} \tag{29}$$

The very last characteristic we generate is the axial tilt, it can be somewhere between 0 and 90 degrees in relation to the planets orbit. We only have our own solar system to go on here, and from that we determine that minor tilts are more common, but major ones are not unprecedented. Just like the rotation period, planets close to their star are forced into alignment and have very little tilt, while planets further away may diverge more. To mimic this we calculate a distance modifier in equation (30) that shifts the mean tilt closer to 0 for planets near the star. This distance modifier is then accounted for when calculating the axial tilt in equation (31) where we generate our last number from U(0,1) called U_{11} which has the seed of U_5 plus 10^8 multiplied with the planet number.

$$\text{Distance Modifier} = \text{DM} = \min\left(\frac{\left(\text{Distance}/\text{SR}^{0.9}\right)^3}{10^5}\ ,51.42857\right) \tag{30}$$

$$\text{Axial tilt} = \text{DM} \times \left({U_{11}}^5 + 10 \times \left(0.5^5 + \left(U_{11} - 0.5\right)^5\right)\right) \tag{31}$$

7. Discussion

Based on data and some rather general observations we have generated a massive, albeit rather empty, world. We split our artificial world into cubes aligned in a grid and determined the number of stars in every cube in a way that created a spiral shape, reminiscent of a spiral galaxy. This was not a particularly accurate way of doing it as stars tend to be grouped together in clusters and these clusters then form the shape of a much less symmetrical spiral (or whatever shape the galaxy in question has).

When determining the characteristics of the stars we choose to be more accurate even though we limited ourselves to the main-sequence stars. Going by actual observational data we came up with a way of generating the characteristics that would mimic their actual distribution. Lastly we generated additional stars and planets for the solar systems and determined their orbits in a way that, while hardly being accurate, managed to mimic accuracy.

The approach is a mix between analysing observational data to retrieve the best fitting probability distributions for the stars and letting more anecdotal data and astronomical hypothesis inform the probability distributions for things like the likelihood of binary star systems and the characteristics of planets. As our knowledge of other solar systems expand due to new discoveries by the Kepler space observatory and future missions, it ought to be easy to find a much better model for generating planets.

The world generated by the equations in this paper may not be particularly interesting in its current state, but it could be taken so much further using the same principles. We could generate moons, asteroid fields and planetary rings. We could determine the atmospheric and planetary composition of the celestial bodies. We could generate supernovas, pulsars, quasars, black holes and additional galaxies.

Going by the planets position, rotation period, axial tilt and atmospheric composition we could determine temperature and climate. For the right kinds of planets we could then set a probability for life and generate alien plants and creatures. All the planets could get procedurally generated surfaces using the diamond-square algorithm, fractals, Brownian motion, simplex noise and other techniques.

Vladimir Romanyuks SpaceEngine (2015) is a good example of taking a few of those extra steps and using more accurate algorithms.

8. References

Celestia, computer software 2013. Available from: <http://www.shatters.net/celestia>. [18 December 2015].

C++ Resource Network 2015, rand. Available from: <http://www.cplusplus.com/reference/cstdlib/rand/>. [21 December 2015].

Croswell, K 2002. The Brightest Red Dwarf. Available from: <http://kencroswell.com/thebrightestreddwarf.html>. [2 January 2016].

Duric, Nebojsa (2004). *Advanced astrophysics*, Cambridge University Press, Cambridge.

Ierusalimschy, Roberto (2003), *The Mathematical Library*. Available from: <http://www.lua.org/pil/18.html>. [21 March 2016].

Kaltenegger, L & Traub, WA 2009, 'Transits of Earth-like Planets', *The Astrophysical Journal*, vol. 698, no. 1, pp. 519-527.

Lada, CJ 2006, 'Stellar Multiplicity and the IMF: Most Stars Are Single', *The Astrophysical Journal Letters*, vol. 640, no. 1, pp. 60-63.

LeDrew, G 2001, 'The Real Starry Sky', *Journal of the Royal Astronomical Society of Canada*, vol. 95, no. 1, pp. 32-35.

Leadwerks Game Engine, computer software 2016. Available from: <http://www.leadwerks.com>. [21 march 2016].

Palma, C 2014, *Luminosity and Apparent Brightness*. Available from: <https://www.e-education.psu.edu/astro801/content/l4_p4.html>. [12 January 2016].

Parish, J 2007, *Roguish Charm*. Available from: <http://www.1up.com/features/essential-50-rogue>. [18 January 2016].

Sourceware 2015. Available from: <https://sourceware.org/git/?p=glibc.git;a=blob_plain;f=stdlib/random_r.c;hb=HEAD>. [21 December 2015].

SpaceEngine, computer software 2015. Available from: <http://en.spaceengine.org/>. [4 December 2015].

Tian, F & Toon, O 2005, 'Transonic Hydrodynamic Escape of Hydrogen from Extrasolar Planetary Atmospheres', *The Astrophysical Journal*, vol. 621, no. 1, pp. 1049-1060.

Introduction to Random Number Generators, 2007. Available from: <http://www3.nd.edu/~mcbg/tutorials/2006/tutorial_files/randomNum/howItworks.html>. [15 December 2015].

Stellar Luminosity Calculator, 2014. Available from: <http://astro.unl.edu/classaction/animations/stellarprops/stellarlum.html>. [16 January 2015].

Wichman, GR 1997, *A Brief History of "Rogue"*. Available from: <http://www.wichman.org/roguehistory.html>. [18 January 2015].

Appendix 1

```
#######################################################################
#####          Stellar density at central plane of virtual spiral galaxy

win.graph(6,6)
par(las=1, mar=c(4,4,4,4))
resolution = 800
limits = c(-resolution/2, resolution/2)
plot(resolution, axes=FALSE, ann=FALSE, xlim=limits, ylim=limits)
axis(1,-1:1*resolution/2,las=1,lab=c(-5000,0,4999))
axis(2,-1:1*resolution/2,las=1,lab=c(-5000,0,4999))
box()
mtext("Y", side=2, line=3, cex=1.2)
mtext("X", side=1, line=2.5, cex=1.2)
oldpercent=0
for(i in 1:resolution){
    for(j in 1:resolution){
        x = 10000*(i-resolution/2)/resolution
        y = 10000*(j-resolution/2)/resolution
        distance = (x^2+y^2)^0.5
        angle = atan(y/x)
        core = 1 - (distance/200)^2
        arms = exp(-distance/1500)*0.5*sin((0.5*distance)^0.35-angle)^2+
        0.5-distance/10000
        density = max(core,arms,0)
        if(is.na(density)){density = 1}
        color = paste0('gray',round(100-density*100))
        lines(i-resolution/2,j-resolution/2, type='p', cex=0.1, pch=15, col=color)
    }
    percent = round(i/resolution*100)
    if(percent != oldpercent){print(c(percent," %"),quote=FALSE)}
    oldpercent = percent
}
```

Appendix 2

```
#####################################################################
#####                Luminosity transformation parameter estimation

p = c(0,0.76,0.89,0.96,0.99,0.998,0.9999997,1.0000000)
lum = c(0.000158,0.086,0.58,1.54,4.42,21.2,26800,78100000)
diff2 = Inf; result = NA; lum2 = NA; lum1={}
for (i in 1:100){
    for (j in 1:100){
        for (k in 1:100){
            for (l in 1:100){
                a = i/10^5; b = j; c = k/10; d = l/10
                for (m in 1:length(p)){
                    lum1[m] = min(a+b*(-log(1-p[m])/c)^d, 78100000)
                }
                diff1 = sum(abs(1-lum/lum1))
                if (diff1 < diff2){
                    result = matrix(c(a,b,c,d), nrow=4,ncol=1)
                    lum2=lum1
                    A=a; B=b; C=c; D=d
                    diff2=diff1
                }
            }
        }
    }
    print(c(i,"%"), quote=FALSE)
}
par(mfrow=c(1,2));lumseq={}
a="log(Luminosity)";b="log(Generated Luminosity)"
for (m in 1:10000){lumseq[m] = min(A+B*(-log(1-m/10000)/C)^D, 78100000)}
plot(p,log(lum), xlab=expression('U'[4]),ylab=a)
lines(1:10000/10000,log(lumseq),col='red')
plot(log(lum),log(lum2),type='o',xlab=a,ylab=b)
abline(c(0,0),c(1,1),col='red')
comparison=matrix(c(lum,lum2),nrow=length(p),ncol=2)
dimnames(comparison) = list(p,c("Actual","Estimate"));comparison
dimnames(result) = list(c("A =","B =","C =","D ="),"Estimates:"); result
```