UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Simon Prii

Procedural Generation of Gas Giants

Bachelor's Thesis (9 ECTS)

Supervisor: Mathias Plans, MSc

Tartu 2025

Procedural Generation of Gas Giants

Abstract:

This thesis presents a new way to procedurally generate gas giants. First, real life gas giants are briefly studied as well as previous work on the topic assessed. Then an algorithm is developed to improve upon previous work by first mapping textures on a sphere. The simulation is then brought to life by giving it movement based on velocity vecors. Then the previously studied visuals of gas giants are added using different methods of implementation. Finally configuration is added to be able to export the simulation as well a demonstration application to show the algorithm at work.

Keywords:

Procedural generation, texture mapping, gas giants

CERC: P170 Computer science, numerical analysis, systems, control

Protseduuriline Gaasihiiglaste Genereerimine

Lühikokkuvõte:

Lõputöö kirjeldab uut viisi kuidas protseduuriliselt gaasihiiglasi genereerida. Esimesena uuritakse lühidalt tegelikke gaasihiglasi ja hinnatakse varasemalt tehtud töid. Seejärel luuakse uus algoritm, mille eesmärk on varasemaid töid parendada. Seda tehakse esmaselt tekstuure sfäärile vastendades. Seejäral tuuakse simulatioon ellu, andes sellele kiirusvektoritel põhineva liikumise. Järgmisena lisatakse gaasihiiglastel leiduvad visuaalid. Lõpuks lisatakse konfiguratsioon simulatsiooni tekstuuri salvestamiseks ja näidisrakendus algoritmi demonstreerimiseks.

Võtmesõnad:

Protseduuriline generatsioon, tekstuuri vastendamine, gaasihiiglased

CERC: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Table of Contents

1. Introduction

Before recent years the documentation and research on space giants like gas giants has been quite a challenging task. Still, space and planets are regularly depicted in movies, games and other media. That has been made possible because of continuous progress in the field of computer graphics and simulations. Over the years the simulations have gotten more and more realistic by using improved algorithms and computing power. Yet most simulations still lack some qualities and aspects. For example, different natural phenomena found on gas giants, missing configurability as well as immersion breaking visual issues. This thesis aims to fill these missing gaps and approve upon already existing simulations. The result is a new algorithm that negates previously mentioned shortcomings.

This thesis is split into five main parts. First, different storms and natural phenomena are briefly studied to get a better understanding of gas giants. Secondly, previously done work is reviewed by highlighting methods of implementation as well as shortcomings. Afterwards, a new and improved algorithm is devised that solves these shortcomings. Then, the algorithm is showcased using a demonstration application, and finally, results and improvements are analysed.

In Appendix one the accompanying files are described. These files include the source code for the demonstration application as well as a few visuals of generated gas giants. Files located in the source code folder are explained and include a small manual for the use of the demo application.

ChatGPT and other large language models (LLMs) were used in the synthesis of this thesis by using them as a learning aids.

2. Natural Phenomena Found on Gas Giants

To simulate effective gas giants, a better understanding of existing ones is needed. Gas giants prohibit a selection of different natural phenomena that might not be found on other celestial bodies. Luckily our solar system is home to four different looking gas giants: Jupiter, Saturn, Neptune and Uranus. In this chapter a closer look at Jupiter and Saturn is given to get a better understanding of what kind of phenomenon they prohibit. Specifically a brief overview is given of the Great red spot and the jets that can be seen on Jupiter, the ring of Saturn and finally the hexagon shape found on the north poles of Saturn and Jupiter. This info can be used later on, to create a more diverse and meaningful simulation.

2.1. Jets

One of the most distinct features of Jupiter is its jets. These jets are organized into a large eastward flowing super jet and differentiating sequence of smaller eastward and westward jets as seen on Figure 1. This pattern has been constant throughout the years of observation[1]. The jets are created by the heating and cooling of gases in Jupiter's atmosphere. Gases are heated by the core of Jupiter as well as sunlight hitting the surface of Jupiter. Looking at Figure 2, part a shows the zonal velocity captured by spacecraft *Cassini* and part b shows a controlled simulation of velocities caused by Jupiter's inner heat (magenta) and solar heat (light blue). This shows that the direction of the larger eastward jet strongly depends on the inner heat of the planet. Without the rising inner heat, the jet would be moving westward instead.

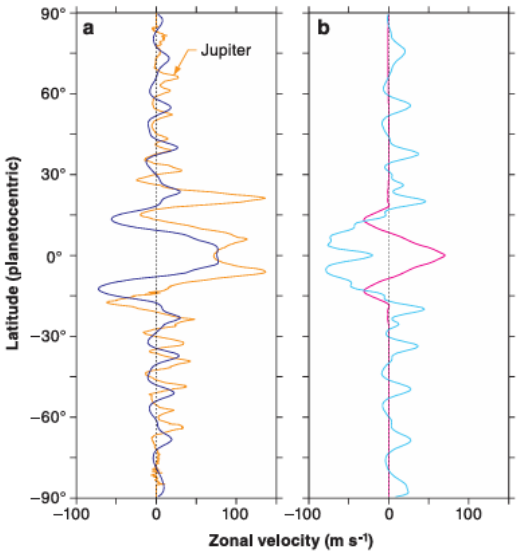


Figure 1. Movement of gas based on heat souce[1].

[1] https://doi.org/10.1175/2008JAS2798.1

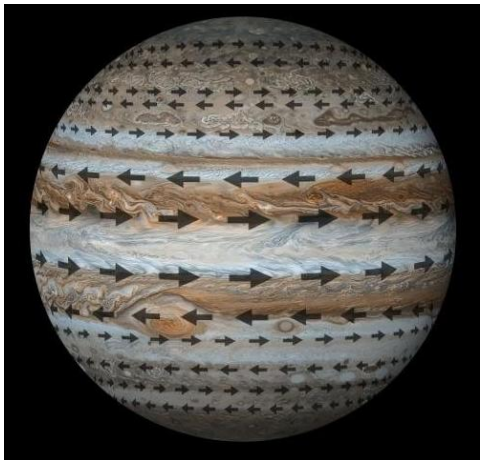


Figure 2. Movement of eastward and westward jets[2]

The movement of rising warm and declining cold gases also creates smaller storms in between the jets. These small storms transport angular momentum from westward to eastward jets[1].

2.2. Great Red Spot

The Great Red Spot (GRS) is arguably one of the most notorious storms found on gas giants to date. The storm has been studied throughout the years by a lot of different spacecraft including Juno[3], which has been orbiting the gas giant since 2016. The taken images of the GRS show that the vortex has a slower rotating inner region and a faster rotating outer region.

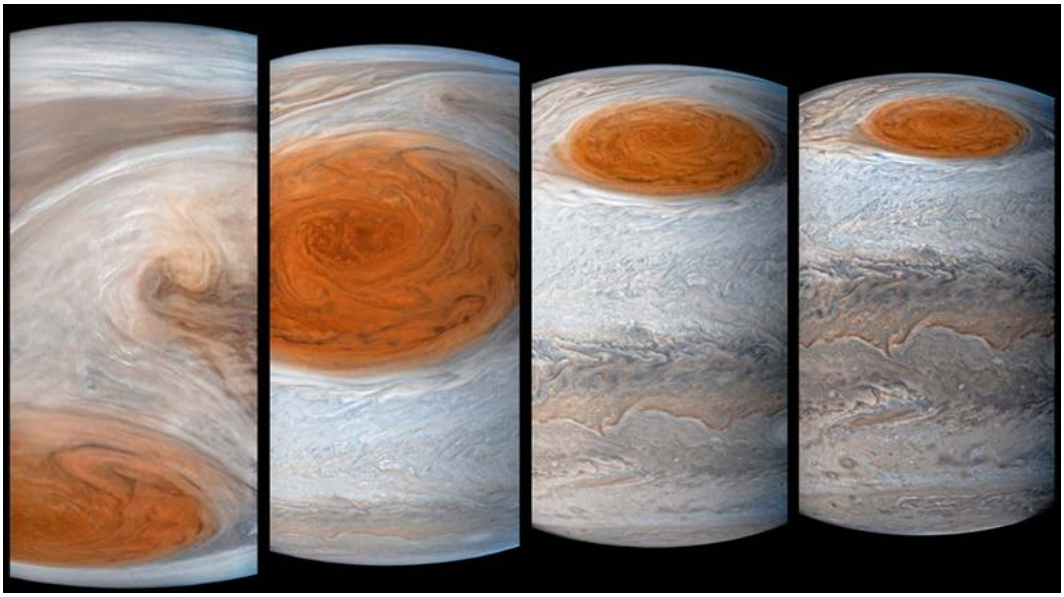


Figure 3. JunoCam images of GRS[3]

[2] https://www.jpl.nasa.gov/images/pia24964-jets-at-jupiter/
[3] https://iopscience.iop.org/article/10.3847/1538-3881/aada81

The JunoCam took images of the GRS with wavelengths red green and blue (RGB) which were later combined to create four colored images as demonstrated by Figure 3.

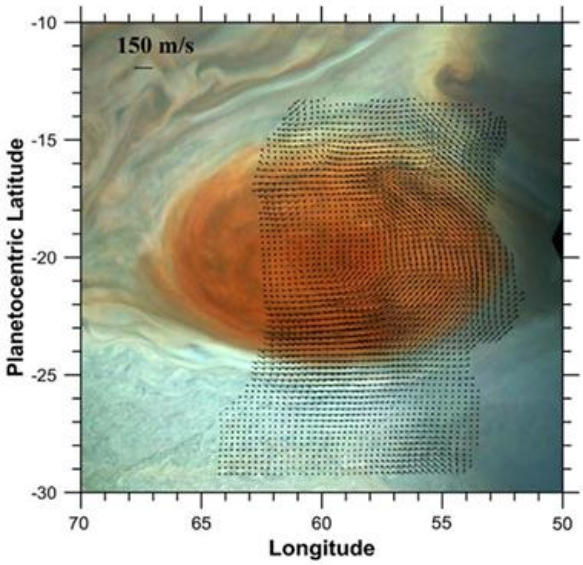


Figure 4. GRS wind field vorticity map[3]

Figure 3 gives a good idea, how huge vortices of similar gas can look like by showing how the colors of the storm can look completely different from the rest of the environment. Figure 4 shows a brief but detailed vorticity map on the movement of gases. The vorticity map helps give an idea how the wind acts near and inside the GRS. It also helps explain the mixing of different gases in the storm.

2.2 Northern Polar Storms

Saturn's North Polar Hexagon[4], shown on Figure 5, was first discovered by Godfrey in the last century using the images from the Voyager spacecraft. There has been speculation[5] that the hexagon is formed by anticyclones present near the north pole that mold the dominant circulation flow into a hexagon shape. Something similar has been discovered at Jupiter's northern pole.

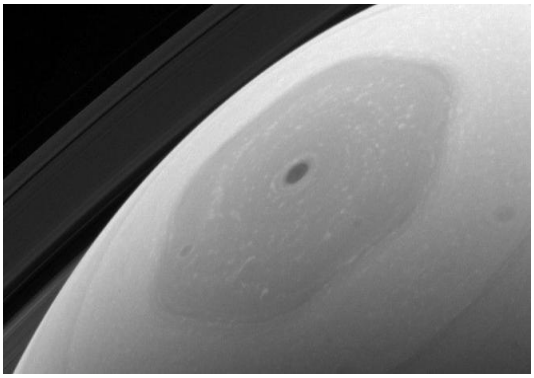
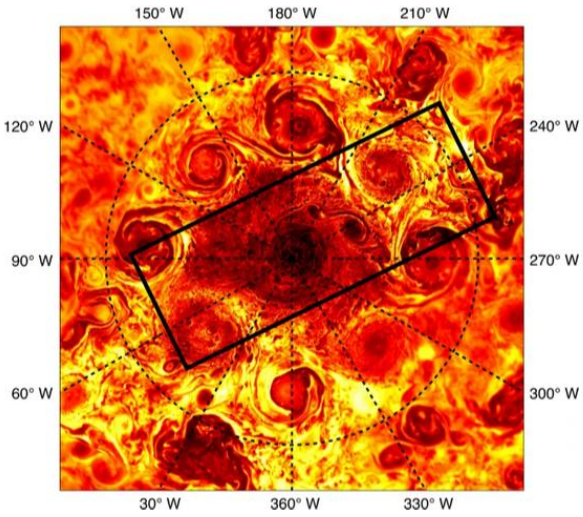


Figure 5. Saturn's polar hexagon[4]

[4] https://science.nasa.gov/mission/cassini/science/saturn/hexagon-in-motion/
[5] https://academic.oup.com/mnras/article/469/4/4133/3828084

For Jupiter, the presence of smaller storms is confirmed by the images taken by Junos Infrared camera[6] as seen on Figue 6.

From the image various sized vortices can be seen surrounding ‘a single polar cyclone.’ The vortices have a diameter of approximately 5,000 km as smaller cyclones have a diameter of about 1000 km. The smaller vortices with the diameter range of about 100 km and less survive for only about a couple of hours to several days, while the 1000 km to 5,000 km last years[6]. This information can be used later on, to increase the realism aspect of the simulation.



Figue 6. Infrared image of Jupiter's northern area[6]

2.3 Planetary rings

It is believed that our solar system hosts at least 6 planetary ring systems. This includes the ring system found on Saturn, Jupiter, Neptune and Uranus[7]. The most known and visible of the planetary rings is undoubtedly Saturn's ring system[8] shown on Figure 8. The ring system comprises a multitude of rings that contain different sized particles or so called space debris. The space rocks and dust particles of the debris have different velocities and although the rings look uniform there are a lot of collisions and chaos happening on a smaller scale. Bigger particles influence other particles through gravity as well as magnetic fields.

Figure 7. Saturn's rings[8]

Figure 8. Saturn's shepherd moon Daphnis[8]

[6] https://www.nature.com/articles/s41567-021-01458-y
[7] https://baas.aas.org/pub/2021n4i258/release/1
[8] https://www.planetary.org/articles/best-pictures-saturn-rings

These fields and changes influence the density of different parts of the rings[9]. For this reason, although Saturn seems to have one giant ring, it is actually a multitude of rings varying in densities and sizes as well as distance from the central gas giant. Figure 7 shows that the lines or so called gaps in between the rings are formed by Saturn's orbiting moons. These moons influence the look of the ring system by pulling smaller debris from the corners of the rings with a stronger gravitational pull than other particles found in the ring system[9].

Having a clearer understanding of various phenomena observed on gas giants, it is now possible to more effectively evaluate previously done work. In the following chapter previous work is viewed and shortcomings discussed.

[9] https://oulurepo.oulu.fi/bitstream/handle/10024/37670/isbn978-952-62-2116-8.pdf?sequence=1&isAllowed=y

3. Previously Created Simulations

Procedural generation of gas giants has been done before using a number of different methods. This chapter will touch upon the previously done work and describe the algorithms used. The peculiarities and shortcomings of these simulations are pointed out and explored. All of the methods described use noise[10] as their base, to add a variable and natural feel to the simulations. In the context of computer graphics, noise refers to a random pattern of values commonly used to add variation to visual elements as best seen on Figure 9. Noise is also continuous, which means it can have derivatives. This is important because of how some of the simulations are made.

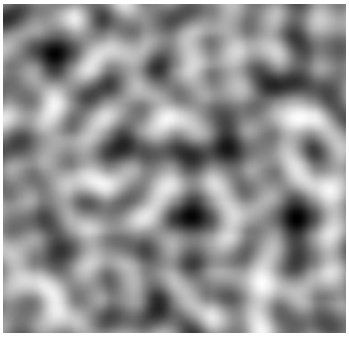


Figure 9. Gradient noise [10]

3.1 Procedural Generation: Simple, Shader-based Gas Giants (and other planets)

A post by Stefan Strömers, username "kchnkrml" [1], includes a short clip of shader simulating the moving texture of a gas giant. For the base of the shader he uses mainly noise called fractal Brownian motion (from now fBm). This algorithm is used to imitate the movement and waviness seen on the gas giant simulation on Figure 10 [1]. As seen on Figure 11, This is achieved by layering multiple instances of noise together with each of them having a smaller amplitude.

Figure 10. Gas giant simulation made by Stefan Strömer [1]

[10] https://gamedevframework.github.io/v0.2.0/noise.html

The simulation also includes five different colors, from which three are combined into one. The colors are combined with the waviness made by fBm and differ depending on the location of the wave [1]. Lower values of fBm correspond to lighter colors and higher values to darker values.

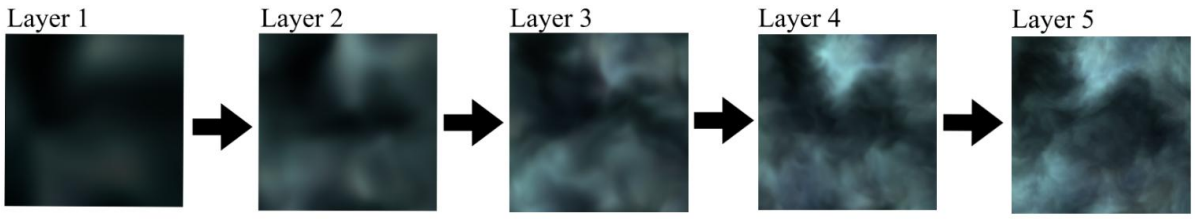


Figure 11. Multiple layers of fBm

Although Stefans simulation gives off an effective visual, it is missing most of the natural phenomenons found on a gas giant. It is missing jets, storms, rings, as well as a polar storm, which would all give the simulation a more meaningful visual.

3.2 Procedural Gas Giants

An article published by Barthélemy Paléologue in 2023 [2] describes the procedural generation of a gas giant seen on Figure 13 using squirrel noise and domain warping. Barthélemy Paléologue uses "squirrel noise" to determine the randomness of the simulation, but mentions that the choice of noise is not necessarily important. To mitigate the effect of randomness and have more control over the noise, Barthélemy recalculates the values obtained from the noise using the Box-Muller transformation. Then he uses the domain warping technique on the resulting noise texture. Domain warping is a type of recursive noise, where the output of one layer of noise is the input of the next[11] as demonstrated on Figure 12.

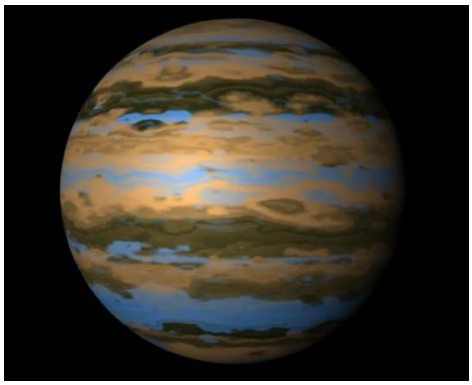


Figure 13. Barthélemy Paléologues simulation of a gas giant [2]

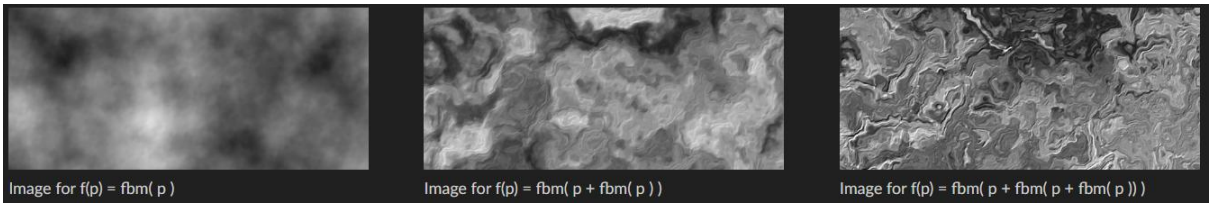


Figure 12. Domain warping demonstrated with fBm[11]

[11] https://iquilezles.org/articles/warp/

For the colors of the simulation, Barthélemy Paléologue chooses three different colors from the HSV (hue, saturation, color) color space: two lighter and one darker. For each point on the sphere, the noise value at that point is used three times. Once in the domain warping technique and two other times to achieve a suitable color. This gives off a similar look to Stefan Strömers gas giant with the illusion of depth in the simulation. Finally, the article points out the horizontal cloud bands on Jupiter, which are created by stretching the generated noise [2]. This leaves the simulation with missing storms and other visuals commonly found on real gas giants.

3.3 Procedural Generation of Gas Giant Planets

A talk by Stephen M. Cameron [3], username "smcameron", describes the procedural generation of gas giants using a noise called curl noise. The aforementioned noise is used to simulate a circulation effect of gas. Cameron simulates millions of tiny particles and makes them move based on the previously mentioned curl noise. This use of separate particles for a simulation is called the Lagrangian approach[12]. Using the Lagrangian approach, the calculations for each particle are done separately, which makes it quite computation heavy. The particles move around and act like tiny markers that draw the final texture. More specifically the movement of particles is determined by the velocity field[13] provided by the noise. A velocity field can be thought of as a grid of vectors, where each vector shows where the particle should move next. This is best illustrated on Figure 14. The longer the vectors are the further the particles move creating the illusion of an area with a faster movement.

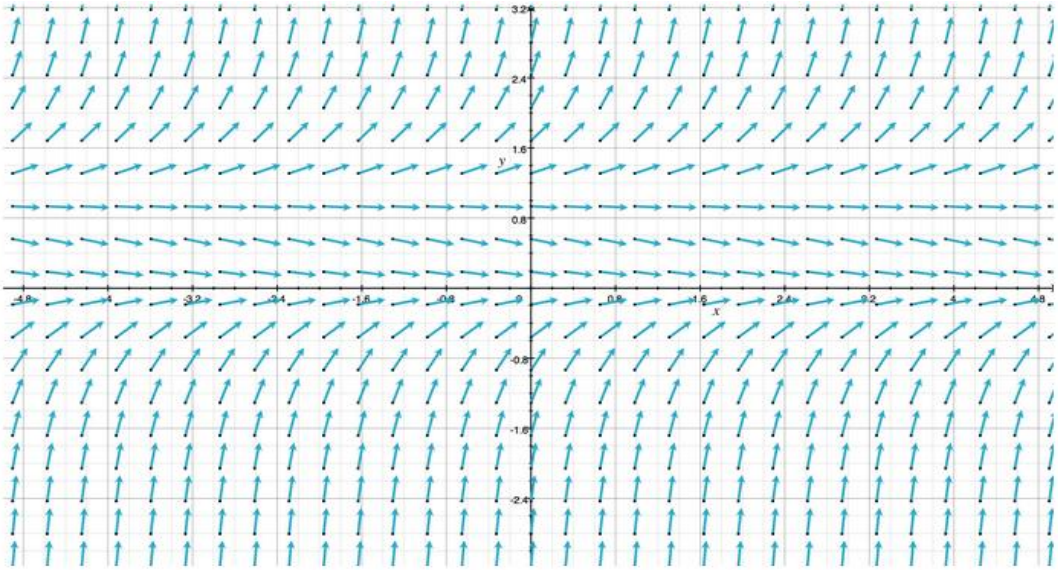


Figure 14. Velocity field[13]

[12] https://www.sciencedirect.com/science/article/abs/pii/S1352231014000946
[13] https://www.khanacademy.org/math/multivariable-calculus/thinking-about-multivariable-function/ways-to-represent-multivariable-functions/a/vector-fields

He also describes the process of mapping the simulation on a sphere [3], adding in a thing called a cubemap[14] shown on Figure 15. As demonstrated on Figure 16, a cubemap is commonly used in computer graphics to map textures to surfaces and is a form of a texture map[15]. It consists of 6 cube faces, where each of them is a 2D texture. It is commonly used to implement environmental mapping or skyboxes, but it is also used to put textures on spherical meshes. In this case a cubemap is used to project the drawn sandy looking texture to a sphere, giving it an appropriate color scheme. Figure 17 shows the final output of the simulation, making use of a sandy cubemap[16].

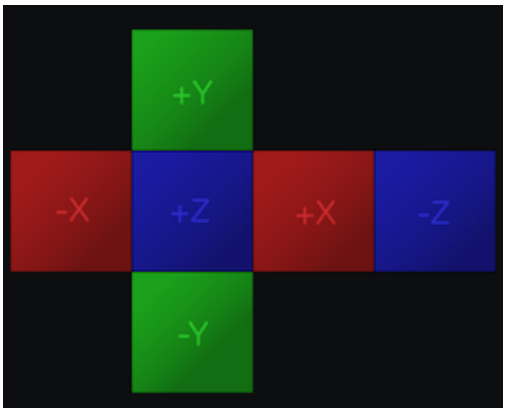


Figure 15. Cubemap texture[14]

Figure 16. Texture map[15]

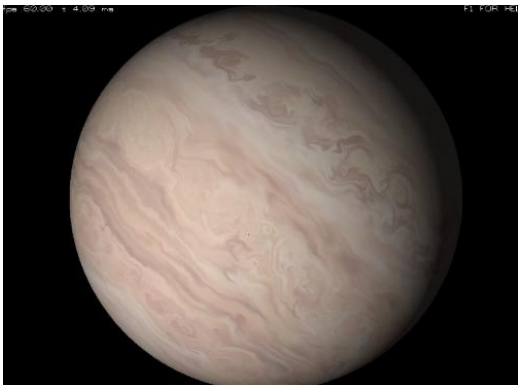


Figure 17. Simulation made with curl noise[16]

Stephen M. Cameron generates a texture based on curl noise and uses tiny particles that move based on that noise to draw the final texture. He then applies the made texture to a texture map called cubemap. Cameron wraps the texture map on a sphere which gives off an effect of a gas giant [3]. Although the simulation gives off a visually appealing result, it can only be executed on a CPU. The problem of using a CPU is that it is not great for parallel tasks, which a particle simulation. However visually effective, the simulation is still missing planetary rings as well as polar visuals.

[14] https://docs.unity3d.com/550/Documentation/Manual/class-Cubemap.html
[15] http://www.vallaste.ee/
[16] https://imgur.com/a/gaseous-giganticus-output-9LipP

3.4 Work by Emil Dziewanowski

Emil Dziewanowski is a technical artist whose personal portfolio page has well explained and documented development of a gas giant like simulation [4]. The development consists of two main parts. The creation of the simulation part and the mapping of the simulation part.

In the simulation section he first describes a number of different methods of simulating motion. The Euler method is used to simulate particles and their movement[12]. It works in a similar way to the Lagrangian method that was previously used by Stephen M. Cameron. When the Lagrangian method uses separate particles, the Euler method uses groups of particles. This makes it faster but less accurate.

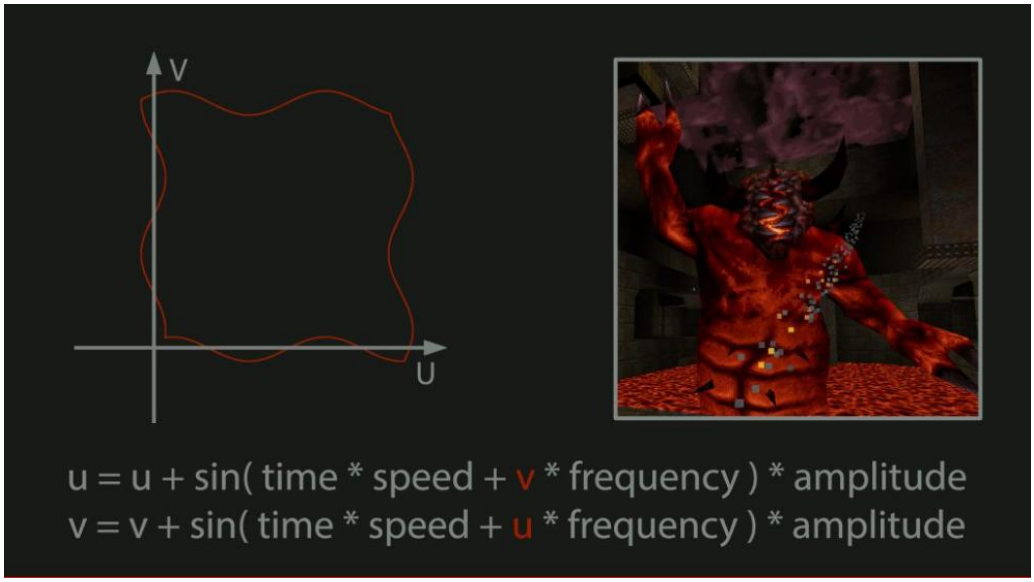


Figure 18. UV distortion [4]

The next section of Emils simulation touches upon the problems associated with curl noise. The first being that with stationary noise the simulation gets stuck. The vortices move at first, but after a while the movement stops. Emil fixes this by adding distortion to the noise making it move in a wavy pattern as seen on Figure 18. This changes the simulation to a more chaotic one creating and dissolving vortices. The second issue is the blending of colors. Mixing is used to keep the simulation smooth. The problem arises when after a while all the colors of the original texture have been mixed and blended into one single color thus hiding the movement of the simulation. The fix, as shown in Figure 19, involves sampling the original texture at certain intervals. This keeps the colors of the original texture more present in the simulation. The sample points are then blended into the simulation by adding previously mentioned distortion and a noise texture. The final issue he encounters is the blurring of the texture when

blending.To compensate for the occurring loss of detail he uses modified sharpening. This causes some artefacts which he removes by limiting the modified values to a certain range.

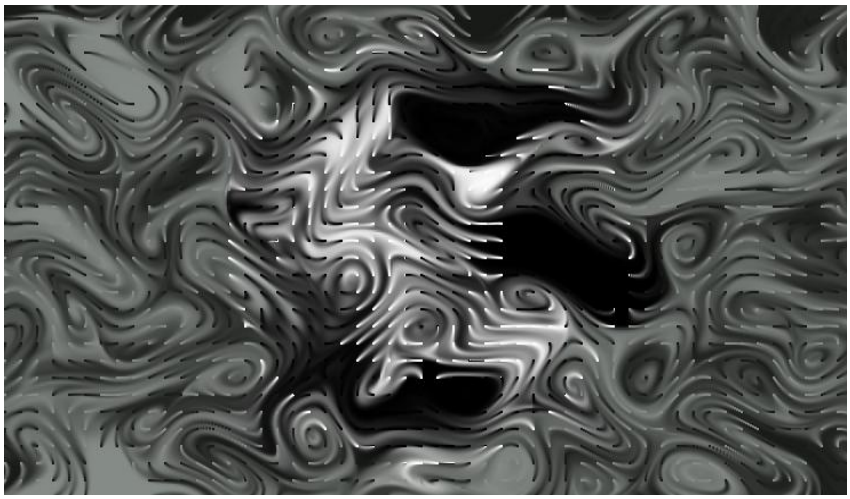


Figure 19. Sampling at certain intervals [4]

Last part of Emils simulation deals with creating different flow patterns to use in the simulation. Specifically he speaks about cyclones, jets and storms. Cyclones are achieved by creating a larger area in the noise where rotation is applied. This is similar to what is happening in the original curl noise simulation mentioned before, but on a bigger, more notable scale. The jets are achieved by creating blurred stripes in the noise to simulate fast movement of gases. In real life opposite flows usually create small vortices or curls. Because the curls aren't created naturally, they are simulated with small areas, where curls are applied. The final flow pattern is storms. Emil archives these storms by essentially using noise to create a chaotic looking flow. This is also the base of the simulation.

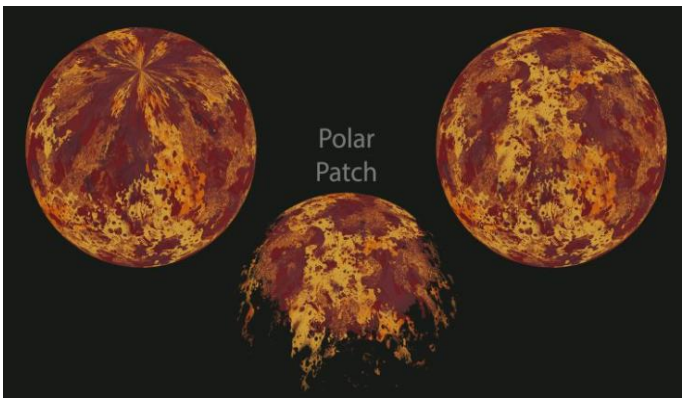


Figure 21. Polar patch [4]

Figure 20. Visible seam on sphere[17]

The mapping of the simulation is done using a sphere map. He dismisses the use of a cubemap due to high performance costs. The use of a sphere map requires only one 2D texture and is thus considered the more optimal option [4]. This is the better option when taking into account

the performance but comes with a cost. This being the seam[17] on where the texture starts and ends seen on Figure 20 as well as an unnatural distortion on the south and north pole of the sphere seen on Figure 21. He fixes the south and north pole distortions by using a patch of texture to overlay the problematic area. The seam however is not fixed but hidden on the other side of the sphere or planet [4]. The final result of Emils simulation can be seen on Figure 22, where a purple color scheme is used.

Figure 22. Emil's simulation of a gas giant [4]

Although all four previously mentioned projects used efficient ways to generate gas giants, they all have some kind of shortcomings. Using fBm gives the gas giant simulation an interesting depth, but it still lacks the motion and patterns similar to existing gas giants. Simulations using curl noise seem to fix this but have their own issues regarding how the particles are managed. Completeness would certainly be added with adding various weather effects as well as fixing general issues with existing algorithms.

[17] https://www.jig.com/help-center/creating-immersive-360-images-with-skydomes

4. Algorithm

This section will go over the logic behind the development of the algorithm. To move the gas, the algorithm uses velocity fields as did the previous simulations of Emil Dziewanowski [4] and Stephen M. Cameron [3]. Because we are dealing with computer graphics and mostly particles, the simulation should mainly run on a GPU. This approach has better rendering times and particle movement compared to running the simulation on a CPU. Because of this, most of the particle movement logic will be written in shaders. Shaders lend themselves well to the Eulerian simulation method, therefore it is used in this algorithm.

4.1 Shaders

Although this algorithm can be run on CPU, the main advancement of this work is that it is stupidly parallelizable. Therefore it is implemented on GPU shaders. Shaders are pieces of code that execute on GPUs. They control two parts of the standard 3D pipeline. Firstly, they can change the location and attributes of the 3D primitives such as points, triangles, triangle fans, etc. Secondly, after rasterization of the primitives, they control the color of the pixels which will be presented on the screen or put into a rendered texture. This shader stage is called fragment shader or pixel shader and will be heavily used to speed up this algorithm.

4.2 Simulation mapping

Fragment shaders can render onto 2D textures, which means that 2D coordinates are used to identify each pixel. One of these coordinates is called UV coordinates, which are mostly used for texture sampling. UV coordinates are normalized, which means that they range from zero to one. Because the Eulerian simulation method is used, each rendered pixel can be thought of as a group of gas particles on the planet and all of them have a color and a unique UV coordinate. Usually, UV coordinates start at the lower left corner with coordinates $(0, 0)$, or upper left corner depending on the graphics library, and end with $(1, 1)$. The UV coordinates are illustrated on Figure 23.

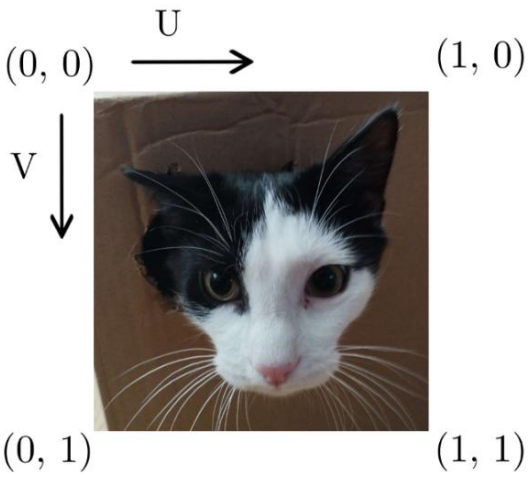


Figure 23. UV coordinates shown on a texture

Gas giants are spherical. Therefore the simulation of them should also be done on the surface of a sphere. This means that it is happening in three dimensions. An algorithm is needed to change the 2D UV coordinates to 3D UV coordinates. There are two types of 3D UV coordinates. The first type is for a 3D texture, then 3D UV coordinates can be similar to 2D UV coordinates but with an extra dimension. The second type is for a cubemap texture, where the 3D UV coordinates are vectors that show the sampling position from the center of the cube. This is illustrated in Figure 24, where the magenta vector goes through the magenta square, which is the area where the texture is sampled. Cubemaps were used by Cameron [3] in his simulation and they will be used in this algorithm as well, although in a different way. 3D UV coordinates do not have to be normalized, but when they are, they are also surface normals of the unit sphere.

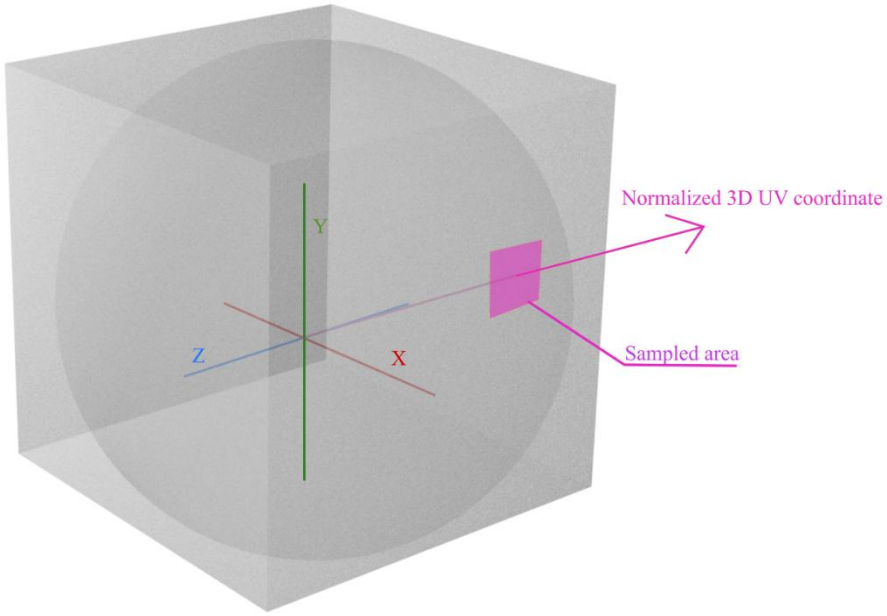


Figure 24. Cubemap texture sampling vector

For each cubemap face, the corresponding UV coordinates need to be translated to 3D space separately and in the correct order. The overview of the algorithm is seen on Figure 25. Firstly, a 2D UV coordinate is transformed to a 3D vector as if it were on a 3D unit cube. Then, that vector is normalized, meaning that it becomes a location on a unit sphere. As discussed before, this location can be used as a 3D UV coordinate, a 3D location for the simulation, and as a surface normal.

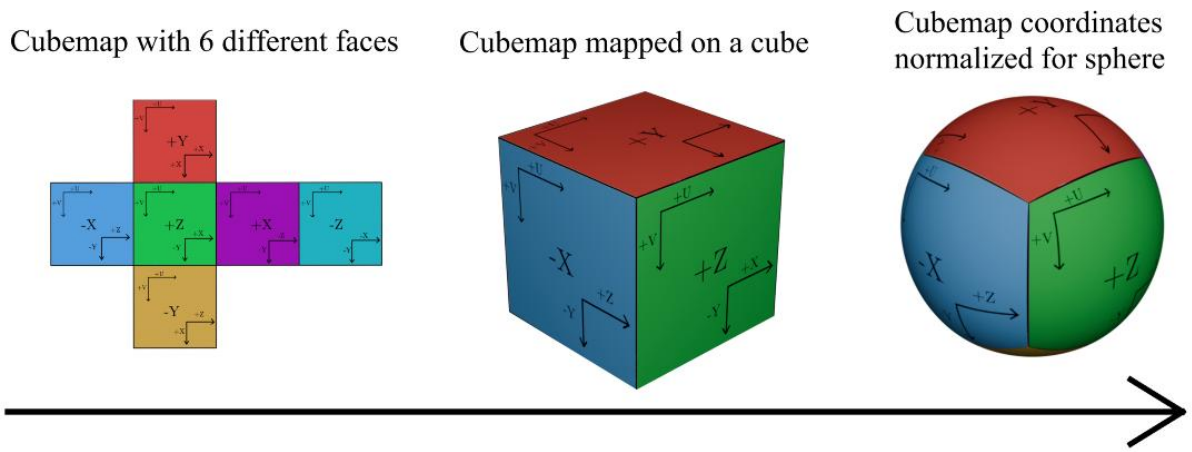


Figure 25. General idea of mapping a cubemap on a sphere

The most complex part of this algorithm is the move from the 2D UV coordinate of the cubemap face to a 3D location on the unit cube. The first step of this transformation is the mapping of the UV axis from $[0, 1]$ to $[-1, 1]$. As seen on Figure 26, this moves the coordinate $(0, 0)$ from the corner to the center of the face. This is done because the unit cube has to be centred at $(0, 0, 0)$, meaning that it has corner coordinates $(\pm 1, \pm 1, \pm 1)$.

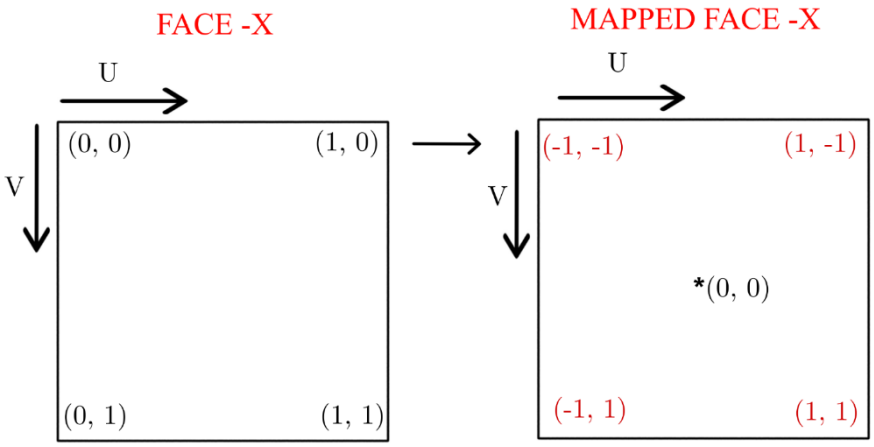


Figure 26. UV coordinates of a face from [0, 1] to [-1, 1].

The rest of the steps of the algorithm are illustrated on Figure 27. In step (1), the UV coordinate is brought into 3D space by adding a third coordinate and setting it to 0. In step (2), the rotation is determined by a change of base matrix M, which changes the coordinates so that they align with the 3D cube that the texels are being mapped to. As illustrated in Figure 28, for face -X, the matrix will translate the coordinates (u, v, 0) to (0, -v, u) by changing the base of U to +Z and the base of V to -Y. This will be done for each face separately using the corresponding change of base matrix.

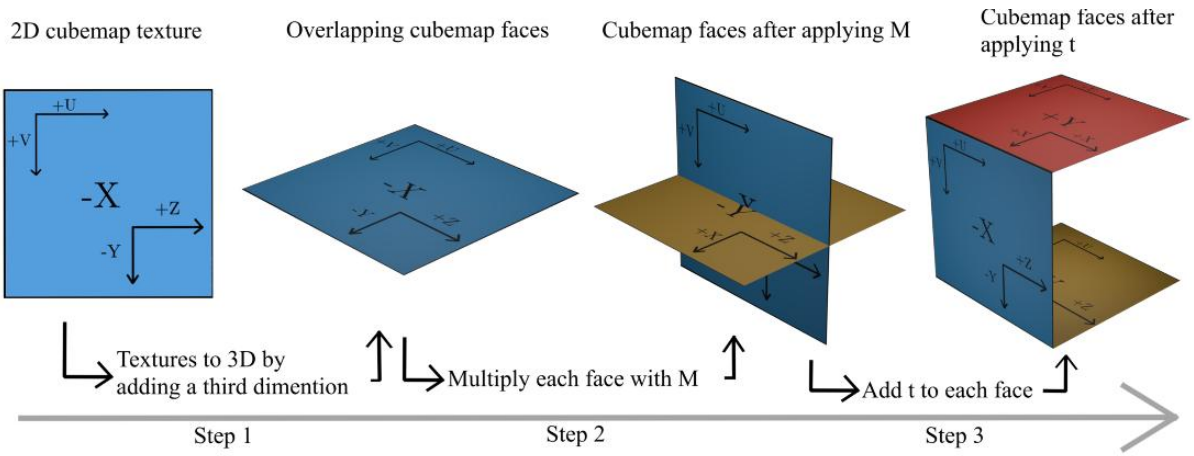


Figure 27. Mapping of a cubemap

Finally, in Step (3), the resulting coordinate is then given a unique location on the unit cube by adding a vector t. Vector t is a signed basis vector matching one of the faces of the cubemap texture (-X, +X, -Y, +Y, -Z, +Z). For example the face -X corresponds with coordinates (-1, 0, 0). As the last step, to get the 3D position on a unit sphere, the vector is normalized.

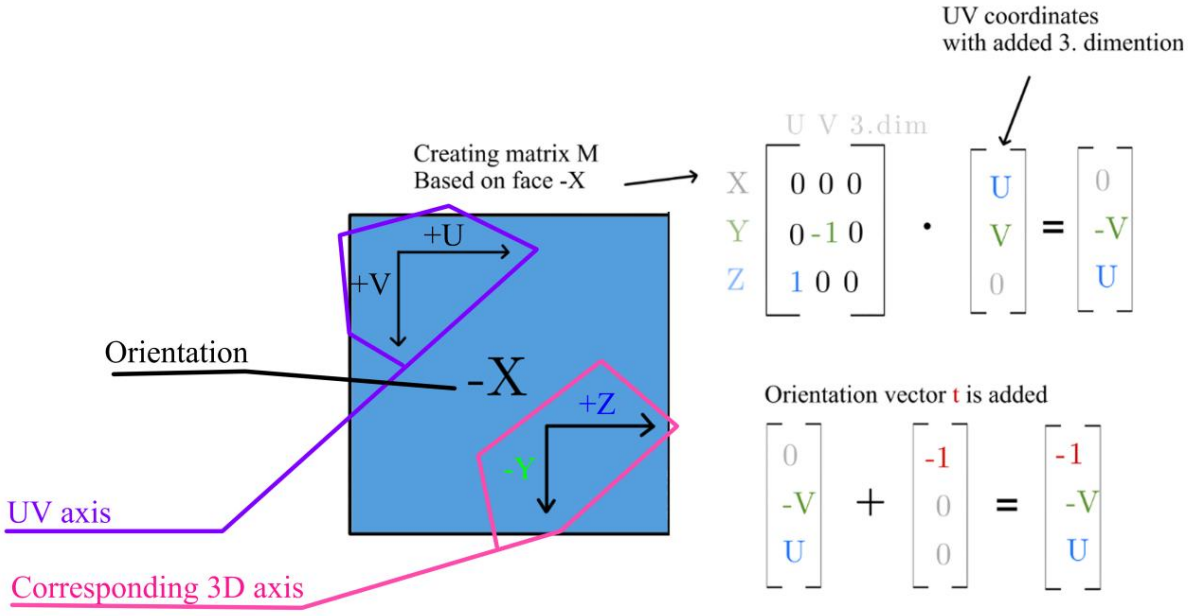


Figure 28. Cubemap face calculations

An example of the use of this algorithm is seen in Figure 29. In the first step, point (0.2, 0.7) is highlighted in blue. It is then mapped from [0, 1] to [-1, 1] in the second step, which results in point (-0.6, 0.4). Then, a third dimension is added in step (3) to get (-0.6, 0.4, 0.0). To bring the texel to a unique 3D space coordinate the vector is multiplied with M in step (4). In this case, the matrix for face -X is used, which is the matrix in Figure 28.

The result is (0, -0.6, -0.4). In step (5), the orientation vector t is added to get the coordinates (-1.0, -0.6, -0.4). In the final step the vector is normalized to get the position on the unit sphere, which brings the final mapped coordinates of point (0.2, 0.7) to (-0.81, -0.48, -0.32).

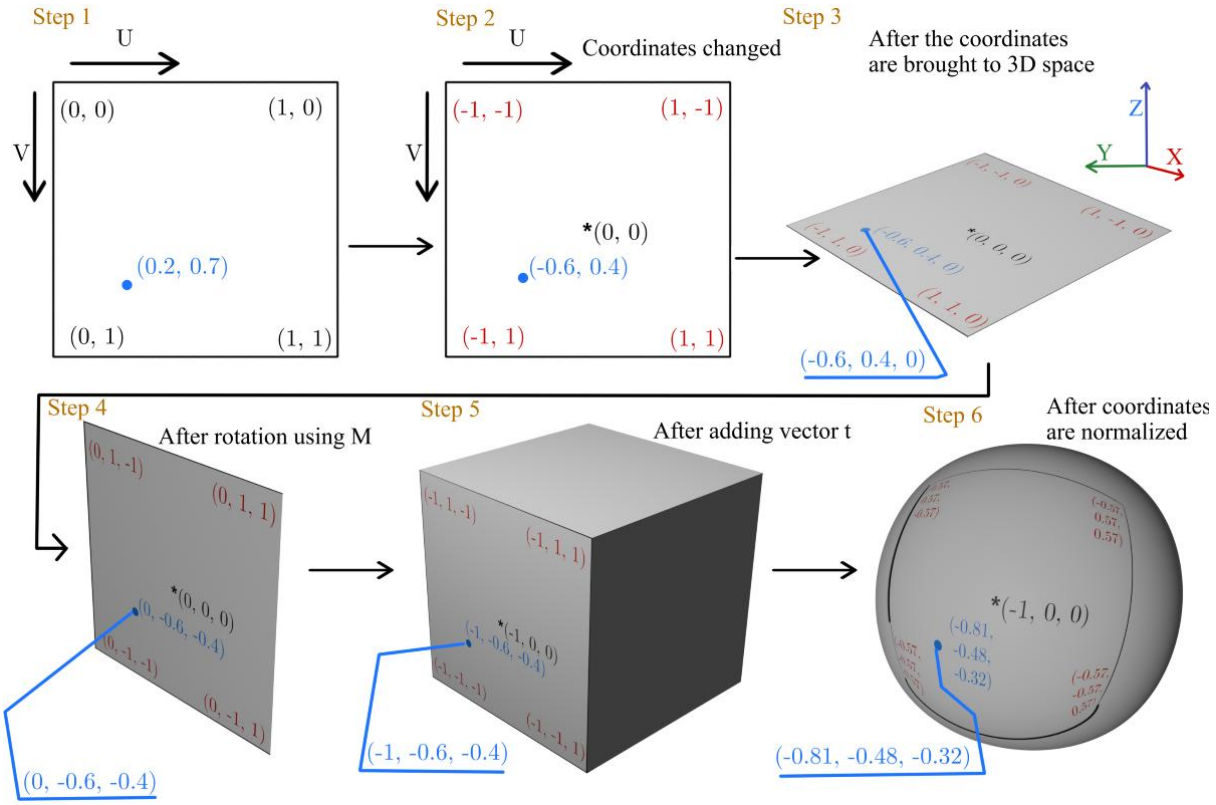


Figure 29. Cubemap face UV coordinates translated to 3D coordinates on a sphere

4.3 Simulation

Now that the gas particles have been given 3D locations, the next step is to make them move and blend with other particles. For this, velocity fields are used. A velocity field is a field of velocity vectors which determine the speed and direction of the particles at their locations. The velocity field is not constant. It is sampled each iteration and can be different each time.

Because pixel shaders cannot render into the same texture as the one they are sampling from, a double buffer pattern is used, which uses two cubemap textures as buffers. The back buffer is used to sample the color from a previous iteration and the front buffer is used as the output of the current iteration. The whole simulation iteration is illustrated in Figure 30. The pixel (or the particle) which is rendered is marked with green. In step (1), the location of the particle is mapped onto a sphere, as described in 4.2 Simulation mapping. Next, the velocity field is sampled at that location to get a velocity vector, which is colored blue. It is added to the location of the particle to get the sampling location X. In step (3), the sampling location is used to sample the color from the previous iteration. That color is the output of the algorithm. Finally, in step

(4), the buffers are switched for the next iteration, making the output of this iteration the input of the next iteration. As an optional step (5), the output can be visualized on the screen by texturing a sphere with the contents of the newly switched back buffer.

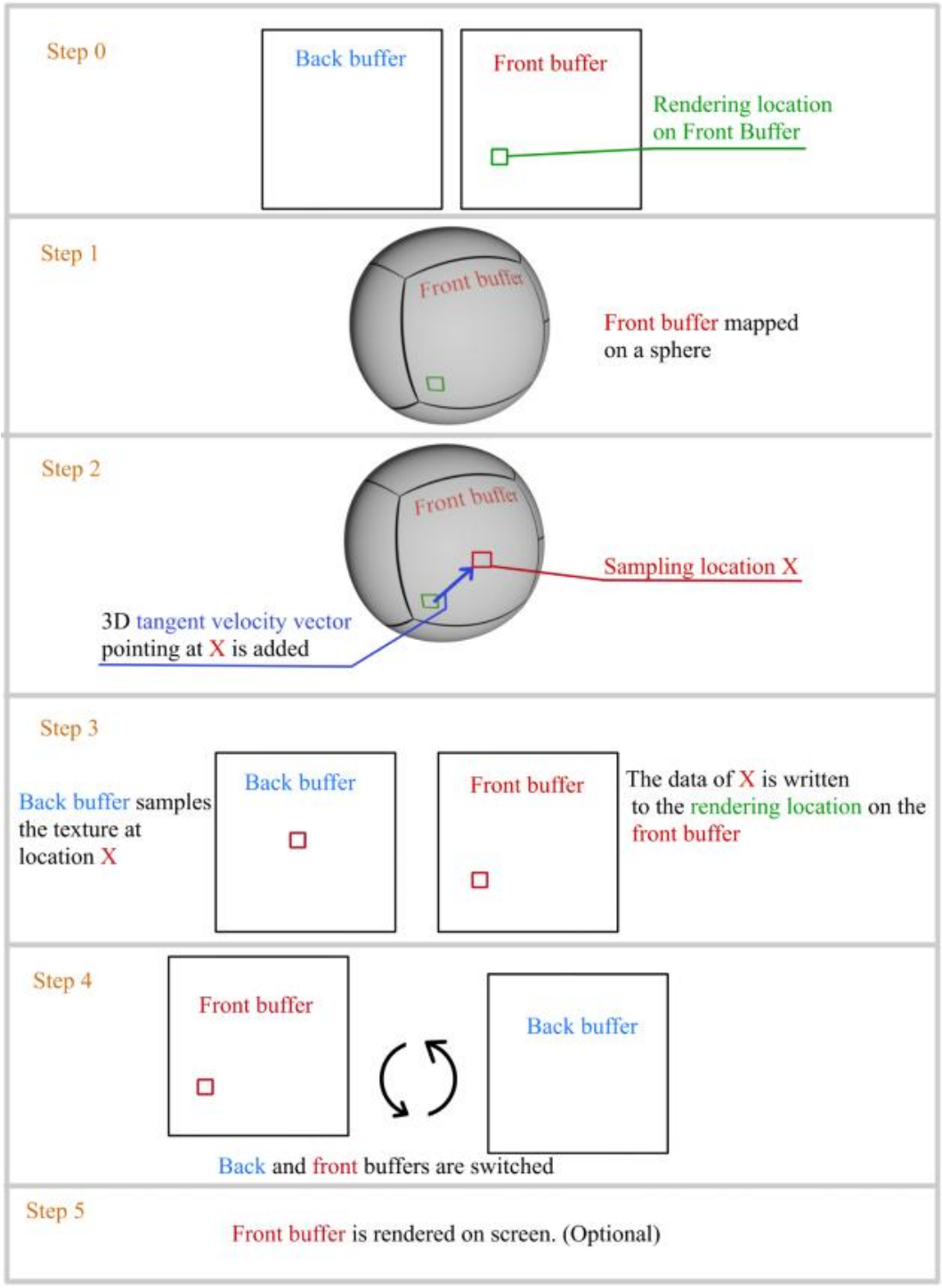


Figure 30. Double buffer pipeline

For this to work correctly the velocity vectors must be tangent to the corresponding positions on the sphere. As illustrated by Figure 31, a tangent plane is a plane that touches the surface of

a sphere at a point[18]. If the velocity vector is not tangent to the surface of the sphere, like the green velocity vector, some energy is lost. This is illustrated by the magenta line, whereupon the 3D UV sampling, the effective wind magnitude becomes smaller. The best scenario for projection would be the red vector as it starts and ends on the surface of the sphere. This idea can be approximated by making the projected vector small enough as indicated by the blue vector. The vector starts on the sphere, but isn't long enough for the curvature to be much different from a straight line.

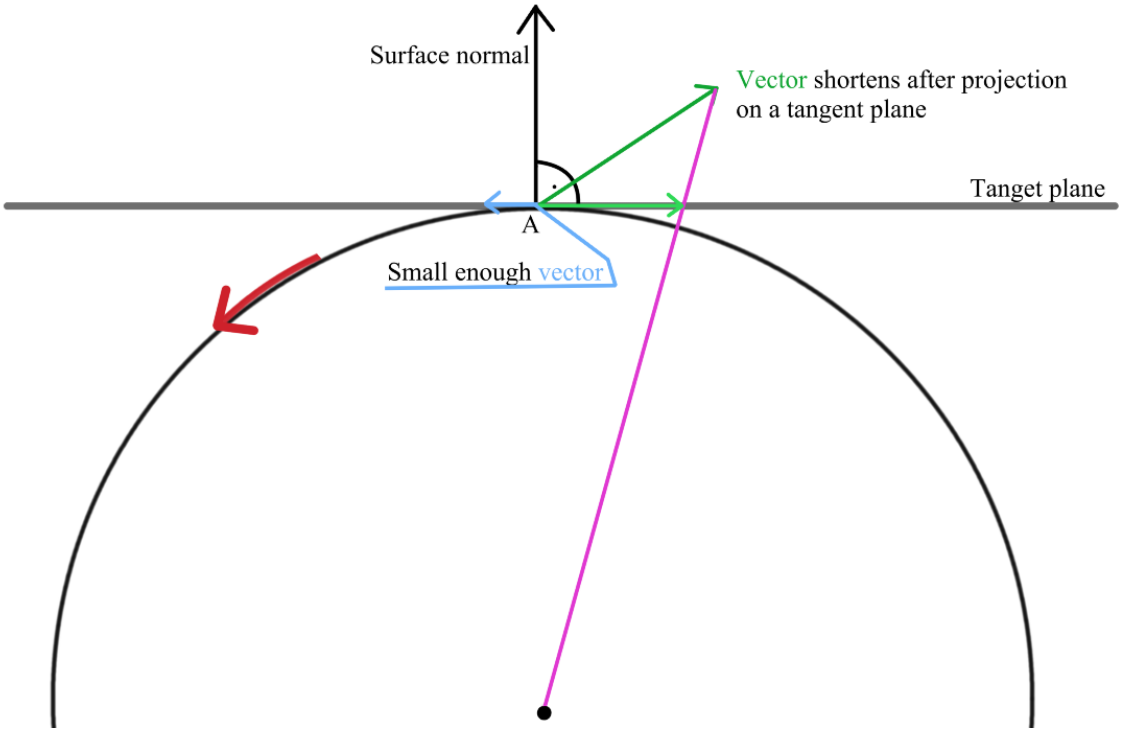


Figure 31. Projection of a velocity vector

The output of the simulation depends heavily on the velocity field. Some velocity fields move particles uniformly, giving it an orderly look. Others are more chaotic. Thus the selection of the velocity field can be used to implement different natural phenomena discussed in 2. Natural Phenomena Found on Gas Giants

4.4 Natural phenomena

The velocity fields are the main way to determine the visuals on the simulated gas giant. There are two types of velocity fields, which are jets and storms. While storms are localized, jets are a global phenomenon.

[18] https://www.khanacademy.org/math/multivariable-calculus/applications-of-multivariable-derivatives/tangent-planes-and-local-linearization/a/tangent-planes

4.4.1 Jets

As discussed in 2.1. Jets, jets are the bands where gases move horizontally in altering directions. The construction of the velocity field for this kind of movement is done in two parts. First, the velocity field is initialized to a global eastward direction. The direction of these vectors can then be flipped at certain altitudes using a sine wave. The sine wave can be modified to change the behaviour of the jets in many different ways.

To get the global eastward velocity field, a cross product is done between the north axis and the normal of the surface at the simulated location. As previously stated, the normalized 3D UV coordinate doubles as the surface normal. The process of using a cross product is illustrated on Figure 32, where the north axis is colored in blue and the surface normals are green. The cross product is colored red. Finally, the velocity field is normalized for the next operation.

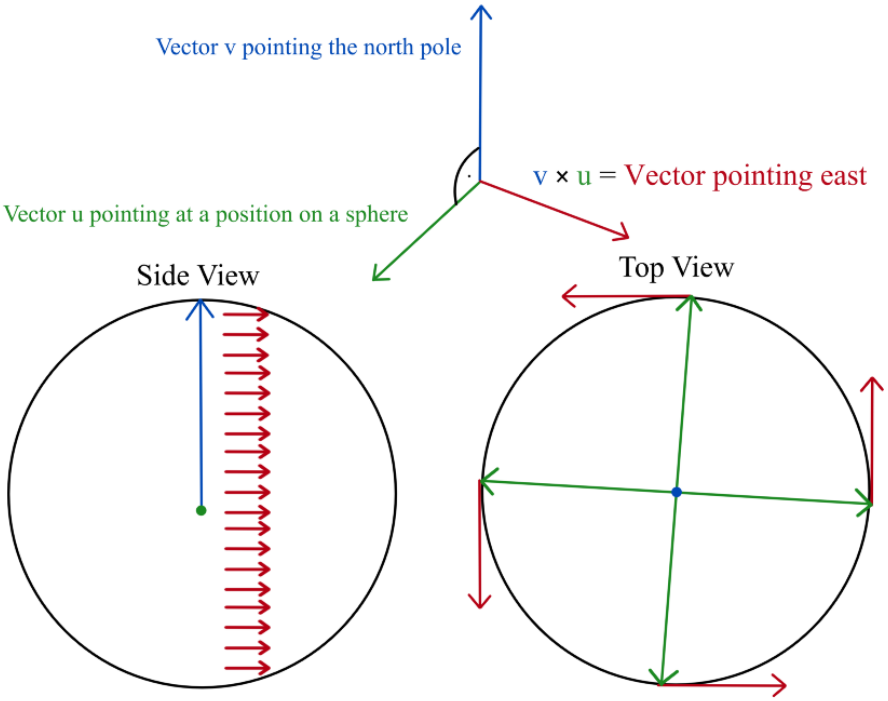


Figure 32. Cross product bewteeen the north axis and the surface normal.

To get the alternating flow bands, the velocity vectors are multiplied by a sine wave. As illustrated on Figure 34, the sine wave changes along the north axis and it alters the speed and direction of the velocity vectors depending on the altitude. Positive values of the wave (represented by red) correspond to the jets moving to east and the negative values (represented by blue) to west. This approach also creates initial round vortices at the southern and northern parts of the sphere, as seen on Figure 33.

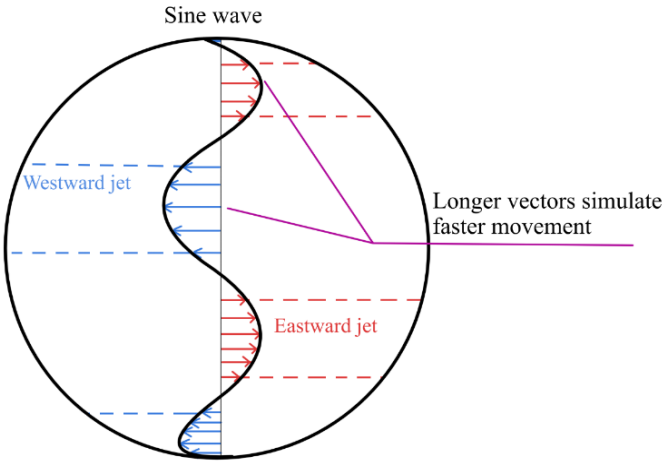


Figure 34. Sine waves alters jets direction.

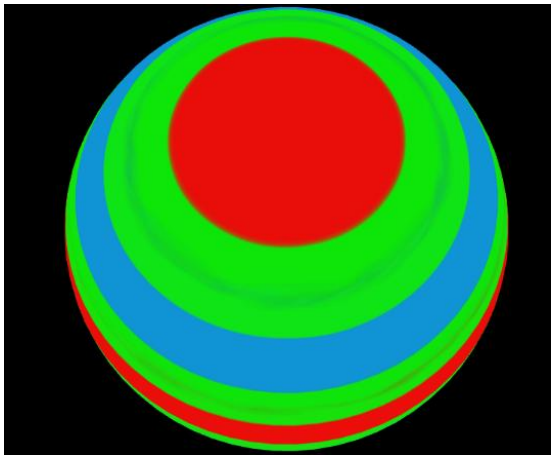


Figure 33. Initial polar vorticy.

The sine wave can then be modified by the speed and frequency parameters. The speed parameter changes the length of the velocity vectors thus changing the speed at which the gases in the bands move. The frequency parameter stretches and folds the vertical sine wave creating jets of various widths. This is illustrated on Figure 35, where frequency 15.5 gives a lot of narrow bands but frequency 1 creates few but wider jets.

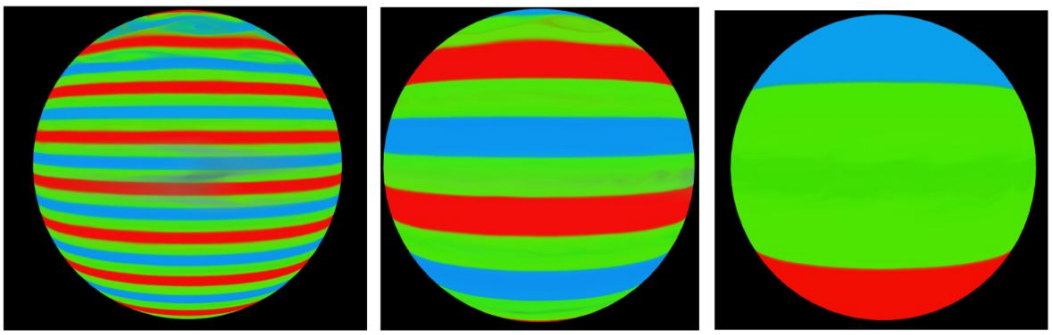


Figure 35. Jets at different frequencies.

The color of the jets can also be determined with the same sine wave. The colors are continuously added every simulation iteration to keep them saturated. For example, in Figure 35 the positive sine values create red bands and the negative ones blue bands. Because the sine wave is continuous, the areas between the jets will have velocity vectors with very low or even zero magnitudes. These areas can be given a third color, for example green. But usually, they are filled with storms.

4.4.2 Storms

One of the most notable visuals of gas giants are their storms. The storms can be created in two different ways. More specific storms can be created using a cross product, similar to what was used in jet logic, and more random storms by using curl noise. One of the most notable natural phenomena are the big storms of the gas giants.

4.4.2.1 Big storm

The big storm is meant to imitate the Great Red Spot of Jupiter. The velocity vector of it is generated similarly to the jets but instead of using the north pole as the up vector, the location of the storm is used instead. While the jets are global, the big storm is not. The magnitude of the velocity vectors gets smaller the further away they are from the epicenter of the big storm. The rate of change is determined by the size parameter. Similarly to jets, the speed parameter changes the magnitude of velocity vectors to influence the strength of the storm. Finally, a separate color parameter makes it stand out. The big storm is added to the total velocity fields (which includes the jets and other storms) with linear interpolation. This means that different phenomena start influencing each other. As seen in Figure 36, the circular storm changes into an oval shape because the jets near the storm overpower the edges of it.

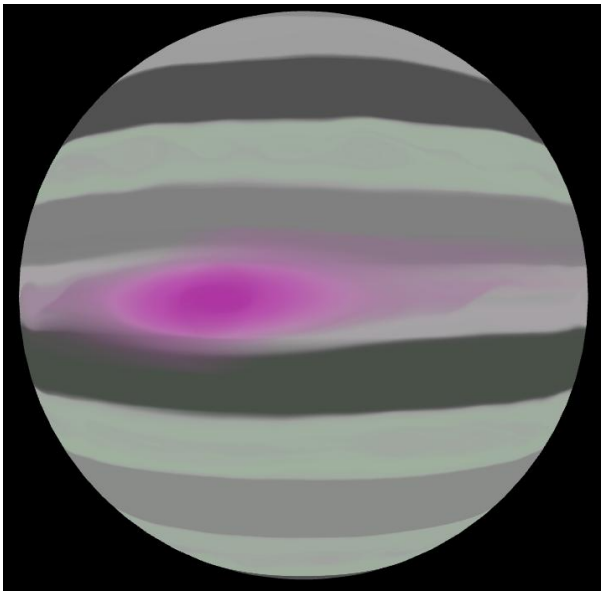


Figure 36. Big storm.

4.4.2.2 Polar hexagon

This interaction between storms and jets can also be used to create velocity fields that produce polar hexagons as seen on Saturn. To create the N-gon shape, medium sized storms are created, one for each side of the N-gon. The storms are placed approximately on the edges of the previously formed polar vortex that naturally formed from jet logic. This is illustrated in Figure 37, where in the step (1), the polar vortex is observed. In step (2), a medium sized storm is added to the edge of that vortex. When the velocity fields are added together, they gain the

average direction and have a bigger magnitude, which produces straight line winds. Finally, the N-gon pattern is created when N storms are placed around the polar vortex regularly.

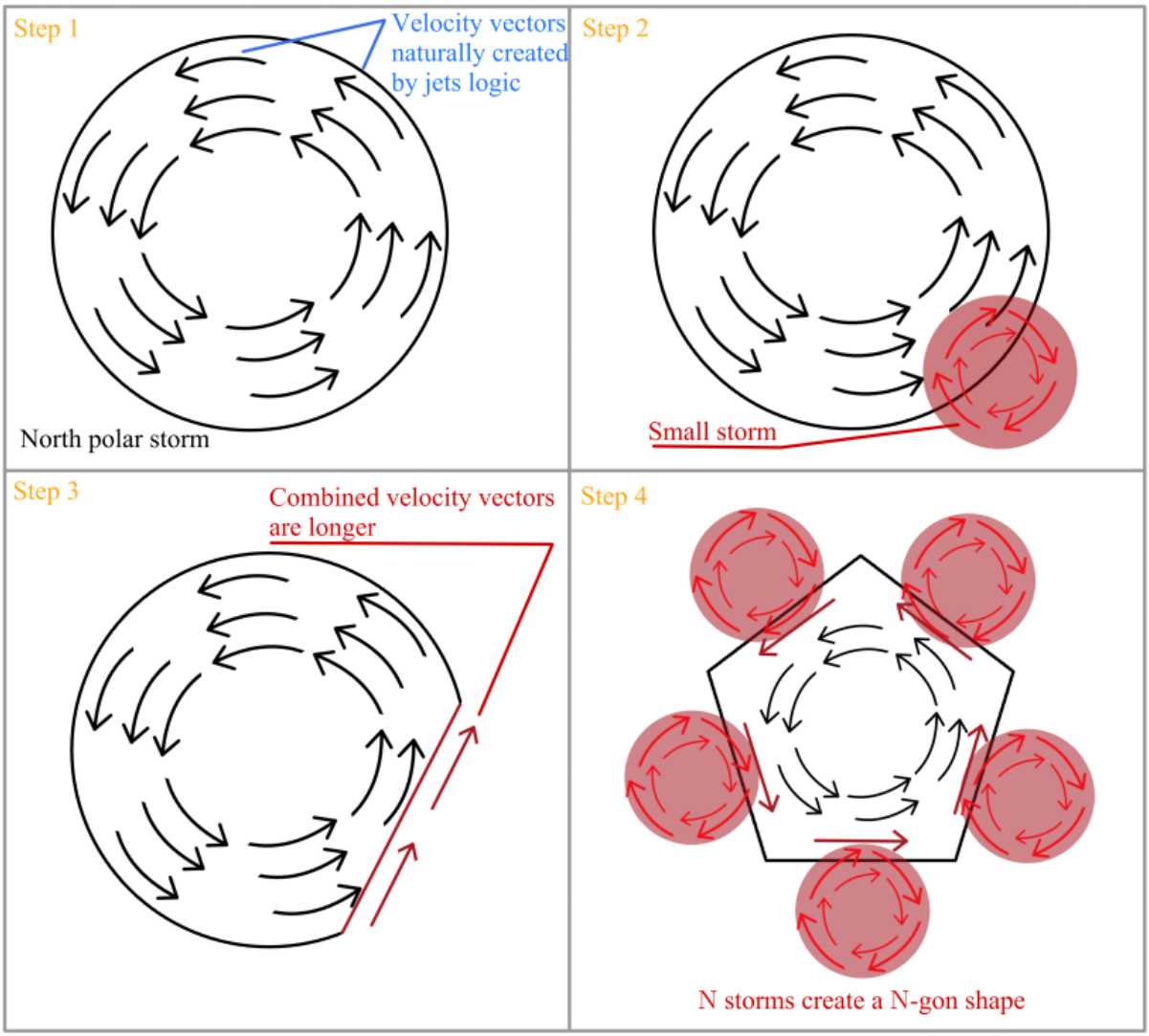
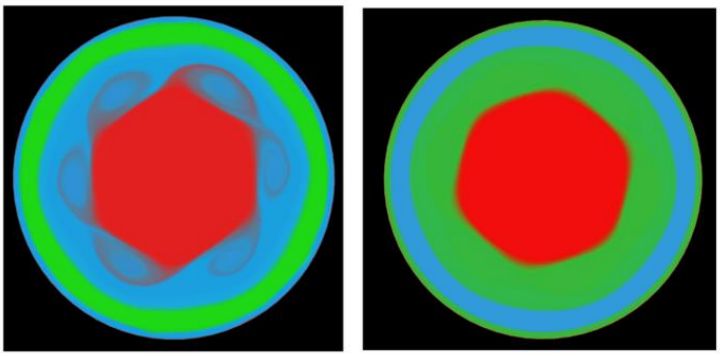


Figure 37. The creation of an n-gon.

The storms that create the polar hexagon can be configured with the speed parameter. Lower speeds do not change the velocity vectors much, which means that the hexagon is subtle or not even noticeable, as shown on Figure 38 example B.



Example A Example B

Figure 38. Examples of polar hexagon.

The high speeds create a perfect hexagon, which might look unnatural. A good balance of the jet speed and the hexagon storm speed is shown on Figure 38 example A, where the corners are sharper but still rounded.

4.4.2.3 Small storms

At the locations where the eastward and westward jets meet, the jet velocity vectors have small magnitudes. This means that there is no uniform directional movement. The alternating directions of the jets naturally create storms in the areas in between the jets. These are added using curl noise which adds randomness to the simulation. Curl noise is a type of gradient noise, meaning it is differentiable and it can output gradient and curl vectors. These vectors can be used as velocity vectors to create said storms. This is similar to what Emil [4] did. However, unlike Emil, this algorithm is on a 3D surface. Therefore, the noise has to be modified so its gradients are tangent to the surface of the unit sphere. The curl operation is also implemented differently from a regular 3D curl.

The gradient noise works by interpolating a grid of gradients to form a continuous field. In 3D, these gradients might not be tangent to the unit sphere, therefore when they are interpolated, they might not produce valid velocity vectors. This is fixed by projecting these gradients onto the unit sphere with (1). More specifically, a vector rejection is done with the surface normal. The vector rejection is similar to the vector projection, with the difference being that instead of the gradient being projected on the normal vector, it is instead projected perpendicular to the normal vector[19]. This means that it is on the tangent plane. The projection is done twice. Firstly, the gradients are projected before the interpolation. Then, they are projected again after the interpolation. This is done because the interpolation is not linear and therefore it might not be tangent anymore.

$$\mathrm{rej}_{\mathbf{b}}(\mathbf{a}) = \mathbf{a} - \mathrm{proj}_{\mathbf{b}}(\mathbf{a}) = \mathbf{a} - \frac{\mathbf{a}\cdot\mathbf{b}}{\mathbf{b}\cdot\mathbf{b}}\mathbf{b} = \mathbf{a} - (\mathbf{a}\cdot\hat{\mathbf{b}})\hat{\mathbf{b}} \quad (1)$$

The gradient of the noise can be converted to the curl of the noise. Usually, on the 2D surface, the curl is calculated with (2), where ψ is the value of the noise at (x, y).

$$\vec{v}(x,y) = \left(\frac{\partial\psi}{\partial y}, \ -\frac{\partial\psi}{\partial x}\right) \quad (2)$$

The curl is like a gradient, but it is rotated 90º clockwise. The curl signifies the vorticity of the values of the noise. In 3D, the formula is a bit more complicated. Instead of using a single value noise, it needs three different noise values. This is not suitable for the algorithm. Therefore the concept of 2D curl is used instead.

[19] https://raw.org/book/linear-algebra/dot-product/

Since the simulation is happening on the tangent plane, the 2D curl could be calculated on that plane. The gradient can be acquired from the projected gradient noise. As stated, the 2D curl is gradient with a 90º rotation clockwise. That rotation can be added with a cross product between the gradient and the surface normal. The cross product keeps the resulting vector on the tangent plane. Therefore, this creates a velocity field with many small vortices that follow the peaks and valleys of the gradient noise as illustrated on Figure 39. The result of adding curl noise on the whole sphere is illustrated by Figure 40 where the sphere is covered by small mixing curls.

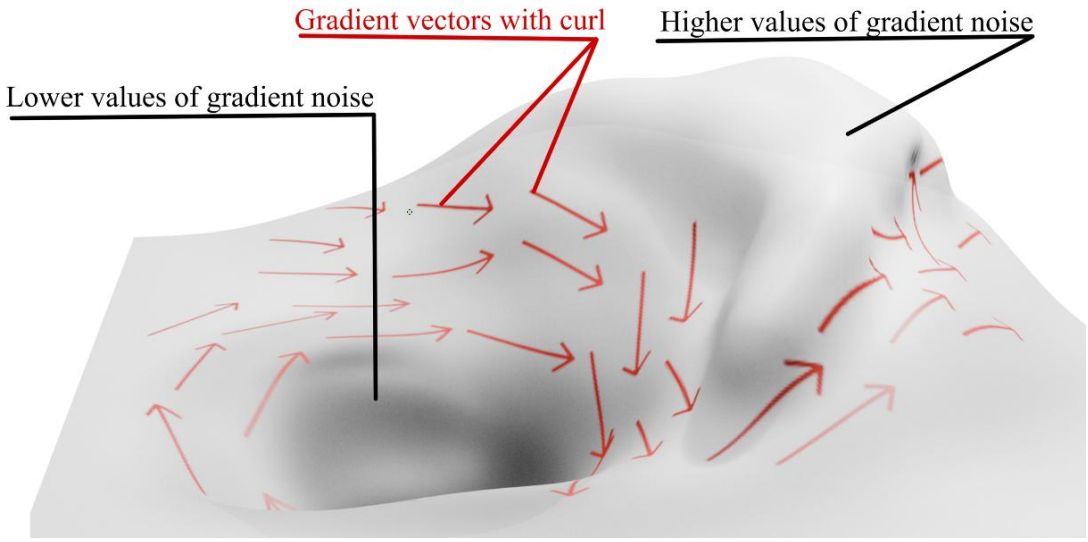


Figure 39. Peaks and valleys of gradientn noise.

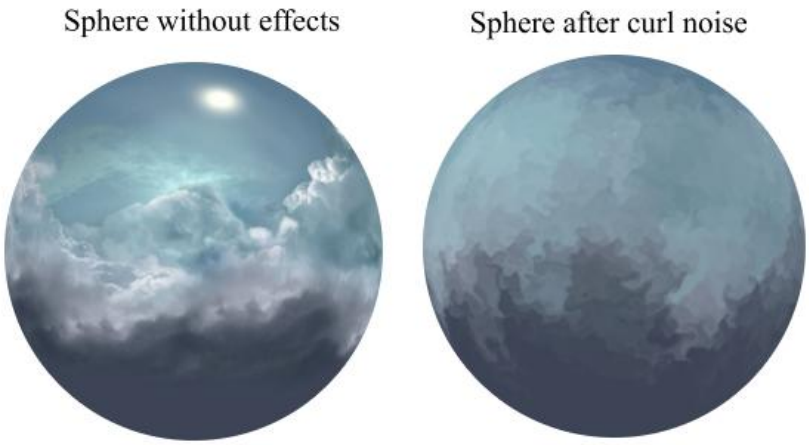


Figure 40. Initial storms on a sphere.

Rotation directions of said vortices are determined by the movement and directions of the surrounding two jets. As seen on Figure 41, it is controlled with a cosine going along the north axis. For the storms to not overtake the jets, a fade is applied. The closer the storm is to the jet, the less influence the vortices have on the final velocity field.

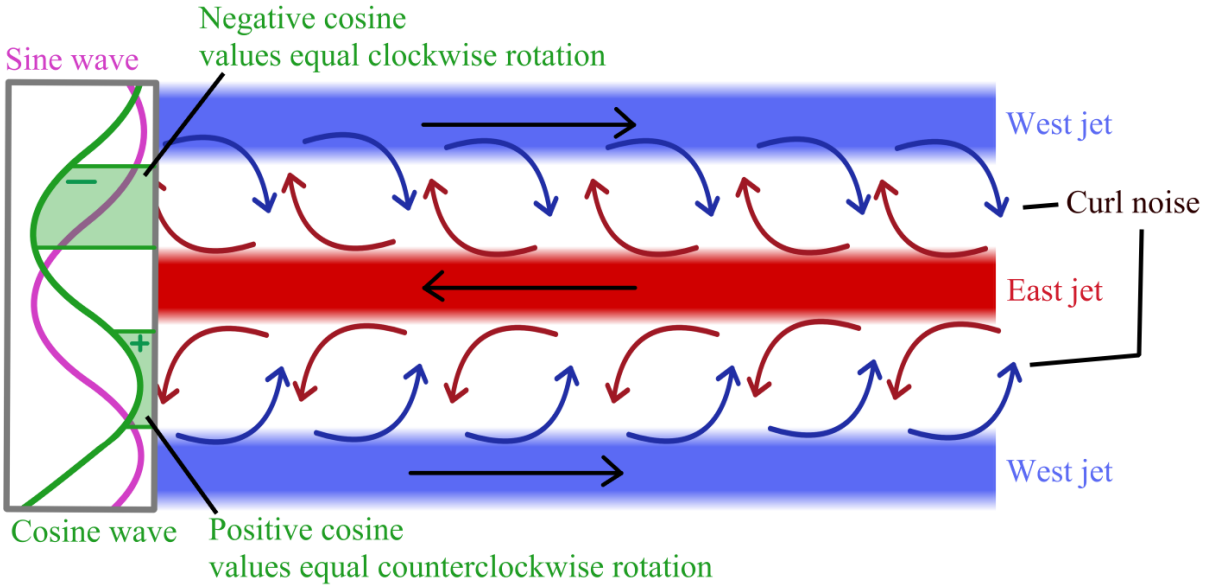


Figure 41. Vortices direction determined by cosine.

These Storms can be modified by changing the underlying curl noise as well as the velocity vectors. As seen on Figure 42, by changing the frequency of the noise, the size of the small storms can be controlled.

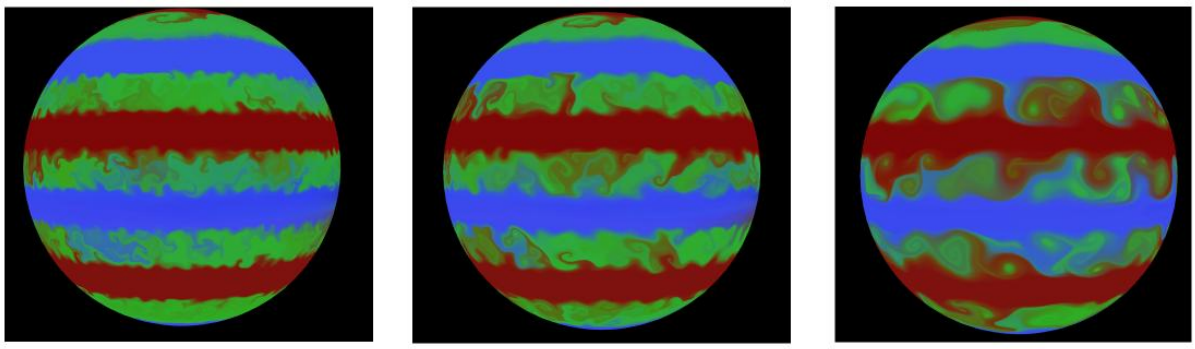


Figure 42. Simulation at different noise frequencies.

The speed parameter is used to change the magnitude of the curl vectors. This works in a similar way to the speed values used for the other storms and jets, by changing the length of velocity vectors.

4.5 Simulation output

To save the simulation at different states a method is needed to transfer the entire surface of the sphere to a 2D texture. This can be achieved by using cylindrical equal area projection. The surface of the sphere is mapped on a cylinder, after which the cylinder is unrolled to create a 2D texture. At each longitude, the data on the sphere is sampled onto the cylinder. The red arrows on Figure 43 show the locations on the sphere to be sampled.

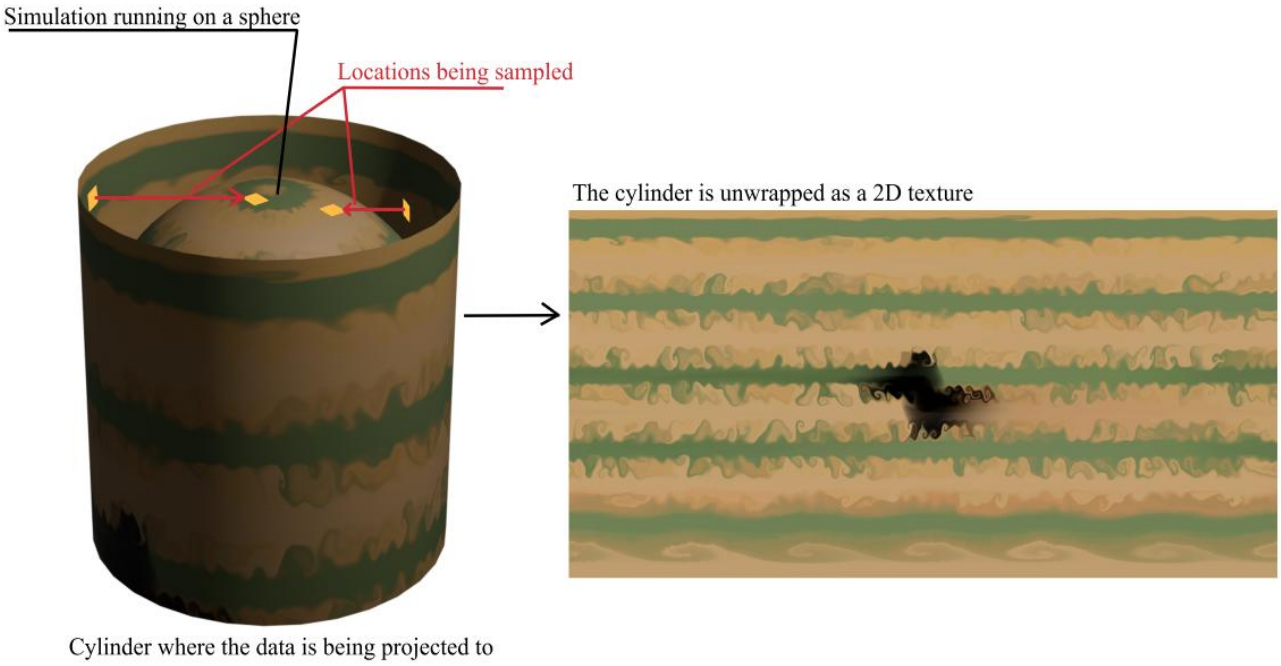


Figure 43. Cylindrical projection.

When rendering the projection, the first thing that has to be done is the conversion of UV coordinates (u, v) from [0, 1] to [-1, 1] to get (u', v'). In the cylindrical equal area projection, both altitude and longitude have positive and negative values (east and west, north and south). Then, the radius of the sphere at given altitude v is calculated by (3).

$$r = \sqrt{1 - v^2} \tag{3}$$

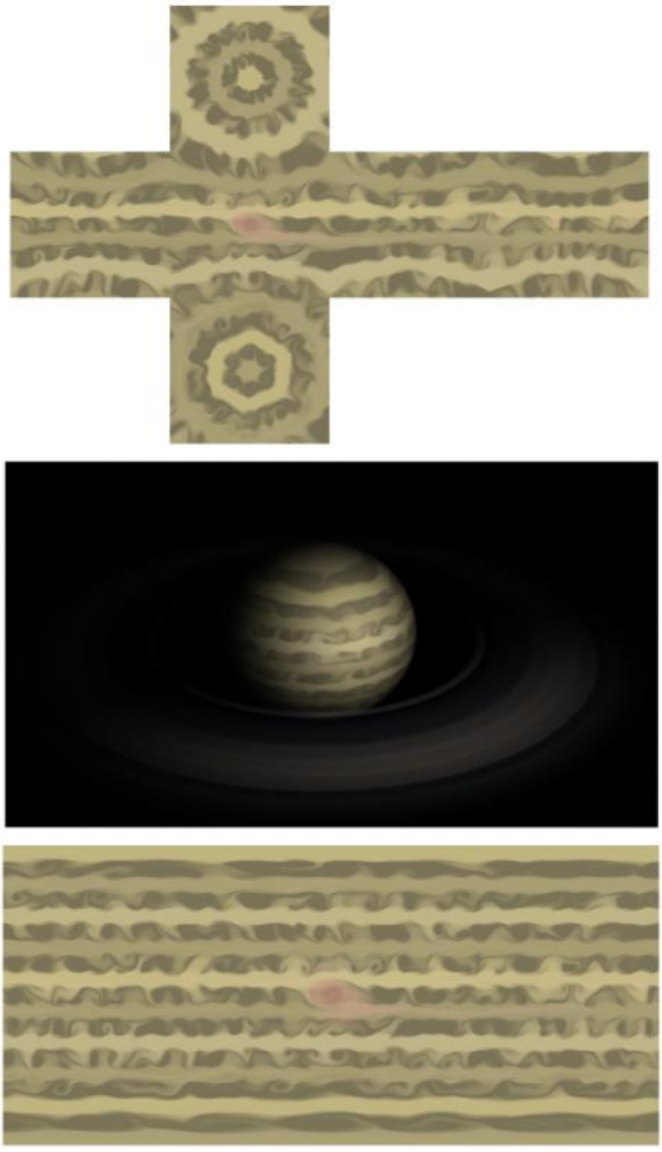
Then, the longitude u is converted into radians with $\theta = \pi \cdot u$. This can then be used to find the x and z coordinates of the 3D sampling vector (4).

$$\vec{s} = (r \cdot \cos(\theta), v, r \cdot \sin(\theta)) \tag{4}$$

Finally the vector is normalized to ensure the sampling vector is pointing to the surface of the sphere.

5. Algorithm Demonstration Application

The best way to show the algorithm is to demonstrate it using a demo application. For the purpose of easy availability the demo application was created in a web environment. A higher level programming language like Javascript was chosen so as to speed up the development process. To not create everything from complete scratch a Javascript framework is used called Babylon.jsfootnote. The framework comes with tools that help create basic 3D scenes, user controls as well as a complete system to work with render pipelines and shaders.



1. The simulation is done and written on a cubemap texture.
2. The simulation is mapped on a sphere with lighting and rings.
3. Cylindrical projection is used to capture and export the surface of the sphere.

Figure 44. Demo application scenes.

Three main scenes were created. They are illustrated on Figure 44. The first scene is for the main simulation of the gas giants. It implements the algorithm described in 4. Algorithm. The second scene uses the data obtained from the first scene and maps it on a 3D sphere as well as

adding basic lighting and rotation to the sphere. This scene is used to observe the course of the algorithm. It provides visual feedback for when the parameters are changed. Finally, the cylindrical projection scene is for capturing the data in the cubemap and converting it into a panorama, ready to be exported and used in other 3D software. This process is described in 4.5 Simulation output.

The first and the third scene are fully calculated in the fragment shaders. This was achieved by creating a plane in 3D space and then looking at it with an orthographic camera such that the whole render target is covered with the plane. That also means that each pixel of the output is determined by one single fragment shader incantation. Note that for rendering a cubemap, six different render calls are needed – one for each face of the cubemap. For both scenes, the output resolution is configurable. Higher resolutions produce better quality simulations in exchange for lower performance.The 3D scene uses the texture that was captured by the first scene and maps the data on a sphere that represents a gas giant. Through the equator of the sphere a plane is added with a planetary ring texture. The sphere and the plane are given an animation loop that rotates the objects on the north axis as well as simple lighting to improve the look of the simulation. Finally the background of the 3D scene is set black to simulate the environment of space. This scene is the main way to see the output of the algorithm.

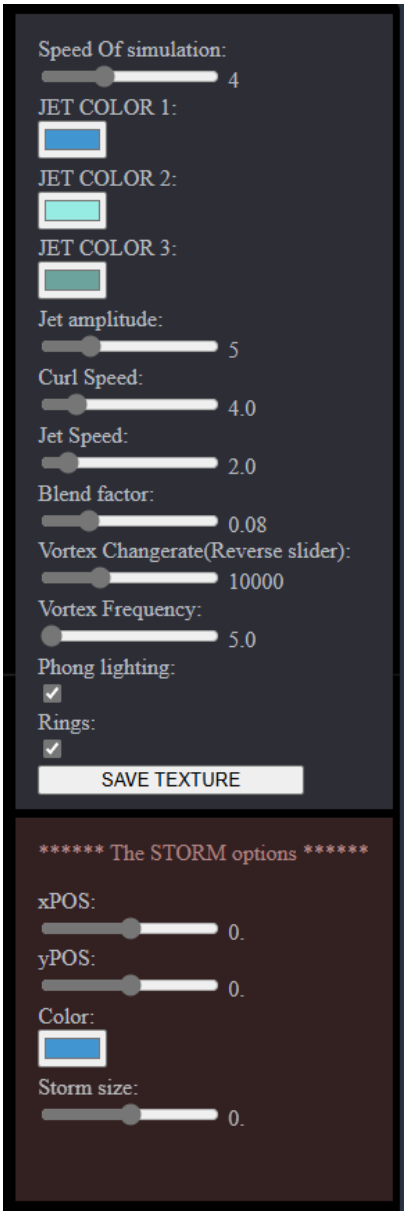


Figure 45. Configuration UI.

To change the course of the algorithm, different parameters are given sliders and color pickers. This user interface is illustrated on Figure 45. The UI includes all of the parameters mentioned in 4. Algorithm for changing the size, speed, colors, and locations of jets and storms. The configuration UI is added to the top right side of the screen so as to not disturb the other possible operations of the browser. For example, changing the tabs on the browser Microsoft Edge. As this feature appears on the left side of the browser, overextending to the left from the configuration menu might cause unwanted interaction. The input is added using

eventListeners that wait for input from the user and the changed value is passed to the shaders in real time. This changes the visuals in the second scene immediately.

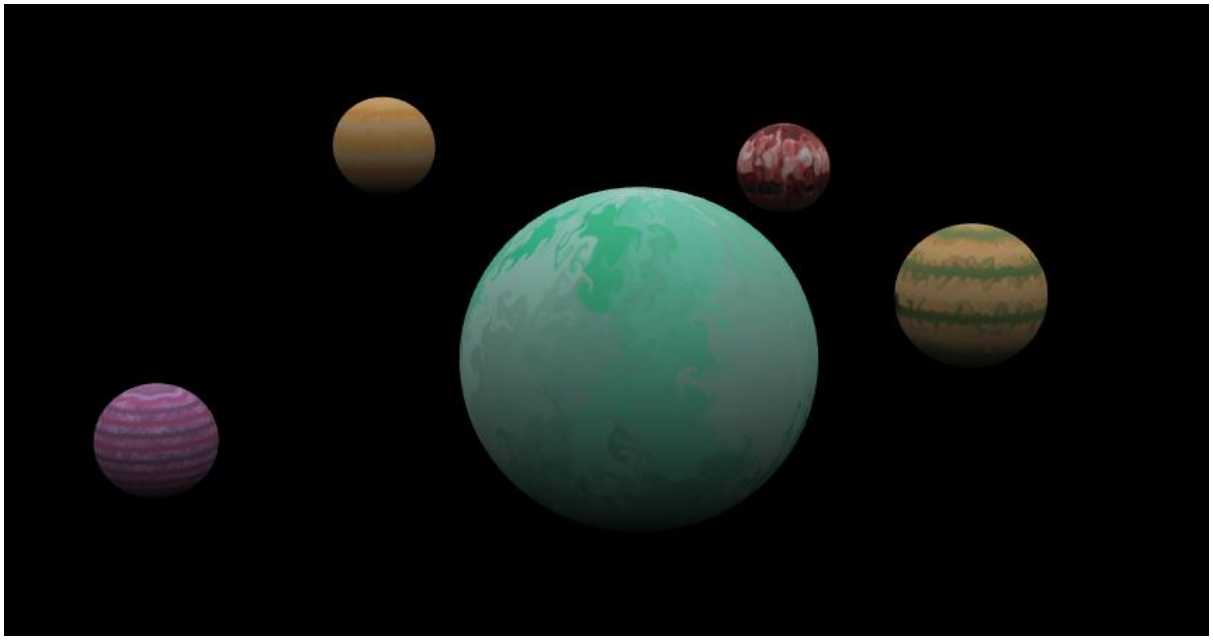


Figure 46. Previously done simulations.

The configuration UI also includes a save button that activates the third scene and saves the output in the server. The panorama is saved as a PNG file. The saved textures can then be displayed on a collection of 5 spheres, as shown on Figure 46, to show previously created simulations. This solution was implemented for showcasing the algorithm during expos.

6. Results

The goal of the thesis was to create an improved algorithm to procedurally generate gas giants based on previously done work. The created algorithm uses parts from both Emil's [4] and Cameron's [3] simulations. Thus the improvement and shortcomings of the newly created algorithm are discussed and compared to the previously mentioned simulations. This is done mainly by comparing the different simulations created using said algorithms.

6.1 The improvements

As seen on Figure 47, the colors of the produced algorithm differ in many ways compared to Emils [4] and Camerons [3] simulations. The previous simulations rely a lot more on the base texture and make less use of colors added during the sampling process. This brings forth the requirement of a third party software to change the original texture. This does not have to be done in the new algorithm. Using chosen colors instead of sampling from a texture improves the speed of the simulation because texture sampling is slow. As the chosen colors are sampled each iteration of the simulation, the problem that Emil encountered, where all the colors blend into gray, is negated.

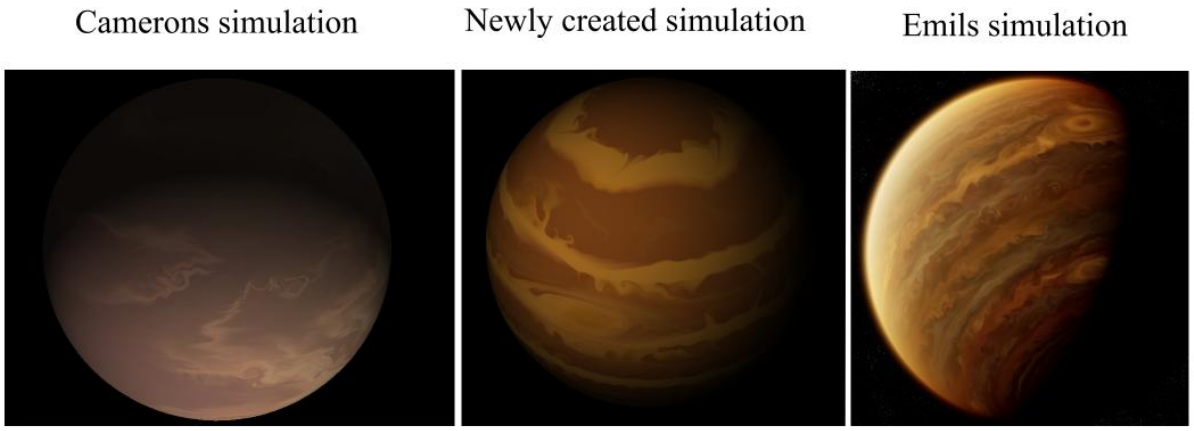


Figure 47. Comparison between simulations.

A similar improvement is made by the velocity fields. Because Emils simulation uses textures as velocity fields, the movement of a storm requires manual changing of the velocity texture making it a cumbersome task. Procedurally generating the velocity vectors makes the generation process more dynamic by adding the possibility of real time changes.

A noticeable part of Emils [4] simulation was the seam created by using a 2D texture. This has been fixed in the new simulation by using a cubemap texture which removed the seam as well as the unnatural polar distortion. Using cubemap textures allows a dynamic polar hexagon to be added which was previously missing in both of the previous simulations.

6.1.1 Performance

As previously mentioned, Camerons [3] simulation was done using the Lagrangian approach for simulating particles as well as running the simulation on a central processing unit (CPU). This made the whole simulation quite slow as a single complete simulation with eight million particles took from three to nine minutes on a CPU *Dual Core i7* using six threads to look complete. This has been greatly improved by using a graphical processing unit (GPU). The newly created algorithm brings simulation times to real time as each iteration is 24 ms with 100,663,296 particles (4096 × 4096 × 6 pixels) on *AMD Radeon RX 6750 XT GPU*. Computing the velocity field took Cameron two minutes, while the new algorithm does this every iteration of the simulation. 20 iterations of Camerons simulation took about 30 seconds while the newly created simulation does the simulation in half a second. That is about 62 times faster. Note that Camerons benchmark was done on CPU that is 11 years old, however, most of the speedup is gained from using GPU. Fast rendering times come with an improvement on development time as new features give visual feedback at a greater pace.

6.2 Potential Improvements

The current algorithm could be improved and further developed in a number of different ways. In this section the improvements and further developments are discussed, including mostly visual changes.

For each of the natural phenomena, the velocity fields are currently quite uniform in nature. For example, the velocity fields of the jets are currently creating bands with relatively similar widths. As real life gas giants like Jupiter have bands that vary in size quite a lot, this is not ideal. Adding the ability to change the sizes of bands at certain altitudes would greatly increase the look of the simulation. This improvement would include the ability to remove bands completely at certain altitudes. The change would allow certain areas of the sphere to be more oriented towards storms, creating a more chaotic but realistic simulation.

6.2.1 Storms

Similar improvement could be made on storms that currently have minimal configurability. The medium and bigger storms could be made to change sizes after a certain time as they do on real gas giants. The changing of sizes would make simulation more varied as currently the big storm is the only storm having this feature. This could be combined with an addition of small vortices of different colors. Currently this effect is missing as using curl noise for smaller storms does not create any noticeable vortices with different colors.

The polar hexagon could also be improved by making the locations and sizes of influencing storms configurable. Changing the influencing storms would allow the creation of more varied shapes on the northern pole.

The big storm can be made more realistic by adding directions for the gases to exit, as well as enter the storm. This would give the storm more influence and allow for the color of the storm to blend more into the jets giving the jets a more varying color scheme near the storm.

6.2.2 Color

As previously mentioned, Emils [4] and Camerons [3] simulations use a base texture for sampling color. This makes the particles of their simulations stand out more compared to the newly created algorithm, by adding more options of color. Currently, a maximum of 4 colors are blended into the simulation and thus lacks depth compared to Emil’s and Cameron's simulations. This could be improved in the future by using different shades of the same color creating the illusion of depth.

6.2.3 Planetary rings

Currently the planetary rings are given in the form of a texture, which lacks any sort of procedural generation. This limits the visuals and configurability of said rings. Completeness would be certainly added by generating the rings dynamically as done with the rest of the simulation. Using particles and a vector field would allow for the implementation of rings with shepherd moons. These moons could be made to influence rings by mixing around debris as found to be happening around real gas giants.

7. Conclusion

This thesis devised a new algorithm for procedurally generating gas giants. After analysing the previous algorithms, multiple areas were discovered that could have been improved. For example lots of algorithms create non continues results, which means there are seams in the produced textures. And in case of seamless algorithms the performance was slow. The new algorithm solved both of these problems.

Before the algorithm was developed, the natural pehenomena on gas giants were studied and categorized. The two main phenomena were jets and storms. While jets are more simple there are many types of storms. The vary in size and location. For example the GRS is the most recognizable storm on Juptier. There are other secondary phenomena that emerge from storms. The one analysed in this work was the polar hexagon. The new algorithm is capable of generating all of these planetary features.

There are two main contributions. First, an algorithm that assigns unique locations on a sphere to cubemap pixels is presented. The essence of it is to transform cubemap face 2D UV coordinates to 3D UV coordinates that are used to sample that cubemap. Secondly, the idea to reduce the set of all possible velocity vectors onto a tangent plane of a sphere is described. This makes it possible to treat the 3D surface as 2D. These two contributions enabled the implementation of many different jets and storms.

This algorithm is demonstrated with a web application. This application exposes all the jet and storm configurations and also exports the simulation output as a panorama texture which is seamless. The application also demonstrated the preformance of the algorithm. Compared to the previous best simulator, there is an order of magnitude improvement.

The algorithm could be improved. Altough it is fast, it still lacks some visuals found in other simulators. The management of color can definetly be improved and would add a layer of depth to the simulation. The storms could be more configurable and new variations could be added. Finally, the planetary rings currently lack any diversity and thus could be improved by adding different options for configurability.

References

[1] S. Strömer, „Procedural generation: Gas giants,“ 2019. Available: https://stroemer.cc/procedural-generation-gas-giants/. (04.12.2024).

[2] B. Paléologue, „Medium,“ 20 June 2023. Available: https://medium.com/@barth_29567/procedural-gas-giants-f2a61bc6bd97. (04.12.2024).

[3] S. M. Cameron, „Github,“ 27 August 2014. Available: https://smcameron.github.io/space-nerds-in-space/gaseous-giganticus-slides/slideshow.html#1. (04.12.2024).

[4] E. Dziewanowski, „emildziewanowski.com,“. Available: https://emildziewanowski.com/flowfields/. (22.03.2025).

Appendix

I. Accompanying Files

The ZIP-file containing the accompanying files to this thesis is structured as follows:

- *Demo* – the folder containing the source code (see Appendix II) of the demo application.
    - *src* – Folder containing javascript and css code used in the demo as well as 5 previously saved simulation located in the uploads folder.
    - *curlNoise.fragment.fx* – Fragment shader where most of the simulation takes place.
    - *curlNoise.vertex.fx* – Vertex shader for the simulation.
    - *cylindricalProjection.fragment.fx* – Fragment shader used for cylindrical projection.
    - *cylindricalProjection.vertex.fx* – Fragment shader used for cylindrical projection.
    - *doneGenerations.html* – HTML file for displaying previously made simulations.
    - *Index.html* – HTML file where the demo application runs.
    - *package.json* – Package manager file containing a list of all the packages to run the application.
    - *planetaryRings.fragment.fx* – Fragment shader for planetary rings.
    - *planetaryRings.vertex.fx* – Vertex shader for planetary rings.
    - *README.md* – Text file including the steps to run the application as well as description configurations.
    - *Ring.png* – Image of planetary rings used to display the rings in the simulation.
    - *sphereRender.fragment.fx* – Fragment shader used to render the simulation in 3D space.
    - *sphereRender.vertex.fx* – Vertex shader used to render the simulation in 3D space.
    - *vite.config.js* – Configuration file needed to run a Vite application.
- *Images* – Folder containing a few images of gas giants created using the demo application.

The source code can also be found in a GitHub repository with the following URL: https://github.com/bimonXdd/ProceduralGasGiant/tree/curlNoise-development

II. License

Non-exclusive licence to reproduce the thesis and make the thesis public

I, **Simon Prii**,

grant the University of Tartu a free permit (non-exclusive licence) to

reproduce, for the purpose of preservation, including for adding to the digital archives of the University of Tartu until the expiry of the term of copyright, my thesis

Procedural Generation of Gas Giants

Supervised by: Mathias Plans

2. grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright;

3. am aware of the fact that the author retains the rights specified in points 1 and 2;

4. confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Simon Prii

15.05.2025