Genera Esfera: Interacting with a Trackball Mapped onto a Sphere to Explore Generative Visual Worlds

Lali Barrière
Dept. de Matemàtiques, Universitat Politècnica de Catalunya, Spain
lali.barriere@upc.edu

Anna Carreras
Faculty of Sciences and Technology, University of Vic, Spain
me@annacarreras.com

Abstract

Genera Esfera[1] [3] is an interactive installation that allows the audience to interact and easily become a VJ (visual DJ) in a world of generative visuals. It is an animated and generative graphic environment with a music playlist, a visual spherical world related with and suggested by the music, which reacts and evolves. The installation has been presented at MIRA Live Visual Arts Festival 2015 [7], in Barcelona. *Genera Esfera* was envisioned, developed and programmed on the basis of two initial ideas: first, to generate our spherical planets we need to work with spherical geometry and program 3D graphics; second, the interaction should be easy to understand, proposing a direct mapping between the visuals and the interface. Our main goal is that participants can focus on exploring the graphic worlds rather than concentrate on understanding the interface. For that purpose we use a trackball to map its position onto sphere rotations. In this paper, we present the interactive installation Genera Esfera, the design guidelines, the mathematics behind the generative visuals and its results.

1 Introduction

The use of computers to create art began in the late 1950s, in labs and research centers where mathematicians and physicists were the first to realize and to be attracted by the expressive potential of the new medium. Since then, a huge community of creative coders has arisen, with a wide range of disparate interests and a continuously increasing number of intersecting disciplines: generative art, interactive art, software art, netart, software poetry, etc. (see [6]). It is very common that digital artists get interested in Mathematics, because it lays at the heart of the computer artist set of tools [2].

Genera Esfera is an interactive installation to discover and play with real-time generated visuals. While listening to music, participants can interact with a spherical interface to explore several worlds and create different graphic effects. Genera Esfera invites the visitors to explore this generative spherical world from different angles and points of view through an interface which is also

[1]Genera Esfera is a catalan expression that could be translated as Generate Sphere. We chose it because it rymes, and it gives the idea of a generative sphere, being a bit less direct.

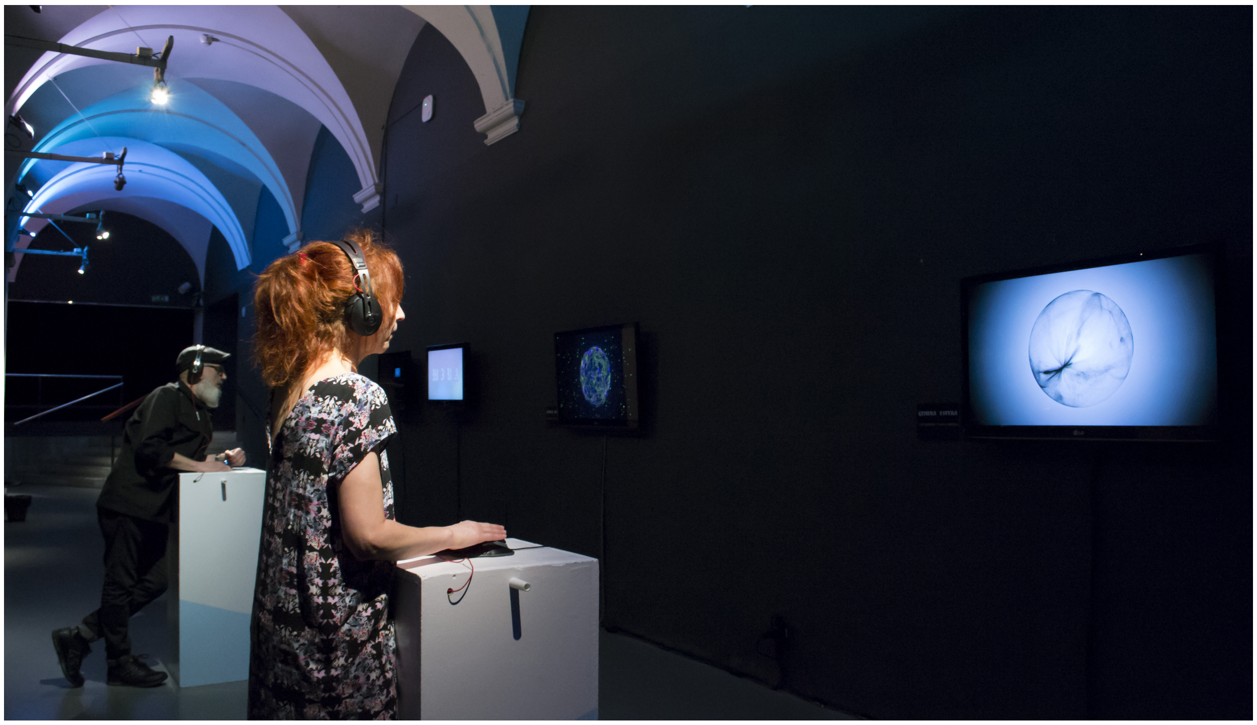


Figure 1: Interacting with Genera Esfera. (Photo by Marta Farràs.)

spherical, and acts as a mirror: a trackball (see Figure 1). The visuals evolve by themselves and react to the participant's interaction changing position, orientation and point of view.

Some interactive installations focus on the development of new technologies, the use of new devices, or the study of new ways to interact. Others focus on the generation of very sophisticated visuals or the design of new programming techniques. Instead, our challenge has been to build an interactive installation on the basis of a simple geometric shape, the sphere, and making use of a familiar, almost classical interactive device, a trackball, to make the interaction natural and easy to understand.

To see the Genera Esfera installation as it was performed and experienced by visitors at MIRA Live Visual Arts Festival 2016, watch our Vimeo video `https://vimeo.com/146750441` [3]. The MIRA Festival is an international festival that takes place yearly in Barcelona. It is a placeholder for the visual arts and new creations that merge music and artistic visuals.

In this paper, we first describe the interactive installation Genera Esfera, how the application is developed, how the interaction is designed and, finally and in detail, the mathematics needed for the generative visuals: a random distribution of points on a sphere, Perlin noise and a vector field on the sphere's surface.

2 General Description

The starting point of Genera Esfera is a music playlist. Each song with its generative visuals constitutes a scene. Using two different playlists and two graphic styles, two versions of Genera

Esfera have been created. These two versions have two mathematical approaches. The visuals and the interaction are based on two main ideas:

- We envision graphic worlds as planets, so we have decided to work with spheres, using spherical geometry and 3D graphics.
- It should be easy to deduce how to interact with Genera Esfera with self-revealing principles of operation. For that purpose, the chosen interface is a trackball, and the trackball rotation is mapped onto a never ending rotation of the spherical planet.

2.1 Hardware

The Genera Esfera installation requires a screen or a wall to project the visuals and a surface to place the trackball. Headphones or, if allowed by the environment, speakers are also needed for the music. The computer that runs the application is inside the base where the trackball stands. (See Figure 1.)

2.2 The Application

The code is built using openFrameworks [10] and prototypes and tests are coded in Processing [11, 14], which are toolkits for creative coding. The final application has the following parts:

1. The music playlist and the music player.
2. The scenes, each scene has a song and a unique set of parameters that establishes its look and behaviour.
3. The sphere animation, which determines the aesthetics of the visuals.
4. The interaction rules, i.e., the actions to be taken when the trackball is rotated or its buttons are pressed.
5. A generic configuration panel, used to set up the installation and choose the parameters that define the different scenes. This panel is hidden when the application is running. (See Figure 2.)

2.3 Music

A set of seven songs is selected for each version of Genera Esfera, according to two criteria: first, the music and the visuals should have a consistent aesthetic and merge smoothly; second, the music should be free-licensed.

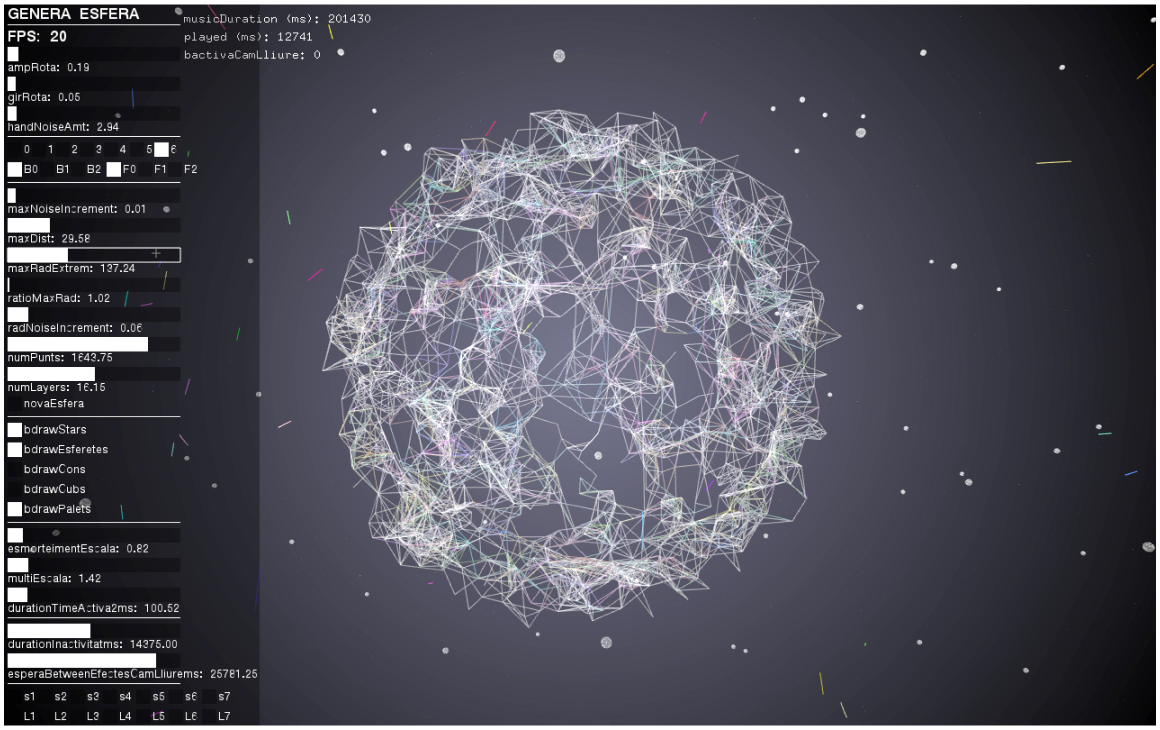
All the music has a Creative Commons license. For these first two versions, it comes form the netlabels: Archaic Horizon, Artico, Nowaki, and Superssonica.

2.4 Interaction

Based on our past experience, we have observed that most festival visitors are not interested in complex interactions. In that sense, successful systems must have principles of operation that are easy to deduce [5, 9]. For that purpose the interface to drive the system response is a trackball.

The sphere is animated, constantly evolving, and it responds to trackball rotations. When participants move the trackball, its rotation is mapped to the sphere, giving the opportunity to explore the world. Two buttons enrich the interaction with some additional effects: i.e. the trackballs left button is related to a sphere growing effect; the right button adds some fleeting elements to the scene. Although the system responds consistently to consistent inputs, it never generates exactly the same visual output twice, because it is sensitive to small differences in user performance, it depends on its current state and rotation, and it adds some randomness to the visual response.

Through these simple interactions the visitors can show their ability to synchronize the presented visual world with the music that is playing [8]. Whenever there is no interaction for one minute, camera rotations and translations are applied to the scene, suggesting that the system is alive and ready to be performed. It is worth to mention that we do not perform audio analysis, and so there is no synchronization between the audio and the visuals, other than the one created by the participants.



```
GENERA  ESFERA
FPS:  20
ampRota:  0.19
girRota:  0.05
handNoiseAmt:  2.94
0  1  2  3  4  5  [✓ 6]
[✓ B0]  B1  B2  [✓ F0]  F1  F2
maxNoiseIncrement:  0.01
maxDist:  29.58
maxRadExtrem:  137.24
ratioMaxRad:  1.02
radNoiseIncrement:  0.06
numPunts:  1643.75
numLayers:  16.15
novaEsfera
[✓ bdrawStars]
[✓ bdrawEsferetes]
bdrawCons
bdrawCubs
[✓ bdrawPalets]
esmorteimentEscala:  0.82
multiEscala:  1.42
durationTimeActiva2ms:  100.52
durationInactivitatms:  14375.00
esperaBetweenEfectesCamLliurems:  25781.25
s1  s2  s3  s4  s5  s6  s7
L1  L2  L3  L4  L5  L6  L7

musicDuration (ms): 201430
played (ms): 12741
bactivaCamLliure: 0
```

Figure 2: The Genera Esfera Estrident, with the GUI. The GUI allows us to set up the parameters: the colors, the radius, the number of points, the maximum distance to which two points can connect, the noise detail, and the radius when the zoom in is triggered. (Screenshot.)

2.5 The Animation

Genera Esfera has two different animations. In next section, we give a detailed description of these two versions. In Section 4, the mathematical tools we used are explained: random distribution of points on a sphere and Perlin noise.

3 Animated Spheres

Genera Esfera has two different versions presented side by side in separate screens, which we call Genera Esfera *Estrident*, for "striking", and Genera Esfera *Caniques*, for "marbles". The difference between these two versions is not only in the music playlist, but also in the aesthetic visual appearance, determined by the programming of the sphere animation. Both animations are parameterized and make use of random elements. By adjusting the parameters we define the variations which will constitute the different scenes. As for the randomness, it is used both at the beginning of the application, to create the spheres, and during runtime, to generate the animation. This allows us to have a fresh, unique animation at every run. This is a common practice in generative artworks.

3.1 Striking Spheres

The first version of Genera Esfera is made by a system of concentric layers of points organically pulsing, connected in an irregular and changing wireframe appearance. (See Figure 2.)

We start by defining a set of ℓ spheres centered at the origin, by choosing the radii, $r_1, \ldots, r_\ell$. A set of n points is distributed on the ℓ spheres. The points assigned to a given sphere are, in turn, distributed uniformly at random on its surface. We say that every sphere, and the set of points on it, form a layer. Then, there is a connection between two points if they are at a distance smaller than a fixed value, even if they belong to different layers.

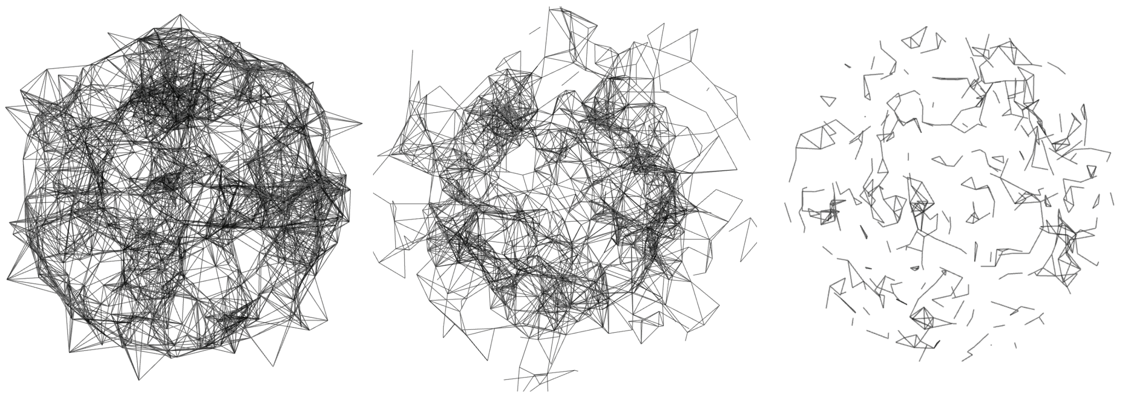


Figure 3: Different appearances of the wireframe, corresponding to different settings.

The animation is driven by the radius variation of each layer, which involves two steps of randomness, in the following way.

- A minimum radius and a maximum increment of the radius, r_{min} and Δ_{Max}, are initially set.
- At every frame, the radius of layer i, r_i, is computed by:

$$r_i = r_{min} + N_i \cdot N \cdot \Delta_{Max}$$

  where N and N_i denote two calls to the Perlin noise function, with different arguments. Since N and N_i are both in $[0, 1]$, the radii of the layers, r_i, organically oscillates between r_{min} and $r_{min} + \Delta_{Max}$.

- Notice that, every frame needs one call to Perlin noise to compute N, which affects the "global" radius of the sphere, and ℓ calls, one per layer, to compute N_i, which defines the evolution of the radius of level i, for every $i = 1, \ldots, \ell$.

This dynamics is written in code as follows:

```
// radius range
maxNoise += maxNoiseIncrement;
delta = ofNoise(maxNoise) * maxDelta;

// each layer corresponds to a radius
for(int i=0; i<numLayers; i++){
  radNoise[i] += radNoiseIncrement;
  layerRadi[i] = minRad + ofNoise(radNoise[i]) * delta;
}
```

In this case we use 1-dimensional Perlin noise, a function of one single parameter that is increased by a small constant, sometimes called "noise detail", at every call. This gives a random but organic appearance to the movement. The arguments of these Perlin noise calls, together with the number of points and the maximum distance to which two points can connect, define the visual appearance of the sphere. Neither the distribution of points in the layers nor their distribution on the corresponding sphere change. (See figure 3.)

3.2 Marbles

The second version of Genera Esfera gets its name from its resemblance to the colored glass swirls of a marble. It is built by means of a particle system, the movement of the particles being ruled by a Perlin noise vector field. The vector field is a field defined on a rectangle and wrapped onto the sphere via an equirectangular map. Changing the vector field parameters has a great impact on the particles' trajectories, as shown in Figure 4.

This version has been technically challenging, because the aesthetic result we wanted needs a lot more computational performance. We work with a set of 1024 leading particles, evolving over the sphere. At every frame, for each leading particle we draw a tail of 512 particles. Therefore, in real-time, more than half a million particles are drawn. This is accomplished by making use of the OpenGL data structure VBO, or Vertex Buffer Object [4].

Next we describe how the noise field is computed and how it determines the behavior of the particles. The field is not a simulation of any real vector field, and it is not defined by its physics. Instead, it is algorithmically defined making extensive use of the Perlin noise function. The following steps are performed at every frame for every particle.

- The elapsed time is represented by t. At every frame it is increased by the constant *timeStep*.
- Each particle has what we call "field coordinates" $(x, y) \in [0, w] \times [0, h]$, which evolve over time.
- Each coordinate of the vector field $(f_x, f_y) \in [0, 1]^2$ is returned by a Perlin noise function call:

$$f_x = N\left(t + phase, complexity \cdot \frac{x}{w} + phase, complexity \cdot \frac{y}{h} + phase\right)$$
$$f_y = N\left(t - phase, complexity \cdot \frac{x}{w} - phase, complexity \cdot \frac{y}{h} - phase\right)$$

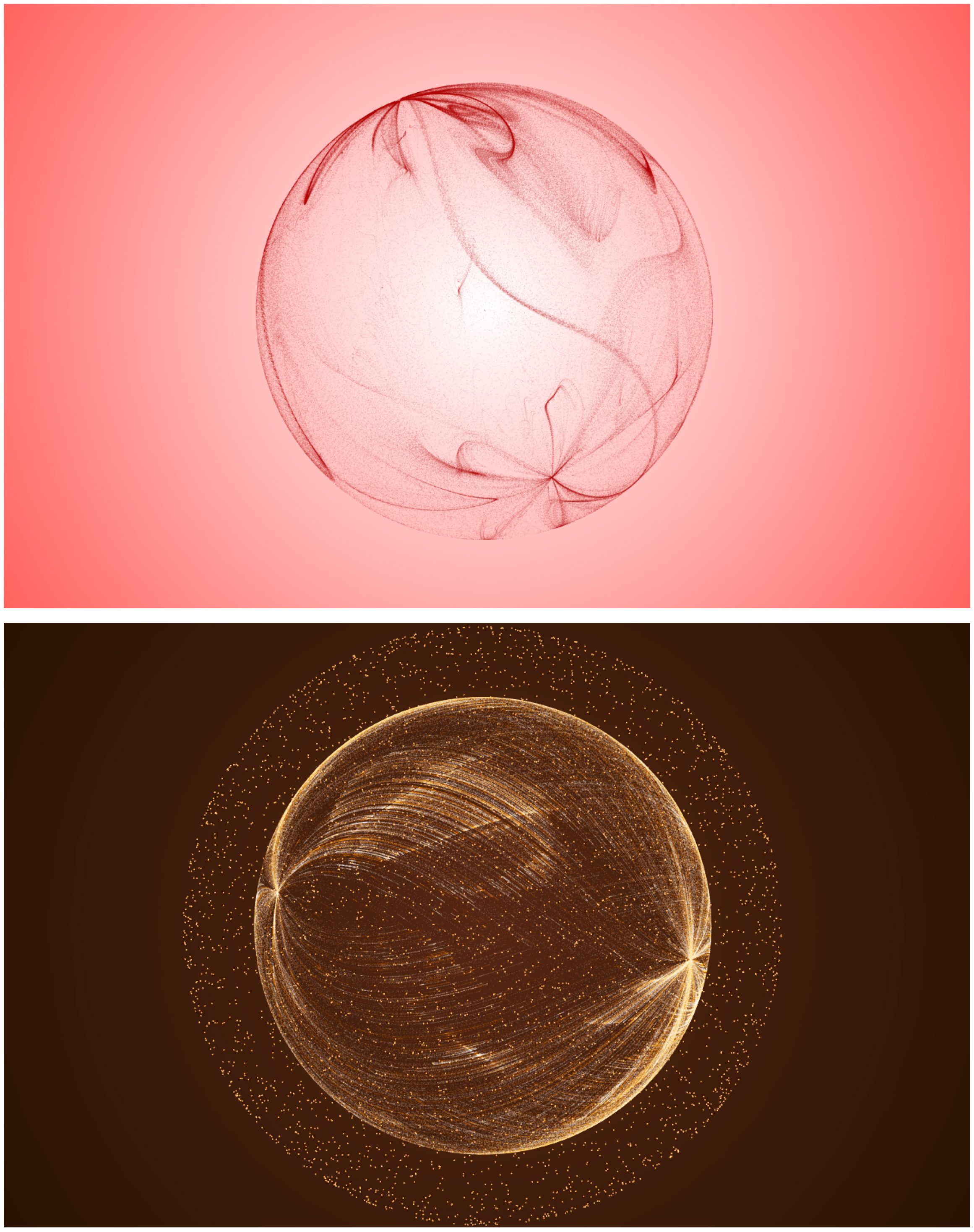


Figure 4: Same code, different settings, give really distinct appearances. In the settings, the colors and the constant parameters that affect the vector field behavior are set up.(Screenshots.)

The parameter *complexity* affects the general evolution of subsequent Perlin noise calls, while the parameter *phase* serves to separate the arguments that define the coordinates f_x and f_y, making them more independent. Notice that we denote by N the calls to the 3-dimensional Perlin noise function.

- The velocity of particle i comes from f_x, f_y and the time t, through a new call to Perlin noise that adds some more randomness. It is also affected by a parameter called *pollenMass*, a measure of dispersion:

$$\left.\begin{array}{l} v_x = speed \cdot (2f_x - 1) \\ v_y = speed \cdot (2f_y - 1) \end{array}\right\}, \text{ where } speed = \frac{1 + N(t+i, f_x, f_y)}{pollenMass}$$

- After these computations, the field coordinates of the particle are set to $(x + v_x, y + v_y)$.
- The angles θ and ϕ that give the spherical coordinates of the particle position are obtained from the field coordinates by the equirectangular map $[0, w] \times [0, h] \to [0, 2\pi] \times [-\frac{\pi}{2}, \frac{\pi}{2}]$.

Adjusting the parameters *timeStep*, w, h, *complexity*, *phase*, and *pollenMass* of the vector field allows us to define a wide range of aesthetically different behaviors (see Figure 4).

Next we show the lines of code where the movement of the particles is computed, by calling the function `getVectorField();` when processing particle i.

```
ofVec2f field = getVectorField(x,y);
float speed = (1 + ofNoise(t+i, field.x, field.y)) / pollenMass;

// add the velocity of the particle to its position
x += ofLerp(-speed, speed, field.x);
y += ofLerp(-speed, speed, field.y);

float theta = ofMap(x, 0,w, 0, TWO_PI);
float phi = ofMap(y, 0,h, -PI/2, PI/2);
```

The function `getVectorField();` makes use of Perlin noise:

```
ofVec2f ofApp::getVectorField(float posX, float posY) {
  float u = ofNoise(t + phase, posX * complexity + phase, poxY * complexity + phase);
  float v = ofNoise(t - phase, posX * complexity - phase, posY * complexity + phase);
  return ofVec2f(u, v);
}
```

4 Random Distributions and Perlin noise

In this section we explain the two types of randomness used in the project. The first type is the uniformly distributed random numbers which are used in many ways over the application, especially when distributing points on the sphere's surface. And the second type is the Perlin noise function, a kind of "controlled randomness", used in the radius variation in Genera Esfera Estrident and in the vector field computation, in Genera Esfera Caniques.

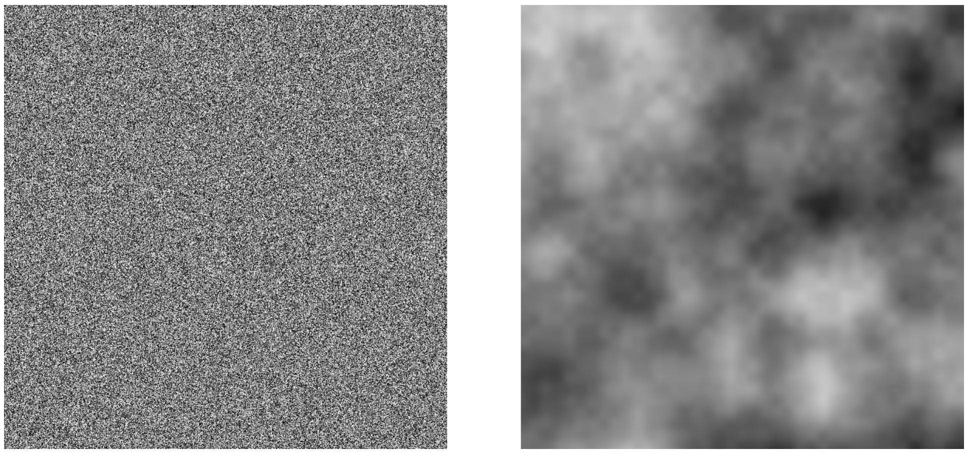


Figure 5: Two different textures to see the difference between random (left) and Perlin noise (right).

4.1 Random numbers

A random number generator gives pseudo-random numbers uniformly distributed in $[0,1]$. As in most of the generative artworks, this function is intensively used.

It is worth mentioning how this function is used to distribute points on the sphere's surface.

Notice that choosing the spherical coordinates of a point, θ and ϕ, uniformly distributed in $[0,2\pi]$ and $[0,\pi]$, respectively, does not give a uniform distribution on the sphere. Indeed, with this distribution, we get a higher concentration of points near the poles. Despite this lack of uniformity, we use this distribution in some of our scenes, for aesthetical reasons. For instance, in both screenshots in Figure 4.

A uniform distribution is obtained by taking u and v uniformly distributed in $[0,1]$, and then by defining the spherical coordinates as:

$$\begin{aligned}\theta &= 2\pi u\\ \phi &= \arccos(2v-1)\end{aligned}$$

This gives us the coordinates $(x,y,z) = (\cos\theta\sin\phi, \sin\theta\sin\phi, \cos\phi,)$ of a random point on the unit sphere.

Observation. The uniform distribution on the unit sphere has density function $f(\Omega) = \frac{1}{4\pi}$, where Ω denotes the solid angle [1] (p. 69). Using spherical coordinates, the differential solid angle is $d\Omega = \sin\phi\, d\theta\, d\phi$, which gives us the distribution $f(\theta,\phi) = \frac{\sin\phi}{4\pi}$ for the 2-dimensional variable (Θ,Φ). Then, the marginal densities are:

$$f_\Phi(\phi) = \int_0^{2\pi} f(\theta,\phi)\, d\theta = \frac{\sin\phi}{2}, \quad \text{and} \quad f_\Theta(\theta) = \int_0^{\pi} f(\theta,\phi)\, d\phi = \frac{1}{2\pi}$$

Therefore, the cumulative density functions are:

$$v = F_\Phi(\phi) = \int_0^{\phi} f_\Phi(\phi)\, d\phi = \frac{1-\cos\phi}{2}, \quad \text{and} \quad u = F_\Theta(\theta) = \int_0^{\theta} f_\Theta(\theta)\, d\theta = \frac{\theta}{2\pi}$$

Our uniform distribution is obtained by choosing u and v uniformly distributed in $[0, 1]$, and then define θ and ϕ by $v = F_\Phi(\phi)$ and $u = F_\Theta(\theta)$.

As one of the reviewers of this paper pointed out, there is another elegant method of uniformly distributing points on a sphere, a particular case of Monte Carlo sampling called rejection sampling, consisting of choosing points on a cube centered at the origin, rejecting those that are not inside the sphere with the same center and diameter of the cube, and then normalizing the accepted ones [15]. As this method does not make any use of trigonometric functions, it is likely to be more efficient than the one we use. However, this is not an issue, because the uniform distribution of points on the sphere is performed only at initialization time.

4.2 Perlin noise

The second kind of randomness is called Perlin noise. Ken Perlin is a mathematician who worked in the simulation of textures for computer graphics in films, starting back in 1981 for the known science fiction film *Tron*. In 1985 he devised the so called Perlin noise algorithm, which has been and still is widely used [12]. In 1997, he won an Academy Award for Technical Achievement from the Academy of Motion Picture Arts and Sciences, an Oscar, for his noise and turbulence procedural texturing techniques in the mentioned film *Tron*.

Perlin noise is widely used to define textures and give the objects more natural appearance, to generate organic movement and, in general, to make randomness smooth. It works with one, two or three arguments, and always returns a number between 0 and 1. By means of increasing its arguments, a series of pseudo-random values is produced. Small increases give smoother results, textures and movements.

To see how Perlin noise works, we focus on its 3-dimensional version. Perlin noise $N(a, b, c)$ is determined by computing a pseudo-random gradient at each of the eight nearest vertices of (a, b, c) on the integer cubic lattice and then doing spline interpolation.

More precisely, for each of the eight points in the set $\{(i, j, k) \mid i \in \{\lfloor a\rfloor, \lfloor a\rfloor + 1\},\ j \in \{\lfloor b\rfloor, \lfloor b\rfloor + 1\},\ k \in \{\lfloor k\rfloor, \lfloor k\rfloor + 1\}\}$, a pseudo-random gradient $\vec{g}_{i,j,k}$ is generated. Then, the eight values obtained by the dot products $\vec{g}_{i,j,k} \cdot (a - i, b - j, c - k)$ are trilinearly interpolated by $s(a - \lfloor a\rfloor)$, $s(b - \lfloor b\rfloor)$, and $s(c - \lfloor c\rfloor)$, where $s(t) = 6t^5 - 15t^4 + 10t^3$.

For more details on Perlin noise, see [12, 13]. Figure 5 illustrates the difference between random and Perlin noise, with a textures drawing example.

5 Further work

Genera Esfera is a work in progress. Still in its infancy, it has enormous potential for future growth. We have four lines of future work:

- We have already started a new version of Genera Esfera to explore new animations and visuals.
- Although the vector field we use resultes in interesting aesthetic variations, other vector fields over the sphere may give interesting results as well.
- We plan to recruit musicians to prepare a playlist specially for Genera Esfera. It may be a selection of existing songs, or a list of songs created for the project.

- Finally, we want Genera Esfera to be open to the collaboration of other visual artists. So we plan to involve other creative coders in the design of other versions of the sphere animation.

6 Conclusion

We have presented Genera Esfera, an interactive installation to explore generative worlds, its design process and the steps to develop it. A trackball with two buttons is used to perform the interaction, making the action possibilities readily perceivable by any potential user. Our work shows that clearly mapped interaction and evolving visuals are interesting enough for a specialized visual arts festival and can attract its visitors.

We have detailed how random numbers and Perlin noise have been used. By taking two different mathematical approaches, we have been able to create two distinct visual worlds: one, consisting of a set of points randomly distributed over a sphere's surface and in several layers, wire-connected when at some distance range; the other, consisting of a particle system over the sphere, driven by a noise field. This was one of our initial goals: the generation of different and interesting aesthetics.

The installation is the starting point to explore new visual graphic results, to find new uses of random numbers and Perlin noise, to add new mathematical tools to our set, and to exploit their potentiality to create generative worlds.

Acknowledgements. The authors are grateful to the two anonymous reviewers, whose comments helped to greatly improve this paper.

References

[1] W. Feller, *An Introduction to Probability Theory and Its Applications*, Vol. 2, 3rd ed. (1971), Wiley.

[2] J. P. Flynt and D. Kodicek, *Mathematics and Physics for Programmers*, 2nd ed. (2012), Course Technology.

[3] Genera Esfera on Vimeo, `https://vimeo.com/146750441` (as of Jan. 16, 2016).

[4] S. Guha, *Computer Graphics Through OpenGL: From Theory to Experiments*, 2nd ed. (2014), Taylor & Francis.

[5] G. Levin and Z. Lieberman. Sounds from Shapes: Audiovisual Performance with Hand Silhouette Contours in "The Manual Input Session". In *Proceedings of NIME '05*, Vancouver, BC, Canada. May 26-28, 2005.

[6] J. Maeda, *Creative Code*, (2004), Thames & Hudson.

[7] MIRA Festival website, `http://www.mirafestival.com` (as of Jan. 16, 2016).

[8] J. Noble, *Programming Interactivity*, (2009), O'Reilly.

[9] D. Norman, *The Design of Everyday Things*, (1988), Basic Books.

[10] openFrameworks website, `http://www.openframeworks.cc` (as of Jan. 16, 2016).

[11] M. Pearson, *Generative Art. A Practical Guide Using Processing*, (2011), Manning.

[12] K. Perlin. An image synthesizer. *ACM Siggraph Computer Graphics*, 19 (3):287–296, 1985.

[13] K. Perlin. Improving Noise. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '02)*, pp. 681–682, San Antonio, Texas. July 21-26, 2002.

[14] Processing website, `http://www.processing.org` (as of Jan. 16, 2016).

[15] C. P. Robert and G. Casella, *Monte Carlo Statistical Methods*, 2nd ed. (1999), Springer-Verlag New York.