# Vectorial drawing demystified. The beauty of Bezier and NURBS curves in 2D and 3D space

**Francesco Paduano**
University of Illinois at Chicago
700 S Halsted St 2029, Chicago, IL 60607
fpadua2@uic.edu

**Lorenzo Di Tucci**
University of Illinois at Chicago
700 S Halsted St 2029, Chicago, IL 60607
ldituc2@uic.edu

### ABSTRACT
Bezier and Nurbs curves are part of the widely underrated technologies for vectorial graphics in 3D computer graphics. Bezier curves and 2D vectorial graphics have brought surging interest especially in scalable-free web application. 3D Bezier and Nurbs curves have only been implemented for modelling purposes in some commercial product such as Maya and 3Ds Max. 3D real time implementation has been marginal so far, especially due to the computational cost that those technologies require. In this paper we present two different showcases of the beauty and the elegance of these technologies, one based on Bezier Curves in the 2D space and the second one based on Nurbs curves in the 3D space.

### Author Keywords
Bezier; Nurbs; Vectorial Drawing; curves;

## INTRODUCTION

### Mathematical Concept
Here we presente the mathematical concepts behind Nurbs and Bezier Curves.

#### *Bezier*
A Bezier curve is a parametric curve. In vector graphics they are used to model smooth curves that, differently from raw images, do not have a finite resolution. In image manipulation programs the combinations of linked Bezier curves is called "path". Paths are not bound by the limits of rasterized images and are intuitive to modify.

Bezier curves are also used in time domain, more in particular in animation and user interface design. The mathematical support for Bezier curves is provided by Bernstein Polynomial. It has been known since 1912 even tough its applicability in computer graphics has been conceived half a century later. Indeed, they have been widely publicized by a french engineer, Pierre Bezier, in 1962. Pierre Bezier used these curves to design automobile bodies for a french company, Renault. However, the first study of these curves has not been done by Mr. Bezier. The mathematician Paul de Casteljau, in 1959, first developed the study of these curves using de Casteljau's alghoritm, a numerically stable method to evaluate Bezier curves in another french automaker, Citroen.

#### *Bezier in Computer Graphics*
Bezier are used in computer graphics mainly to model smooth curves. Thanks to their simplicity, they are easy to display and easy to manipulate. Is possible to apply affine transformations such as translation and rotation simply by applying the respective transform on the control points of the curve. The most common kind of Bezier curves are the quadratic and cubic curves. Intuitively, the computation effort increase with the degree of the curve. Furthermore, it is possible to create composite Bezier Curves. A composite curve is a patching of low order Bezier curves and is used when there is the needing of more complex shapes. Usually, composite curves are commonly referred to as a "path" in vector graphics standard, as for example SVG, and vector graphics program as Adobe Illustrator.

The simplest method to rasterize a Bezier is to evaluate the curve at many closely space points and then scan convert the prozimating sequence of line segments. The problem is this technique don't guarantee a perfect smoothness. This because, the points may be spaced too far apart. It could also generate too many points where the curve is close to linear. A solution to this problem is a method that use recursive subdivision. According to this method, the points of control of the curve are checked for ensure that the approximate line segment is within a small tolerance. If this does not happen, the curve is subdivided in two halfs and the same procedure is applied recursively.

#### *Bezier Mathematical Model*
A Bezier curve is always composed by a set of control points from P0 to Pn, where n is the order of the curve (linear if 1, 2 for quadratic, 3 for cubic and so on). The first and the last point are always the end points of the curve but in general the intermediate control points do not lie on the curve. In this paper we will define the most common Bezier Curves: the cubic Bezier curve.

A cubic Bezier curve is defined in the plane (or in higher-dimensional space) by four points: P0,P1,P2 and P3. The curve starts from P0 and ends in P3 and uses the direction given by point P2. Usually, the curve does not pass through points P1 and P2 because this points are needed only to provide information regarding the direction. The distance between point P0 and P1 gives information regarding the distance traced by the curve in direction P2 before moving to P3.

The cubic Bezier curve can be defined as a linear combination of two quadratic Bezier curves:

$$\mathbf{B}(t) = (1-t)\mathbf{B}_{\mathbf{P}_0,\mathbf{P}_1,\mathbf{P}_2}(t) + t\mathbf{B}_{\mathbf{P}_1,\mathbf{P}_2,\mathbf{P}_3}(t) \text{ , } t \in [0,1].$$

The explicit form of the curve is:

$$\mathbf{B}(t) = (1-t)^3\mathbf{P}_0 + 3(1-t)^2 t\mathbf{P}_1 + 3(1-t)t^2\mathbf{P}_2 + t^3\mathbf{P}_3 \text{ , } t \in [0,1].$$

There is the possibility that, for some choices of points P1 and P2, the curve may intersect itself or contain a cuspide.

Given any 4 distinct points, is always possible to create a Bezier curve. It is also possible to compute the control points of the Bezier curve given the starting point, the ending one and the points along the curve.

*Nurbs*

Nurbs is the acronym of Non-uniform rational basis spline. It is a mathematical model used in computer graphics in order to generate and represent curves and surfaces. They are mostly used in 3D modelling and animation, computer-aided design(CAD), manufacturing (CAM) and engineering (CAE). Its wide use is thank to its great flexibility and precision for handling both analytic and modeled shapes. They can also be efficiently handled by computer programs and allow for easy human interaction. Nurbs surfaces are function of two parameters mapping to a surface in 3D space whose parameters are the control points. This surfaces can represent simple geometrical shapes in a compact form. Their success has been possible thanks to them intuitiveness and predictability. Control points are always connected directly to the curve (surface), and if they are not, they act as if were connected by a rubber hand.

*Nurbs mathematical model*

We present in this paper only the mathematical model for a NURBS curve, since they have been used in the latter example.
The general formulation of a NURBS curve is

$$C(u) = \sum_{i=1}^{k} \frac{N_{i,n}w_i}{\sum_{j=1}^{k} N_{j,n}w_j}\mathbf{P}_i = \frac{\sum_{i=1}^{k} N_{i,n}w_i\mathbf{P}_i}{\sum_{i=1}^{k} N_{i,n}w_i}$$

Where the term $N_{i,n}$ is called *basis function* and it is defined recursively as follow:

$$N_{i,n} = f_{i,n}N_{i,n-1} + g_{i+1,n}N_{i+1,n-1}$$

The index *i* present the *ith* control point, and *n* corresponds with the degree of the basis function.
The mathematical definition of the functions $f_{i,n}$ and $g_{i+1,n}$ is not reported in this report. Their value is strongly related with the *knot vector* values.

**Computational Effort**

*Stencil, then cover*

*NVIDIA Corporation* presented in [1] a novel approach to path rendering with GPU: The *Stencil, then cover* (StC).

After having createed a path object *StC* render the object in two main steps:

- **Step 1**: *Stencil* the path object into the stencil buffer. GPU provides fast stenciling of filled or stroked paths
- **Step 2**: *Cover* the path object and stencil test against its coverage stenciled by the prior step. In addition, application can configure arbitrary shading during the step and add more details later.

In conclusion, this method supports the union of functionality of all major path rendering standards with the hardware support.

*Nurbs*

Nurbs curves are not a widespread model in computer graphics because they are computationally expensive. In the paper [2] is presented a new method to evaluate and display trimmed Nurbs surfaces using the GPU. This kind of surfaces, are nowadays tassellated into triangles before being sent directly into the graphic cars. This, because there is not native hardware support for this kind of geometry. Previously, Nurbs display method relied on evaluate the curves after first approximating Nurbs patches with lower degree Bezier patches. In this paper, the authors, with their evaluation method, discovered that, for interactive display of a large number of trimmed Nurbs surfaces, the GPU-based evaluation of the exact surface is a viable option.

Furthermore, this paper highlight that the Nurbs topic is a not very considered technology. However, thanks to the evolution of the hardware, nowadays is possible to deal which these kind of curves.

## SHOWCASE I: ARTISTIC BEZIER IN A MAGNETIC FIELD

The first showcase illustrated in this paper shows the capabilities of Bezier curve in 2D drawing. This example has the merely purpose of showing the artistic capabilities of vectorial drawing in a new and original implementation.

### Concept

The original idea that stand behind this example is: *what would happen if pen ink was captured by an electromagnetic field?*
Our purposes focus on developing a tool so that the artist is able to compose an artwork. This process take place in two steps:

- **step 1**: draw the electromagnetic field by placing *electromagnetic charges* and *directional forces* on the canvas

- **step 2**: spread the ink on the canvas by using the mouse of a *wacom graphic tablet*

The ink released on the canvas is captured by the electromagnetic field and it is spread on the canvas.
This novel concept has no literature precedents. The only work that might be considered similar to our research is described in [3]

### Mathematical Background

In vector calculus, a vector field is a function which assign to every point of the space a vector. In this implementation every point of the canvas in mapped to an electromagnetic force. The resulting force is calculated as the sum of the effect of all the *electromagnetic objects* placed in the canvas. The *electromagnetic objects* designed for the purposes of the example are of three kind:

- **attractor**: an attractor has a position and an intensity. The intensity is a float positive value. An attractor applies a force which direction is always pointed towards its position and the magnitude is proportional to the attractor intensity and inversely proportional to the square distance

- **repulsor**: a repulsor is the same of the attractor but the force direction is the opposite

- **directional force**: a directional force has a position, a direction and an intensity. It applies a force which direction is the assigned force direction, and the magnitude is proportional to the attractor intensity and inversely proportional to the square distance

Each drop of ink has a *size, color, mass*. *Size* and *color* control the stroke and the color of the line, whereas the *mass* is used to compute its velocity and acceleration.

Furthermore, the user can change the value of two parameters:

- **drop life**: The life of a drop of ink expressed in milliseconds. When a drop is released on the canvas a timer start. The ink opacity is proportional to the time left to live, and reach 0 when time elapsed equalize the ink drop life.

- **ink viscosity**: The viscosity of the ink. When the field force is computed on the single ink drop it's considered also a *viscosity friction force* contribute which follows the relation

$$friction = -v_{drop} \times inkViscosity$$

where $v_{drop}$ is the current velocity of the ink drop.

### Implementation

The implementation has been made in Javascript and HTML5 Canvas. The HTML Canvas offer high - level tools for vetorial drawing and fair good performances.
When a mouse click or the pressure of a graphical tablet is detected ink drops are generated on a canvas. The *size* of the drop is proportional to the detected pressure of the pen. If a mouse is used, it is a fixed value.
At regular intervals the acceleration, velocity and position of the drops are updated according to the electromagnetic forces on the canvas.

### Results

**Figure 1. Screenshot of the application with two forces**

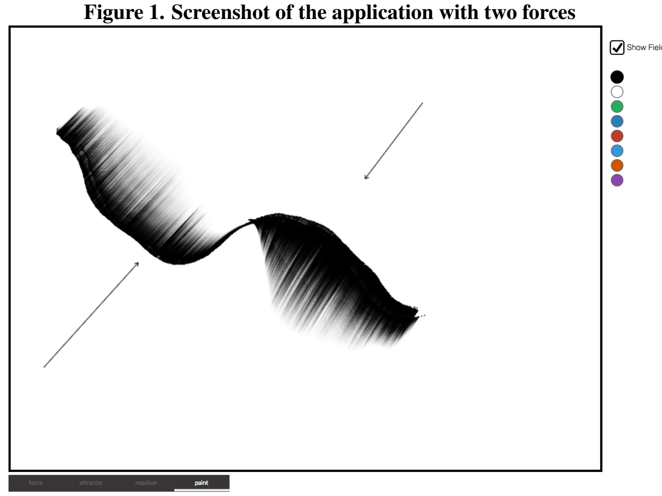


Figure 1 is a screenshot of the application. On the bottom there are 4 buttons which let the user to choose between 4 different modalities. The first three modalities let the user to add respectively a directional force, an attractor and a repulsor whereas the fourth modality lets the user to paint the canvas.
On the left is is possible to show or hide the symbolic representation of the electromagentic field and to choose the color of the ink from a predefined color palette.
Figure 1 shows a simple case of how the ink placed on the canvas is affected by the electromagnetic field.
By changing the value of *drop life* and *viscosity* the artist can realize quite different effects. Figure 2 illustrates the different results for two different parameters configuration. Both drawings are made with the same vectorial field. The blue drawing on the left has been made with a *drop life* equals to 12000 and a *viscosity* factor equals to 0.5. The red drawing on the right presents the following parameters: *drop life* = 2000 and *viscosity* = 0.0. A low viscosity value let the ink to aquire

**Figure 2. Two different value for viscosity and drop life on the same vectorial field. On the left drop life = 12000 and viscosity = 0.5. On the right drop life = 2000 and viscosity = 0.0**

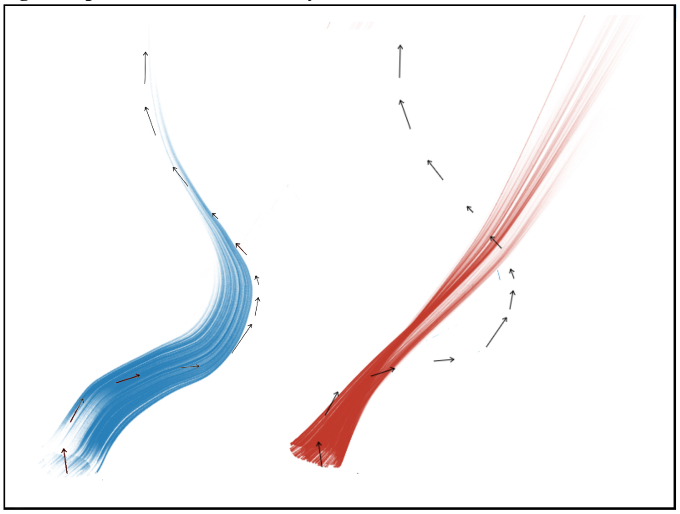


**Figure 3. Screenshot of a more complex field**

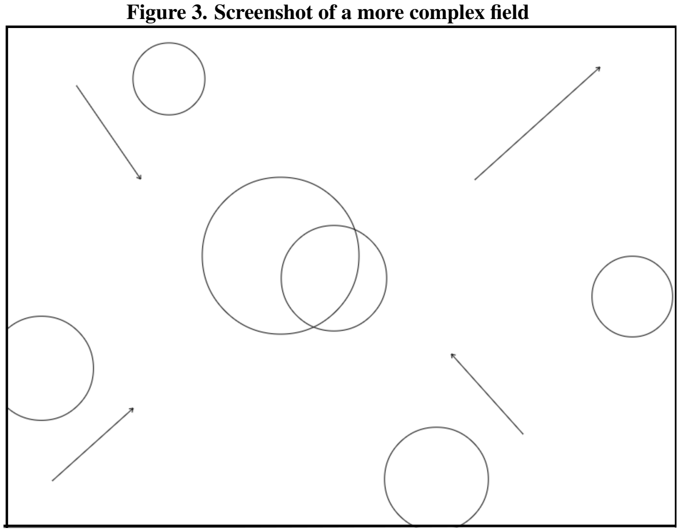
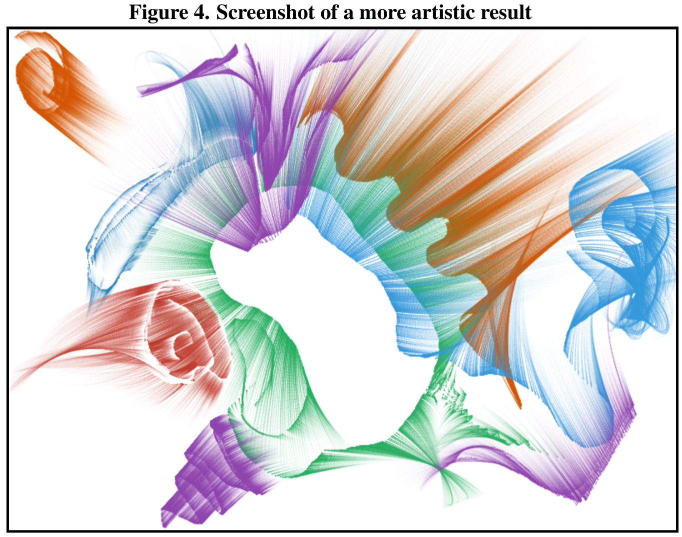


**Figure 4. Screenshot of a more artistic result**



a high acceleration and to squirt on the canvas without pronunced changes of direction. An high value of viscosity slow down the ink drops and make them follow more accurately the forces of the vectorial field.

Figure 3 reports a more complex example of vectorial field. The *electromagnetic objects* placed on the canvas are of two kinds. The arrows are *directional forces*, *attractor* and *repulsor* are represented by circles.

Figure 4 is a more artistic realization which makes use of the vectorial field of Figure 1 underneath.

**Figure 5. Screenshot of two flowers**

## SHOWCASE II: PAINTING NURBS ON A MESH

In this second showcase we want to present a quite spectacular effect obtained by intensively using raytracing and NURBS curve in a 3D scene.

### Concept

The scene is composed by one single invisible mesh, in this case an human head.
The basic concept is to trace a bundle of curves on the mesh. The curves flows on the mesh surface and delineate the shape of the object. The lines are not casted inside the mesh but are projected in a way that border on the object. In this way, we have as effect that the line are drawing the borders of the object. The mesh is actually made visible by the curves that follow its layout.
This method is not constrained to a single geometry. It is indeed possible to draw every kind of object. Obviously, the quality of the drawing increases as the number of lines drawn increases. Therefore, if lots of lines are used there would be the possibility to see more peculiarities of the object, but the computational effort required will be higher.

### Method

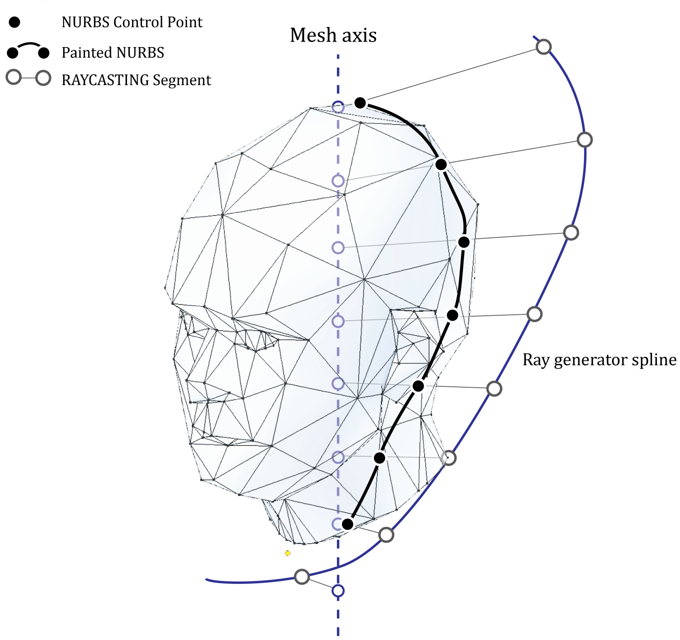



**Figure 6. Schematic View - Nurbs**

In this section we present the method used to achieve the described effect.
In the first place the mesh is loaded by the program and stored in memory. Then, **n** *Ray Generator splines* are generated around the mesh. A *Ray Generator spline* is a NURBS curve which twists around the object at a fixed distance from the vertical axis of the object.
Each *Ray Generator spline* is then subdivided in **m** equal segments. A ray is projected from each vertex of each segment toward the vertical axis of the mesh. If the ray intersect the mesh a *NURBS Control Point* is generated. The set of *NURBS Control Point* generated from each *Ray Generator spline* are the control points of the *painted NURBS*.
A *painted NURBS* is a NURBS curve that is rendered on the screen which strokes the shape of the loaded mesh.
The execution of the program is divided in time intervals of variables length. At the beginning of each time interval a number **s** of *painted NURBS* is generated. The *painted NURBS* change its opacity procedurally in order to obtain the effect like it was 'flowing' along the mesh.
The computationally - expensive part of the software is due to the huge number of raycast operation performed, which is $n*m$. At the start-up of the application the user can customize the following parameters:

- **number of control lines**: the number **n** of *Ray Generator splines* used. Lower is the number, more detail of the mesh bight be lost.
- **rays for line**: number **m** of segments the *Ray Generator splines* is subdivided in. That is the number of raycast computed for each *Ray Generator splines*
- **curling**: how many times every *Ray Generator splines* twists around the vertical axis of the mesh. A curling index equals to zero means that all the *Ray Generator splines* are perfectly vertical lines
- **curling randomness**: a randomness factor added to the curling index
- **line speed**: speed of the flow of the *painted NURBS*
- **line life**: the time that takes every *painted NURBS* to disappear
- **lines spawn**: how many *painted NURBS* are generated every time interval

### Implementation

The application has been implemented with *Three.js*[1], a popular and powerful Javascript library for WebGL.
Our implementation relies on *Three.js* mainly for the helper functions for loading an external *obj* file and for the raycasting operations.
The utility used for rendering the NURBS curve is built in in the *Three.js* engine. It relies on a standard implementation which tesselate and pass a triangulated geometry to the vertex shader.

### Results

In this section we present some screenshots of our program obtained with different configurations.
For each tested configuration we presents two screenshot. One is taken after ten seconds from the start of the application when the lines are still partially drawn. The second screenshot is taken after 20 seconds when the lines has reached a stable density.
The following table reports the parameters values for *configuration 1*

[1] http://threejs.org/

| Parameter | Value |
|---|---|
| number of control lines | 700.0 |
| rays for line | 40 |
| curling | 3 |
| curling randomness | 1 |
| line speed | 2.0 |
| line life | 4.0 |
| line spawn | 5 |

Figure7 shows the effect obtained after 10 seconds of execution. Note how skewed the lines are. This effect is obtained by setting the curling parameter to 3 and the curling randomness to 1. Figure8 depicts the scene after 20 seconds. Note that the geometry is not completely filled up with lines. This is mostly due to the relatively low line life and low value of line spawn.

The following table reports the parameters values for *configuration 2*

| Parameter | Value |
|---|---|
| number of control lines | 1500.0 |
| rays for line | 50 |
| curling | 0 |
| curling randomness | 0 |
| line speed | 0.5 |
| line life | 12.0 |
| line spawn | 12 |

Figure9 and Figure10 shows respectively the effect obtained after 10 and 20 seconds of execution. The lines all parallels to each other are obtained by setting both the *curling* and *curling randomness* set to 0.
The geometry is almost completely wrapped by lines. This effect is realized by setting a relatively high value of *line spawn* and *line life*.

**Figure 7. Configuration 1 after 10 seconds**

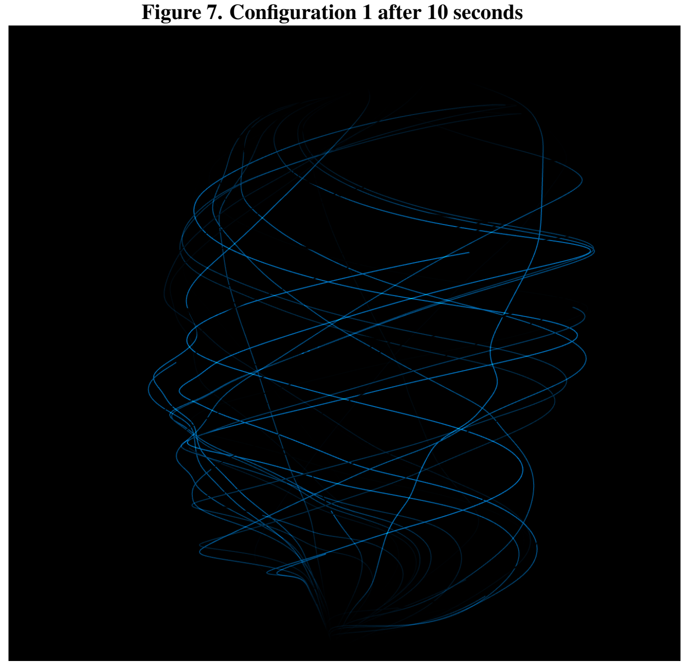


**Figure 8. Configuration 1 after 20 seconds**

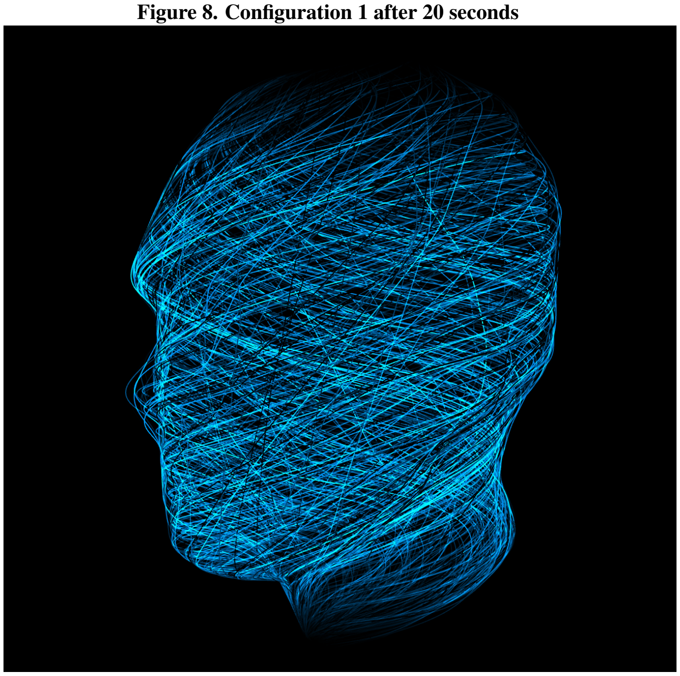


**Figure 10. Configuration 2 after 20 seconds**

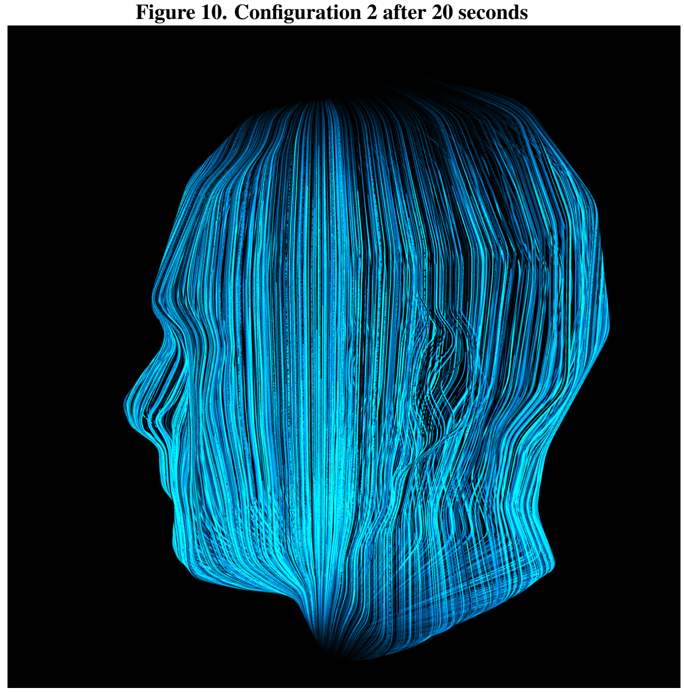
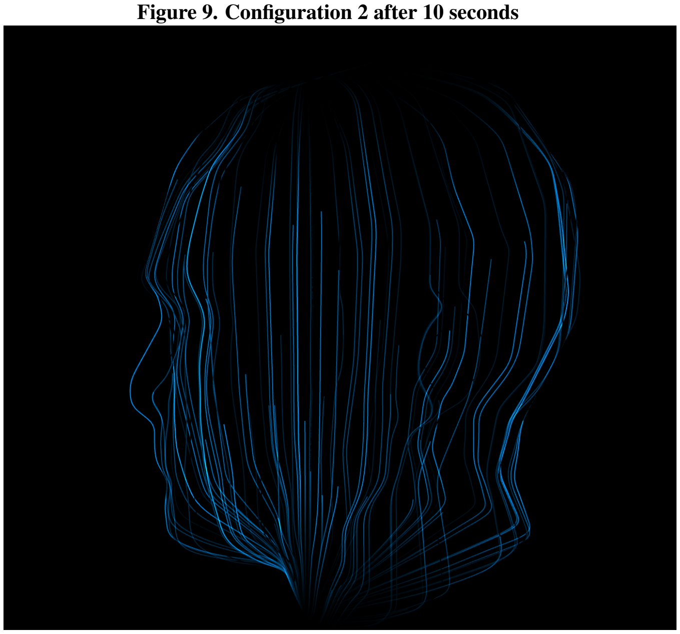


**Figure 9. Configuration 2 after 10 seconds**



## CONCLUSION

The use of Bezier curves to draw objects and artistic features is not new practice. The literature is full of examples that creates a very interesting effect.
On the contrary, we believe that the use of Nurbs curves to create artisisc effect in 3D computer graphics haven't been explored enough. Our work demonstrate that the effects that can be obtained with the pure use of these technologies are very neat and spectacular.
All this effects can be executed real-time nowadays thanks to the surging evolution of the hardware. The developments made in processors and GPUs allows the extensive use of this techniques that in the past were prohibitiive due to the high computational effort.
The showcases that have been presented in this paper show an intensive use of vectorial primitives to accomplish the task given. The environment in which these primitives are executed is the web. This has been done because, nowadays, the web is the instrument to which all the new technologies are moving to. His accessibility and popularity are the keys point of its success.

## REFERENCES


1. Mark J. Kilgard, Jeff Bolz - NVIDIA Corporation (2011) *GPU-accelerated Path Rendering*. ACM, (2012).
2. Adarsh Krishnamurthy, Rahul Khardekar, Sara McMains, - University of California, Berkeley (2011), *Direct evaluation of NURBS curves and surfaces on the GPU*. Proceeding SPM '07 Proceedings of the 2007 ACM symposium on Solid and physical modeling Pages 329-334.
3. D. A. Schroeder, D. Coffey, and D. F. Keefe (2010), *Drawing with the Flow: A Sketch-Based Interface for Illustrative Visualization of 2D Vector Fields*