USING A PROCEDURAL COLOR PALETTE IN GEOGEBRA

Juan Carlos Ponce Campuzano

School of Environment and Science, Griffith University, Australia

Abstract

Creating smooth and aesthetically pleasing color transitions in GeoGebra can be challenging, especially when manually adjusting RGB or HSV values. This paper presents a simple yet powerful procedural color palette function based on cosine transformations, allowing for effortless dynamic color generation. By leveraging this function, users can achieve continuous color variation without the complexity of switching between color spaces. We explore the mathematical foundation of this method and its practical benefits in mathematical visualization.

Keywords: procedural color, GeoGebra scripting, color function, color palettes, trigonometric functions

1 INTRODUCTION

In mathematical visualization, color is an essential tool for highlighting patterns, distinguishing elements, and improving clarity. GeoGebra offers built-in tools for color selection, but dynamically adjusting colors often requires converting between RGB (Red, Green, and Blue) and HSV (Hue, Saturation, and Value) models, which can be tedious. Inspired by the work of Iñigo Quilez (nd), this paper explores an alternative approach: a procedural color palette function that generates smooth color transitions using cosine functions. This method enables efficient and automated color generation, which is particularly useful in GeoGebra.

2 THE PROCEDURAL COLOR PALETTE FUNCTION

Mathematical Formulation

The procedural color palette function is defined as:

$$\textbf{color}(x) = \mathbf{A} + \mathbf{B} \cdot \cos\big(2\pi(\mathbf{C} \cdot x + \mathbf{D})\big).$$

Here:

- x is the input parameter (e.g., a normalized value between 0 and 1),
- $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ are vector parameters controlling the color transformation,
- The function applies the cosine transformation component-wise to vectors.

This formulation allows for continuous and smooth color variations without requiring manual adjustments.

Applying the Cosine Function to a Vector

In OpenGL Shading Language (GLSL) and many shading languages, when you apply $\cos(x)$ to a vector, it operates *component-wise*. This means that if $v = (v_1, v_2, v_3)$ is a three-dimensional vector, then

$$\cos(v) = (\cos(v_1), \cos(v_2), \cos(v_3)).$$

This element-wise function application simplifies procedural color generation, as the cosine transformation is applied uniformly across the RGB components.

Interpretation and Applications

By modifying the parameters $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$, users can generate a wide range of color palettes.

A	**B**	**C**	**D**
0.5, 0.5, 0.5	0.5, 0.5, 0.5	1, 1, 1	0, 0.333, 0.667
			

Table 1. Rainbow colors equivalent to HSV.

A	**B**	**C**	**D**
1, 0.5, 0.5	0.5, 0.5, 0.5	0.75, 1, 0.666	0.8, 1, 0.333
			

Table 2. Green-Magenta-Cyan gradient.

A	**B**	**C**	**D**
0.5, 0.5, 0.5	0.5, 0.5, 0.5	0.8, 0.8, 0.5	0, 0.2, 0.5
			

Table 3. Orange-Blue gradient.

The parameters **A** and **B** control brightness and contrast of the color.

A	B	C	D
1, 1, 1	**0.5, 0.5, 0.5**	1, 1, 1	0, 0.333, 0.666
			

Table 4. High brightness.

A	B	C	D
0.5, 0.5, 0.5	**1, 1, 1**	1, 1, 1	0, 0.333, 0.666
			

Table 5. High contrast.

The parameter **C** controls the frequency of the colors and **D** the type of colors. The cosine-based structure of the function ensures smooth transitions, which makes it particularly useful for visualizing gradients, animations, and structured color schemes.

A	B	C	D
0.5, 0.5, 0.5	0.5, 0.5, 0.5	**2, 2, 2**	0, 0.333, 0.666
			

Table 6. The rainbow colors repeat.

A	B	C	D
0.5, 0.5, 0.5	0.5, 0.5, 0.5	1, 1, 1	**1.36, -1.36, -0.8**
			

Table 7. A different color palette.

The function **color**(x) is based on an approach introduced by Iñigo Quilez (nd) for procedural color generation. His method, widely used in shader programming and computer graphics, simplifies color transitions without requiring complex transformations between color spaces. By applying this method in GeoGebra, we enable more efficient color manipulation in mathematical visualizations.

2.1 GeoGebra Tool for Visualizing the Color Palette Function

To facilitate the exploration of the function

$$\mathbf{color}(x) = \mathbf{A} + \mathbf{B} \cdot \cos\big(2\pi(\mathbf{C} \cdot x + \mathbf{D})\big),$$

can access the interactive GeoGebra tool (Ponce Campuzano (2025)), available at:

`https://www.geogebra.org/m/rnzrfxph`

This tool allows users to dynamically modify the parameters $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ and observe the resulting color palette in real time. In addition, it provides a list of predefined color palettes and an option to export the vector values of $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$, enabling seamless integration into GeoGebra or other environments for further experimentation.

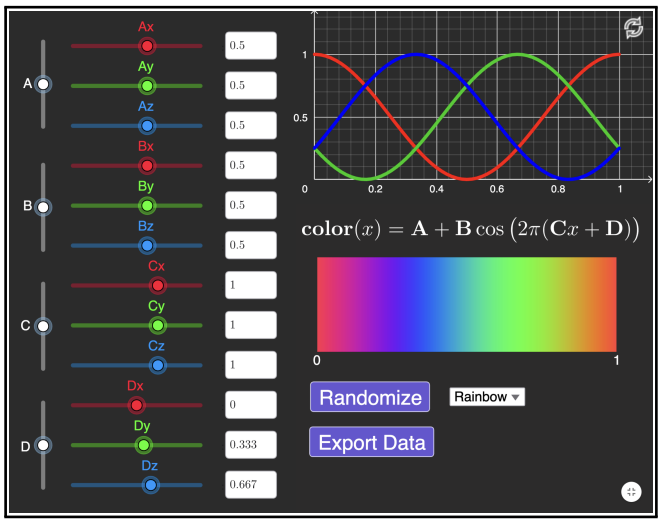


Figure 1. Tool for exploring color palettes.

3 Implementation in GeoGebra

3.1 Initial Setup

To implement the vector function **color**(x) in GeoGebra, we need to consider it in terms of the RGB components. That is

$$\begin{aligned} \text{functionRed}(x) &= A_x + B_x \cdot \cos\big(2\pi(C_x \cdot x + D_x)\big), \\ \text{functionGreen}(x) &= A_y + B_y \cdot \cos\big(2\pi(C_y \cdot x + D_y)\big), \\ \text{functionBlue}(x) &= A_z + B_z \cdot \cos\big(2\pi(C_z \cdot x + D_z)\big), \end{aligned}$$

where $\mathbf{A} = (A_x, A_y, A_z)$, $\mathbf{B} = (B_x, B_y, B_z)$, $\mathbf{C} = (C_x, C_y, C_z)$, and $\mathbf{D} = (D_x, D_y, D_z)$.

Remark 1. *For this implementation, I used GeoGebra Classic 5 for desktop. However, this method also works for the recent versions of GeoGebra, Classic 6 or Suite.*

Now, using the tool mentioned in the previous section, we select a color palette. For example, in the GeoGebra Input box we type the following values, line by line:

```
A = (0.5, 0.5, 0.5)
B = (0.5, 0.5, 0.5)
C = (1, 1, 1)
D = (0, 0.33, 0.67)
```

Then we input each one of the RGB components, defined as functions:

```
functionRed(x) = x(A) + x(B) * cos(2pi(x(C) * x + x(D)))
functionGreen(x) = y(A) + y(B) * cos(2pi(y(C) * x + y(D)))
functionBlue(x) = z(A) + z(B) * cos(2pi(z(C) * x + z(D)))
```

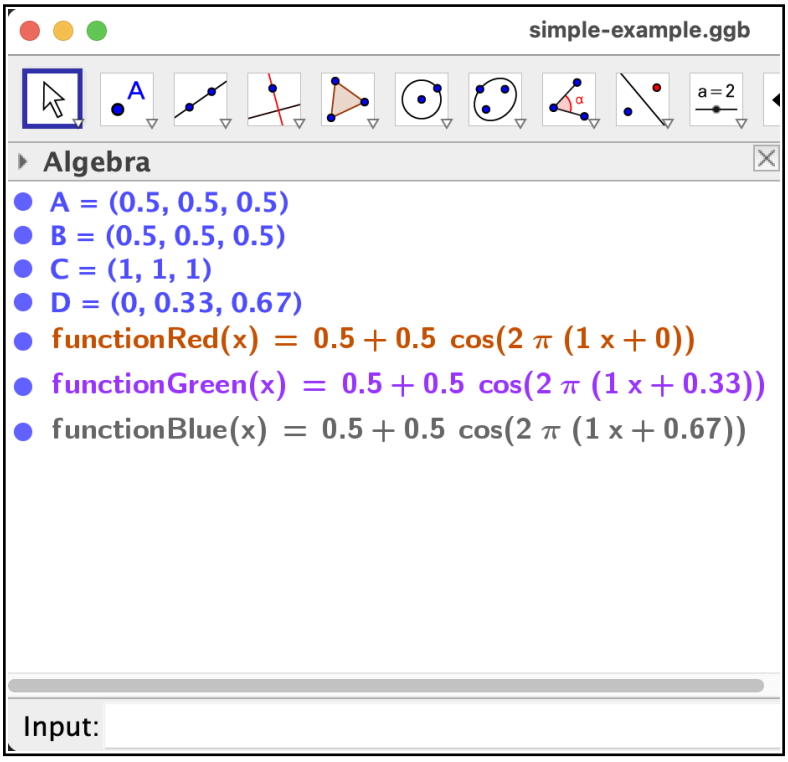


Figure 2. Vectors A, B, C, D and RGB functions.

Remark 2. *By default, GeoGebra assigns a color to each function as it is created. These colors are arbitrary and do not correspond to the names we assign to the functions.*

3.2 Simple example

To use the RGB functions, first create a slider from 0 to 1, with increment of 0.01; and a circle with radius 1, at the origin:

```
t = Slider(0, 1, 0.01, 1, 200)
c = Circle((0, 0), 1)
```

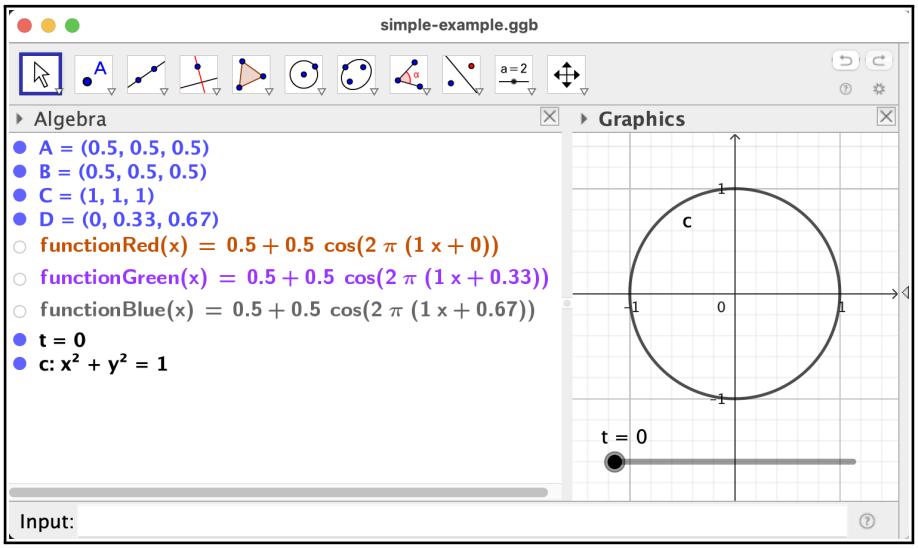


Figure 3. Circle and slider in the graphics view.

Now, we can use the `SetDynamicColor` command (GeoGebra, 2025c), in particular,

```
SetDynamicColor(<Object>, <Red>, <Green>, <Blue>, <Opacity>)
```

to apply the color using the RGB color scheme and change its opacity. So in the input box, we type:

```
SetDynamicColor(c, Min( 1, Max( 0, functionRed(t) ) ), Min(1 , Max( 0,
    functionGreen(t) ) ), Min( 1, Max( 0, functionBlue(t) ) ), 1)
```

The `SetDynamicColor` command sets the RGB color components of the circle with the corresponding RGB functions, that we defined previously. By the way, instead of using this command, we can manually type the RGB functions in the corresponding inputs of the circle's properties, as shown in Figure 4.

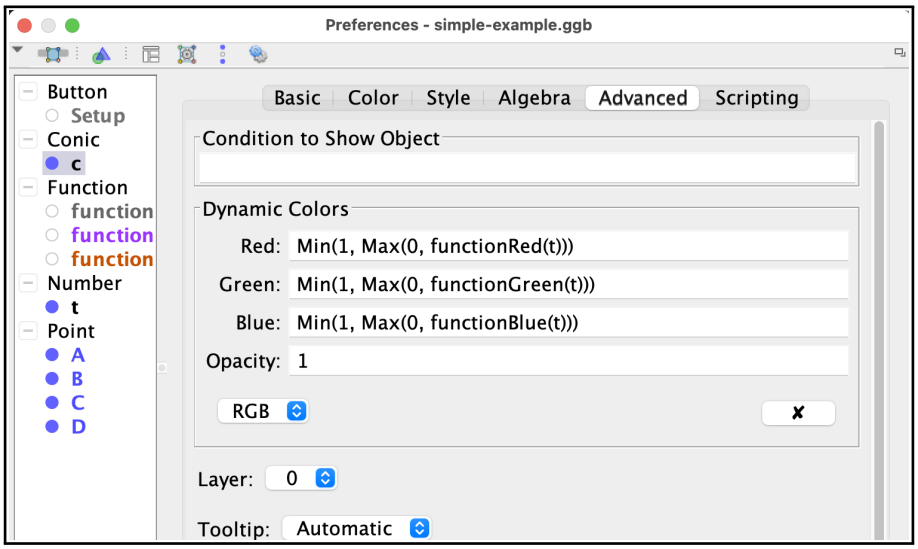
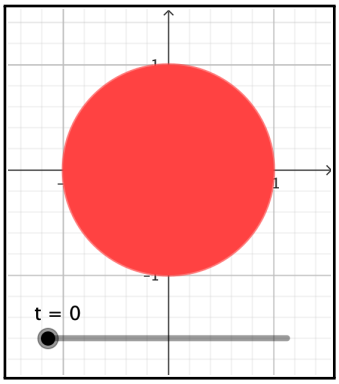


Figure 4. Settings of the circle. The color scheme of the object is always RGB by default.

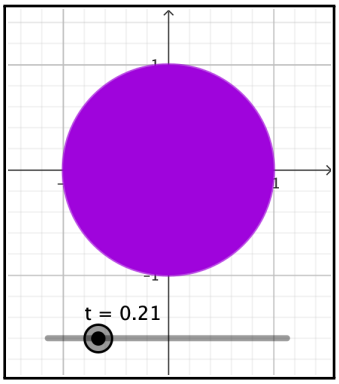
Remark 3. *Notice also that to enforce the values of our RGB functions to be within the interval* $[0, 1]$ *we introduce the* `Min` *and* `Max` *commands.*

- `Max(0, functionRed(t))`: *Ensures the function never goes below 0.*
- `Min(1, ...)`: *Ensures the function never goes above 1.*

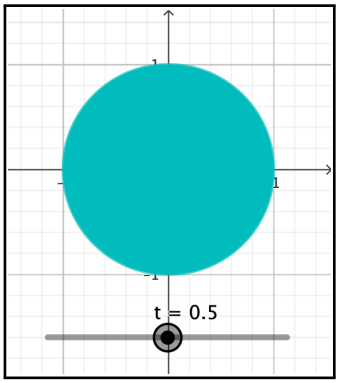
The color of the circle can be easily adjusted by dragging the slider, as shown in Figure 5. In particular, this implementation produces a similar color scheme to HSV while keeping the default RGB color scheme of the object (see Figure 4).



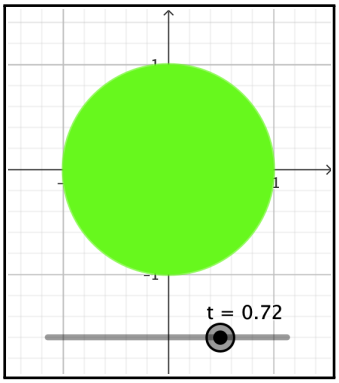
(a) Red `t=0`.



(b) Purple `t=0.21`.



(c) Light Blue `t=0.5`.



(d) Light Green `t=0.72`.

Figure 5. Different values of `t` for different colors. This color palette is similar to the HSV scheme.

3.3 Advanced examples

Now, let us see a couple of more advanced examples where this color implementation is particularly useful.

Example 1

Here we will use the method described in Ponce Campuzano (2021). First, we create a class of circles with specific labels:

```
Execute( Zip("C"+i+" = Circle((4 * random(), 4 * random()), RandomUniform(0.2,
    1))", i, 1..10) )
```

Here, we also used the `Zip` and `RandomUniform` commands, see GeoGebra (2025d) and GeoGebra (2025a).

This time we will use a different color palette Orange-Blue, defined in Table 3. Again, using the `SetDynamicColor` command, we will set the colors of the circles with the color functions:

```
Execute( Zip("SetDynamicColor(C"+i+", Min(1, Max(0, functionRed("+i+"/10))), Min
    (1, Max(0, functionGreen("+i+"/10))), Min(1, Max(0, functionBlue("+i+"/10))),
     1)", i, 1..10) )
```

The result is shown in Figure 6. And of course, using the color palette from the simple example, we obtain an HSV color scheme as shown in Figure 7.

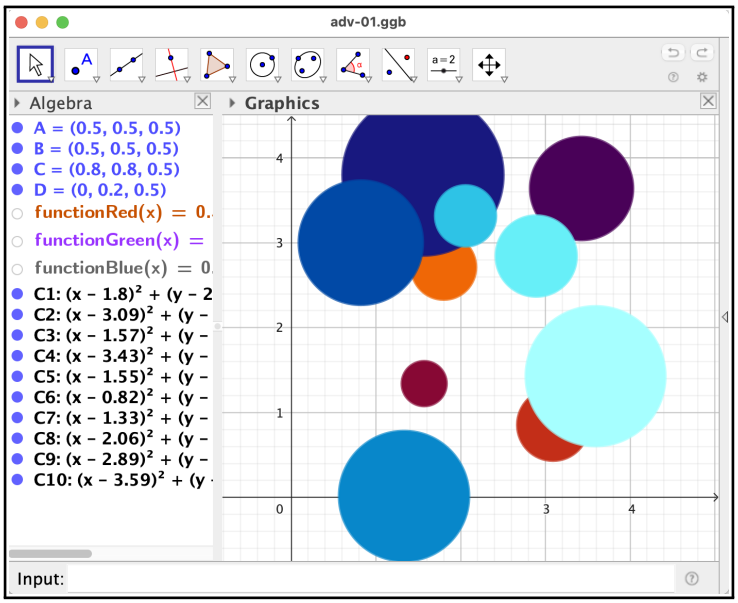


Figure 6. Circles colored with the Orange-Blue color palette.

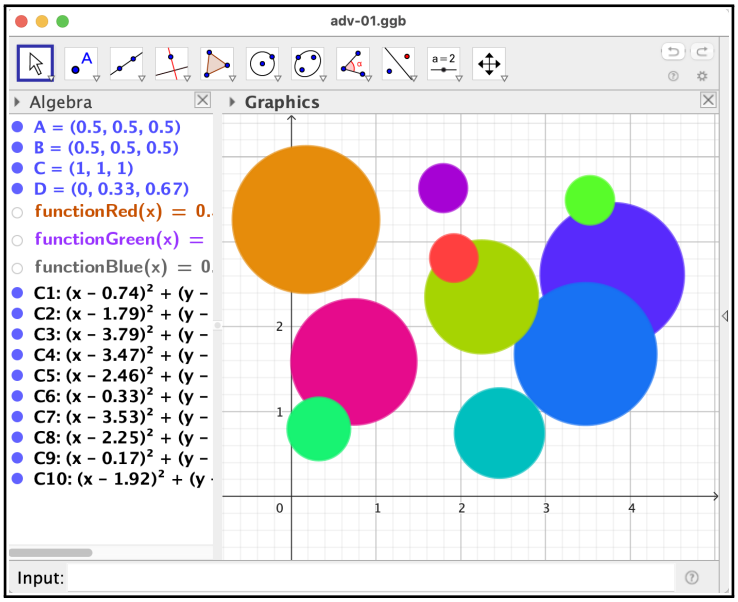


Figure 7. Circles colored with the Rainbow color palette (similar to HSV).

Example 2

Now, let us make a more interesting construction. We will use the parametric curve for the infinity symbol:

$$\begin{cases} x(t) &= \dfrac{\sqrt{2}\cos t}{1+\sin^2 t} \\ \\ y(t) &= \dfrac{\sqrt{2}\cos t \sin t}{1+\sin^2 t} \end{cases}$$

The main idea is to define a number of labeled circles, with different radii, that move on the infinity curve to create the effect shown in Figure 8.

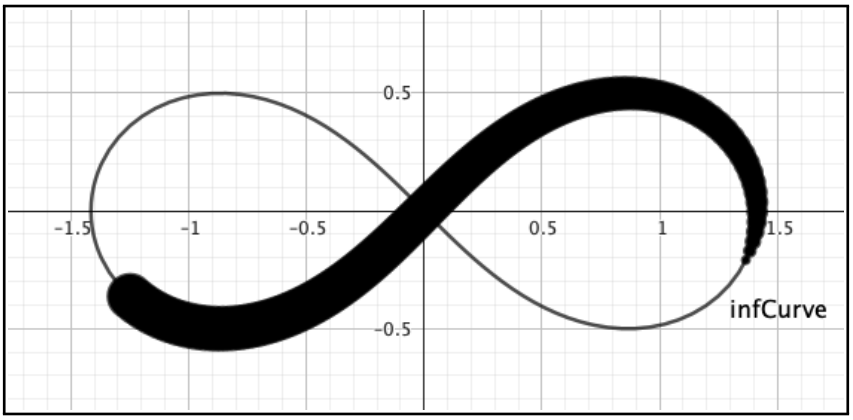
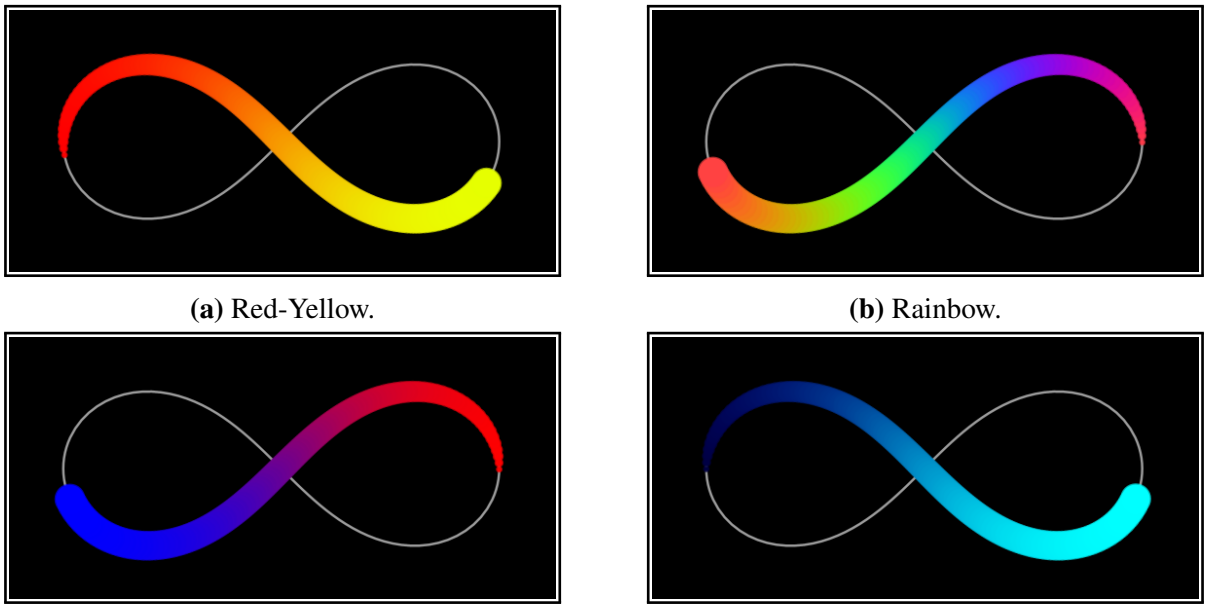


Figure 8. Parametric curve with circles of different radii.

To accomplish this, we just need to input the following GeoGebra commands, line by line:

```
infCurve = Curve( sqrt(2)*cos(t)/((sin(t))^2 + 1),
             sqrt(2)*cos(t)*sin(t)/((sin(t))^2 + 1) , t, 0, 2pi )

t = Slider(0, 2pi, 0.01, 2, 200)

LPs = Sequence(infCurve(k + t), k, 0, 3, 0.03)

m = Length(LPs)
Lm = 1..m

Execute(Zip("C"+k+"=Circle(Element(LPs, "+k+"), 0.01 * ln("+(k+1)+"))", k, Lm))

Execute( Zip("SetDynamicColor(C"+k+", Min(1, Max(0, functionRed("+k+"/m))), Min
    (1, Max(0, functionGreen("+k+"/m))), Min(1, Max(0, functionBlue("+k+"/m))),
    1)", k, Lm) )
```

With the `Sequence` command we create a list of points on the curve (see North American GeoGebra Journal Staff (2021) and GeoGebra (2025b)). Then we use again the method from Ponce Campuzano (2021) to create the labeled circles and color them dynamically with the `SetDynamicColor` command. After hiding the labels and the auxiliary objects, we obtain the construction shown in Figure 8 (see Edwards and Quinlan (2021)). Finally, we introduce the RGB functions to explore different color palettes, as shown in Figure 9.



(a) Red-Yellow.

(b) Rainbow.

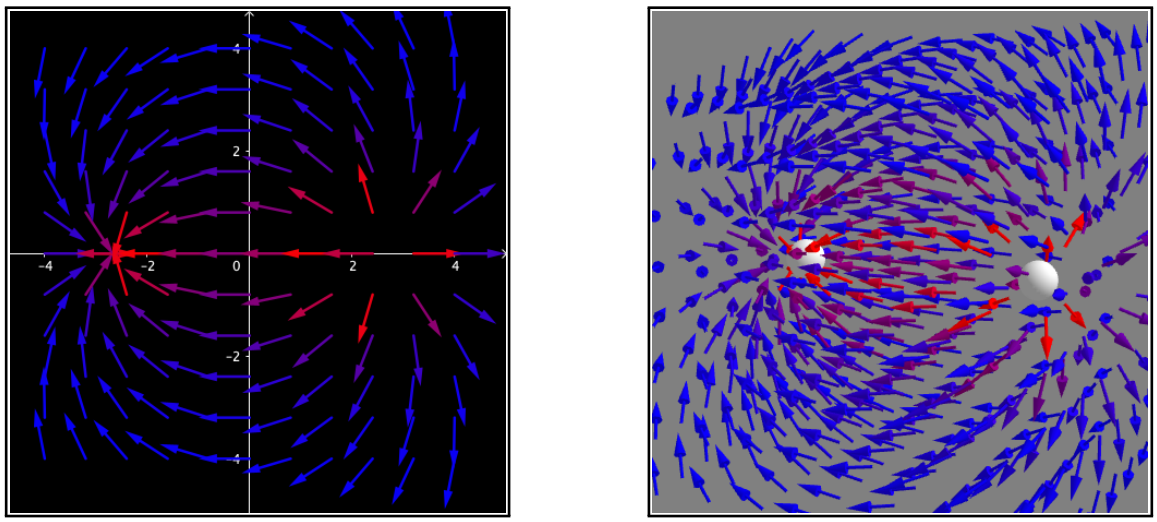
(c) Blue-Red.

(d) Blue-Cyan.

Figure 9. The infinity symbol with different color palettes.

4 Final Comments

The procedural color palette function presented in this paper provides an efficient and flexible approach to generating smooth color transitions in GeoGebra without the need for manual RGB-HSV conversions. By leveraging cosine-based transformations, this method enables the creation of diverse color gradients through simple parameter adjustments. This approach can be particularly useful for visualizing mathematical and physical phenomena, such as plotting magnetic fields in 2D or 3D within GeoGebra, where smooth color transitions can enhance the representation of vector magnitudes and directions, as shown in Figure 10.



(a) 2D.

(b) 3D.

Figure 10. Vector fields.

To facilitate the exploration of RGB color functions, an online tool has been developed which enables users to dynamically visualize the effect of different parameter choices and obtain the corresponding color data with ease. This tool provides an intuitive interface for experimenting with smooth transitions and can be particularly useful for designing custom palettes for mathematical visualizations and interactive learning environments. Finally, the full GeoGebra code for the initial setup is available in the appendix. The live demos for the advanced examples, including the GGB code, are all available at:

`https://www.geogebra.org/m/x8evngvf`

REFERENCES

Edwards, T. and Quinlan, J. (2021). Layering: Showing and hiding objects. *North American GeoGebra Journal*, 9(1):8–10.

GeoGebra (2025a). Random Uniform Command. `https://geogebra.github.io/docs/manual/en/commands/RandomUniform/`. [Online; accessed 14-Feb-2025].

GeoGebra (2025b). Sequence Command. `https://geogebra.github.io/docs/manual/en/commands/SetDynamicColor/`. [Online; accessed 14-Feb-2025].

GeoGebra (2025c). SetDynamicColor Command. `https://geogebra.github.io/docs/manual/en/commands/SetDynamicColor/`. [Online; accessed 14-Feb-2025].

GeoGebra (2025d). Zip Command. `https://geogebra.github.io/docs/manual/en/commands/Zip/`. [Online; accessed 14-Feb-2025].

North American GeoGebra Journal Staff (2021). Lists and sequences. *North American GeoGebra Journal*, 9(1):25–30.

Ponce Campuzano, J. C. (2021). On coloring different objects of the same class. *North American GeoGebra Journal*, 9(1):31–35.

Ponce Campuzano, J. C. (2025). Tool for exploring color palettes. `https://www.geogebra.org/m/rnzrfxph`. [Online; accessed 14-Feb-2025].

Quilez, I. (n.d.). Better color palettes for procedural art. `https://iquilezles.org/articles/palettes/`. [Online; accessed 14-Feb-2025].

Juan Carlos Ponce Campuzano, (j.poncecampuzano@griffith.edu.au), teaches mathematics and works on the design and integration of online learning modules and interactive mathematical applets at the School of Environment and Science, Griffith University, Australia. Juan Carlos's professional interests include the design and development of open-source mathematics applets and interactive online books. Learn more about his projects here: `https://www.jcponce.com/p/projects.html`.

APPENDIX - SIMPLE EXAMPLE CODE

```
#=============================================================
# Define points for the color palette
#=============================================================
A = (0.5, 0.5, 0.5)
B = (0.5, 0.5, 0.5)
C = (1, 1, 1)
D = (0, 1/3, 2/3)

#=============================================================
# Define RGB functions
#=============================================================

functionRed(x) = x(A) + x(B) * cos(2pi(x(C) * x + x(D)))
functionGreen(x) = y(A) + y(B) * cos(2pi(y(C) * x + y(D)))
functionBlue(x) = z(A) + z(B) * cos(2pi(z(C) * x + z(D)))

#=============================================================
# Create slider and circle
#=============================================================

t = Slider(0, 1, 0.01, 1, 200)
c = Circle((0, 0), 1)

#=============================================================
# Apply color with the SetDynamicColor command
#=============================================================

SetDynamicColor(c, Min(1, Max(0, functionRed(t) )), Min(1, Max(0,
   functionGreen(t) )), Min(1, Max(0, functionBlue(t) )), 1)
```