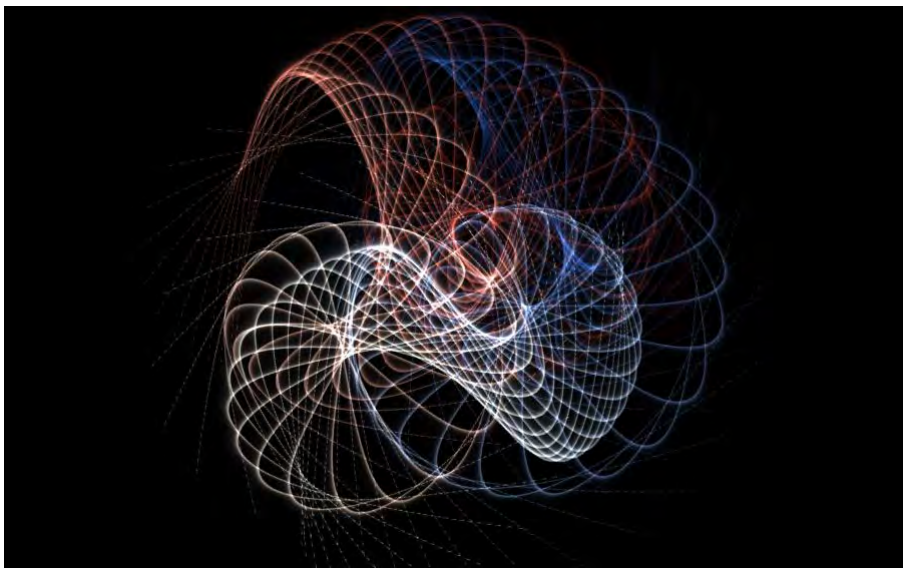


BEZIER CURVE PARTICLE SYSTEM

Pairing a GLSL Shader and 3D Particles

TUTORIAL

Isadora has great tools for developing particle systems. In this tutorial the steps for preparing a GLSL Shader and then pairing it with a 3D Particles module is explored.

Bonemap

2020

Two Point Bezier Curve Particle System

1 Preparing the GLSL Shader

In this tutorial we will build a particle system in Isadora.
In this system we will use a GLSL Shader adapted from The Book of Shaders by Patricio Gonzalez Vivo and Jen Lowe. Use the link below to view the source code for the GLSL Shader we will be adapting.

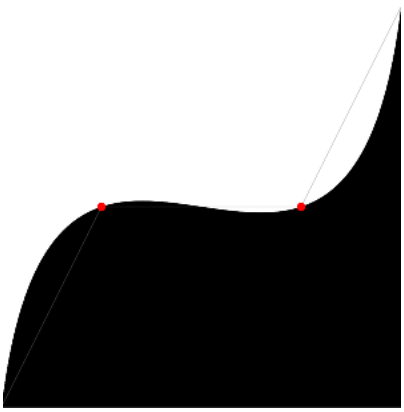
https://thebookofshaders.com/edit.php?log=160414041756

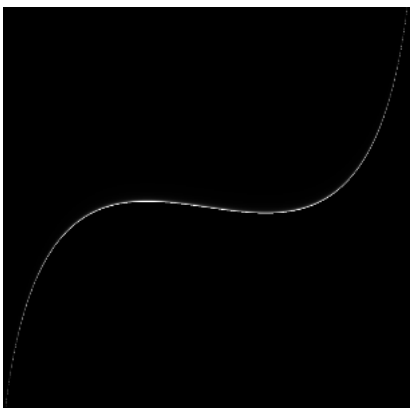
What is happening?

We have accessed source code that animates control points to demonstrate a Bezier curve. Wow – this is great – so let's adapt the code to produce a custom Isadora module.

What we will do

- Simple modification of the source code to work as a GLSL module in Isadora.
- The module will have inputs added to control the Bezier points in the horizontal and vertical (x and y) directions.
- Modify the code so that the black and white halves become a tapering white line
- The grey lines and the red dots indicating the relationship between the control points will be removed.


output of the source code

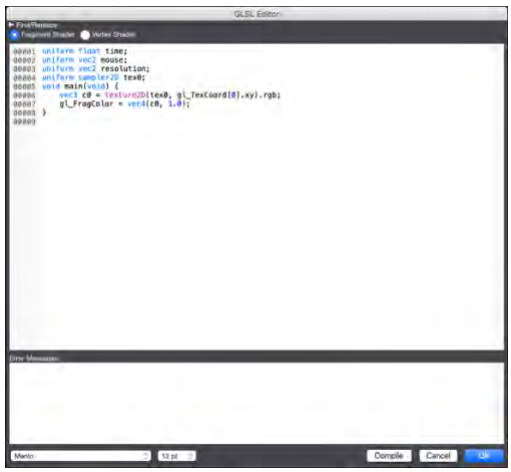

output after modification

There is a download from the Isadora community plugins page here:
https://troikatronix.com/plugin/two-point-bezier-shape/
The file from the Isadora Plugins page has pre-prepared code to work as an Isadora module. It has control point inputs already configured. In addition the red dots and straight lines have been removed. It will require a little more modification of the code to achieve the tapering white line. It will save you some time so go ahead and download it.

The following list of actions prompts you to organise a functioning GLSL Shader actor within an Isadora patch window.

1. download the 'Two Point Bezier Shape' file from the Isadora Plugins Page
2. open the file in a text reader application
3. copy the code – all of it!
4. open Isadora
5. place a new 'GLSL Shader' actor in the Scene Editor patch window
6. double click the 'GLSL Shader' actor to reveal the code editor
7. delete the existing code
8. paste the copied code into the GLSL code editor window
9. activate the code by clicking the 'compile' button
10. close the code editor window by clicking OK

The GLSL Shader actor in Isadora allows you to paste in shader code from other sources



Double clicking on the GLSL Shader actor reveals the code editor where you can cut, paste and modify the text.

Before we get started with tweaking the source code, consider accessing the 'how to' pages from the Isadora website: https://support.troikatronix.com/support/solutions/articles/13000025645-glsl-shader-actor-tutorial

The official Isadora GLSL Shader Actor Tutorial provides advice on managing the adaptation of shader source code from other sources.

Modifying the code

With the GLSL Shader now in place we can perform a few modifications to the code to reach our goal. Open the GLSL Shader actor code editor. This is the same GLSL Shader actor where the 'Two Point Bezier Shape' text has been pasted in from the Isadora plugin page.

Look at the left hand side of the code editor. There are numbers running down that indicate individual lines of code in a numeric sequence. There are only a few lines of code that we will be modifying to make the expected changes to the shader.

At line 00017 and 00018 we are adding additional 'special comment lines' that Isadora recognises as inputs when it compiles the GLSL module. These are called

'uniform variable' and allow a stream of information from the CPU to interact in real-time with the shader.

// ISADORA_FLOAT_PARAM(spreadL, cp5, 0.0, 0.2, 0.00025, "line spread");

```
00011
00012
00013 // ISADORA_FLOAT_PARAM(Control1X, cp1, 0.0, 1.0, 0.25, "Control Point 1X");
00014 // ISADORA_FLOAT_PARAM(Control1Y, cp2, 0.0, 1.0, 0.5, "Control Point 1Y");
00015 // ISADORA_FLOAT_PARAM(Control2X, cp3, 0.0, 1.0, 0.75, "Control Point 2X");
00016 // ISADORA_FLOAT_PARAM(Control2Y, cp4, 0.0, 1.0, 0.5, "Control Point 2Y");
00017 // ISADORA_FLOAT_PARAM(spreadL, cp5, 0.0, 0.2, 0.00025, "line spread");
00018 // ISADORA_FLOAT_PARAM(vertS, cp6, 0.0, 1.0, 0.00025, "vertical shift");
00019
00020 uniform float Control1X;
00021 uniform float Control1Y;
00022 uniform float Control2X;
00023 uniform float Control2Y;
00024 uniform float spreadL; // spreads the line to a gradient
00025 uniform float vertS; // shifts the curve on the vertical axis
00026
```

This screen capture indicates the additional code required at line 17 & 18, 24 & 25 within the GLSL editor to modify the Two Point Bezier Shape code.

For an explanation of this interaction and the Isadora special comment line see the official Isadora GLSL Shader Actor Tutorial.

The uniform variables are declared at line 00024 and 00025:

uniform float spreadL;
uniform float vertS;

The code at line 00170 is modified to accept the uniform variables that have been declared at line 0024 and 0025.

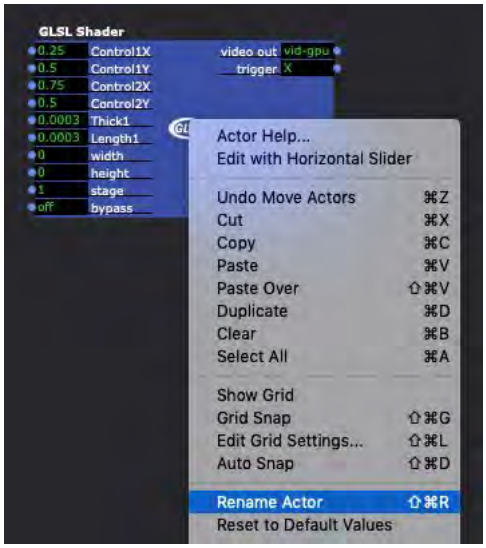
vec3 color = vec3(smoothstep(l, st.y+(vertS),l+px+(spreadL)));

```
00168
00169     //RM: changing the order here shifts the figure and ground output & (vertS) shifts the curve on
      the vertical axis, (spreadL) spreads the line to a gradient
00170     vec3 color = vec3(smoothstep(l, st.y+(vertS),l+px+(spreadL)));
00171
```

This screen capture indicates the additional code required at line 170 within the GLSL editor to modify the Two Point Bezier Shape code.

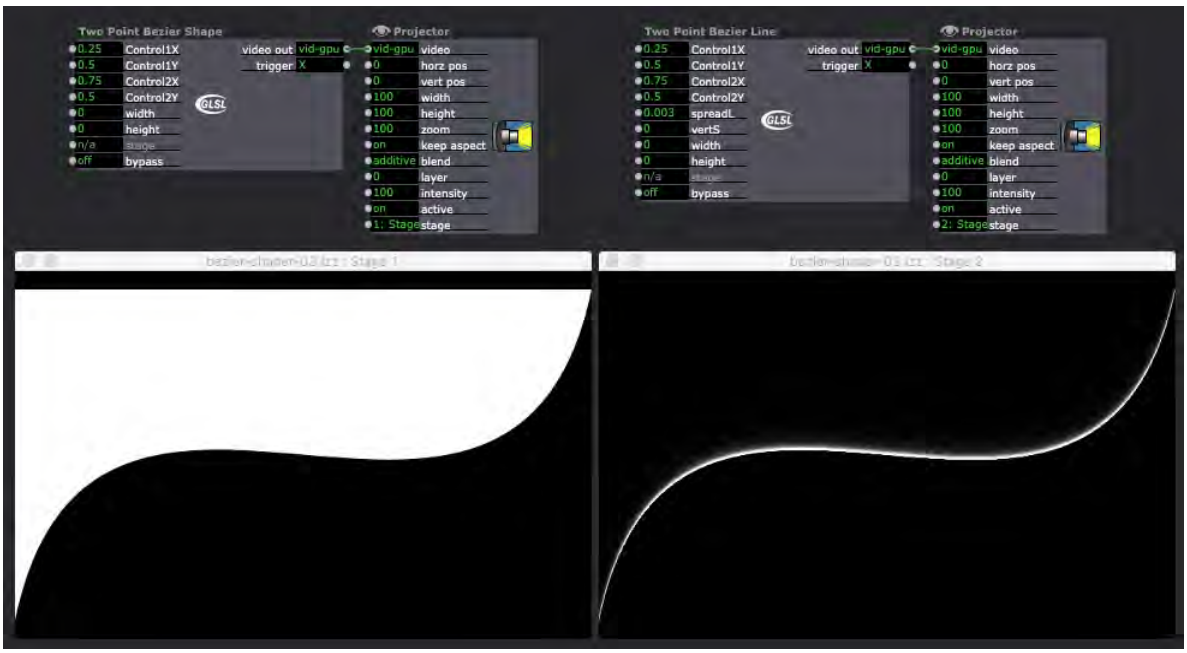
Ensure that you have modified the code to be exactly as it appears above. The re-arrangement of the elements in the following segment is responsible for generating the curve with the appearance of a line: (l, st.y+(vertS),l+px+(spreadL)).

To clean this up let's rename the GLSL Shader to something other than its default name. You are able to save the code by cutting and pasting it into a text document then saving the text document to the Isadora GLSL plugin folder on your computer. To access the correct folder: Isadora / Help menu / Open Plugins Folder / TroikaTronix GLSL Shaders. More instruction about saving your code is available through the official Isadora GLSL Shader Actor Tutorial.



Consider renaming the actor to something other than its default name

What we have achieved so far is the modification of the default plugin ‘Two Point Bezier Shape’. It is now a new instance that you have renamed. There are two new inputs that affect the attributes of the curve.

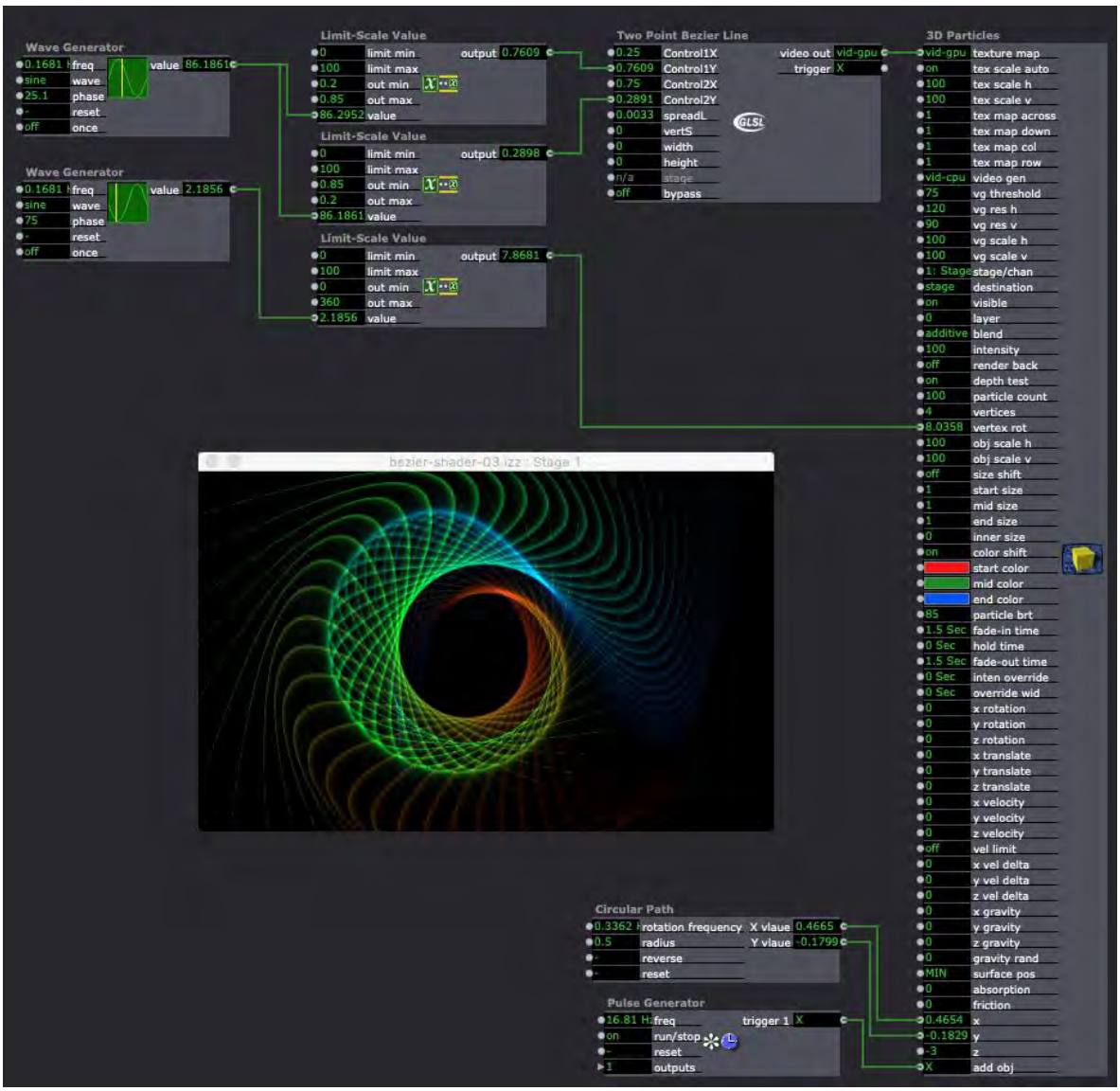


A screen capture indicating the variation achieved through code modification of a default GLSL shader. On the left is the output of the source shader. On the right is the output of the modified shader.

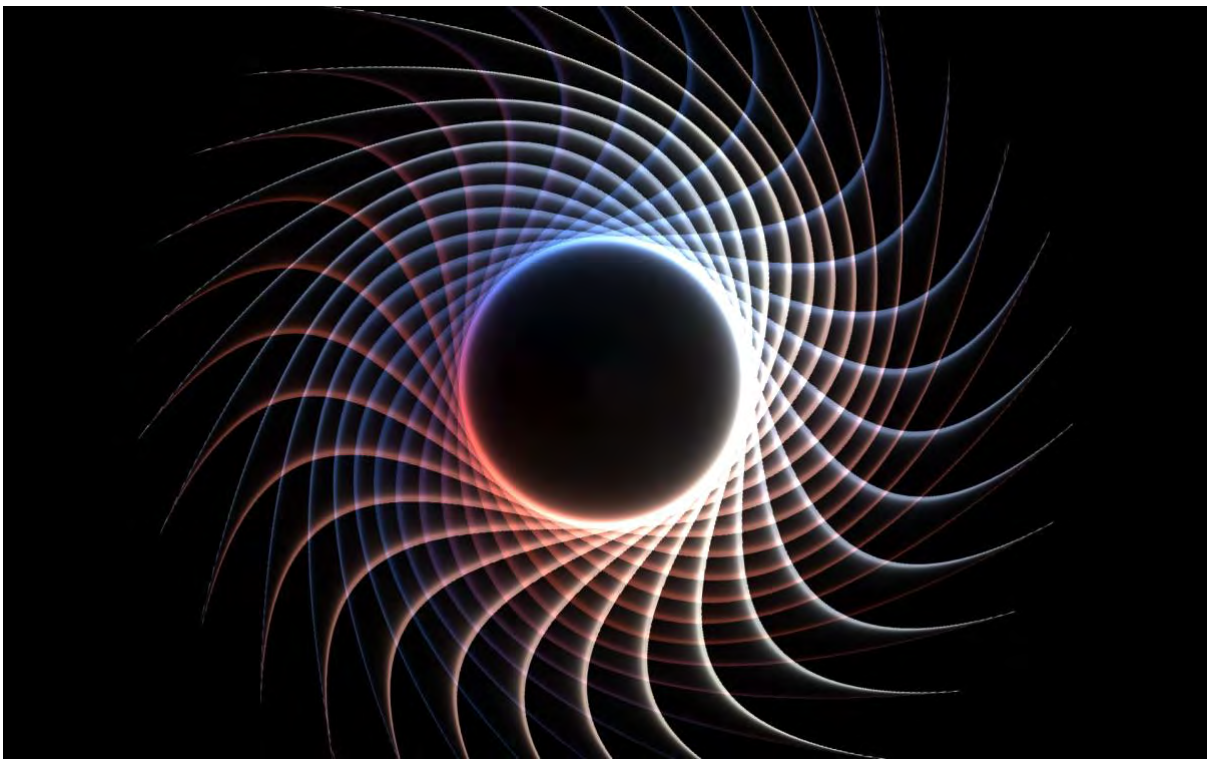
2 Incorporating the modified shader actor into a 3D Particles system

There are a few conditions that are required before we will see anything. Use a patch cable to link the ‘video out’ of the GLSL actor to the ‘texture map’ input of the 3D Particles. Isadora’s ‘3D Particles’ actor takes a 2D image as an input at the ‘texture map’ parameter. You can now add a ‘Pulse Generator’ to the patch and link its output to the ‘add obj’ parameter input at the very bottom of the 3D Particles actor. Also increment the ‘z’ parameter input to negative 3 (-3). Now activate the ‘Stage’.

This 3D particle system is relatively simple considering that we have spent a bit of time preparing the GLSL Shader that is paired with it. In the example represented in the screen grab below, three parameters of movement are implemented using ‘Wave Generator’ actors. The first is through the control point parameters of the GLSL Shader actor. The second is at the 3D Particles actor ‘vertex rot’ input. And the third is a circular movement configured at the 3D Particles ‘x’ and ‘y’ inputs at the bottom of the actor. I have used the ‘Circular Path’ User Actor that is available for download from the Isadora Plugins page: https://troikatronix.com/plugin/circular-path-by-bonemap/. There is infinite variation possible and by tinkering with the parameters you can achieve many types of elegant and beautiful effects.



The full patch is relatively simple, but finding and entering the right balance of numerical properties takes time.



Above and below: An example screen capture of the stage output