Processing Demo

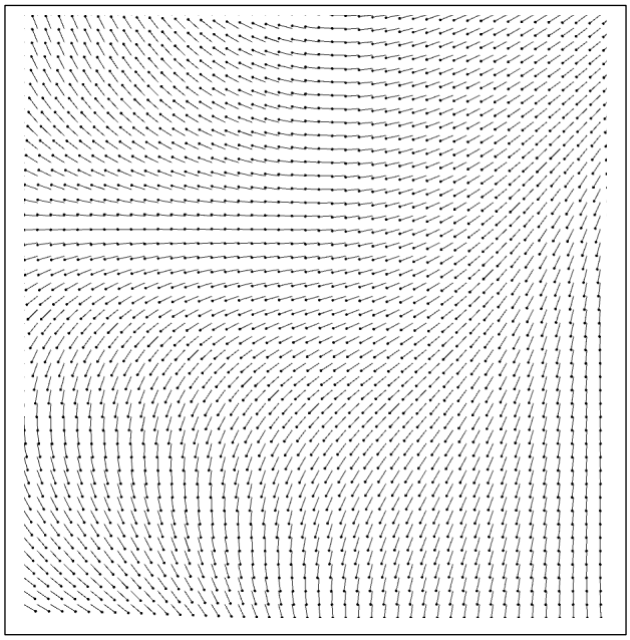
Introduction

Processing is a powerful tool for creating all sorts of interesting visual output. Over the years, the Processing community has developed extensive documentation and written countless algorithms and libraries, which allow programmers to create generative/interactive art, and data visualisations, and with the P5.js library for JavaScript, dynamic web pages as well.

Below are some examples of different techniques for generating interesting visualisations. Read through the notes and closely identify the sections of code we have suggested, try changing some of the values to see how the visualisations change.

Vector Fields

Vector fields are used in mathematics and physics to represent the motion of particles through different media, for example, modelling currents in water.



In the GIF shown here, we can see a vector field made of arrows which move around, pointing in the direction of flow.

In Processing we can use vector fields creatively to generate digital artwork. In the code example below, we see the implementation of a somewhat complex vector field. Run the programme and make changes to the highlighted sections to see how it affects the vector field.

Code EX.1 Vector_Fields

This vector field is generated using two separate equations for the X and Y positions of each of the 20,000 particles being drawn.

The size of the field can be changed by increasing or decreasing the number for the 'scale' variable. Try changing this number 10 to something else.

The more interesting effects come when we make changes to these equations.

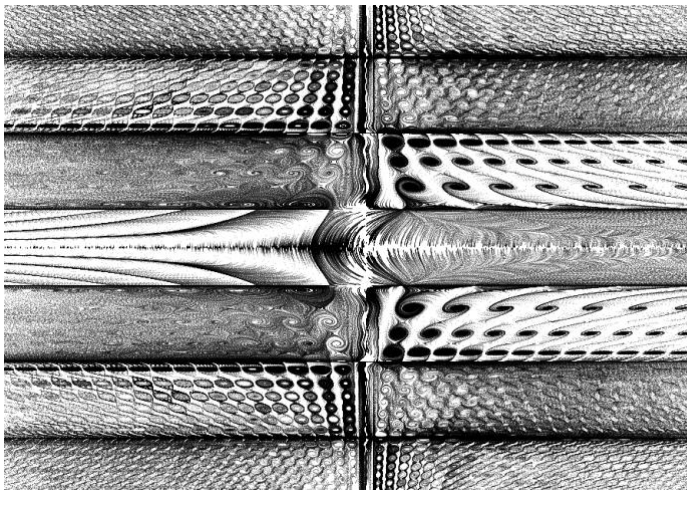
The first image shown is the field generated by the code above.

However, what if we change the first equation to u = cos(x) * sin(y*y) and the second to v = sin(y) * tan(x*y). You can see the resulting second image has a different style.

Experiment by changing these two equations, and try using different mathematical operators, such as addition, subtraction, or division.

Change the sin cos and tan functions around, have some fun and see if you can create something you like. If you get an image you would like to save, simply click on the image and it will save a copy to the project folder.

```
void setup () {
  size(940, 630);
  //fullScreen(P2D, 2);
  background(255);

  for (int i = 0; i < 20000; i++) {
    particles.add(new Particle(random(0, width), ranc
  }
}

void draw() {
  for (Particle ballz : particles) {
    ballz.update();
    ballz.show();
  }
}

PVector vector__field (float x, float y) {
  float scale = 10; //change the resolution of field
  x = map(x, 0, width, -scale, scale);
  y = map(y, 0, height, -scale, scale);

  //make changes to both these equations!
  float u = x * sin(y*y);
  float v =  cos(y) * tan(x*y);

  return new PVector(u, v);
}
```



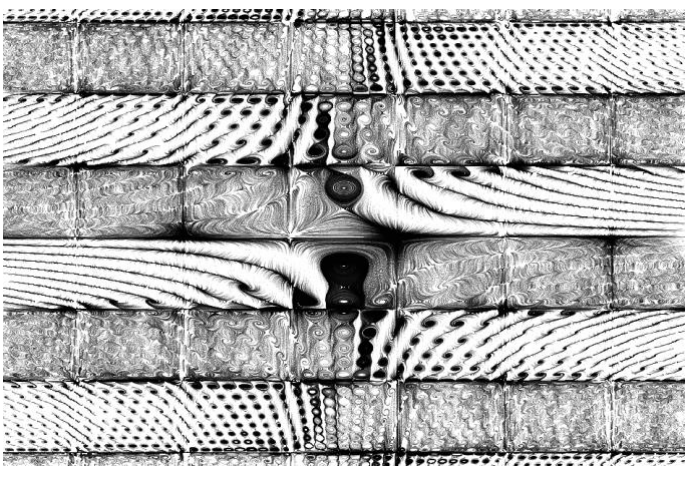


Image Processing

Processing gives us a lot of tools for transforming images and video. Glitch art is a popular technique which uses algorithms to manipulate the pixels of an image to produce a new composition of pixels.

EX2 Image_Slice

In this example, we have some code that lets us load an image stored inside the project folder. The algorithm then divides the image into equally sized columns and slices the image up. The resulting slices are then moved around randomly using something known as perlin noise to create a glitchy watery effect.

Try changing the values for the slice width and the noise scale, what effect do these have?

Now try finding some images you like from the internet, save them to the data folder and then load them into the programme to see what effect the algorithm has on the images you choose.

```
int columns, rows;
int w = 35;
float noiseScale = 0.001;

PImage img;
PImage newImg;
int sx, sy;

void setup() {
  fullScreen(P2D);

  img = loadImage("data/vgogh.png");

  sx = width - img.width;
  sy = height - img.height;
```

To load a file, simply replace the file name with the name of the file you have downloaded in the loadImage() function, making sure to include the file format.