Algorithmic Art Workbook #13 3D glasses and AI

Table of Contents

MIT Licence 4
Workbook #13 Quick Content Summary 5
Module G Unit #1 3D glasses 7
Sketch G1.1 index.html file 8
Sketch G1.2 a box 9
Sketch G1.3 the anaglyph 11
Sketch G1.4 the scene 12
Sketch G1.5 a tunnel 14
Sketch G1.6 pushing it back 16
Sketch G1.7 parallel lines 18
Sketch G1.8 translation 20
Sketch G1.9 rotate 22
Sketch G1.10 moving depth 24
Sketch G1.11 lots of them 26
Sketch G1.12 a continuation 28
Sketch G1.13 adding 3D 30
Using 3D text 32
Sketch G1.14 starting text 33
Sketch G1.15 the font 34
Sketch G1.16 WEBGL 36
Sketch G1.17 the anaglyph 38
Sketch G1.18 Dodecahedron 40
Sketch G1.19 face the pentagon 42
Sketch G1.20 resizing 44
Sketch G1.21 draw the dodecahedron 46
Sketch G1.22 rotate 49
Sketch G1.23 the anaglyph 52
Module G Unit #2 typography 56
Download and Upload font 57
Uploaded font 59
Sketch G2.1 display text 60
Sketch G2.2 breath of life 62
Sketch G2.3 dashed outline 64
Sketch G2.4 what’s the point 66
Sketch G2.5 lines, letters and arrays 68
Sketch G2.6 getting the point 70
Sketch G2.8 3D extruded text 72
Sketch G2.9 geometry 73
Sketch G2.10 let there be lights 75

Sketch G2.11 steering behaviour 77
Sketch G2.12 pt points 78
Sketch G2.13 vehicles 80
Sketch G2.14 steer 82
Sketch G2.15 good behaviour 85
Module H Unit #1 introduction to AI 90
What is ml5.js? 91
Key elements of ml5.js 92
BodyPose 93
HandPose 94
FaceMesh 95
ImageClassifier 96
SoundClassifier 97
ml5 NeuralNetwork 98
Module H Unit #2: colour predictor 101
The index.html 102
Sketch H2.1 creating the data 103
Sketch H2.2 Sliders 104
Sketch H2.3 adding the neural network 106
Sketch H2.4 adding the data 108
Sketch H2.5 normalising the data 110
Sketch H2.6 training the neural network 112
Sketch H2.7 classifying the colour 115
Sketch H2.8 the best prediction 118



MIT Licence

Copyright ©2026 Warren George

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Brief summary:

Permissions (what you can do)

Commercial use
Modification
Distribution
Private use

Limitations (what is not covered)

Liability
Warranty



Workbook #13 Quick Content Summary

This is where you can have some fun with a range of applications. The first unit is making 3D more 3D, and by that I mean with 3D glasses. You will need some for that unit; they are very inexpensive. You can get cardboard or plastic frame ones, but what you do need are the type that have red and blue lenses.

The second unit is about using a new feature available in version 2.x in p5.js, where you can do some interesting stuff with text, both 2D and 3D. For that, you will need to download a font that can be used in this fashion.

The final units are where you play with AI. Not generative AI such as with Large Language Models or Diffusion models, but with pretrained models that you can use in your browser, which pick up features of your hand or face. Most of the pre-trained will be explored in workbook #14, but in this workbook there is an introduction to building your own model.

If you are really interested in this wonderful topic, then I have a series available (on my website) for you.



Algorithmic Art
Module G
Unit #1
3D glasses

Module G Unit #1 3D glasses

For this unit, we will need a pair of 3D glasses. They are very cheap to buy online and usually come in packs of ten (if you need that many). They have a red lens (LEFT) and a blue lens (RIGHT); that is the sort you will need for this unit. The frames can be made of card, but I recommend getting the plastic frames as they are more robust and work well.

Figure 1





Sketch G1.1 index.html file

You may well have gone to see films like Avatar in 3D, where the images leapt out at you. We can do the same with p5.js and 3D shapes. You are not going to create an Avatar-quality scene, but you can at least have fun seeing the 3D shapes come to life. To create a 3D image in the sense of it jumping out of the screen, we can take a little shortcut by using a library that someone has very kindly created to do all the donkey work.

We will need to use a library for this unit called p5.anaglyph, and it needs to go in the index.html file. The script is as follows:

```
<script src="https://cdn.jsdelivr.net/gh/jdeboi/p5.anaglyph@main/
                dist/p5.anaglyph.min.js"></script>
```

You add it the usual way through the `index.html` file.

```
<!DOCTYPE html>
<html lang="en"><head>
  <script src="https://cdn.jsdelivr.net/npm/p5@2.3.2/lib/p5.js"></script>
  <script src="https://cdn.jsdelivr.net/gh/jdeboi/p5.anaglyph@main/dist/
p5.anaglyph.min.js"></script>
    <link rel="stylesheet" type="text/css" href="style.css">
    <meta charset="utf-8">
  </head>
  <body>
    <main>
    </main>
    <script src="sketch.js"></script>
</body></html>
```

Notes

Just add it in the usual way.



Sketch G1.2 a box

Starting sketch, to get things rolling, we create a simple, rotating, 3D box.

```
let angle = 0

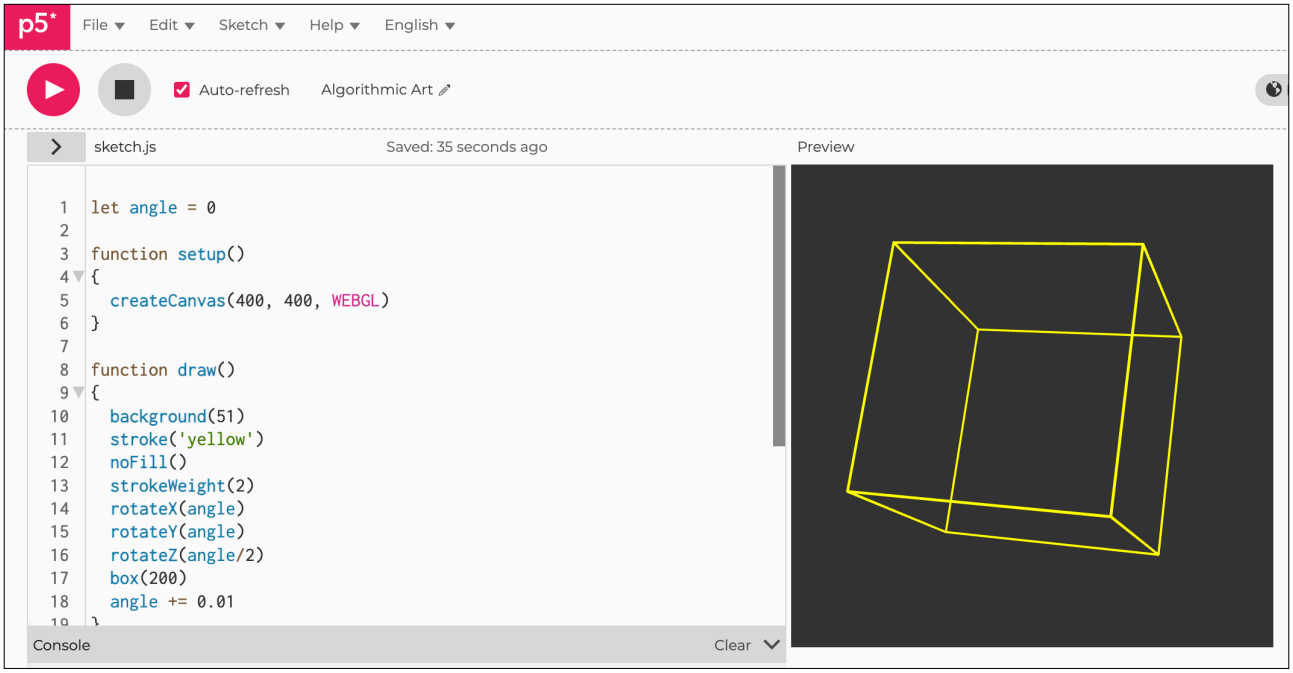
function setup()
{
  createCanvas(400, 400, WEBGL)
}

function draw()
{
  background(51)
  stroke('yellow')
  noFill()
  strokeWeight(2)
  rotateX(angle)
  rotateY(angle)
  rotateZ(angle)
  box(200)
  angle += 0.01
}
```

Notes

We have a lovely rotating box. Not properly 3D enough yet.

Figure G1.2


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 35 seconds ago
Preview
let angle = 0
function setup()
{
createCanvas(400, 400, WEBGL)
}
function draw()
{
background(51)
stroke('yellow')
noFill()
strokeWeight(2)
rotateX(angle)
rotateY(angle)
rotateZ(angle/2)
box(200)
angle += 0.01
Console
Clear



Sketch G1.3 the anaglyph

Adding the Anaglyph library to the sketch.

```
let angle = 0
let anaglyph

function setup()
{
  createCanvas(400, 400, WEBGL)
  anaglyph = createAnaglyph(this)
}

function draw()
{
  background(51)
  stroke('yellow')
  noFill()
  strokeWeight(2)
  rotateX(angle)
  rotateY(angle)
  rotateZ(angle)
  box(200)
  angle += 0.01
}
```

Notes

Nothing changes yet.

Code Explanation

`anaglyph = createAnaglyph(this)`	Part of the library that allows us to use an object



Sketch G1.4 the scene

We need to create a separate function that will take in all the values we have currently. This will be what we call our scene. The scene will hold the 3D effect for when we use the 3D glasses.

We need to create another function to handle the new scene, which will inherit all the features of the anaglyph library. We cannot do this in draw alone. Here we need to identify the new scene as we draw the cube in its new guise. We can use any name (within reason), but I plumbed for `graphic`.

! Notice we have transferred nearly all the code from `draw()` into this new function. Now it is time to put on your 3D glasses.

```
let angle = 0
let anaglyph

function setup()
{
  createCanvas(400, 400, WEBGL)
  anaglyph = createAnaglyph(this)
}

function draw()
{
  anaglyph.draw(scene)
  angle += 0.01
}

function scene(graphic)
{
  graphic.background(51)
  graphic.stroke('yellow')
  graphic.noFill()
  graphic.strokeWeight(2)
  graphic.rotateX(angle)
  graphic.rotateY(angle)
  graphic.rotateZ(angle)
  graphic.box(200)
}
```

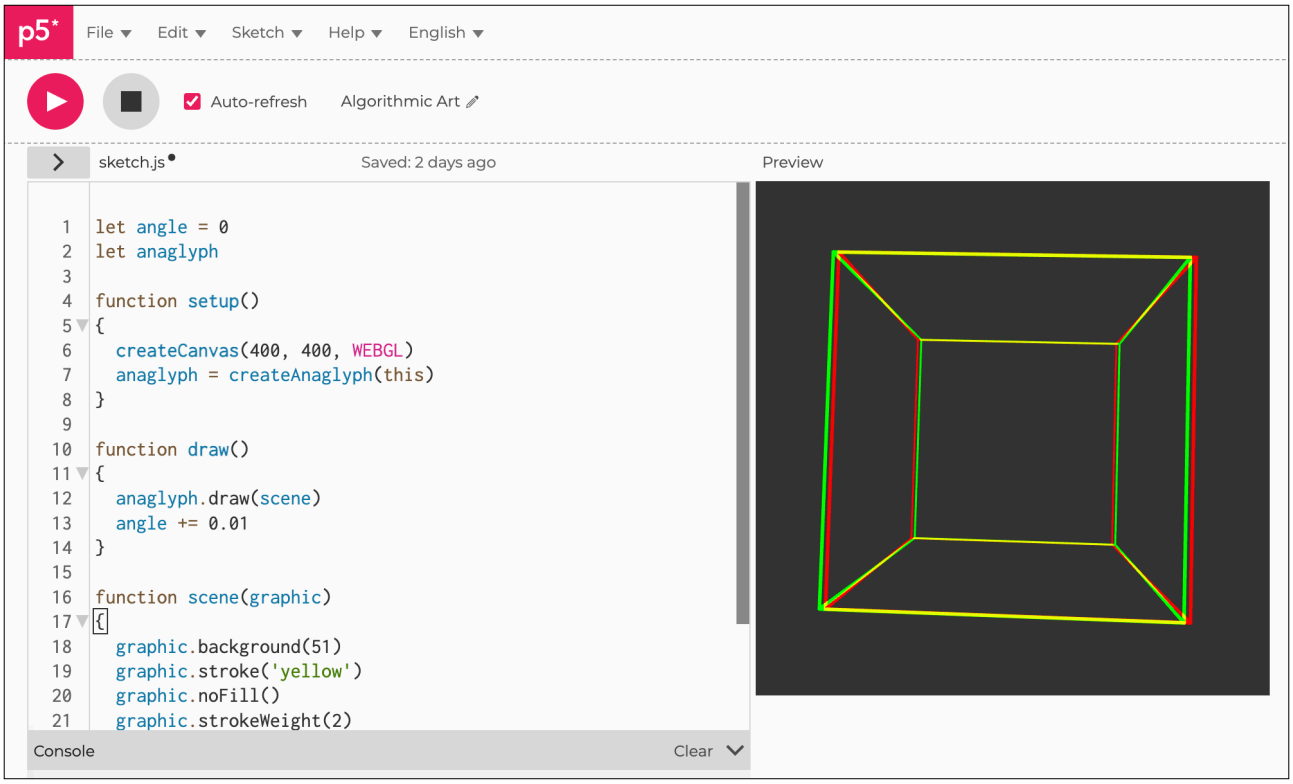
Notes

As you don your glasses, you will see the cube spring into life. You could include the angle being incremented in the `scene()` function, but it seems to spin twice as fast, so I have left it in the `draw()` function.

Challenges

1. Try other shapes.
2. How would you have more than one box?

Figure G1.4



Sketch G1.5 a tunnel

! This is our starting sketch.

We are going to create a tunnel using the anaglyph effect. This will use 2D shapes to create a 3D effect, but all will be revealed. We want a square with the dimension as a variable, `boxWidth`. This allows us to play with different values, but this is key anyway as we will be moving forwards over time.

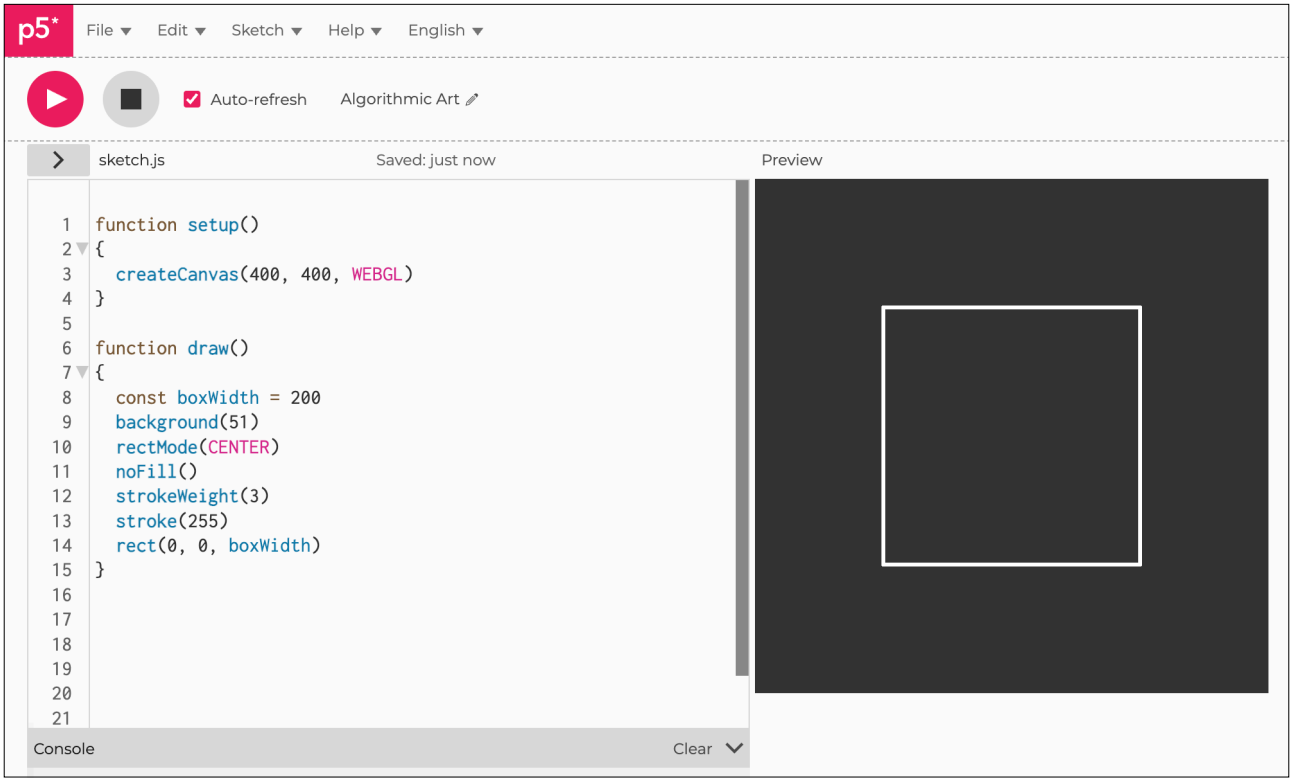
```
function setup()
{
  createCanvas(400, 400, WEBGL)
}

function draw()
{
  const boxWidth = 200
  background(51)
  rectMode(CENTER)
  noFill()
  strokeWeight(3)
  stroke(255)
  rect(0, 0, boxWidth)
}
```

Notes

Using white for the lines just has a better effect later on. The datatype `const` means it cannot be changed, it is fixed, a sort of non-variable variable.

Figure G1.5


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: just now
Preview
function setup()
{
createCanvas(400, 400, WEBGL)
}
function draw()
{
const boxWidth = 200
background(51)
rectMode(CENTER)
noFill()
strokeWeight(3)
stroke(255)
rect(0, 0, boxWidth)
}
Console
Clear



Sketch G1.6 pushing it back

We want to push the square further back. We translate the square using all three elements: the x, y, and z components. This makes the square seem smaller.

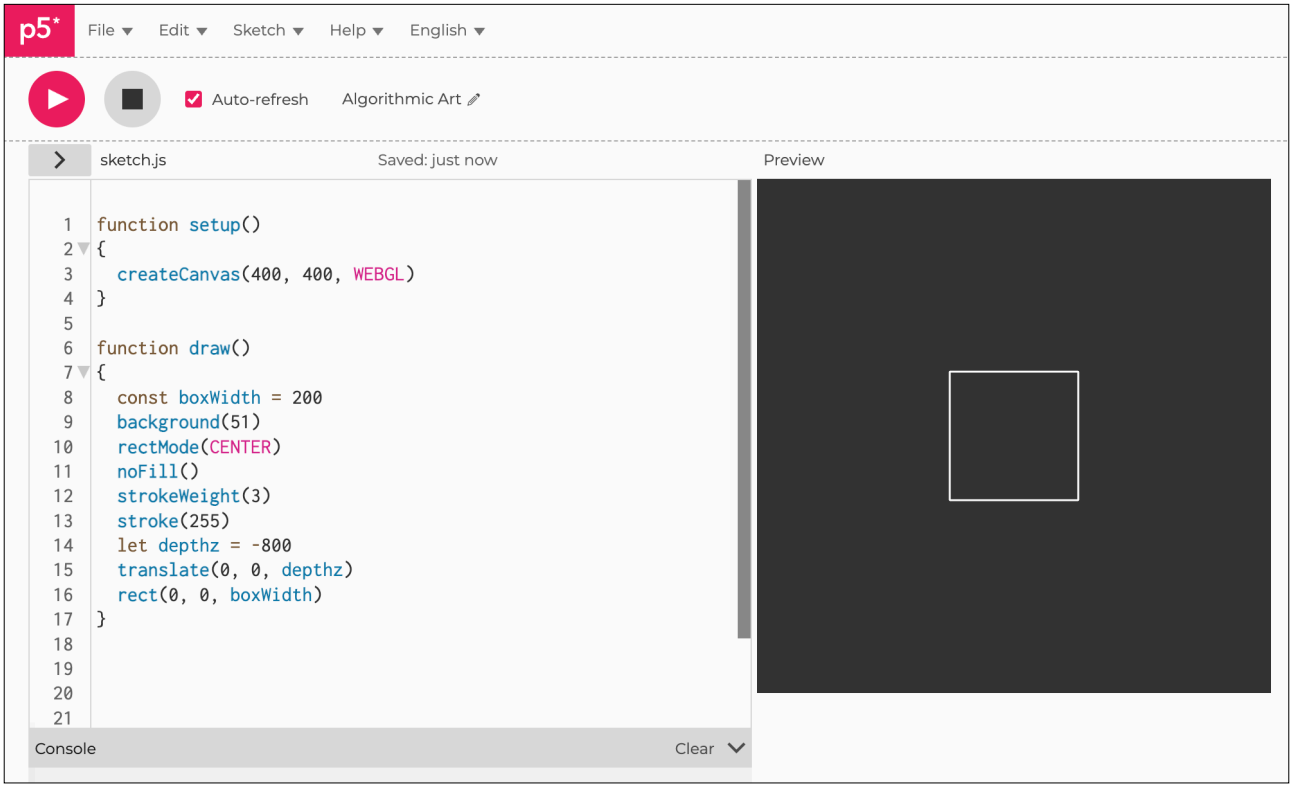
```
function setup()
{
  createCanvas(400, 400, WEBGL)
}

function draw()
{
  const boxWidth = 200
  background(51)
  rectMode(CENTER)
  noFill()
  strokeWeight(3)
  stroke(255)
  let depthz = -800
  translate(0, 0, depthz)
  rect(0, 0, boxWidth)
}
```

Notes

The same-size square just further away in the z direction, hence the variable depthz.

Figure G1.6


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: just now
Preview
function setup()
{
createCanvas(400, 400, WEBGL)
}
function draw()
{
const boxWidth = 200
background(51)
rectMode(CENTER)
noFill()
strokeWeight(3)
stroke(255)
let depthz = -800
translate(0, 0, depthz)
rect(0, 0, boxWidth)
}
Console
Clear



Sketch G1.7 parallel lines

We will be moving the square forwards shortly, but for the moment, we want to create the effect of a cube. So, we want the parallel lines that the square will follow. These are two squares. Think of them as where the walls meet the ceiling.

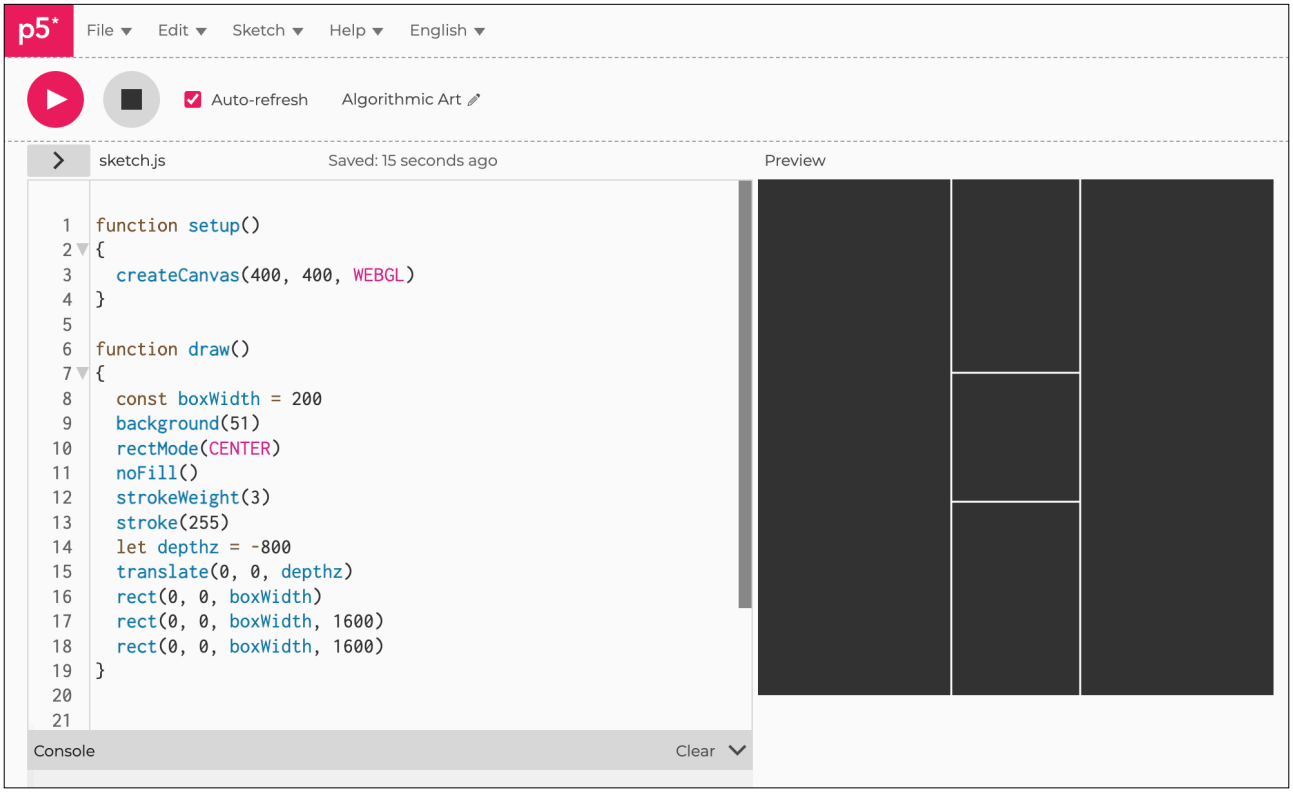
```
function setup()
{
  createCanvas(400, 400, WEBGL)
}

function draw()
{
  const boxWidth = 200
  background(51)
  rectMode(CENTER)
  noFill()
  strokeWeight(3)
  stroke(255)
  let depthz = -800
  translate(0, 0, depthz)
  rect(0, 0, boxWidth)
  rect(0, 0, boxWidth, 1600)
  rect(0, 0, boxWidth, 1600)
}
```

Notes

This looks very uninspiring. All will become clear, honestly.

Figure G1.7


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 15 seconds ago
Preview
function setup()
{
createCanvas(400, 400, WEBGL)
}
function draw()
{
const boxWidth = 200
background(51)
rectMode(CENTER)
noFill()
strokeWeight(3)
stroke(255)
let depthz = -800
translate(0, 0, depthz)
rect(0, 0, boxWidth)
rect(0, 0, boxWidth, 1600)
rect(0, 0, boxWidth, 1600)
}
Console
Clear



Sketch G1.8 translation

We will translate these last two rectangles. We will also add in some `push()` and `pop()` otherwise all the translations will be compounded. We want to keep them separate for when we move the square.

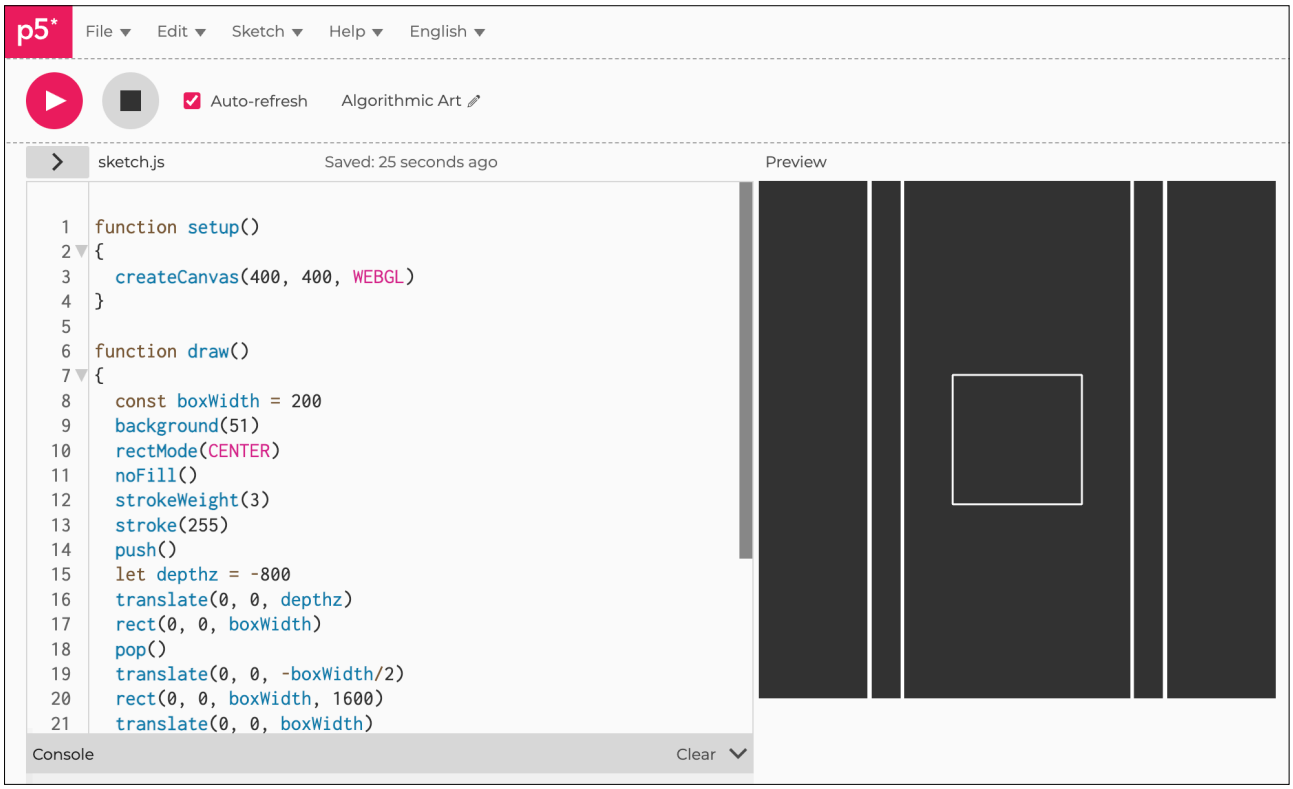
```
function setup()
{
  createCanvas(400, 400, WEBGL)
}

function draw()
{
  const boxWidth = 200
  background(51)
  rectMode(CENTER)
  noFill()
  strokeWeight(3)
  stroke(255)
  push()
  let depthz = -800
  translate(0, 0, depthz)
  rect(0, 0, boxWidth)
  pop()
  translate(0, 0, -boxWidth/2)
  rect(0, 0, boxWidth, 1600)
  translate(0, 0, boxWidth)
  rect(0, 0, boxWidth, 1600)
}
```

Notes

Still looks unpromising!

Figure G1.8


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 25 seconds ago
Preview
function setup()
{
createCanvas(400, 400, WEBGL)
}
function draw()
{
const boxWidth = 200
background(51)
rectMode(CENTER)
noFill()
strokeWeight(3)
stroke(255)
push()
let depthz = -800
translate(0, 0, depthz)
rect(0, 0, boxWidth)
pop()
translate(0, 0, -boxWidth/2)
rect(0, 0, boxWidth, 1600)
translate(0, 0, boxWidth)
Console
Clear



Sketch G1.9 rotate

Now see what happens when we rotate the two rectangles.

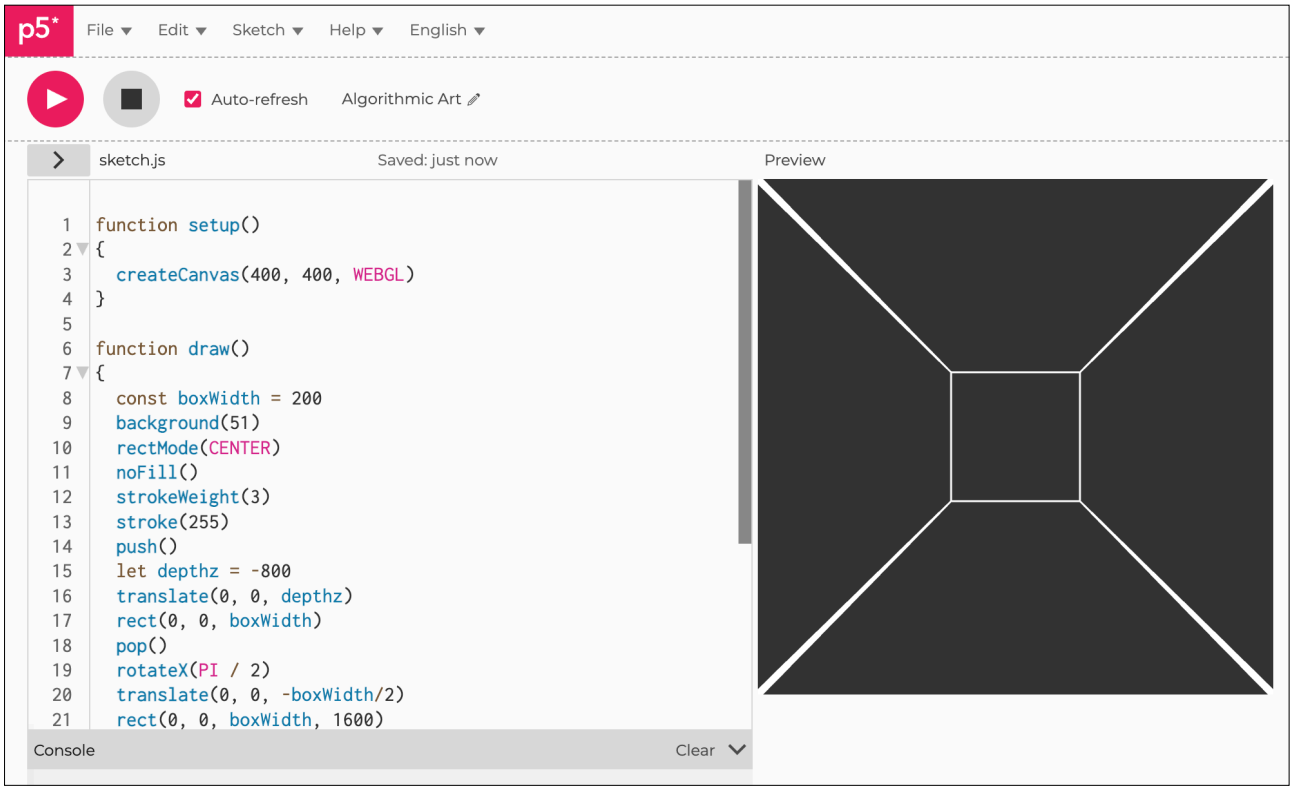
```
function setup()
{
  createCanvas(400, 400, WEBGL)
}

function draw()
{
  const boxWidth = 200
  background(51)
  rectMode(CENTER)
  noFill()
  strokeWeight(3)
  stroke(255)
  push()
  let depthz = -800
  translate(0, 0, depthz)
  rect(0, 0, boxWidth)
  pop()
  rotateX(PI / 2)
  translate(0, 0, -boxWidth/2)
  rect(0, 0, boxWidth, 1600)
  translate(0, 0, boxWidth)
  rect(0, 0, boxWidth, 1600)
}
```

Notes

It now looks like the square has edges radiating out from it towards you. PI is the value of 180° in degrees. It starts to make sense (at least to me it does).

Figure G1.9


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: just now
Preview
function setup()
{
createCanvas(400, 400, WEBGL)
}
function draw()
{
const boxWidth = 200
background(51)
rectMode(CENTER)
noFill()
strokeWeight(3)
stroke(255)
push()
let depthz = -800
translate(0, 0, depthz)
rect(0, 0, boxWidth)
pop()
rotateX(PI / 2)
translate(0, 0, -boxWidth/2)
rect(0, 0, boxWidth, 1600)
Console
Clear



Sketch G1.10 moving depth

This bit can get a little complicated. We want to move the square forwards. We have a count to help advance the square incrementally.

```
let count = 0

function setup()
{
  createCanvas(400, 400, WEBGL)
}

function draw()
{
  const boxWidth = 200
  background(51)
  rectMode(CENTER)
  noFill()
  strokeWeight(3)
  stroke(255)
  push()
  let depthz = count - 800
  translate(0, 0, depthz)
  rect(0, 0, boxWidth)
  count += 2
  pop()
  rotateX(PI / 2)
  translate(0, 0, -boxWidth/2)
  rect(0, 0, boxWidth, 1600)
  translate(0, 0, boxWidth)
  rect(0, 0, boxWidth, 1600)
}
```

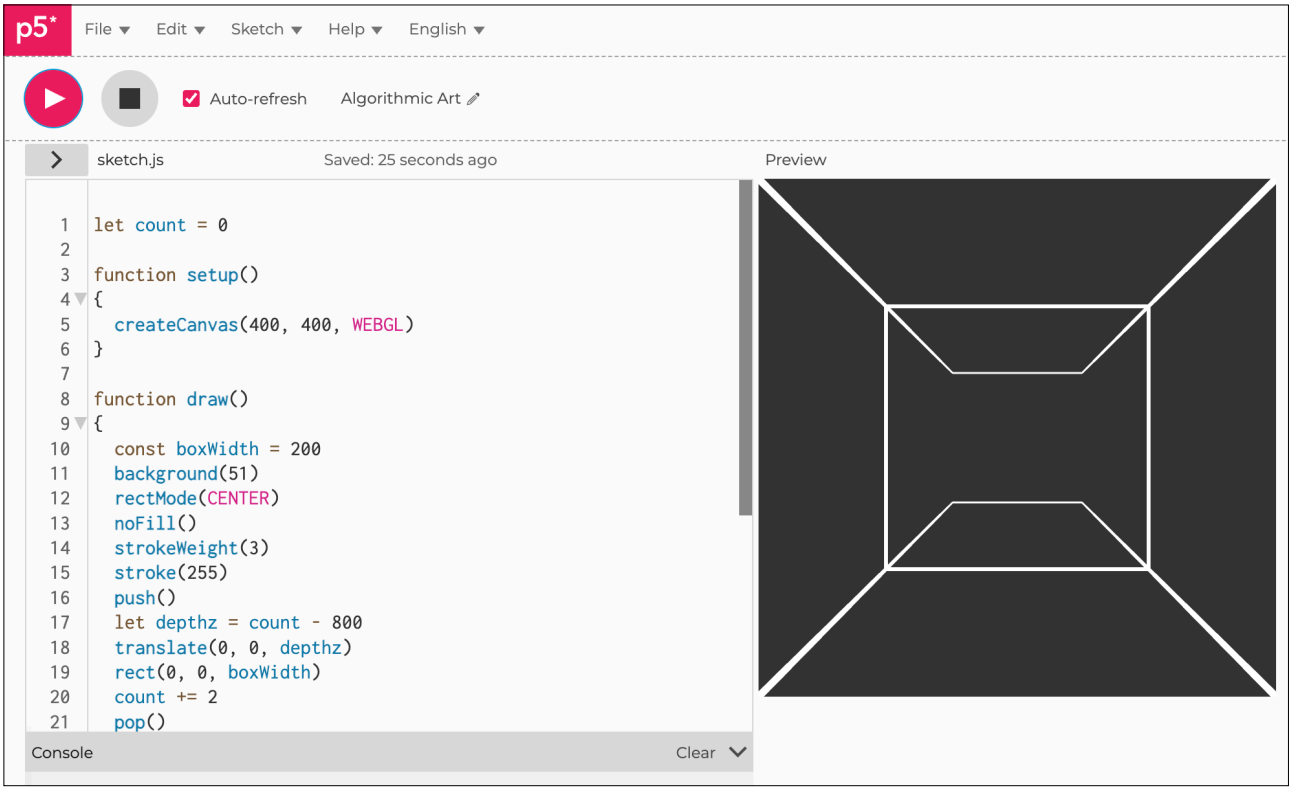
Notes

You should see the square slowly moving towards you. We start with the count on zero and the depth at -800, and the count increasing so that the depth eventually becomes zero.

Challenge

Try some other values (other than 800) to see what is going on.

Figure G1.10


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 25 seconds ago
Preview
let count = 0
function setup()
{
createCanvas(400, 400, WEBGL)
}
function draw()
{
const boxWidth = 200
background(51)
rectMode(CENTER)
noFill()
strokeWeight(3)
stroke(255)
push()
let depthz = count - 800
translate(0, 0, depthz)
rect(0, 0, boxWidth)
count += 2
pop()
Console
Clear



Sketch G1.11 lots of them

We want lots of squares all moving together, spaced out in intervals. We will create 30 squares. We will slow them down a little bit by changing the count. The distance between the squares is 100, so each square is drawn 100 from the next one and then moves them forward at a slow speed to 0.05.

```
let count = 0

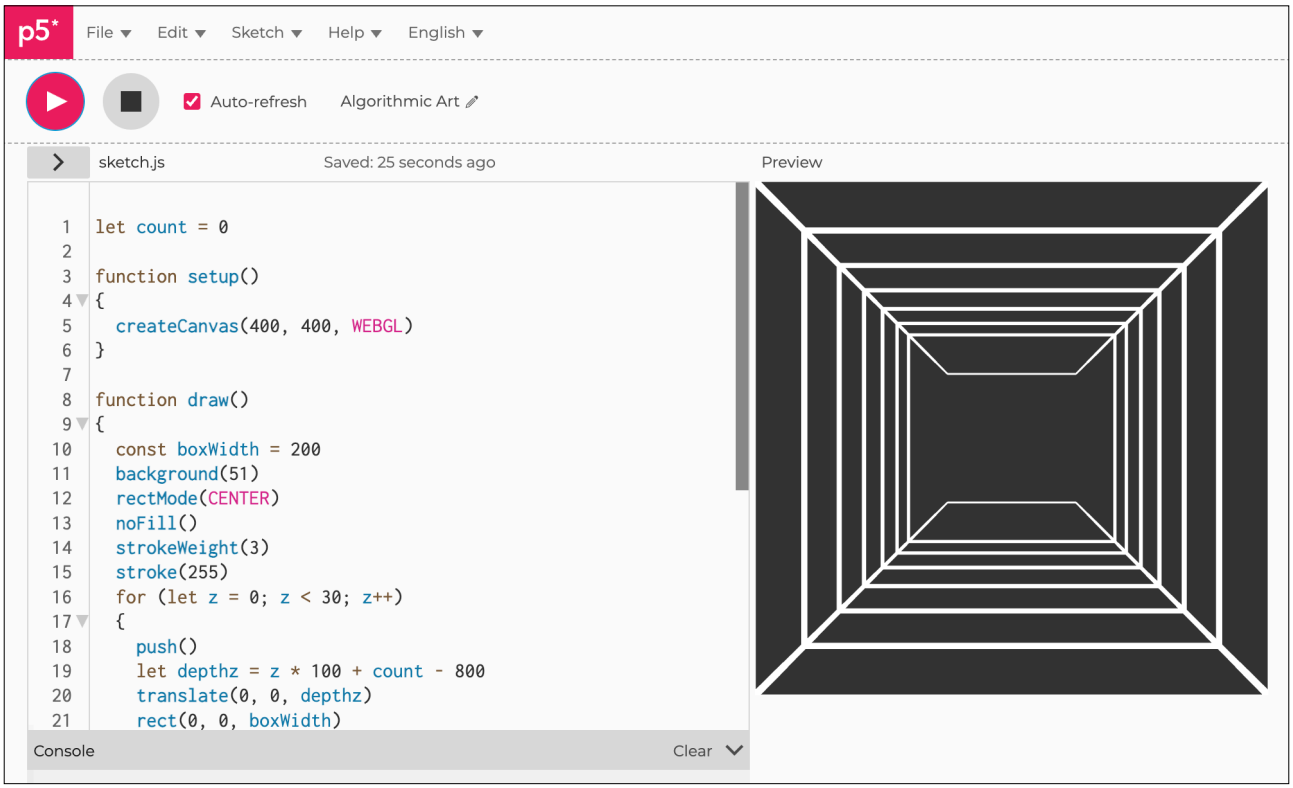
function setup()
{
  createCanvas(400, 400, WEBGL)
}

function draw()
{
  const boxWidth = 200
  background(51)
  rectMode(CENTER)
  noFill()
  strokeWeight(3)
  stroke(255)
  for (let z = 0; z < 30; z++)
  {
    push()
    let depthz = z * 100 + count - 800
    translate(0, 0, depthz)
    rect(0, 0, boxWidth)
    count += 0.05
    pop()
  }
  rotateX(PI / 2)
  translate(0, 0, -boxWidth/2)
  rect(0, 0, boxWidth, 1600)
  translate(0, 0, boxWidth)
  rect(0, 0, boxWidth, 1600)
}
```

Notes

This is great except for one tiny detail: we eventually run out of squares. We need to find a way to keep drawing them continuously.

Figure G1.11



Sketch G1.12 a continuation

The next step is to redraw them. To do that, we need to reset the count back to zero once a certain number is reached. In this case, we will pick a number such as 2000 (because I know it works). Then subtract 800 to push it back to the right place.

```
let count = 0

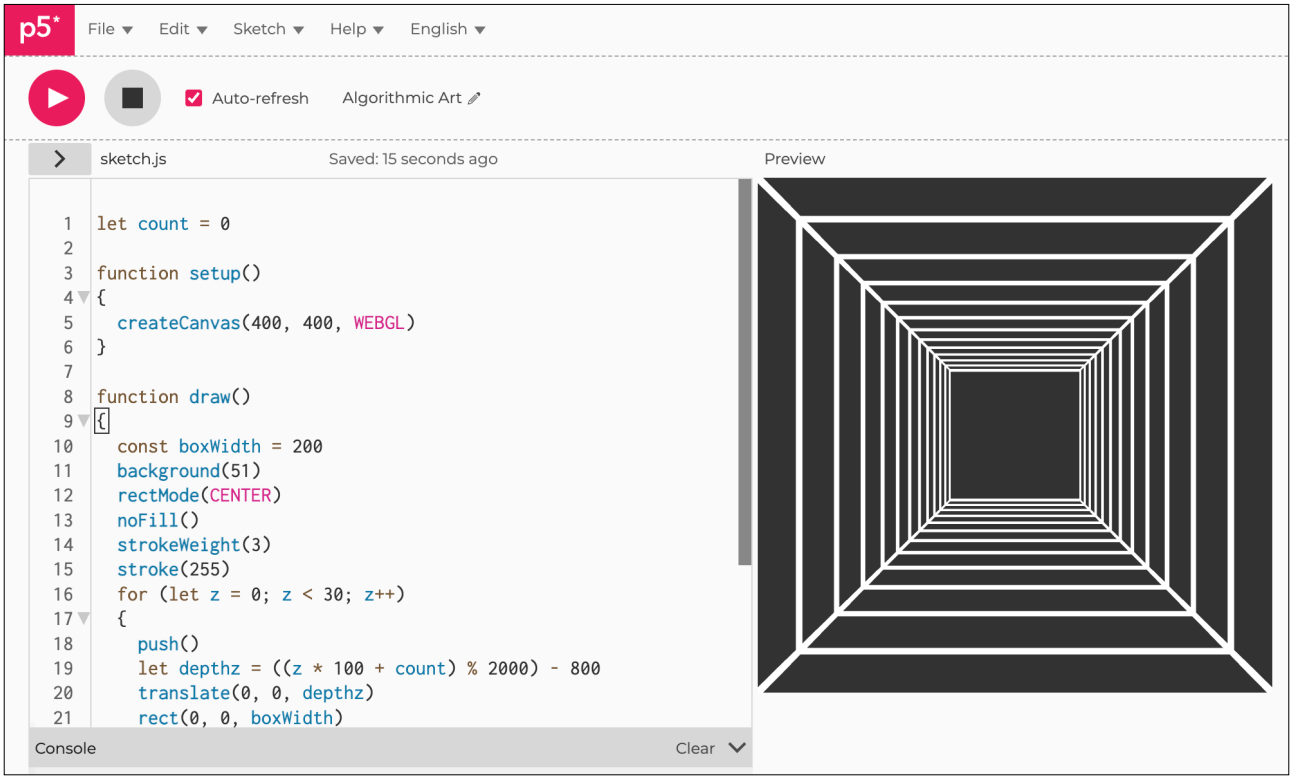
function setup()
{
  createCanvas(400, 400, WEBGL)
}

function draw()
{
  const boxWidth = 200
  background(51)
  rectMode(CENTER)
  noFill()
  strokeWeight(3)
  stroke(255)
  for (let z = 0; z < 30; z++)
  {
    push()
    let depthz = ((z * 100 + count) % 2000) - 800
    translate(0, 0, depthz)
    rect(0, 0, boxWidth)
    count += 0.05
    pop()
  }
  rotateX(PI / 2)
  translate(0, 0, -boxWidth/2)
  rect(0, 0, boxWidth, 1600)
  translate(0, 0, boxWidth)
  rect(0, 0, boxWidth, 1600)
}
```

Notes

You now have a continuous tunnel. But it is not 3D yet.

Figure G1.12


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 15 seconds ago
Preview
let count = 0
function setup()
{
createCanvas(400, 400, WEBGL)
}
function draw()
{
const boxWidth = 200
background(51)
rectMode(CENTER)
noFill()
strokeWeight(3)
stroke(255)
for (let z = 0; z < 30; z++)
{
push()
let depthz = ((z * 100 + count) % 2000) - 800
translate(0, 0, depthz)
rect(0, 0, boxWidth)
Console
Clear

Sketch G1.13 adding 3D

We introduce the anaglyph bits and pieces.

! What I suggest is copying the entire `draw()` function, create the `scene()` function and pasting it all into the `scene()` function. Then you can start to add in the `graphics` prefix where necessary.

```
let count = 0
let anaglyph

function setup()
{
  createCanvas(400, 400, WEBGL)
  anaglyph = createAnaglyph(this)
}

function draw()
{
  anaglyph.draw(scene)
}

function scene(graphics)
{
  const boxWidth = 200
  graphics.background(51)
  graphics.rectMode(CENTER)
  graphics.noFill()
  graphics.strokeWeight(3)
  graphics.stroke(255)
  for (let z = 0; z < 30; z++)
  {
    graphics.push()
    let depthz = ((z * 100 + count) % 2000) - 800
    graphics.translate(0, 0, depthz)
    graphics.rect(0, 0, boxWidth)
    count += 0.05
    graphics.pop()
  }
  graphics.rotateX(PI / 2)
```

```
  graphics.translate(0, 0, -boxWidth/2)
  graphics.rect(0, 0, boxWidth, 1600)
  graphics.translate(0, 0, boxWidth)
  graphics.rect(0, 0, boxWidth, 1600)
}
```

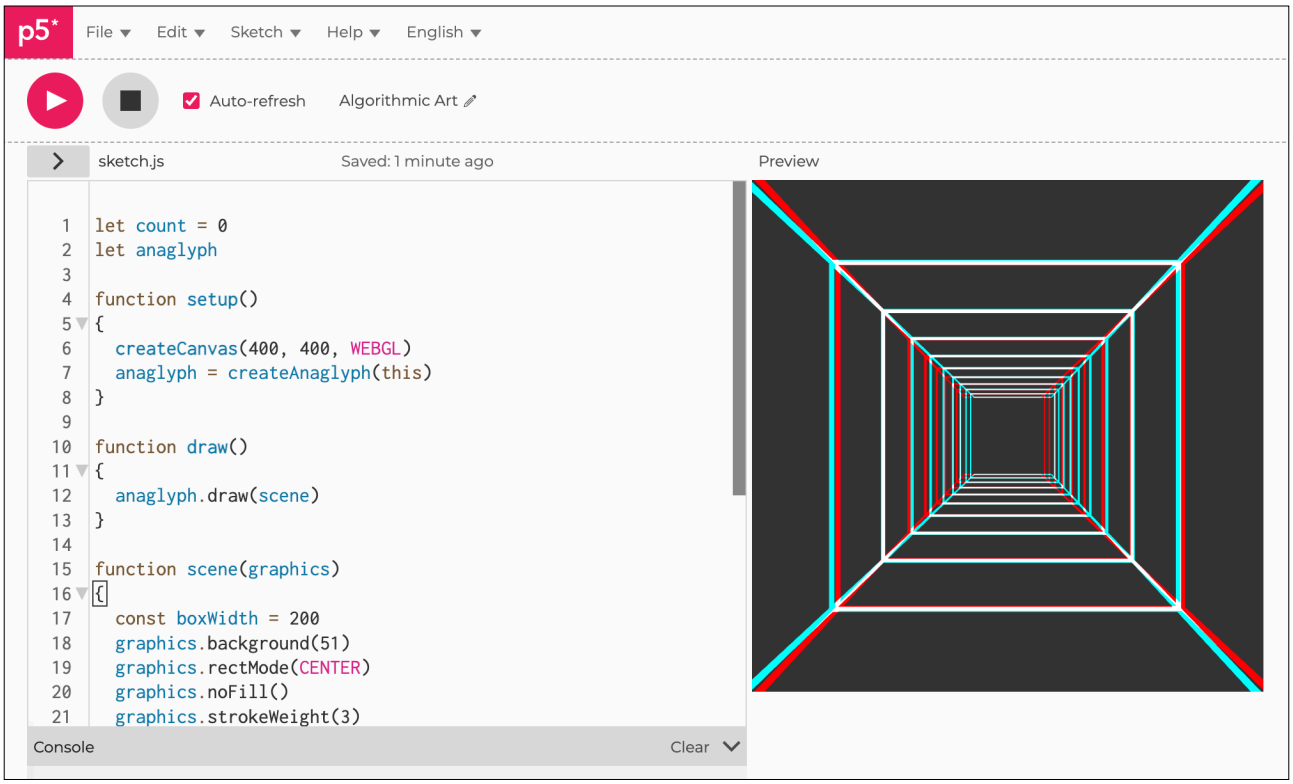
Notes

This is just an example to play with. There are lots of elements to this which you could transfer onto some other project. Using white lines, `stroke(255)` for better effect.

Challenge

Could you do the same thing with a triangle?

Figure G1.13



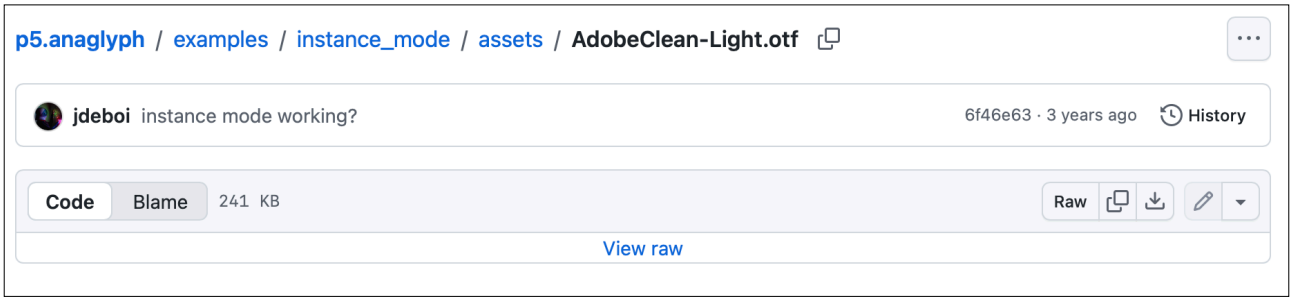
Using 3D text

We need to use a special kind of text file. We can't use just ordinary text with WEBGL. Go to the link below (I should have the link available on the website page). Download it and check where it is downloaded to:

https://github.com/jdeboi/p5.anaglyph/blob/main/examples/instance_mode/assets/AdobeClean-Light.otf

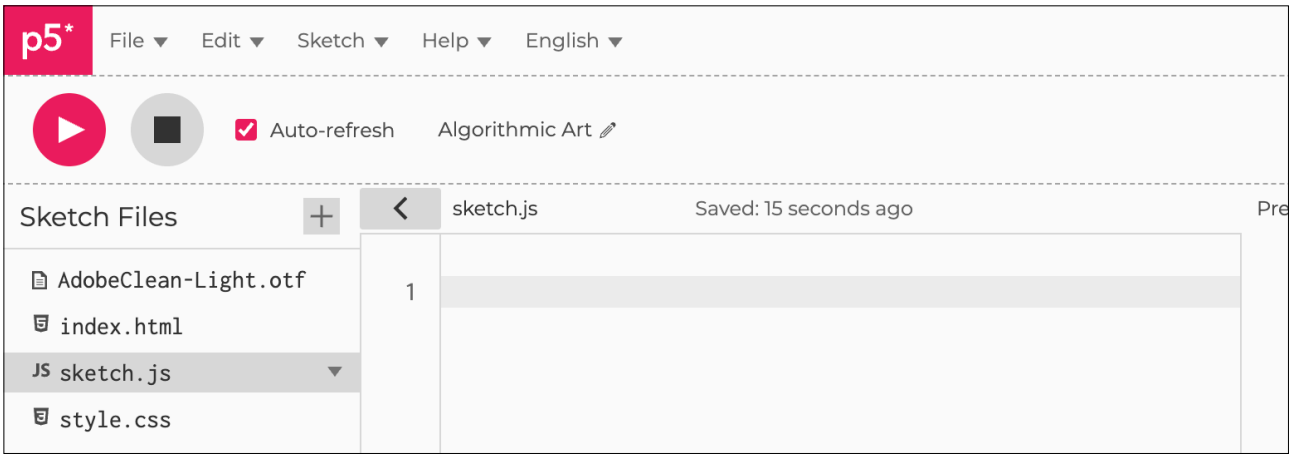
Download the file by clicking on the download button on the far right-hand side.

Figure 2: font download on GitHub



Upload the text file and add it to the list of files in the usual way. You can change the name if you wish but keep the `.otf` suffix.

Figure 3: uploaded to the sketch



Sketch G1.14 starting text

! This is our starting sketch.

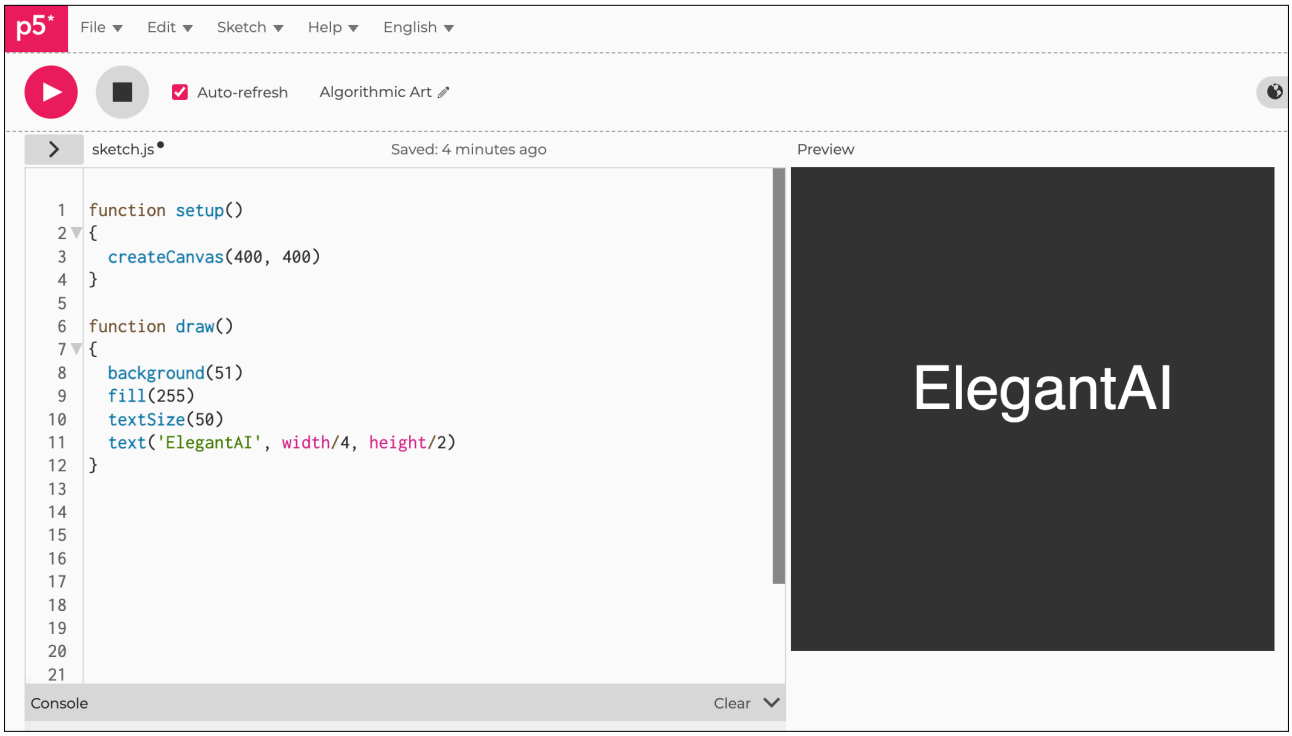
```
function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(51)
  fill(255)
  textSize(50)
  text('ElegantAI', width/4, height/2)
}
```

Notes

This gives us a bog-standard text on canvas.

Figure G1.14





Sketch G1.15 the font

We can change the font. To upload it before we start using it, we need to use asynchronous and await. Make sure the name in the `loadFont()` function matches the exact name of the font file.

```
let font

async function setup()
{
  createCanvas(400, 400)
  font = await loadFont('AdobeClean-Light.otf')
}

function draw()
{
  background(51)
  fill(255)
  textSize(50)
  textFont(font)
  text('ElegantAI', width/4, height/2)
}
```

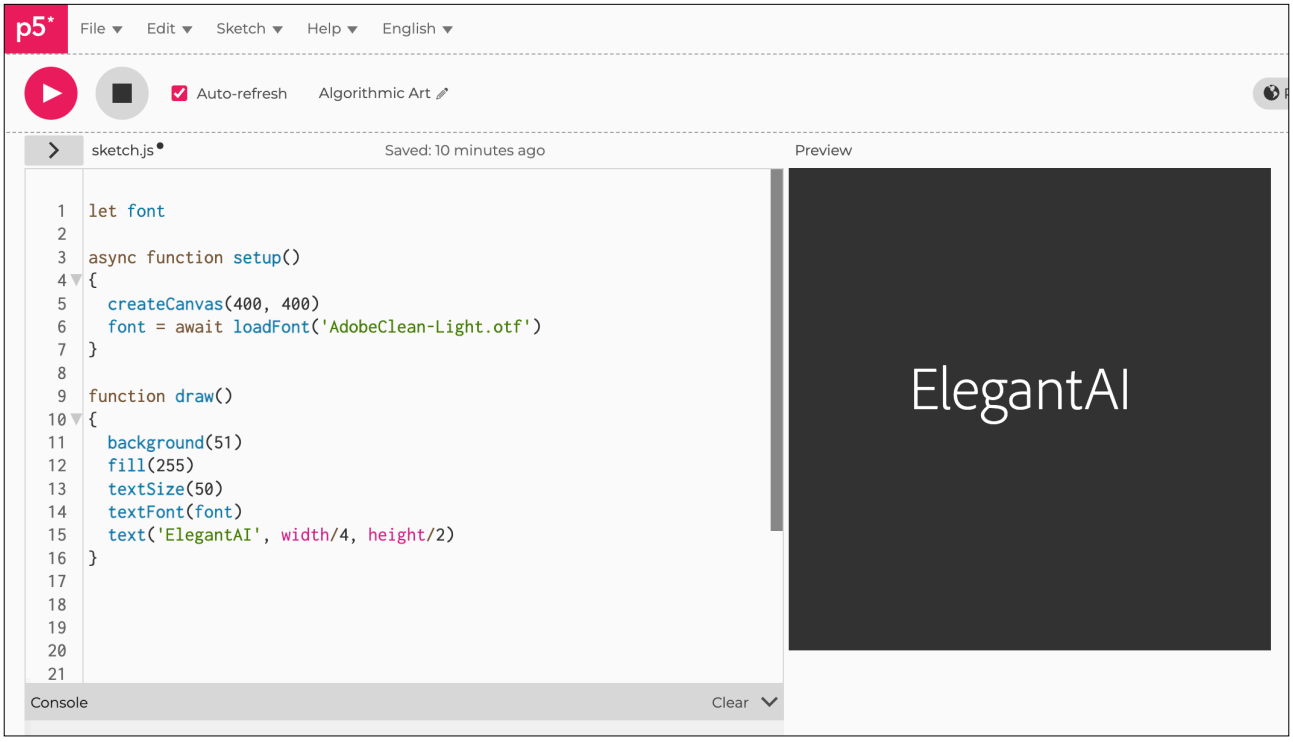
Notes

The font looks very similar. We need this extension to be able to use `WEBGL` and therefore anaglyph.

Challenge

You could try downloading a more interesting font off the internet, but they must be `.otf` or `.ttf` format.

Figure G1.15


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 10 minutes ago
Preview
let font
async function setup()
{
createCanvas(400, 400)
font = await loadFont('AdobeClean-Light.otf')
}
function draw()
{
background(51)
fill(255)
textSize(50)
textFont(font)
text('ElegantAI', width/4, height/2)
}
ElegantAI
Console
Clear



Sketch G1.16 WEBGL

Add the WEBGL and we can now make it rotate about an axis.

```
let font
let angle = 0

async function setup()
{
  createCanvas(400, 400, WEBGL)
  font = await loadFont('AdobeClean-Light.otf')
}

function draw()
{
  background(51)
  fill(255)
  textSize(50)
  textFont(font)
  rotateY(angle)
  text('ElegantAI', 0, 0)
  angle += 0.01
}
```

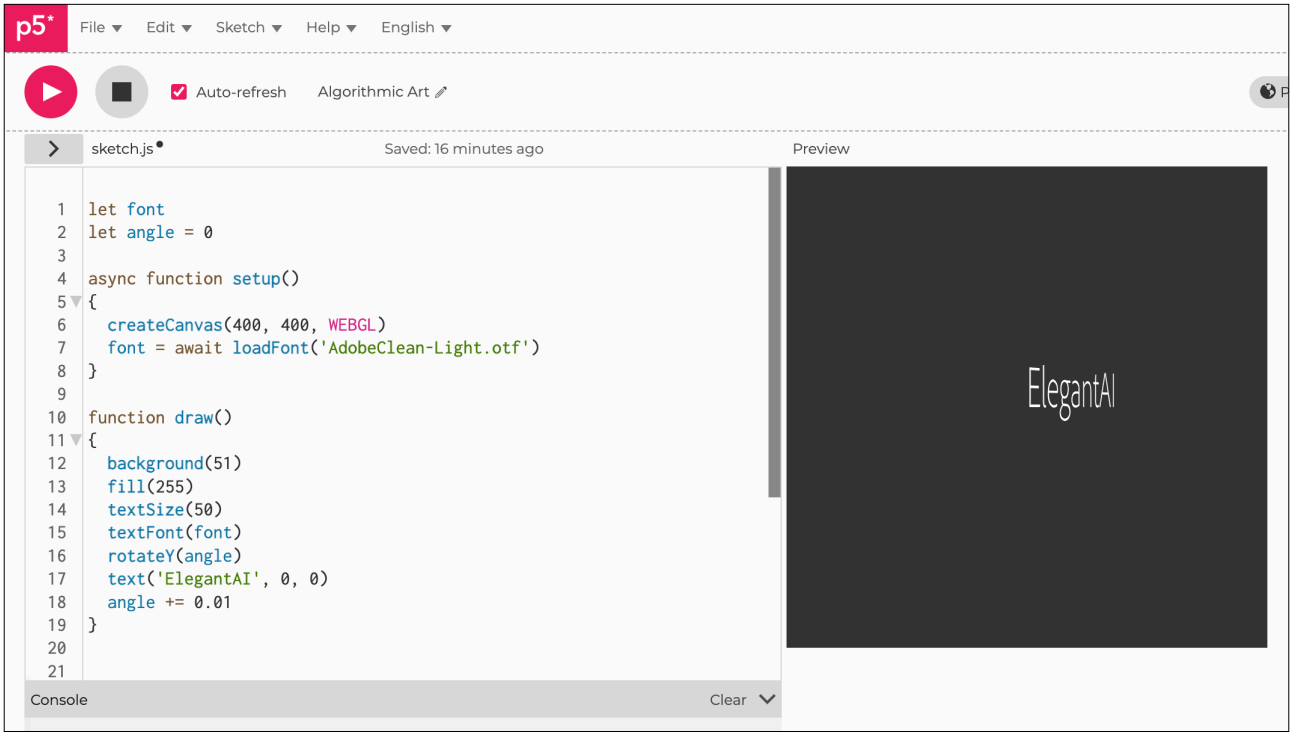
Notes

The text should be rotating slowly. Remember that we set the co-ordinates to 0, 0.

Challenge

Can you rotate it about the centre of the text?

Figure G1.16


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 16 minutes ago
Preview
let font
let angle = 0
async function setup()
{
createCanvas(400, 400, WEBGL)
font = await loadFont('AdobeClean-Light.otf')
}
function draw()
{
background(51)
fill(255)
textSize(50)
textFont(font)
rotateY(angle)
text('ElegantAI', 0, 0)
angle += 0.01
}
ElegantAI
Console
Clear



Sketch G1.17 the anaglyph

Quite a bit of refactoring, as you might imagine, and then put on your 3D glasses. We are also going to rotate it around the centre of the text.

```
let font
let angle = 0
let ananglyph

async function setup()
{
  createCanvas(400, 400, WEBGL)
  font = await loadFont('AdobeClean-Light.otf')
  anaglyph = createAnaglyph(this)
}

function draw()
{
  angle += 0.01
  anaglyph.draw(scene)
}

function scene(graphics)
{
  graphics.background(51)
  graphics.fill(255)
  graphics.textSize(50)
  graphics.textFont(font)
  graphics.rotateY(angle)
  graphics.text('ElegantAI', -100, 0)
}
```

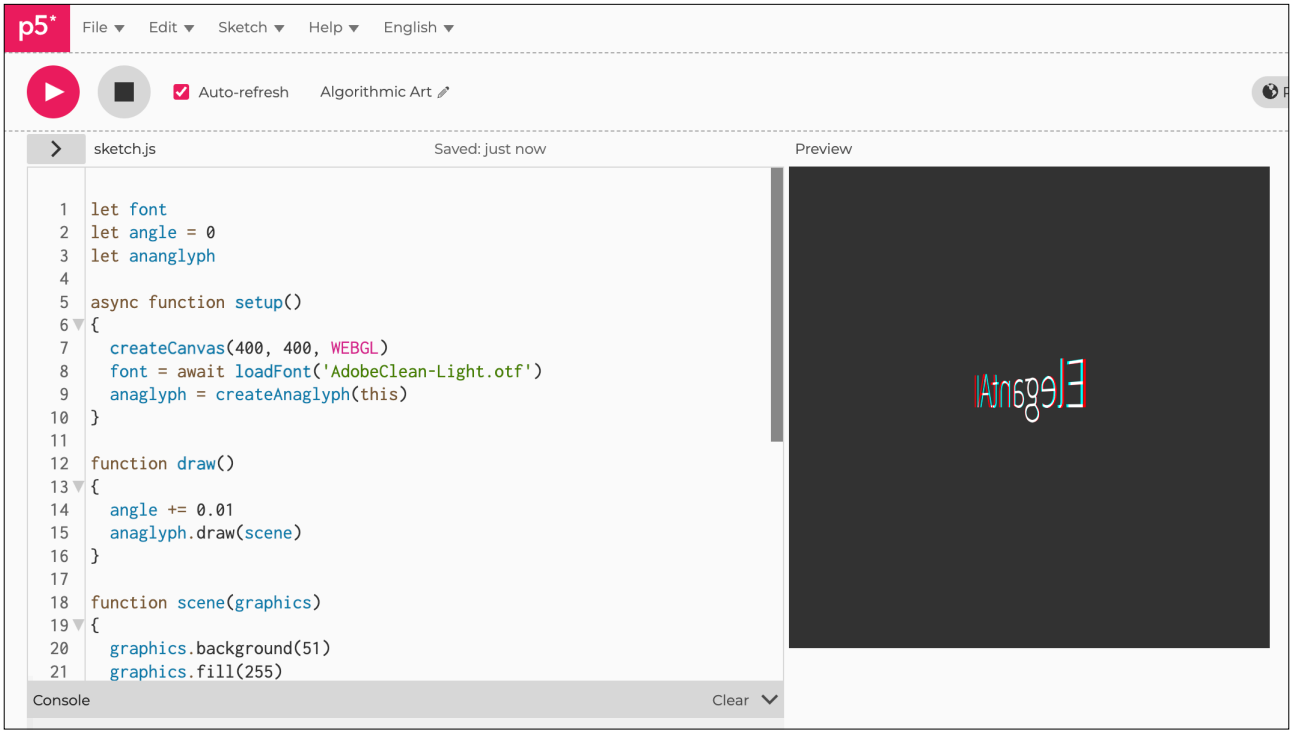
Notes

We rotate it around somewhere in the middle of the text. Also, now it is in 3D.

Challenge

Could you move the text backwards and forward?

Figure G1.17


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: just now
Preview
let font
let angle = 0
let ananglyph
async function setup()
{
createCanvas(400, 400, WEBGL)
font = await loadFont('AdobeClean-Light.otf')
anaglyph = createAnaglyph(this)
}
function draw()
{
angle += 0.01
anaglyph.draw(scene)
}
function scene(graphics)
{
graphics.background(51)
graphics.fill(255)
Console
Clear



Sketch G1.18 Dodecahedron

We can also create an irregular (although this is actually regular) shape using the `vertex()` function. In this case, we are going to create a `dodecahedron`, which means that there is going to be a whole lot of copying and pasting.

A `dodecahedron` has `12` faces, where each face is a `pentagon`. The pentagon has `5` edges and `5` vertices. This means that the `dodecahedron` has a total of `20` vertices.

This is a 3D object. We need the vectors of each vertex, which contain three elements (`x`, `y`, `z`). We need an array to hold all these vertices. We will call it `vert[]`. We also use some maths to calculate the other vertices. We use the `golden ratio`; we used something like this in the phyllotaxis example.

```
let vert = []


function setup()
{
  createCanvas(400, 400, WEBGL)
  const A = (1 + sqrt(5)) / 2
  const B = 1 / A
  vert.push(createVector(1, 1, 1))
  vert.push(createVector(1, 1, -1))
  vert.push(createVector(1, -1, 1))
  vert.push(createVector(1, -1, -1))
  vert.push(createVector(-1, 1, 1))
  vert.push(createVector(-1, 1, -1))
  vert.push(createVector(-1, -1, 1))
  vert.push(createVector(-1, -1, -1))
  vert.push(createVector(0, B, A))
  vert.push(createVector(0, B, -A))
  vert.push(createVector(0, -B, A))
  vert.push(createVector(0, -B, -A))
  vert.push(createVector(B, A, 0))
  vert.push(createVector(B, -A, 0))
  vert.push(createVector(-B, A, 0))
  vert.push(createVector(-B, -A, 0))
  vert.push(createVector(A, 0, B))
  vert.push(createVector(A, 0, -B))
  vert.push(createVector(-A, 0, B))
  vert.push(createVector(-A, 0, -B))
```

```
}
```

Notes

These are just the vertices. Believe it or not, we need to take these vertices in a very specific order and arrange them around each pentagon face. Admittedly, other shapes might have been easier.

Code Explanation

`const A = (1 + sqrt(5)) / 2`	The golden ratio approximately: 1.618033988749895
`const B = 1 / A`	The inverse of the golden ratio is: 0.6180339887498948



Sketch G1.19 face the pentagon

We need to combine the vertices to each face. We have 12 faces, each with 5 vertices. Some of these vertices are shared by 3 faces. The great thing is that someone has done the hard work for us (not me), and we can identify each vertex with an individual index, which will have three elements. We will use this information later.

We need an array to hold these faces, and we will call the array `faces[]`.

```
let vert = []
let faces = []

function setup()
{
  createCanvas(400, 400, WEBGL)
  const A = (1 + sqrt(5)) / 2
  const B = 1 / A
  vert.push(createVector(1, 1, 1))
  vert.push(createVector(1, 1, -1))
  vert.push(createVector(1, -1, 1))
  vert.push(createVector(1, -1, -1))
  vert.push(createVector(-1, 1, 1))
  vert.push(createVector(-1, 1, -1))
  vert.push(createVector(-1, -1, 1))
  vert.push(createVector(-1, -1, -1))
  vert.push(createVector(0, B, A))
  vert.push(createVector(0, B, -A))
  vert.push(createVector(0, -B, A))
  vert.push(createVector(0, -B, -A))
  vert.push(createVector(B, A, 0))
  vert.push(createVector(B, -A, 0))
  vert.push(createVector(-B, A, 0))
  vert.push(createVector(-B, -A, 0))
  vert.push(createVector(A, 0, B))
  vert.push(createVector(A, 0, -B))
  vert.push(createVector(-A, 0, B))
  vert.push(createVector(-A, 0, -B))

  faces.push([0, 16, 2, 10, 8])
  faces.push([0, 8, 4, 14, 12])
```

```
  faces.push([16, 17, 1, 12, 0])
  faces.push([1, 9, 11, 3, 17])
  faces.push([1, 12, 14, 5, 9])
  faces.push([2, 13, 15, 6, 10])
  faces.push([13, 3, 17, 16, 2])
  faces.push([3, 11, 7, 15, 13])
  faces.push([4, 8, 10, 6, 18])
  faces.push([14, 5, 19, 18, 4])
  faces.push([5, 19, 7, 11, 9])
  faces.push([15, 7, 19, 18, 6])
}
```

Notes

To speed up the pasting and copying, I created a line with x's rather than numbers and copied and pasted 12 times. Then I went back through them very carefully, changing the x to the relevant number. A bit tedious but necessary.

Challenge

I am sure you can find a quicker way to do this.



Sketch G1.20 resizing

You may be wondering what this is going to look like as the numbers we have put in seem to be mainly 1's and 0's or thereabouts. We need to increase each vector by 50 so that it is something we will actually be able to see.

```
let vert = []
let faces = []
let r = 50

function setup()
{
  createCanvas(400, 400, WEBGL)
  const A = (1 + sqrt(5)) / 2
  const B = 1 / A
  vert.push(createVector(1, 1, 1))
  vert.push(createVector(1, 1, -1))
  vert.push(createVector(1, -1, 1))
  vert.push(createVector(1, -1, -1))
  vert.push(createVector(-1, 1, 1))
  vert.push(createVector(-1, 1, -1))
  vert.push(createVector(-1, -1, 1))
  vert.push(createVector(-1, -1, -1))
  vert.push(createVector(0, B, A))
  vert.push(createVector(0, B, -A))
  vert.push(createVector(0, -B, A))
  vert.push(createVector(0, -B, -A))
  vert.push(createVector(B, A, 0))
  vert.push(createVector(B, -A, 0))
  vert.push(createVector(-B, A, 0))
  vert.push(createVector(-B, -A, 0))
  vert.push(createVector(A, 0, B))
  vert.push(createVector(A, 0, -B))
  vert.push(createVector(-A, 0, B))
  vert.push(createVector(-A, 0, -B))
  for (let i = 0; i < vert.length; i++)
  {
    vert[i].mult(r)
  }
```

```
  faces.push([0, 16, 2, 10, 8])
  faces.push([0, 8, 4, 14, 12])
  faces.push([16, 17, 1, 12, 0])
  faces.push([1, 9, 11, 3, 17])
  faces.push([1, 12, 14, 5, 9])
  faces.push([2, 13, 15, 6, 10])
  faces.push([13, 3, 17, 16, 2])
  faces.push([3, 11, 7, 15, 13])
  faces.push([4, 8, 10, 6, 18])
  faces.push([14, 5, 19, 18, 4])
  faces.push([5, 19, 7, 11, 9])
  faces.push([15, 7, 19, 18, 6])
}
```

Notes

Nothing to see. You can use `console.log(vert[2])` to see how that works.

Challenge

You can multiply by a different number if you want.



Sketch G1.21 draw the dodecahedron

We have an array of faces. In each face, we have an array which are the indices for the vertices. Basically, we systematically work through each face and draw out each vertex. Each vertex is a point, and we can join these vertices together.

```
let vert = []
let faces = []
let r = 50

function setup()
{
  createCanvas(400, 400, WEBGL)
  const A = (1 + sqrt(5)) / 2
  const B = 1 / A
  vert.push(createVector(1, 1, 1))
  vert.push(createVector(1, 1, -1))
  vert.push(createVector(1, -1, 1))
  vert.push(createVector(1, -1, -1))
  vert.push(createVector(-1, 1, 1))
  vert.push(createVector(-1, 1, -1))
  vert.push(createVector(-1, -1, 1))
  vert.push(createVector(-1, -1, -1))
  vert.push(createVector(0, B, A))
  vert.push(createVector(0, B, -A))
  vert.push(createVector(0, -B, A))
  vert.push(createVector(0, -B, -A))
  vert.push(createVector(B, A, 0))
  vert.push(createVector(B, -A, 0))
  vert.push(createVector(-B, A, 0))
  vert.push(createVector(-B, -A, 0))
  vert.push(createVector(A, 0, B))
  vert.push(createVector(A, 0, -B))
  vert.push(createVector(-A, 0, B))
  vert.push(createVector(-A, 0, -B))
  for (let i = 0; i < vert.length; i++)
  {
    vert[i].mult(r)
  }
```

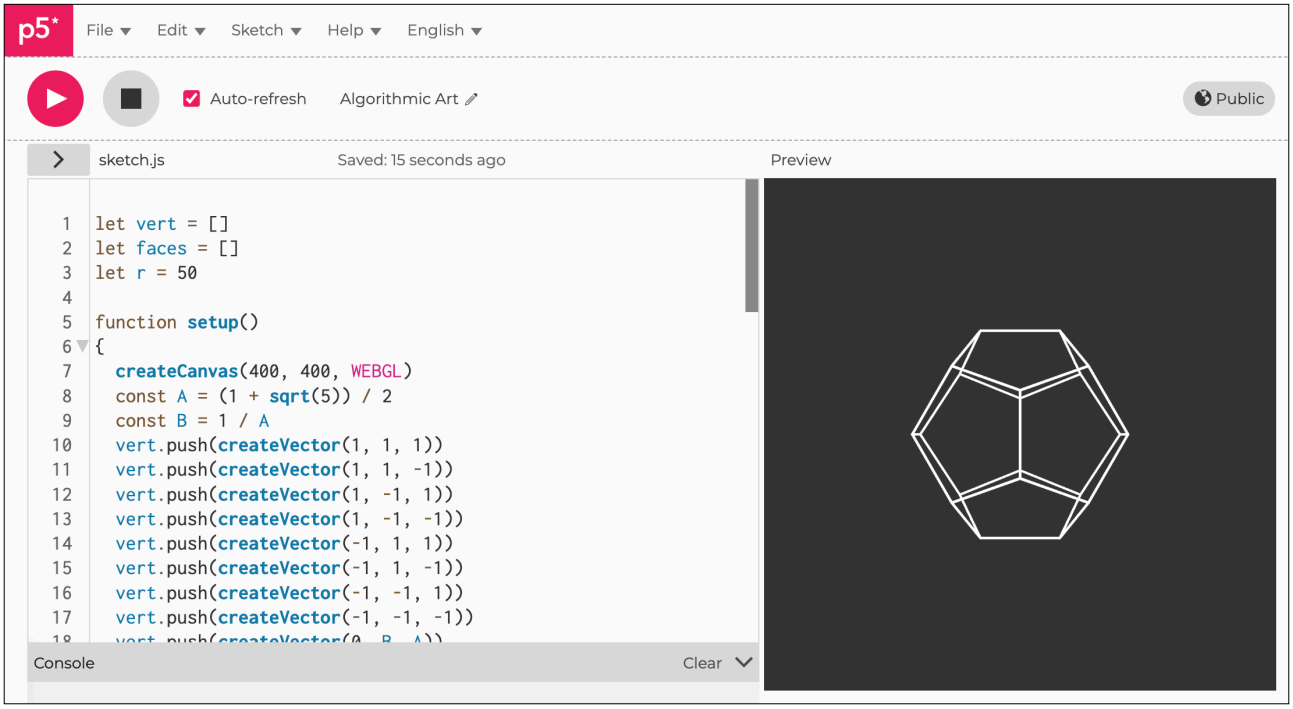
```
  faces.push([0, 16, 2, 10, 8])
  faces.push([0, 8, 4, 14, 12])
  faces.push([16, 17, 1, 12, 0])
  faces.push([1, 9, 11, 3, 17])
  faces.push([1, 12, 14, 5, 9])
  faces.push([2, 13, 15, 6, 10])
  faces.push([13, 3, 17, 16, 2])
  faces.push([3, 11, 7, 15, 13])
  faces.push([4, 8, 10, 6, 18])
  faces.push([14, 5, 19, 18, 4])
  faces.push([5, 19, 7, 11, 9])
  faces.push([15, 7, 19, 18, 6])
}
```

```
function draw()
{
  background(51)
  stroke(255)
  strokeWeight(2)
  noFill()
  for (let i = 0; i < faces.length; i++)
  {
    beginShape()
    for (let j = 0; j < faces[i].length; j++)
    {
      let v = vert[faces[i][j]]
      vertex(v.x, v.y, v.z)
    }
    endShape(CLOSE)
  }
}
```

Notes

We now have something to look at, a static dodecahedron. All that hard work has paid off.

Figure G1.21


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
Public
sketch.js
Saved: 15 seconds ago
Preview
let vert = []
let faces = []
let r = 50
function setup()
{
createCanvas(400, 400, WEBGL)
const A = (1 + sqrt(5)) / 2
const B = 1 / A
vert.push(createVector(1, 1, 1))
vert.push(createVector(1, 1, -1))
vert.push(createVector(1, -1, 1))
vert.push(createVector(1, -1, -1))
vert.push(createVector(-1, 1, 1))
vert.push(createVector(-1, 1, -1))
vert.push(createVector(-1, -1, 1))
vert.push(createVector(-1, -1, -1))
Console
Clear

Sketch G1.22 rotate

We can now rotate the dodecahedron.

```
let vert = []
let faces = []
let r = 50
let angle = 0

function setup()
{
  createCanvas(400, 400, WEBGL)
  const A = (1 + sqrt(5)) / 2
  const B = 1 / A
  vert.push(createVector(1, 1, 1))
  vert.push(createVector(1, 1, -1))
  vert.push(createVector(1, -1, 1))
  vert.push(createVector(1, -1, -1))
  vert.push(createVector(-1, 1, 1))
  vert.push(createVector(-1, 1, -1))
  vert.push(createVector(-1, -1, 1))
  vert.push(createVector(-1, -1, -1))
  vert.push(createVector(0, B, A))
  vert.push(createVector(0, B, -A))
  vert.push(createVector(0, -B, A))
  vert.push(createVector(0, -B, -A))
  vert.push(createVector(B, A, 0))
  vert.push(createVector(B, -A, 0))
  vert.push(createVector(-B, A, 0))
  vert.push(createVector(-B, -A, 0))
  vert.push(createVector(A, 0, B))
  vert.push(createVector(A, 0, -B))
  vert.push(createVector(-A, 0, B))
  vert.push(createVector(-A, 0, -B))
  for (let i = 0; i < vert.length; i++)
  {
    vert[i].mult(r)
  }
  faces.push([0, 16, 2, 10, 8])
```

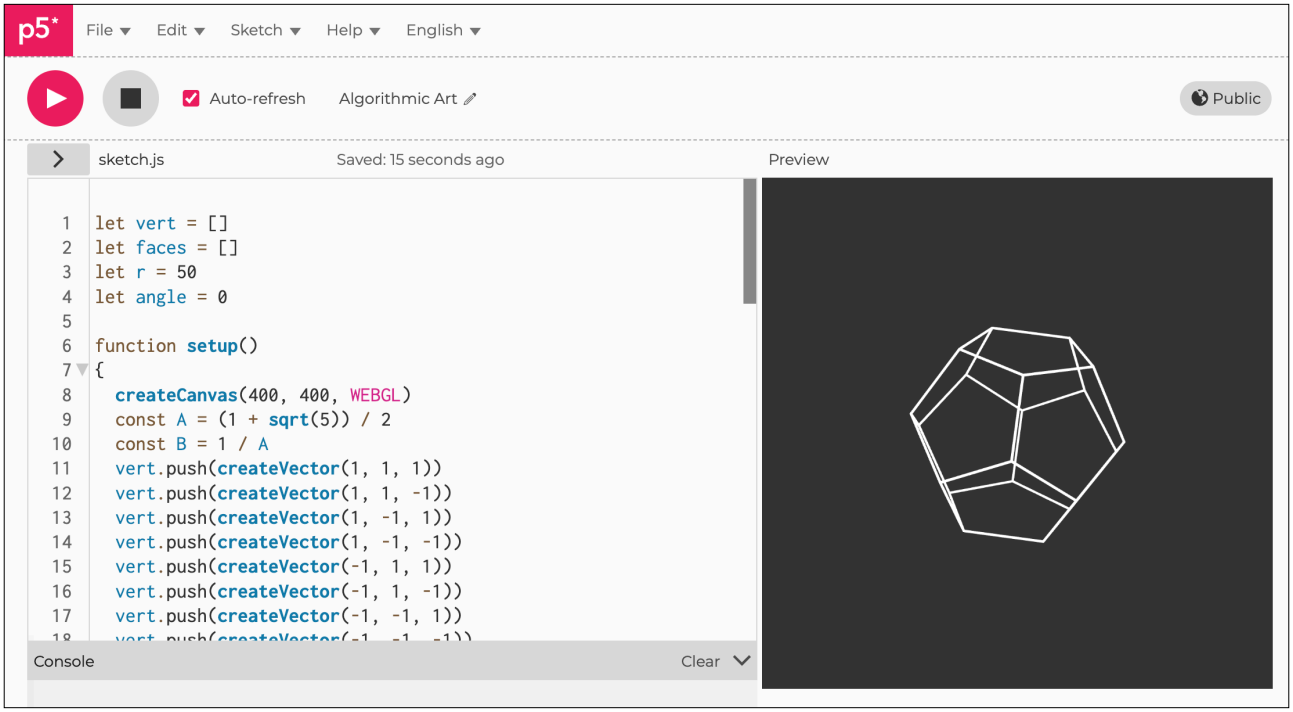
```
  faces.push([0, 8, 4, 14, 12])
  faces.push([16, 17, 1, 12, 0])
  faces.push([1, 9, 11, 3, 17])
  faces.push([1, 12, 14, 5, 9])
  faces.push([2, 13, 15, 6, 10])
  faces.push([13, 3, 17, 16, 2])
  faces.push([3, 11, 7, 15, 13])
  faces.push([4, 8, 10, 6, 18])
  faces.push([14, 5, 19, 18, 4])
  faces.push([5, 19, 7, 11, 9])
  faces.push([15, 7, 19, 18, 6])
}

function draw()
{
  background(51)
  rotateX(angle)
  rotateY(angle)
  stroke(255)
  strokeWeight(2)
  noFill()
  for (let i = 0; i < faces.length; i++)
  {
    beginShape()
    for (let j = 0; j < faces[i].length; j++)
    {
      let v = vert[faces[i][j]]
      vertex(v.x, v.y, v.z)
    }
    endShape(CLOSE)
  }
  angle += 0.01
}
```

Notes

A lovely rotating dodecahedron.

Figure G1.22


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
Public
sketch.js
Saved: 15 seconds ago
Preview
let vert = []
let faces = []
let r = 50
let angle = 0
function setup()
{
createCanvas(400, 400, WEBGL)
const A = (1 + sqrt(5)) / 2
const B = 1 / A
vert.push(createVector(1, 1, 1))
vert.push(createVector(1, 1, -1))
vert.push(createVector(1, -1, 1))
vert.push(createVector(1, -1, -1))
vert.push(createVector(-1, 1, 1))
vert.push(createVector(-1, 1, -1))
vert.push(createVector(-1, -1, 1))
Console
Clear



Sketch G1.23 the anaglyph

Here we bring in the usual changes. One way is to copy most of the `draw()` function and paste it into the new `scene()` function. Then we add the `graphic.` prefix. Time to put your glasses on. I think you will agree it looks great.

```
let vert = []
let faces = []
let r = 50
let angle = 0
let anaglyph

function setup()
{
  createCanvas(400, 400, WEBGL)
  anaglyph = createAnaglyph(this)
  const A = (1 + sqrt(5)) / 2
  const B = 1 / A
  vert.push(createVector(1, 1, 1))
  vert.push(createVector(1, 1, -1))
  vert.push(createVector(1, -1, 1))
  vert.push(createVector(1, -1, -1))
  vert.push(createVector(-1, 1, 1))
  vert.push(createVector(-1, 1, -1))
  vert.push(createVector(-1, -1, 1))
  vert.push(createVector(-1, -1, -1))
  vert.push(createVector(0, B, A))
  vert.push(createVector(0, B, -A))
  vert.push(createVector(0, -B, A))
  vert.push(createVector(0, -B, -A))
  vert.push(createVector(B, A, 0))
  vert.push(createVector(B, -A, 0))
  vert.push(createVector(-B, A, 0))
  vert.push(createVector(-B, -A, 0))
  vert.push(createVector(A, 0, B))
  vert.push(createVector(A, 0, -B))
  vert.push(createVector(-A, 0, B))
  vert.push(createVector(-A, 0, -B))
  for (let i = 0; i < vert.length; i++)
```

```
  {
    vert[i].mult(r)
  }
  faces.push([0, 16, 2, 10, 8])
  faces.push([0, 8, 4, 14, 12])
  faces.push([16, 17, 1, 12, 0])
  faces.push([1, 9, 11, 3, 17])
  faces.push([1, 12, 14, 5, 9])
  faces.push([2, 13, 15, 6, 10])
  faces.push([13, 3, 17, 16, 2])
  faces.push([3, 11, 7, 15, 13])
  faces.push([4, 8, 10, 6, 18])
  faces.push([14, 5, 19, 18, 4])
  faces.push([5, 19, 7, 11, 9])
  faces.push([15, 7, 19, 18, 6])
}

function draw()
{
  anaglyph.draw(scene)
  angle += 0.01
}

function scene(graphic)
{
  graphic.background(51)
  graphic.rotateX(angle)
  graphic.rotateY(angle)
  graphic.stroke(255)
  graphic.strokeWeight(2)
  graphic.noFill()
  for (let i = 0; i < faces.length; i++)
  {
    graphic.beginShape()
    for (let j = 0; j < faces[i].length; j++)
    {
      let v = vert[faces[i][j]]
      graphic.vertex(v.x, v.y, v.z)
```

```
    }
    graphic.endShape(CLOSE)
  }
}
```

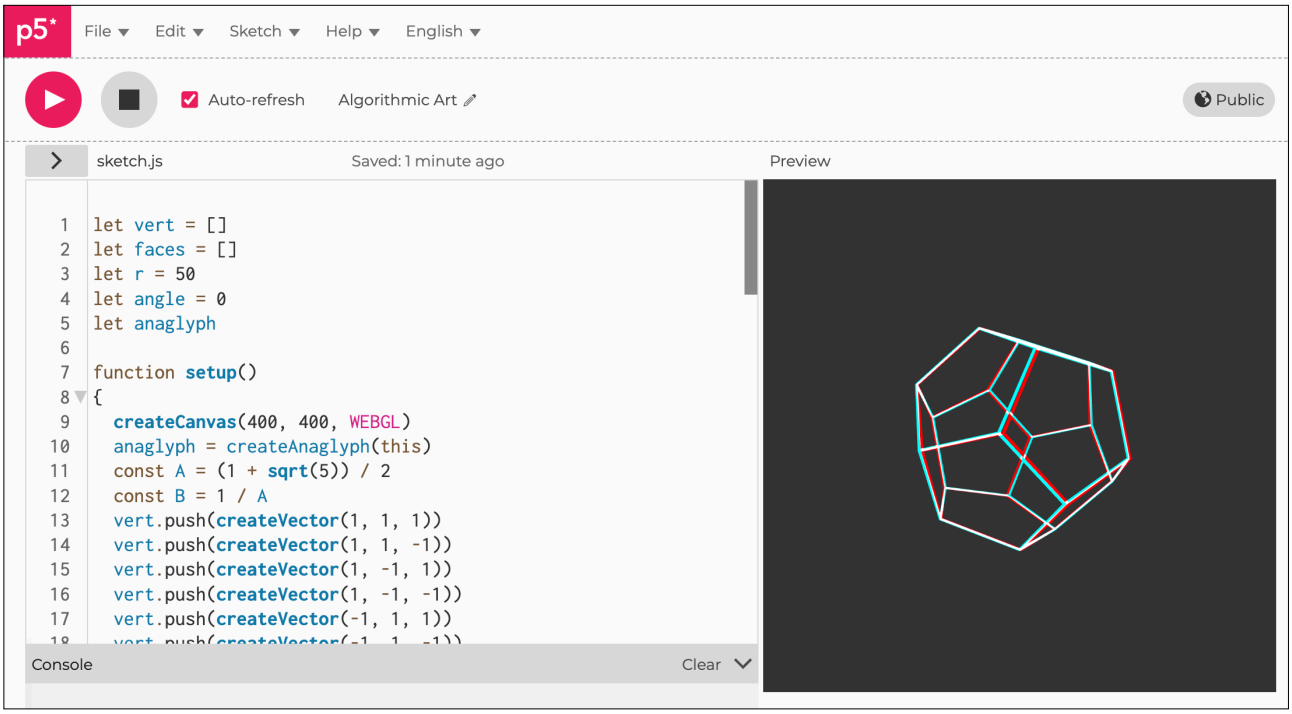
Notes

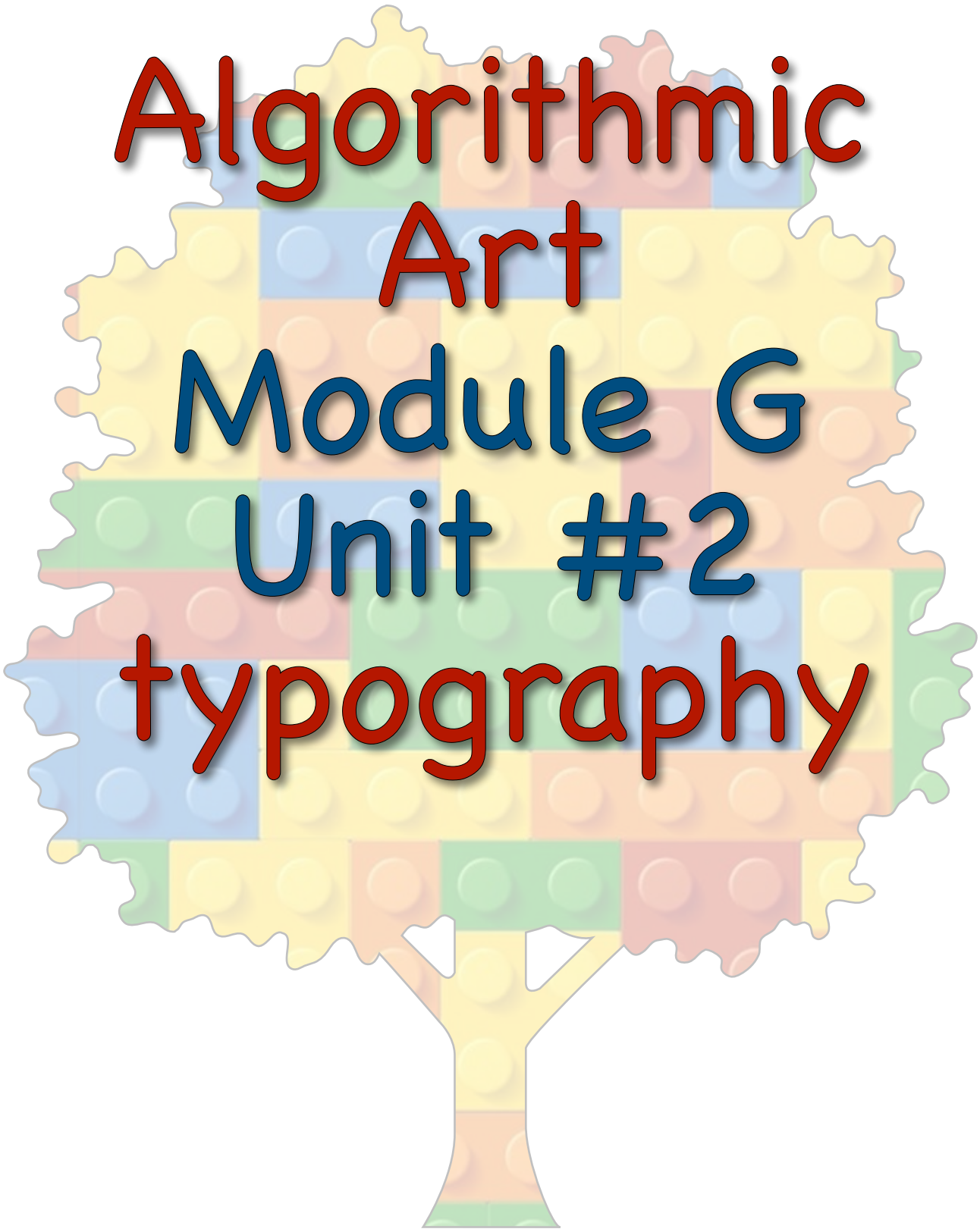
This shows that you do need to have a good understanding of geometry. I have shown you this to whet your appetite.

Challenge

Try your own designs; maybe start with something simpler to start with.

Figure G1.23




Algorithmic
Art
Module G
Unit #2
typography



Module G Unit #2 typography

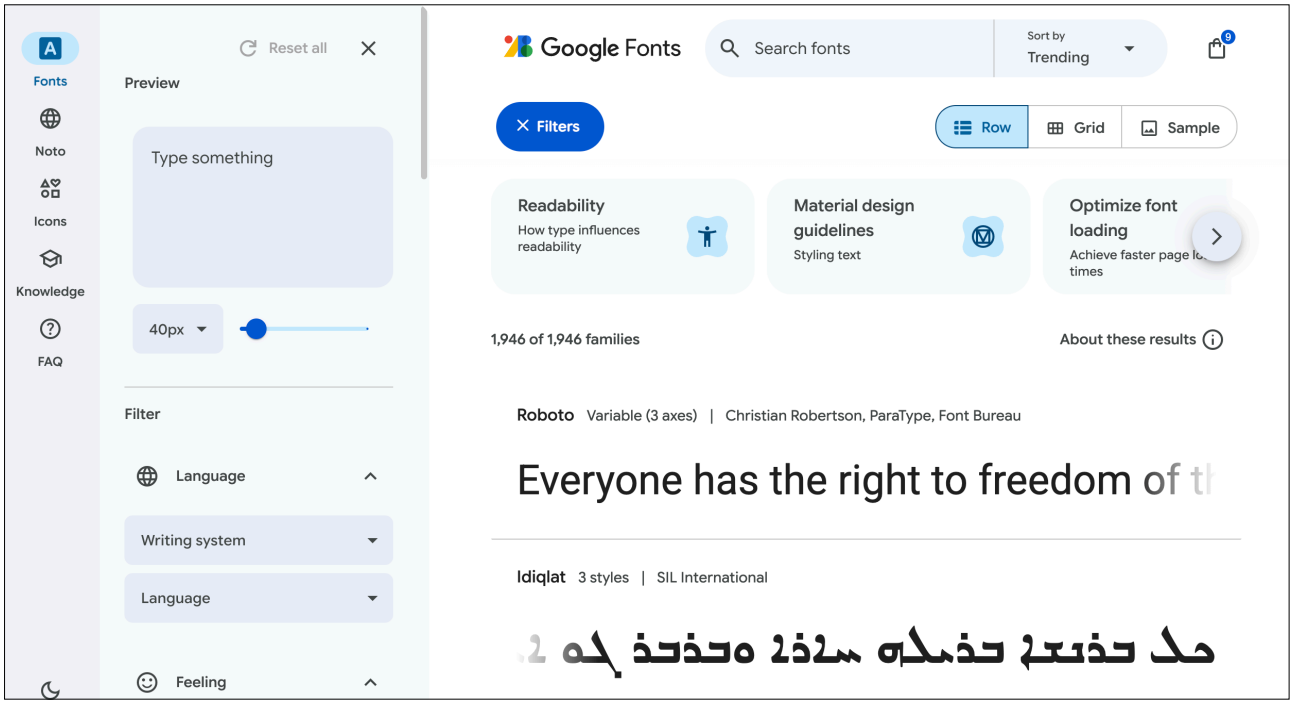
We want to download a font for this unit. The one we want has variable weights. The one I suggest is called `Roboto-VariableFont_wdth,wght.ttf` which is a bit of a mouthful. To download a `.ttf` font, first go to `https://fonts.google.com/`. There are some much more interesting fonts which I would encourage you to explore, but finding a variable weights one is harder than you might think.

Download and Upload font

First of all we need to download the font we need from Google Fonts at:

https://fonts.google.com/

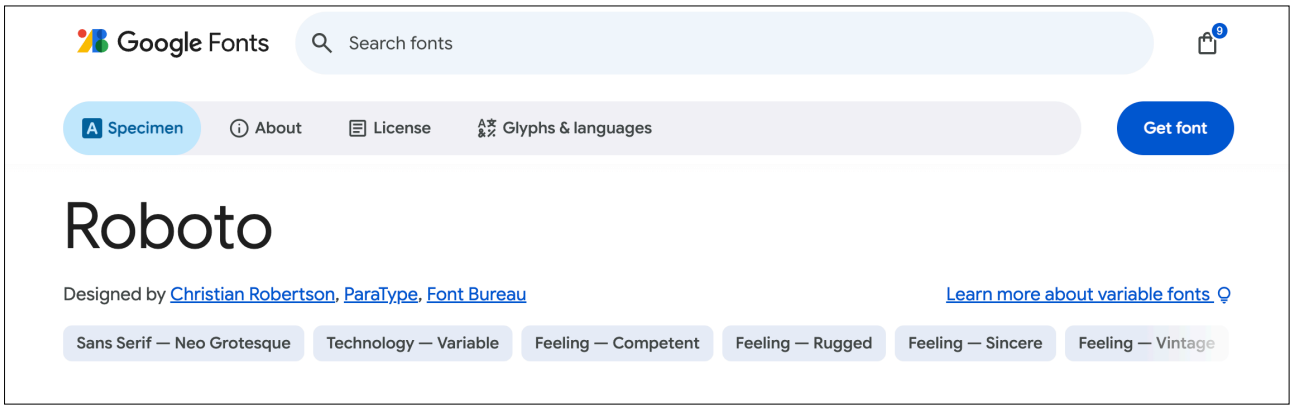
Figure 1: Google fonts



Alternatively you can get a link to this page on my website.

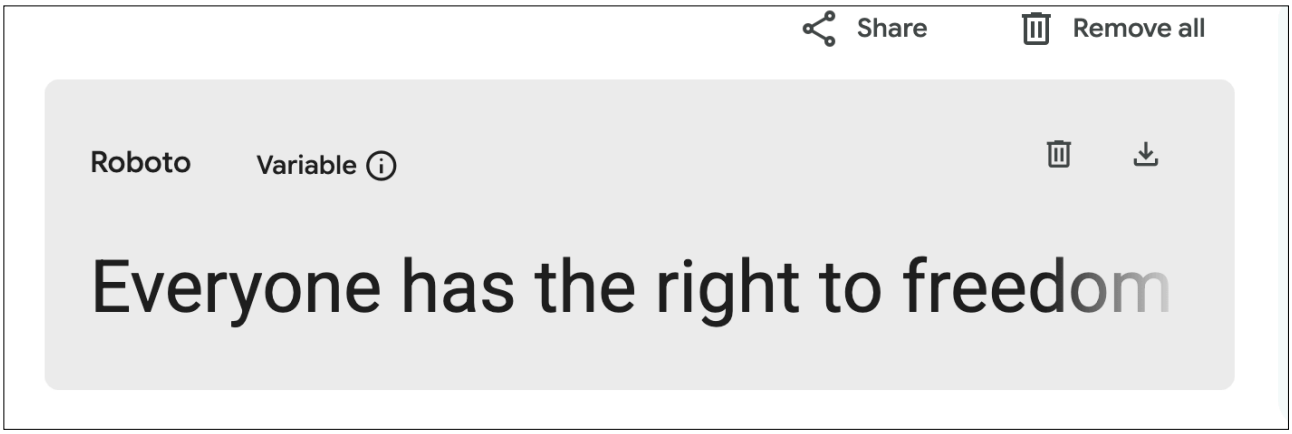
Next, select the family for `Roboto` and click on the `Get font` button (top right).

Figure 2: Roboto family of fonts



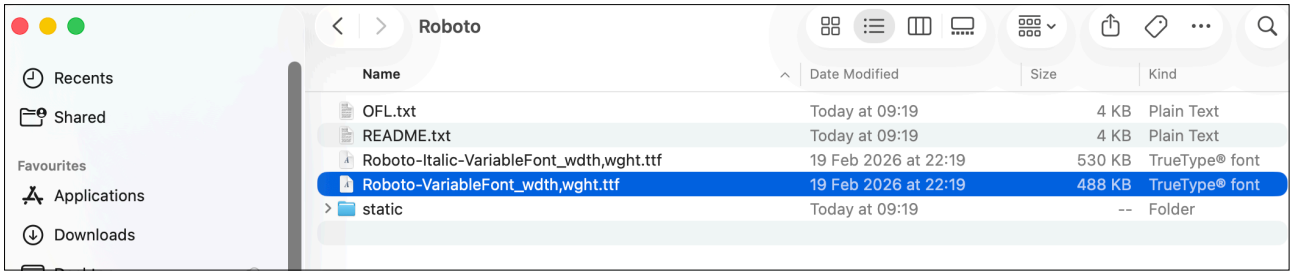
We get a few fonts from this family (nine altogether). Click on the download button.

Figure 3: download button



This will still give you numerous fonts. The one you want to find is the `Roboto-VariableFont_wdth,wght.ttf`. Depending on your computer (and operating system), it will be in a place where downloads are saved. You don't need to do anything with it except find where it is in a folder labelled `Roboto`.

Figure 4: downloaded font somewhere

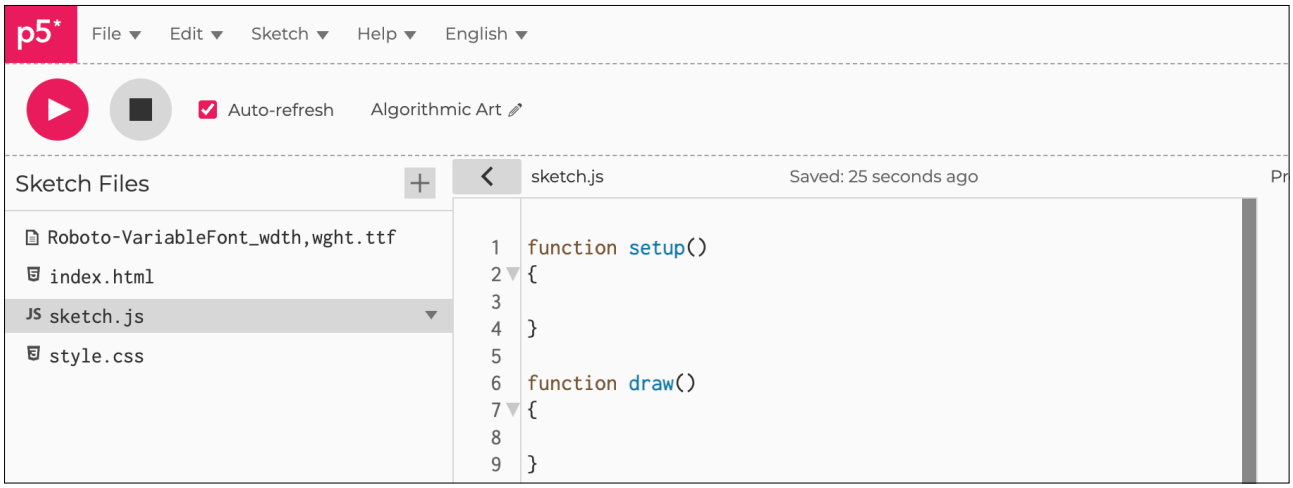




Uploaded font

This is something you should be familiar with. Go to the list of files for your sketch and click on the upload file tab. Click Browse and find the font file and upload it.

Figure 5: uploaded font



Sketch G2.1 display text

! This is our starting sketch.

It will print the text to the canvas. We use async and await to make sure we have uploaded the font before we use it.

```
let font

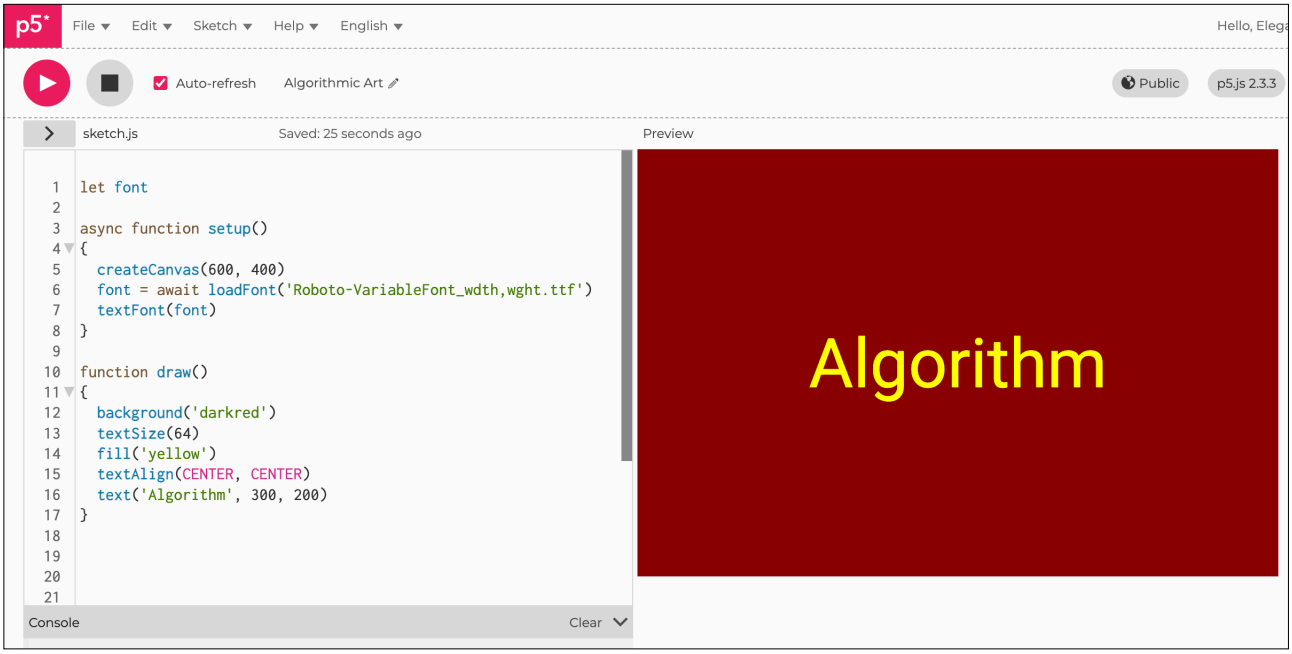
async function setup()
{
  createCanvas(600, 400)
  font = await loadFont('Roboto-VariableFont_wdth,wght.ttf')
  textFont(font)
}

function draw()
{
  background('darkred')
  textSize(64)
  fill('yellow')
  textAlign(CENTER, CENTER)
  text('Algorithm', 300, 200)
}
```

Notes

This will also serve to check that the font is working.

Figure G2.1


p5*
File
Edit
Sketch
Help
English
Hello, Elega
Auto-refresh
Algorithmic Art
Public
p5.js 2.3.3
sketch.js
Saved: 25 seconds ago
Preview
let font
async function setup()
{
createCanvas(600, 400)
font = await loadFont('Roboto-VariableFont_wdth,wght.ttf')
textFont(font)
}
function draw()
{
background('darkred')
textSize(64)
fill('yellow')
textAlign(CENTER, CENTER)
text('Algorithm', 300, 200)
}
Algorithm
Console
Clear



Sketch G2.2 breath of life

Now we can breathe some life into the text. We will get it to expand and contract as a sine wave. The sine wave runs between −1 and +1; we will map this to the weights (thickness of the font).

```
let font
let wt
let angle = 0

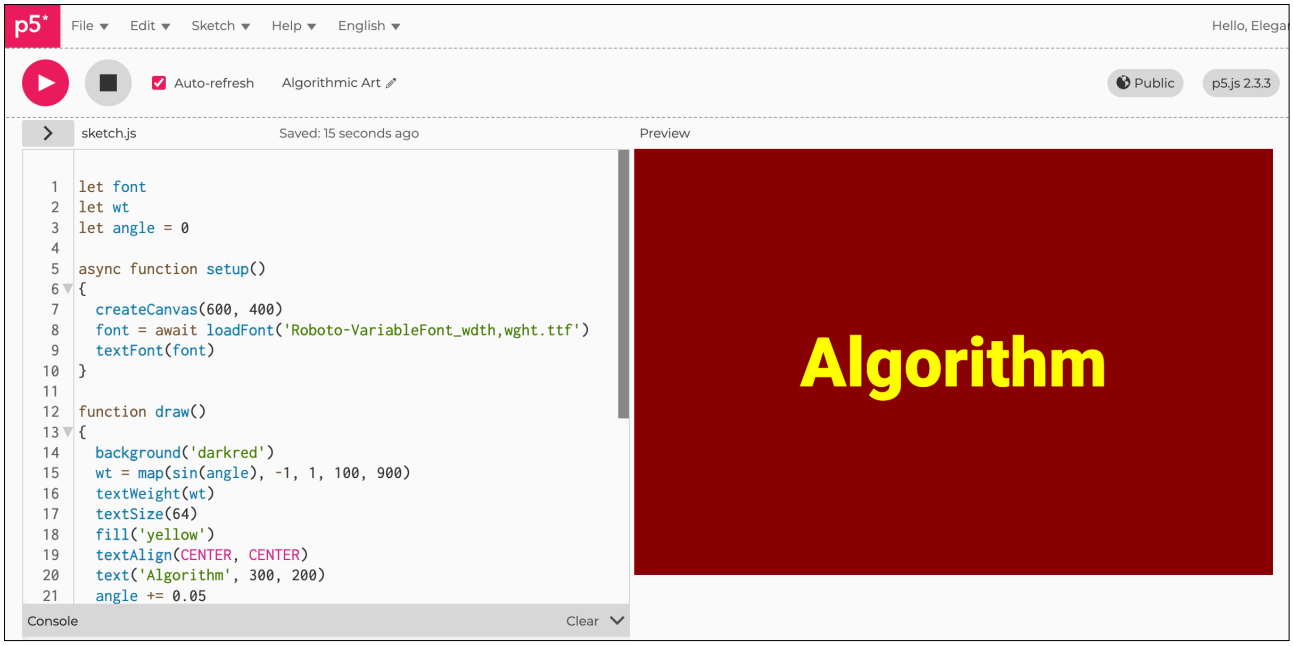
async function setup()
{
  createCanvas(600, 400)
  font = await loadFont('Roboto-VariableFont_wdth,wght.ttf')
  textFont(font)
}

function draw()
{
  background('darkred')
  wt = map(sin(angle), -1, 1, 100, 900)
  textWeight(wt)
  textSize(64)
  fill('yellow')
  textAlign(CENTER, CENTER)
  text('Algorithm', 300, 200)
  angle += 0.05
}
```

Notes

We increment the angle by 0.05. This oscillates the weight from 100 to 900, back to 100, and so on.

Figure G2.2


p5*
File
Edit
Sketch
Help
English
Hello, Elega
Auto-refresh
Algorithmic Art
Public
p5.js 2.3.3
sketch.js
Saved: 15 seconds ago
Preview
let font
let wt
let angle = 0
async function setup()
{
createCanvas(600, 400)
font = await loadFont('Roboto-VariableFont_wdth,wght.ttf')
textFont(font)
}
function draw()
{
background('darkred')
wt = map(sin(angle), -1, 1, 100, 900)
textWeight(wt)
textSize(64)
fill('yellow')
textAlign(CENTER, CENTER)
text('Algorithm', 300, 200)
angle += 0.05
Algorithm
Console
Clear

Sketch G2.3 dashed outline

We will create some dashed lines around the outline of the text. We will do this by getting the points that track the outline (not all of them, just spaced out) and then drawing a small rectangle in their place. We will also need to rotate these rectangles as they go round the corners of the letter shapes.

! This is our starting sketch

```
let font

async function setup()
{
  createCanvas(600, 400)
  font = await loadFont('Roboto-VariableFont_wdth,wght.ttf')
  textFont(font)
  textAlign(CENTER, CENTER)
  textSize(200)
}

function draw()
{
  background('darkred')
  fill('yellow')
  text('Art', 300, 200)
}
```

Notes

We will remove: `text('Art', 300, 200)` shortly. Notice also that it is much shorter text and larger for a good reason.

Figure G2.3


p5*
File
Edit
Sketch
Help
English
Hello, Elega
Auto-refresh
Algorithmic Art
Public
p5.js 2.3.3
sketch.js
Saved: 1 minute ago
Preview
let font
async function setup()
{
createCanvas(600, 400)
font = await loadFont('Roboto-VariableFont_wdth,wght.ttf')
textFont(font)
textAlign(CENTER, CENTER)
textSize(200)
}
function draw()
{
background('darkred')
fill('yellow')
text('Art', 300, 200)
}
Art
Console
Clear

Sketch G2.4 what's the point

We can use some clever functions in p5.js version 2.x. We want to get the points in the text. We want to hold all the points in a variable called `allPoints`. We don't want all the points but just a sample of them; for that, we have the option `sampleFactor`, which we will set to `0.1`.

! Note that we have deleted the text line of code in the `draw()` function.

```
let font
let allPoints

async function setup()
{
  createCanvas(600, 400)
  font = await loadFont('Roboto-VariableFont_wdth,wght.ttf')
  textFont(font)
  textAlign(CENTER, CENTER)
  textSize(200)
  let options = { sampleFactor: 0.1 }
  allPoints = font.textToContours('Art', 300, 200, options)
}

function draw()
{
  background('darkred')
  fill('yellow')
  // text('Art', 300, 200)
}
```

Notes

We now have all (some) of the points. Each point is an array of four elements: `x`, `y`, `angle`, and `alpha`. We next need to get those data points and draw them.

Challenge

Change the `sampleFactor` (later on) to see the difference it makes.

Figure G2.4


p5*
File
Edit
Sketch
Help
English
Hello, Elega
Auto-refresh
Algorithmic Art
Public
p5.js 2.3.3
sketch.js
Saved: 15 seconds ago
Preview
let font
let allpoints
async function setup()
{
createCanvas(600, 400)
font = await loadFont('Roboto-VariableFont_wdth,wght.ttf')
textFont(font)
textAlign(CENTER, CENTER)
textSize(200)
let options = { sampleFactor: 0.1 }
allpoints = font.textToContours('Art', 300, 200, options)
}
function draw()
{
background('darkred')
fill('yellow')
}
Console
Clear



Sketch G2.5 lines, letters and arrays

The length of the `allPoints` array is `7`. This is because the letters are made up of segments. For instance, if you had the letter `O`, it would count for two arrays, the inner and outer circle. The `A` has three, the `r` has two, and the `t` has two. This is just the way the font structures the letters.

We take each segment and pull out the points in that array and create another array called `letterPoints`. This will have a much larger number of elements. We can then use that data to draw the points of each letter, creating a dashed line effect. There are other effects you could create.

```
let font
let allPoints
let letterPoints

async function setup()
{
  createCanvas(600, 400)
  font = await loadFont('Roboto-VariableFont_wdth,wght.ttf')
  textFont(font)
  textAlign(CENTER, CENTER)
  textSize(200)
  let options = { sampleFactor: 0.1 }
  allPoints = font.textToContours('Art', 300, 200, options)
}

function draw()
{
  background('darkred')
  for (let i = 0; i < allPoints.length; i++)
  {
    letterPoints = allPoints[i]
    for (let j = 0; j < letterPoints.length; j++)
    {
      fill('yellow')
    }
  }
}
```

Notes

Just taking it one step at a time. Nothing to see yet except a sea of dark red.



Sketch G2.6 getting the point

We use another variable that can pull out and reference individual elements for each point, such as the x, y, and angle components, which we will need to draw the dashed line. This means it will follow the curve of the letter (if there is one). We call this variable pt (point for short).

```
let font
let allPoints
let letterPoints
let pt

async function setup()
{
  createCanvas(600, 400)
  font = await loadFont('Roboto-VariableFont_wdth,wght.ttf')
  textFont(font)
  textAlign(CENTER, CENTER)
  textSize(200)
  let options = { sampleFactor: 0.1 }
  allPoints = font.textToContours('Art', 300, 200, options)
}

function draw()
{
  background('darkred')
  for (let i = 0; i < allPoints.length; i++)
  {
    letterPoints = allPoints[i]
    for (let j = 0; j < letterPoints.length; j++)
    {
      pt = letterPoints[j]
      noStroke()
      fill('yellow')
      rectMode(CENTER)
      push()
      translate(pt.x, pt.y)
      rotate(pt.angle)
      rect(0, 0, 5, 2.5)
```

```
        pop()
      }
    }
}
```

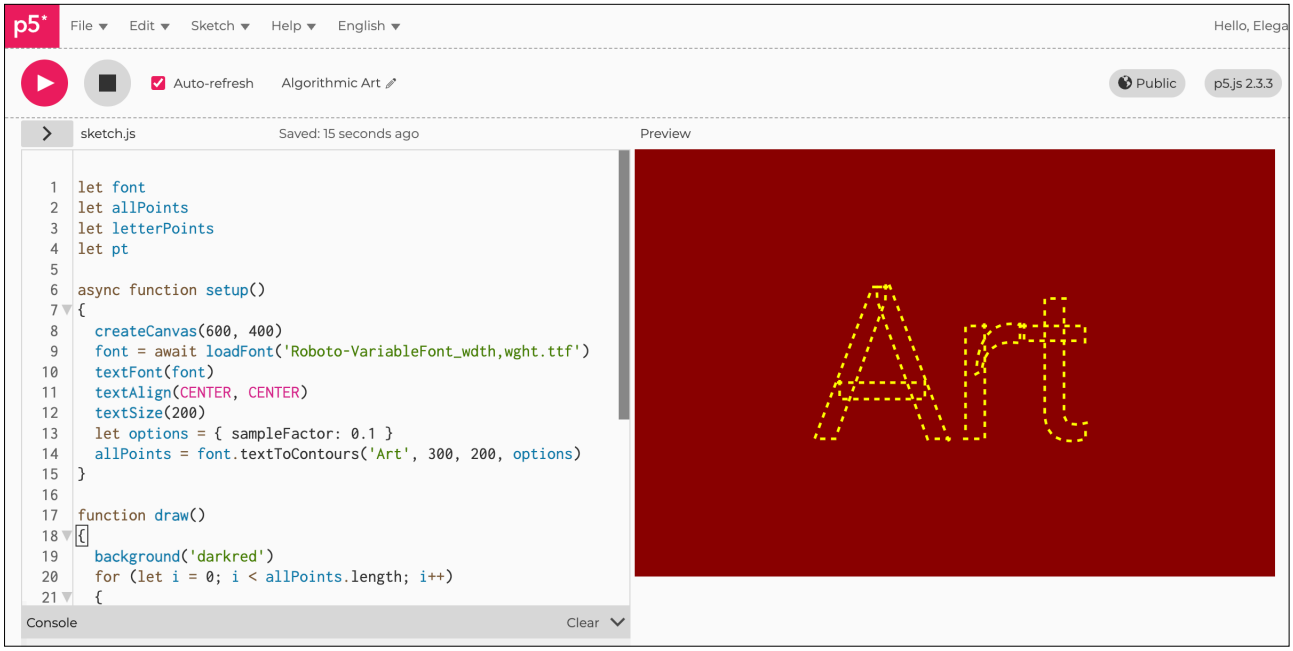
Notes

We have our points as dashes.

Challenge

1. Try other shapes
2. Could you use them to make it 3D (glasses)?

Figure G2.6



Sketch G2.8 3D extruded text

! Our starting sketch, delete what isn't needed and change the highlighted lines of code.

Using the same font, we can extrude the text and make it 3D. I would recommend using version 2.0.5 for this example and for the next one. In the images, I have used the latest version, which does still work but there is a conflict or bug (at present).

```
let font

async function setup()
{
  createCanvas(600, 400, WEBGL)
  font = await loadFont('Roboto-VariableFont_wdth,wght.ttf')
  textFont(font)
  textAlign(CENTER, CENTER)
  textSize(200)
  let options = { extrude: 50, sampleFactor: 0.1 }
}

function draw()
{
  background('darkred')
}
```

Notes

Just a red canvas.



Sketch G2.9 geometry

Instead of getting the contours of the text, now we get the geometry using the `textToModel()` function.

! You will get an error message (ignore it). If it bothers you, use a previous version of p5.js (still version `2.x`), such as `2.0.5` (my recommendation).

```
let font
let textGeom

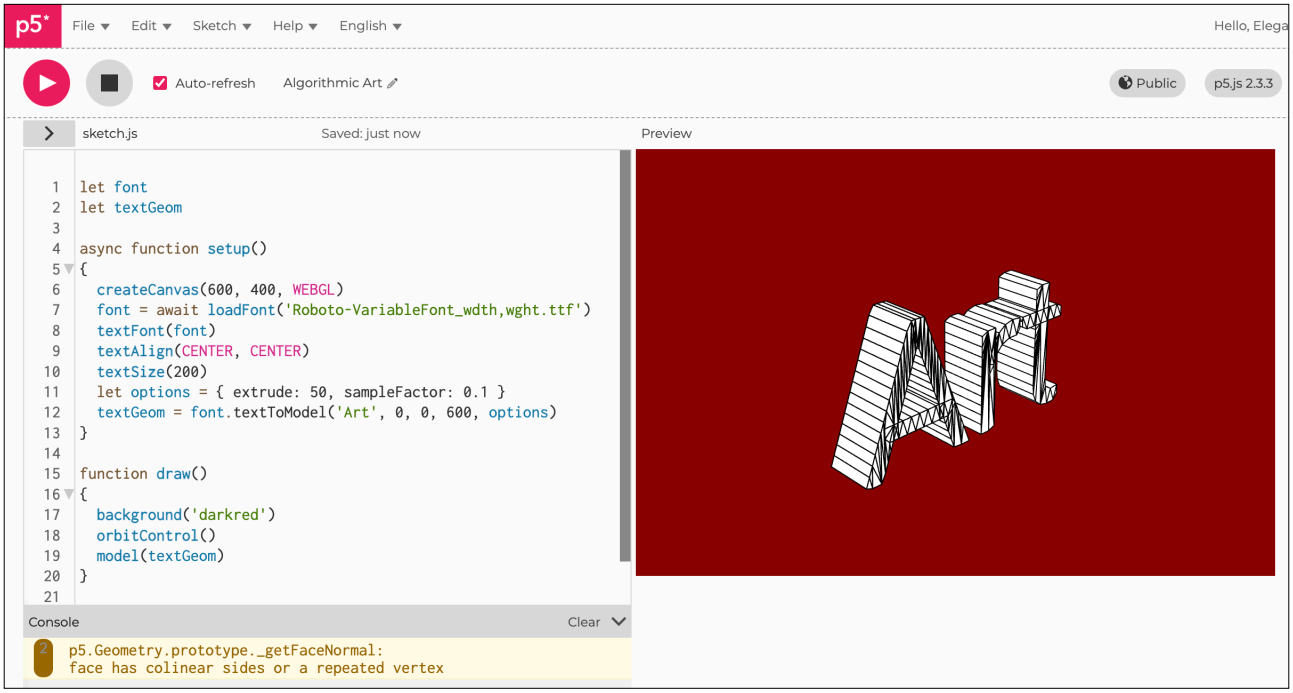
async function setup()
{
  createCanvas(600, 400, WEBGL)
  font = await loadFont('Roboto-VariableFont_wdth,wght.ttf')
  textFont(font)
  textAlign(CENTER, CENTER)
  textSize(200)
  let options = { extrude: 50, sampleFactor: 0.1 }
  textGeom = font.textToModel('Art', 0, 0, 600, options)
}

function draw()
{
  background('darkred')
  orbitControl()
  model(textGeom)
}
```

Notes

We also add the orbital control function so we can rotate it with the mouse.

Figure G2.9


p5*
File
Edit
Sketch
Help
English
Hello, Elega
Auto-refresh
Algorithmic Art
Public
p5.js 2.3.3
sketch.js
Saved: just now
Preview
let font
let textGeom
async function setup()
{
createCanvas(600, 400, WEBGL)
font = await loadFont('Roboto-VariableFont_wdth,wght.ttf')
textFont(font)
textAlign(CENTER, CENTER)
textSize(200)
let options = { extrude: 50, sampleFactor: 0.1 }
textGeom = font.textToModel('Art', 0, 0, 600, options)
}
function draw()
{
background('darkred')
orbitControl()
model(textGeom)
}
Console
Clear
2
p5.Geometry.prototype._getFaceNormal:
face has colinear sides or a repeated vertex



Sketch G2.10 let there be lights

Remove the lines and add some light.

```
let font
let textGeom

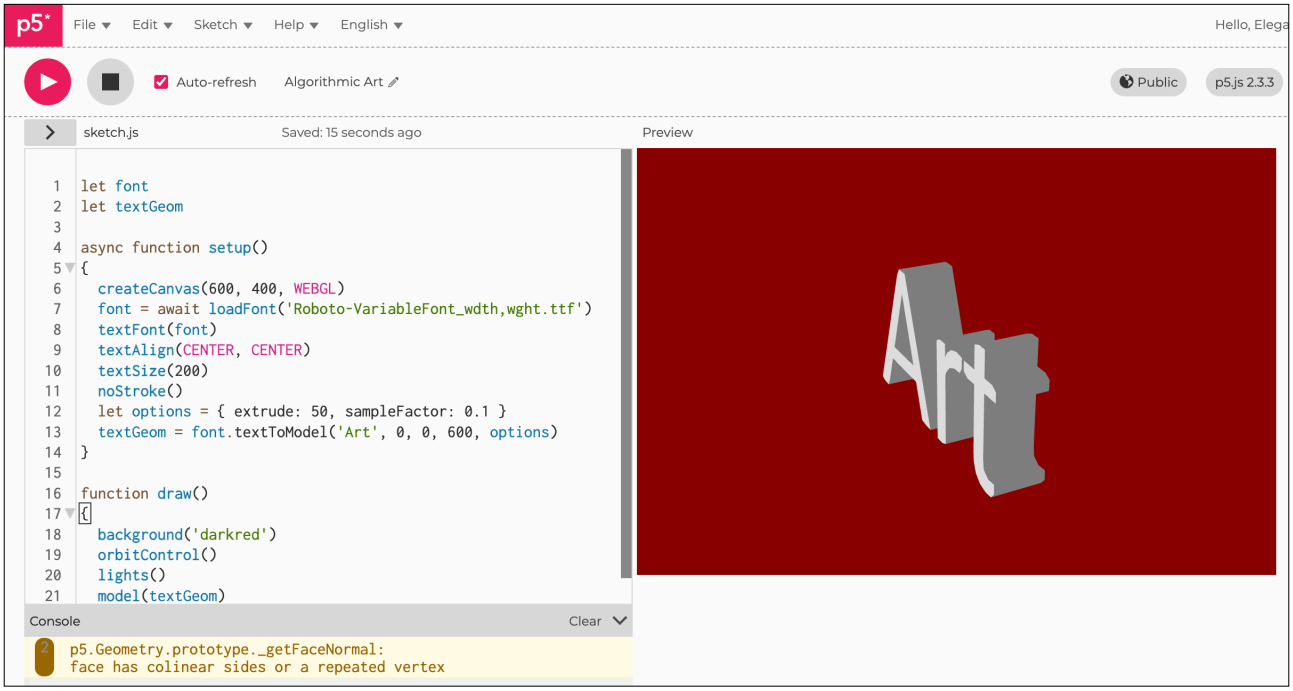
async function setup()
{
  createCanvas(600, 400, WEBGL)
  font = await loadFont('Roboto-VariableFont_wdth,wght.ttf')
  textFont(font)
  textAlign(CENTER, CENTER)
  textSize(200)
  noStroke()
  let options = { extrude: 50, sampleFactor: 0.1 }
  textGeom = font.textToModel('Art', 0, 0, 600, options)
}

function draw()
{
  background('darkred')
  orbitControl()
  lights()
  model(textGeom)
}
```

Notes

You will still get the error message, more of a warning than a problem.

Figure G2.10


p5*
File
Edit
Sketch
Help
English
Hello, Elega
Auto-refresh
Algorithmic Art
Public
p5.js 2.3.3
sketch.js
Saved: 15 seconds ago
Preview
let font
let textGeom
async function setup()
{
createCanvas(600, 400, WEBGL)
font = await loadFont('Roboto-VariableFont_wdth,wght.ttf')
textFont(font)
textAlign(CENTER, CENTER)
textSize(200)
noStroke()
let options = { extrude: 50, sampleFactor: 0.1 }
textGeom = font.textToModel('Art', 0, 0, 600, options)
}
function draw()
{
background('darkred')
orbitControl()
lights()
model(textGeom)
Console
Clear
p5.Geometry.prototype._getFaceNormal:
face has colinear sides or a repeated vertex

Sketch G2.11 steering behaviour

We can use the points on the contours of the text to move them from some random position back to where they belong. The effect is for the text to materialise as the points swarm in. You get the picture, I hope.

! Very important. Please change the latest version to **2.2.3** for this next example. You will get lots of horrible warning messages, which will be some bug in the current (my latest) version. It may well be sorted by the time you are coding this.

! This is our starting sketch using the same font.

```
let font
let points

async function setup()
{
  createCanvas(600, 400)
  font = await loadFont('Roboto-VariableFont_wdth,wght.ttf')
  textFont(font)
  textAlign(CENTER, CENTER)
  textSize(200)
  let options = { sampleFactor: 0.1 }
  points = font.textToContours('Happy', 300, 200, options)
}

function draw()
{
  background('darkred')
}
```

Notes

Nothing to see yet



Sketch G2.12 pt points

Getting all the points (well, most of them). Just to make sure everything is working OK. Comment out the background in `draw()` and let's draw the points in setup for now. We will be changing how we draw them later.

```
let font
let points
let pt

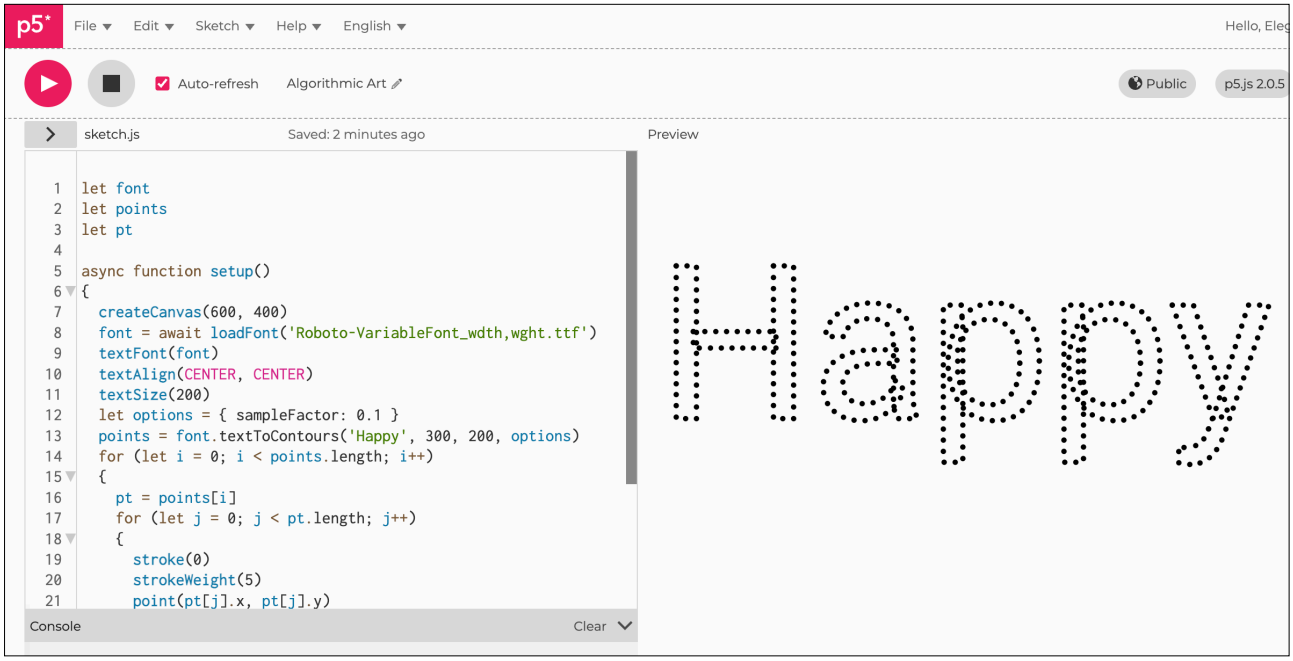
async function setup()
{
  createCanvas(600, 400)
  font = await loadFont('Roboto-VariableFont_wdth,wght.ttf')
  textFont(font)
  textAlign(CENTER, CENTER)
  textSize(200)
  let options = { sampleFactor: 0.1 }
  points = font.textToContours('Happy', 300, 200, options)
  for (let i = 0; i < points.length; i++)
  {
    pt = points[i]
    for (let j = 0; j < pt.length; j++)
    {
      stroke(0)
      strokeWeight(5)
      point(pt[j].x, pt[j].y)
    }
  }
}

function draw()
{
  // background('darkred')
}
```

Notes

We can see the points for the word `Happy`. You will get a incomplete Happy if using the latest version, just change the version 'till you get the full word (I found 2.2.3 worked OK).

Figure G2.12


p5*
File
Edit
Sketch
Help
English
Hello, Eleg
Auto-refresh
Algorithmic Art
Public
p5.js 2.0.5
sketch.js
Saved: 2 minutes ago
Preview
let font
let points
let pt
async function setup()
{
createCanvas(600, 400)
font = await loadFont('Roboto-VariableFont_wdth,wght.ttf')
textFont(font)
textAlign(CENTER, CENTER)
textSize(200)
let options = { sampleFactor: 0.1 }
points = font.textToContours('Happy', 300, 200, options)
for (let i = 0; i < points.length; i++)
{
pt = points[i]
for (let j = 0; j < pt.length; j++)
{
stroke(0)
strokeWeight(5)
point(pt[j].x, pt[j].y)
Console
Clear
Happy



Sketch G2.13 vehicles

We want these points to move, and so we will designate them as vehicles. For that, we need a `vehicle class`. This is the basics of vectors: movement of particles. There isn't room in this unit to cover each in detail. The `maxspeed` and `maxforce` variables are there to stop the particles going berserk.

Just in case you are unfamiliar, `pos` is position, `vel` is velocity, `acc` is acceleration. We have set the starting position randomly along the left-hand edge. Remove the commented-out lines of code. We will need to apply a force to the vehicles and eventually steer them to their allotted places.

```
let font
let points
let pt

async function setup()
{
  createCanvas(600, 400)
  font = await loadFont('Roboto-VariableFont_wdth,wght.ttf')
  textFont(font)
  textAlign(CENTER, CENTER)
  textSize(200)
  let options = { sampleFactor: 0.1 }
  points = font.textToContours('Happy', 300, 200, options)
  for (let i = 0; i < points.length; i++)
  {
    pt = points[i]
    for (let j = 0; j < pt.length; j++)
    {
      // stroke(0)
      // strokeWeight(5)
      // point(pt[j].x, pt[j].y)
    }
  }
}

function draw()
{
  background('darkred')
```

```
}
```

```
class Vehicle
{
  constructor(x, y)
  {
    this.pos = createVector(0, random(height))
    this.vel = createVector(random(1), random(1))
    this.acc = createVector(0, 0)
    this.maxspeed = 10
    this.maxforce = 1
  }

  show()
  {
    stroke('yellow')
    strokeWeight(5)
    point(this.pos.x, this.pos.y)
  }

  move()
  {
    this.pos.add(this.vel)
    this.vel.add(this.acc)
    this.acc.set(0, 0)
  }

  applyForce(f)
  {
    this.acc.add(f)
  }
}
```

Notes

Remember to change the version to 2.2.3 (others do work as well). There is nothing to see yet.



Sketch G2.14 steer

We want to steer the vehicle towards the target, which is the final position. We want it to arrive at the target once it has been steered. We have also added these vehicles to an array of vehicles that we want to steer and move to their desired target spot.

```
let font
let points
let pt
let vehicle
let vehicles = []

async function setup()
{
  createCanvas(600, 400)
  font = await loadFont('Roboto-VariableFont_wdth,wght.ttf')
  textFont(font)
  textAlign(CENTER, CENTER)
  textSize(200)
  let options = { sampleFactor: 0.1 }
  points = font.textToContours('Happy', 300, 200, options)
  for (let i = 0; i < points.length; i++)
  {
    pt = points[i]
    for (let j = 0; j < pt.length; j++)
    {
      vehicle = new Vehicle(pt[j].x, pt[j].y)
      vehicles.push(vehicle)
    }
  }
}

function draw()
{
  background('darkred')
}

class Vehicle
{
```

```
constructor(x, y)
{
  this.pos = createVector(0, random(height))
  this.vel = createVector(random(1), random(1))
  this.acc = createVector(0, 0)
  this.maxspeed = 10
  this.maxforce = 1
}

show()
{
  stroke('yellow')
  strokeWeight(5)
  point(this.pos.x, this.pos.y)
}

move()
{
  this.pos.add(this.vel)
  this.vel.add(this.acc)
  this.acc.set(0, 0)
}

applyForce(f)
{
  this.acc.add(f)
}
```

```
arrive(target)
{
  let desired = p5.Vector.sub(target, this.pos)
  let d = desired.mag()
  if (d < 100)
  {
    let mg = map(d, 0, 100, 0, this.maxspeed)
    desired.setMag(mg)
  }
  else
```

```
    {
      desired.setMag(this.maxspeed)
    }
    let steer = p5.Vector.sub(desired, this.vel)
    steer.limit(this.maxforce)
    this.applyForce(steer)
  }
}
```

Notes

Getting our ducks in a row. I will be exploring vectors and forces in more detail in a forthcoming tutorial series or workbooks about physics simulation. Nothing to see yet.



Sketch G2.15 good behaviour

We need to bring all this together and get the vehicles moving by calling them in the `draw()` function. The `behaviours()` function moves each vehicle to its allotted target position from its random starting point.

```
let font
let points
let pt
let vehicle
let vehicles = []

async function setup()
{
  createCanvas(600, 400)
  font = await loadFont('Roboto-VariableFont_wdth,wght.ttf')
  textFont(font)
  textAlign(CENTER, CENTER)
  textSize(200)
  let options = { sampleFactor: 0.1 }
  points = font.textToContours('Happy', 300, 200, options)
  for (let i = 0; i < points.length; i++)
  {
    pt = points[i]
    for (let j = 0; j < pt.length; j++)
    {
      vehicle = new Vehicle(pt[j].x, pt[j].y)
      vehicles.push(vehicle)
    }
  }
}

function draw()
{
  background('darkred')
  for (let i = 0; i < vehicles.length; i++)
  {
    let v = vehicles[i]
    v.behaviours()
```

```
    v.move()
    v.show()
  }
}

class Vehicle
{
  constructor(x, y)
  {
    this.pos = createVector(0, random(height))
    this.vel = createVector(random(1), random(1))
    this.acc = createVector(0, 0)
    this.target = createVector(x, y)
    this.maxspeed = 10
    this.maxforce = 1
  }

  show()
  {
    stroke('yellow')
    strokeWeight(5)
    point(this.pos.x, this.pos.y)
  }

  move()
  {
    this.pos.add(this.vel)
    this.vel.add(this.acc)
    this.acc.set(0, 0)
  }

  applyForce(f)
  {
    this.acc.add(f)
  }

  arrive(target)
  {
```

```
    let desired = p5.Vector.sub(target, this.pos)
    let d = desired.mag()
    if (d < 100)
    {
      let mg = map(d, 0, 100, 0, this.maxspeed)
      desired.setMag(mg)
    }
    else
    {
      desired.setMag(this.maxspeed)
    }
    let steer = p5.Vector.sub(desired, this.vel)
    steer.limit(this.maxforce)
    this.applyForce(steer)
  }

  behaviours()
  {
    let arrive = this.arrive(this.target)
    this.applyForce(arrive)
  }
}
```

Notes

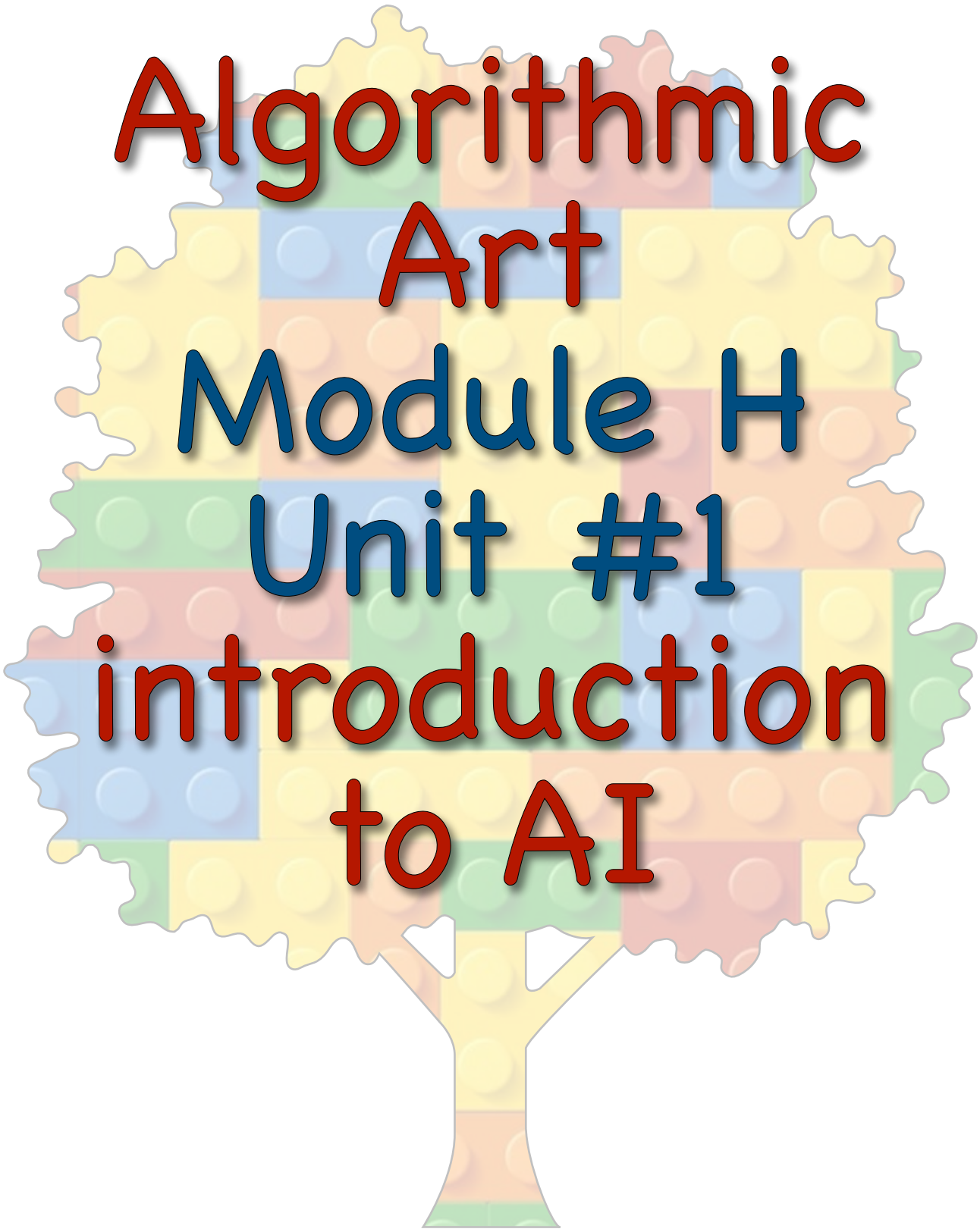
The vehicles (points) should start on the left-hand side edge and rush across the canvas to find their target spot.

Challenges

1. Try other random starting points for effect.
2. Try different max speeds and max force.
3. Add colours.
4. 3D glasses?

Figure G2.15


p5*
File
Edit
Sketch
Help
English
Hello, Eleg
Auto-refresh
Algorithmic Art
Public
p5.js 2.0.5
sketch.js
Saved: 35 seconds ago
Preview
let font
let points
let pt
let vehicle
let vehicles = []
async function setup()
{
createCanvas(600, 400)
font = await loadFont('Roboto-VariableFont_wdth,wght.ttf')
textFont(font)
textAlign(CENTER, CENTER)
textSize(200)
let options = { sampleFactor: 0.1 }
points = font.textToContours('Happy', 300, 200, options)
for (let i = 0; i < points.length; i++)
{
pt = points[i]
for (let j = 0; j < pt.length; j++)
{
vehicle = new Vehicle(pt[j].x, pt[j].y)
Console
Clear
Happy



Algorithmic Art Module H Unit #1 introduction to AI

Module H Unit #1 introduction to AI

Before we jump into using AI to create art, it is worth spending a little bit of time understanding what we are using and where it came from. I have a whole series of workbooks on AI if you are interested after exploring these units.

If you think AI and arts, you may be thinking of Generative AI which uses images it has scraped off the internet to help you create an image from a prompt. That is certainly one use of Ai but that is not what we are going to be doing in the next module.

The other workbooks I mentioned take you step by step through the process of building an AI model (virtually from scratch), adding your own data, and training it. There you will get a deeper understanding of how AI works and what tools you have to use and apply it.

This module is all about using pretrained models which have a specific application. They have already been trained on data. They are not LLMs (such as ChatGPT), there are no prompts. You will be coding it using p5.js through a library called ml5.js. Where the ML is short for Machine Learning and the 5 is a reference to p5.js.

Our AI is going to work through p5.js and will work in the browser. Nothing to download and certainly nothing to pay for (although you can donate). It is a simple line of code in the index.html file as you did with anaglyph.js. So it couldn't be easier in that respect. You don't have to understand the inner workings of AI to use this, but you do need to be able to code, and that is why we are doing it, one small step at a time.

This will open up many possibilities of an interactive approach to Algorithmic Art. Where you go with this is up to you. In the next few pages, I will offer a brief outline of what to expect.



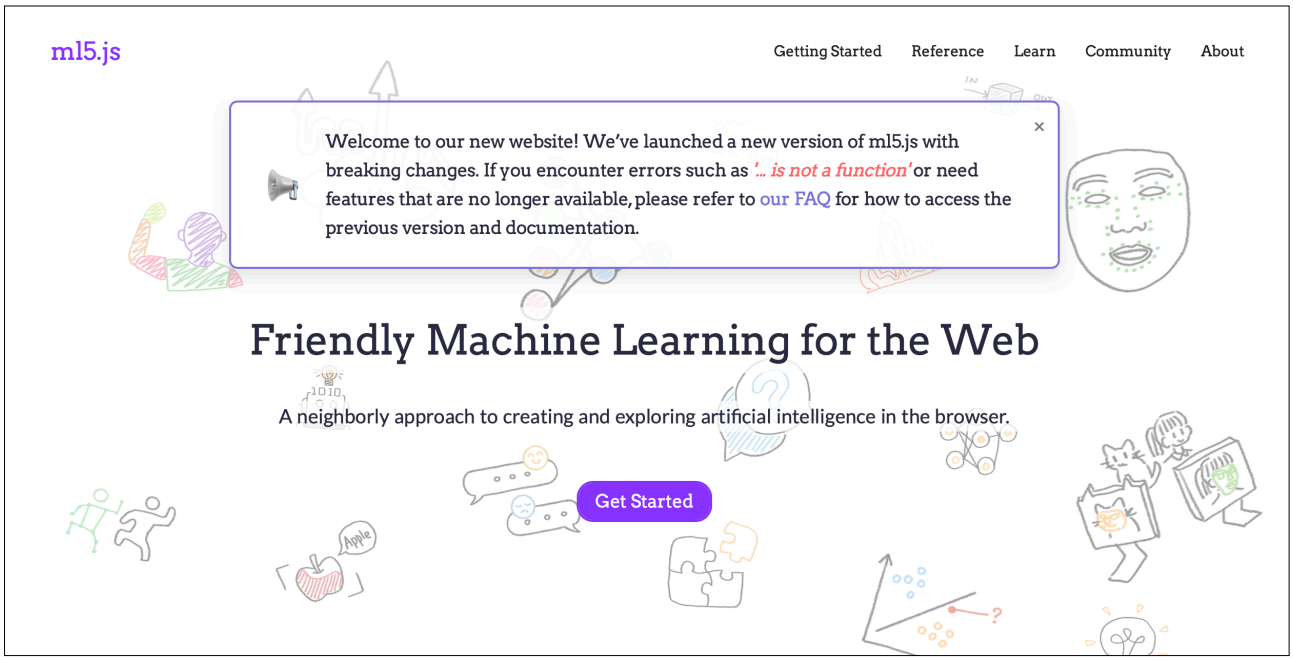
What is ml5.js?

The ml5.js is a library that makes using Machine Learning (AI) accessible to mere mortals like you and me. It is built on TensorFlow.js, which in turn is built on TensorFlow, which is used in Machine Learning with the coding language of Python.

The Processing Foundation has a website where you can look at ml5.js in more detail. The ml5.js library has recently gone through a revision, and so there are, in effect, two websites. I will link you to the new one, which is pretty obvious.

The website is www.ml5js.org.
The home page looks like this (see Fig.1):

Figure 1: home page



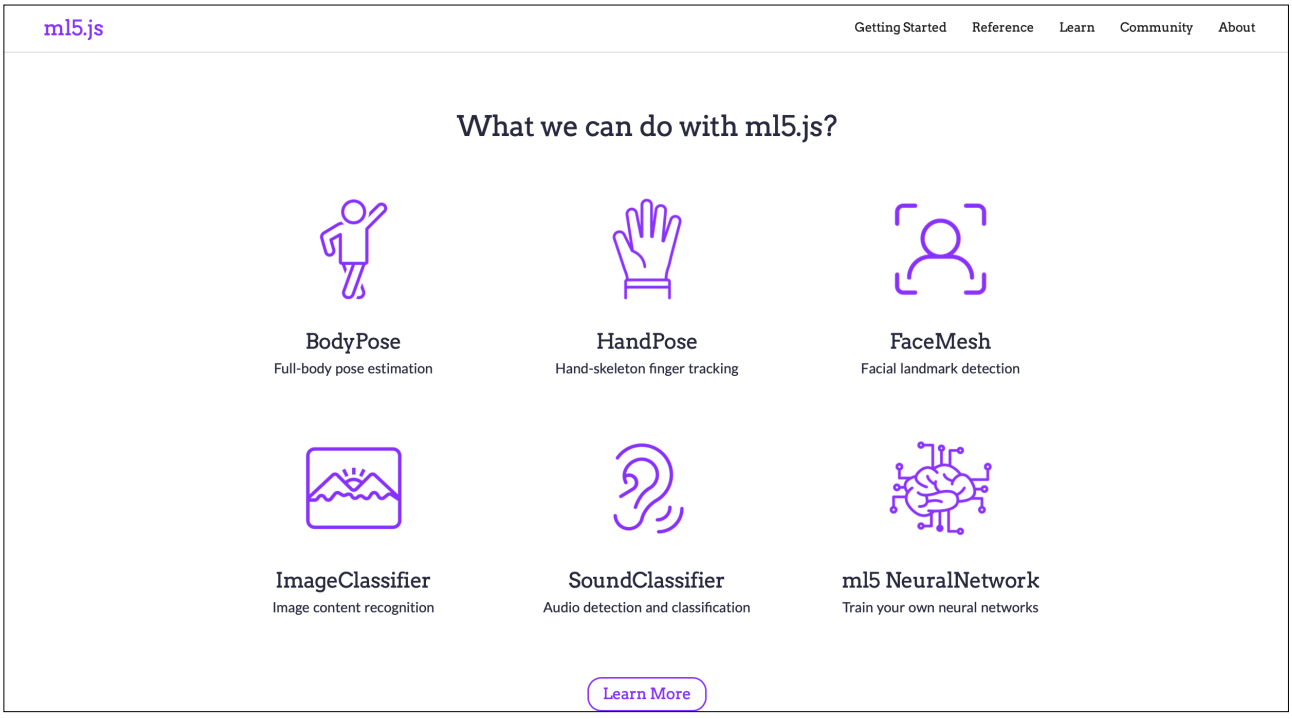
But before you click on the Get Started button, just scroll down a bit and see what is there. Notice it gives a reference to the old version of ml5.js. It is only important if you find some old code that is based on that version, but we won't be at this stage.

As you scroll down, you can see the next window with some graphics (see Fig.2). Here are some examples of the pretrained models I mentioned:

BodyPose
HandPose
FaceMesh
ImageClassifier
SoundClassifier

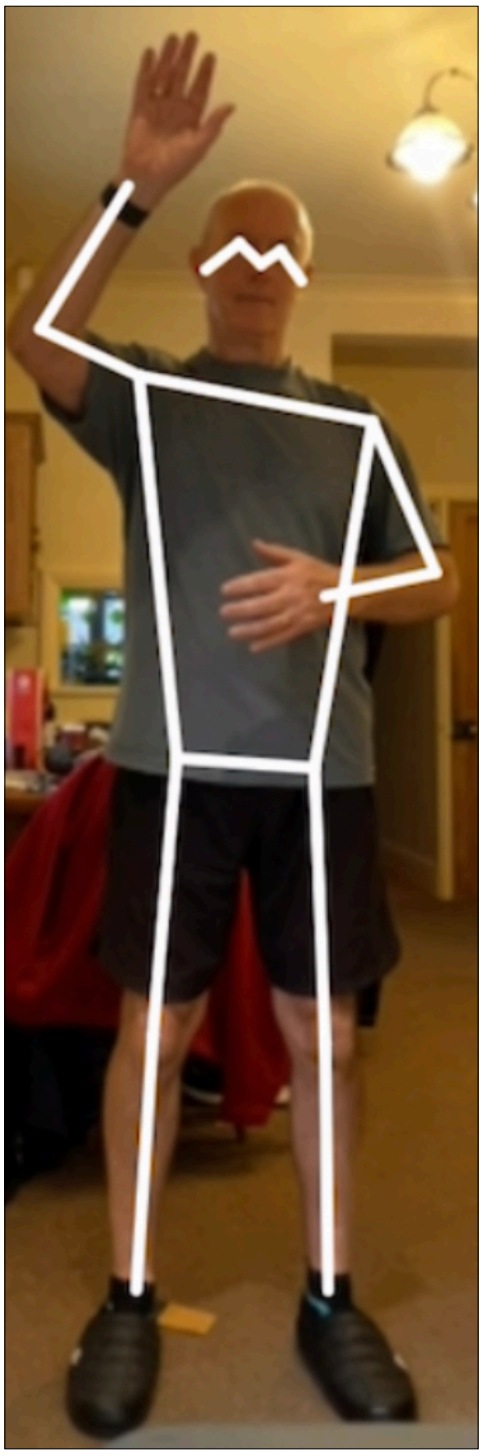
There is a sixth one called ml5 NeuralNetwork, which is where you can build your own model, train it, save it, and use it. If you are interested, then head over to my AI workbooks. I will briefly go through each of these next.

Figure 2: models



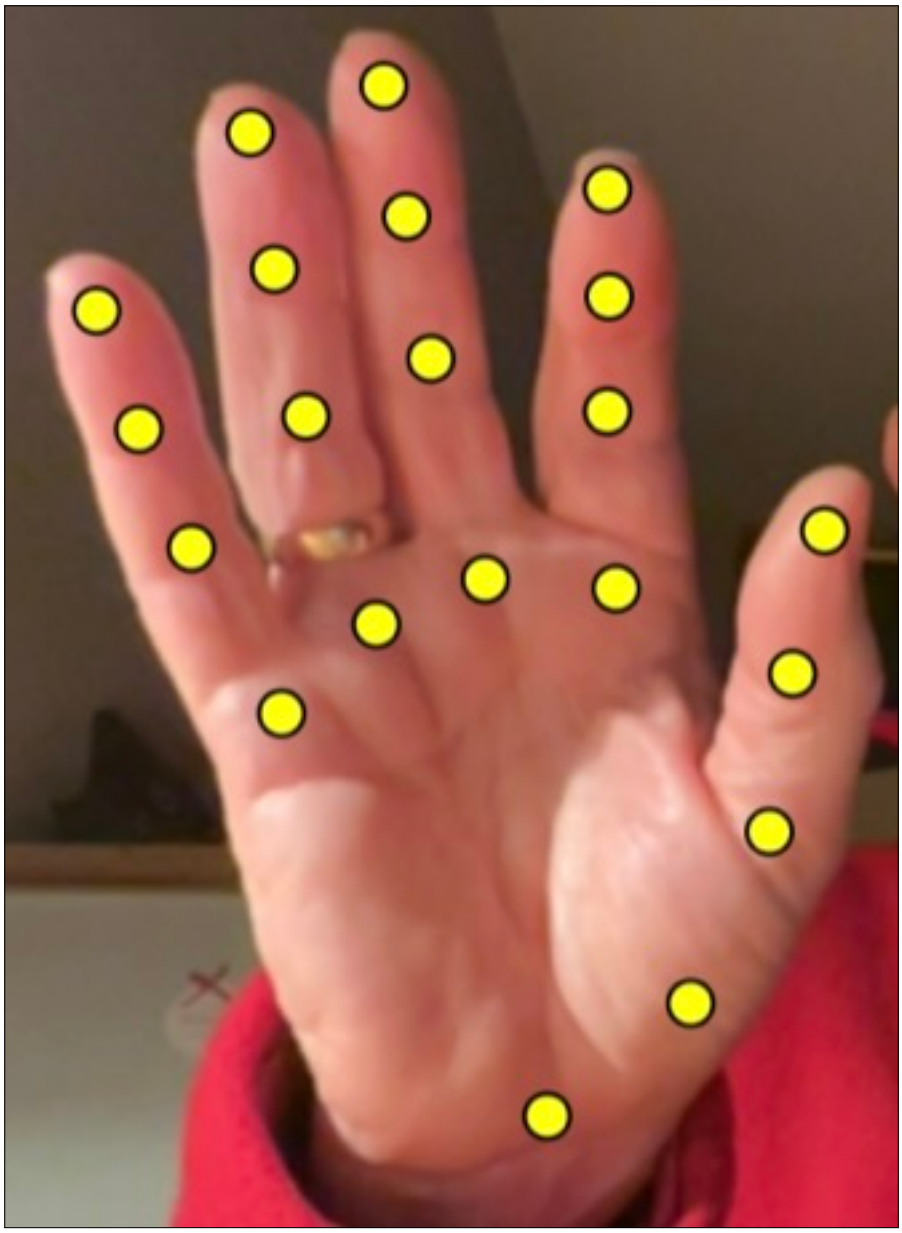
This requires a webcam of some sort pointed at you. When you run this pretrained model, it picks up the key body parts as points, for instance, shoulder, elbow, knee, ankle, and so on. This works in real time, and as you move around, it follows your movement.

Figure 3: BodyPose



Here we have a HandPose pretrained model that maps out the major points on one or both hands. We use this model in our next unit as it is easiest to use sat down in front of your webcam.

Figure 4: HandPose

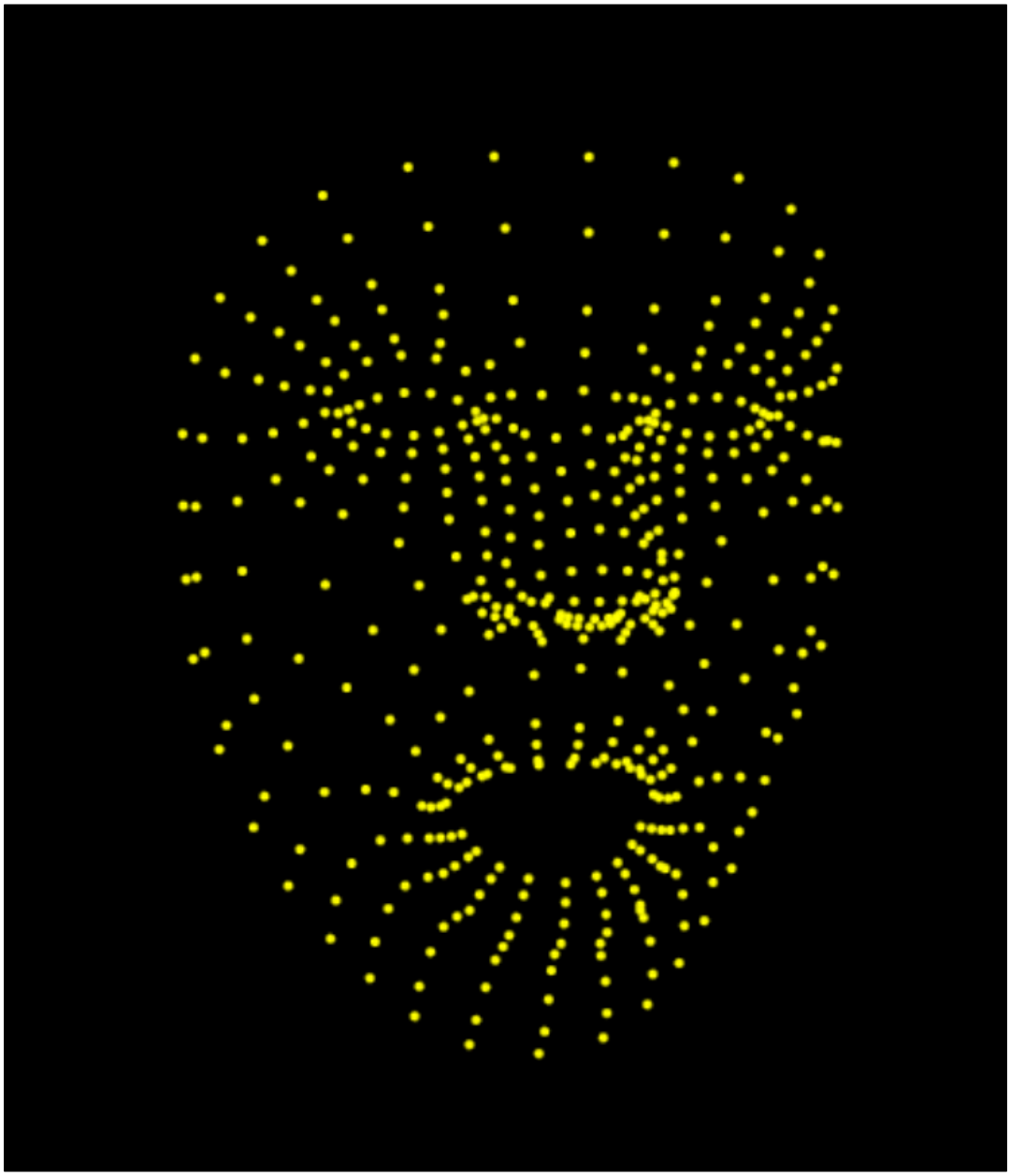




FaceMesh

This model maps lots of points on your face around the eyes, the nose, and mouth. You can use that to add on to your face a mask of your choosing.

Figure 5: FaceMesh



This can take an image or a video and classify what it can see. It gives you the confidence score for its prediction. It can only use data it has trained on, so you might find it provides some funny predictions. Here is the example on the website.

Figure 6: imageClassifier

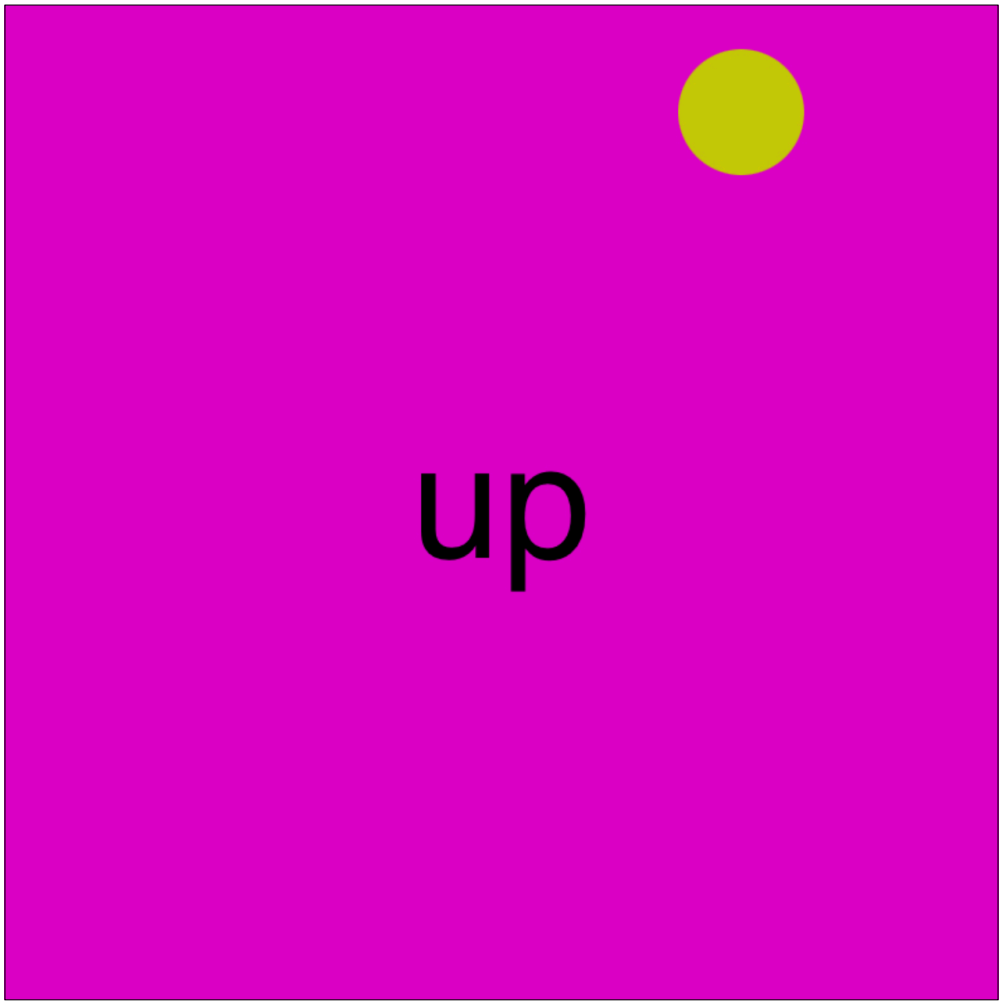


This model is trained on recognising spoken words. There are 18 words, and they are as follows:

Zero, One, Two, Three, Four, Five, Six, Seven, Eight, Nine, Up, Down, Left, Right, Go, Stop, Yes, and No,

You can control or affect some sketch you have written as you speak into the microphone. It only knows those 18 words, but it's something to play around with. In Fig.7 I move a circle around the canvas with my voice, with mixed results.

Figure 7: soundClassifier



This can be anything you have enough data for. In the AI workbooks, there are plenty of examples to work through to give you a feeling for how it works. They are simple examples that don't need a neural network but demonstrate how they work and how you can build one.

You won't be building one in these series of workbooks, but you may come across an idea to use one creatively, so I would recommend working through the AI workbooks as it may give you some ideas.

A neural network is nothing magical; it is simply an algorithm based on some relatively simple maths, much of which is to do with calculus and matrix multiplication. So if you did that at school, you are most of the way there and not far from being able to build your own from scratch.

A quick explanation is that you have inputs; it could be just some data, it could be an image of a dog or a cat. The data passes through the neural network, which has a series of weights (these are random) and then through a series of nodes which collect the data multiplied by the weights, to the output which has the correct result.

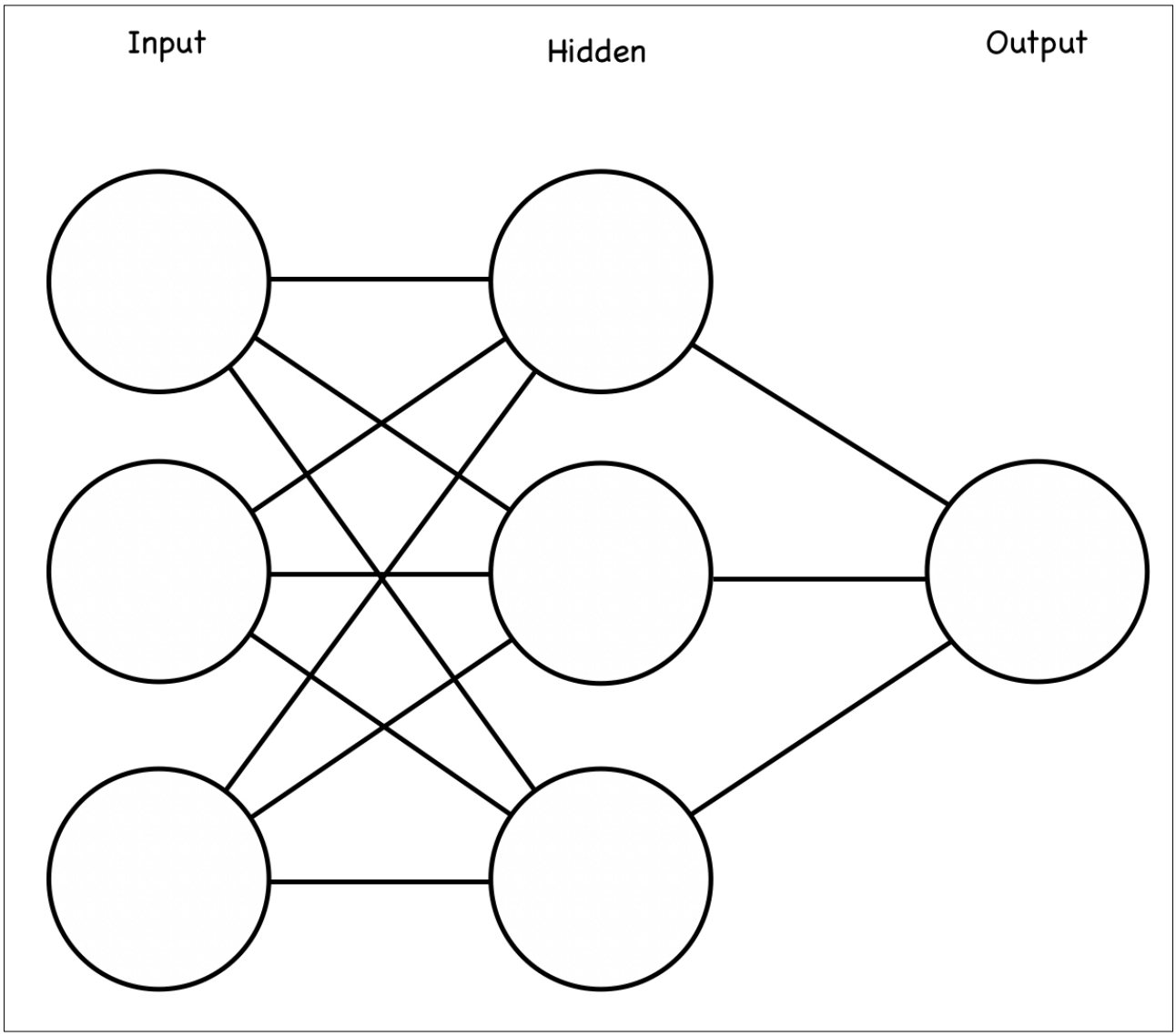
Because the neural network has effectively guessed what the correct result is, there will be an error. This error is passed back through the neural network, and those weights are tweaked slightly, and it has another go at guessing the correct result. It repeats this until there is no or very little error. This is how a model is trained.

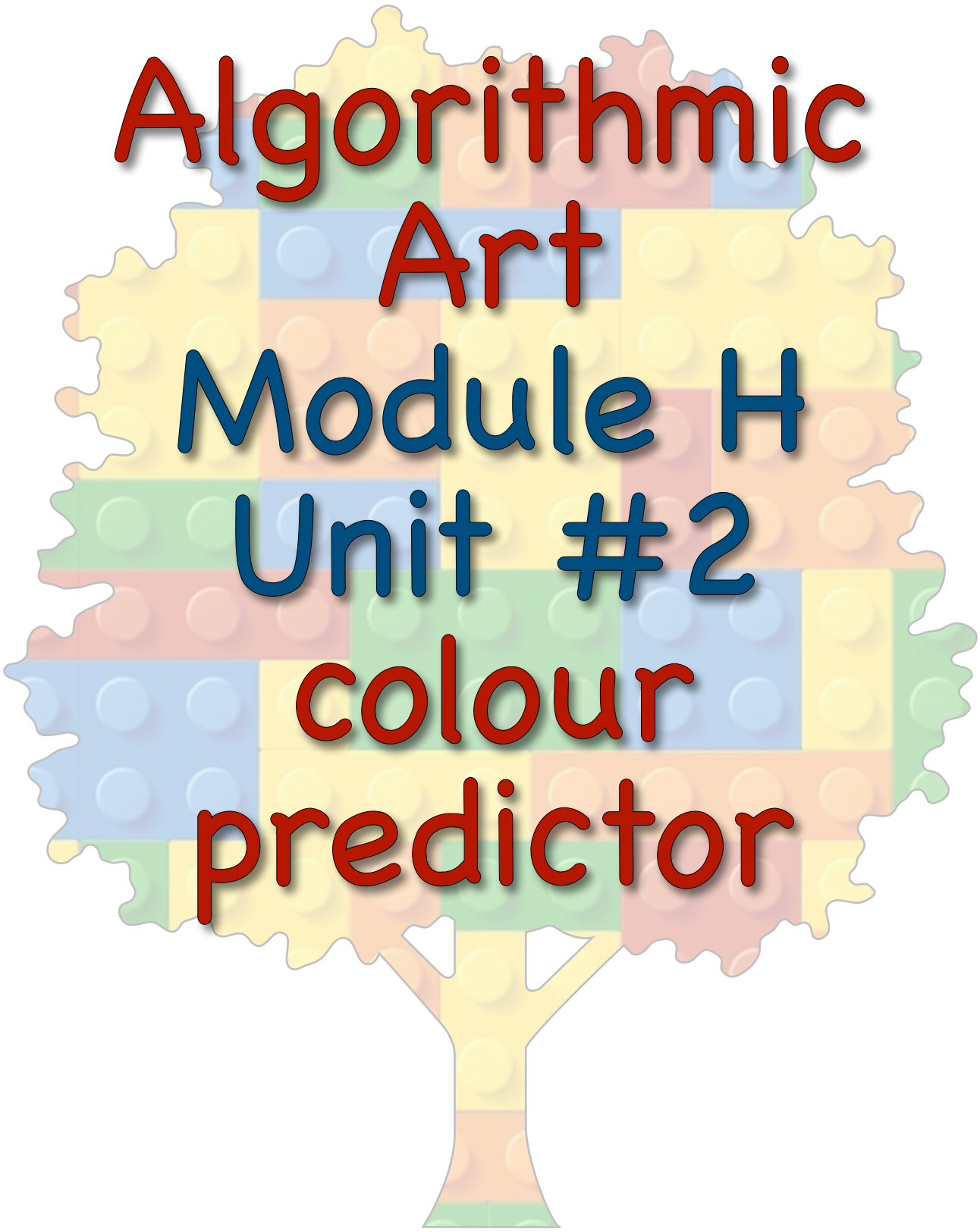
Fig.8 shows a simple diagram of what is going on inside a simple neural network. There are lots of layers and nodes (or neurons), and all it is doing is lots and lots of simple calculations. When you use the Large Language Models such as ChatGPT, they have billions of these nodes (neurons), hence why they are housed in data centres.

We use much smaller ones that we can run in our browser. The simplest is called a perceptron, which is effectively one node (neuron) with, say, two inputs and one output. This is such a vast topic, and for the most part, you do not need to know how it works in the same way you do not need to know how the internal workings of the engine work in your car, but you do need to know when and where to put water, fuel, electricity, or air into the car to keep it moving.

For now, you can just use it. That is complex enough. Enjoy and have a go at seeing what you can create.

Figure 8: diagram of a simple neural network





Algorithmic Art Module H Unit #2 colour predictor



Module H Unit #2: colour predictor

We are going to use ml5.js default architecture to train it on a bunch of colours and get it to predict the colour based on the data given it. We will create this data to train on, and then test the final product which will predict the colour.

The model will try to predict reddish, greenish, and blueish colours. For this, we will provide a small synthetic dataset in the form of a JSON-style file. We will train the neural network on that data and see if it can learn these three classifications for a variety of combinations of the RGB values.

This is, therefore, a classification task as we want it to select, through the built-in softmax function, the most likely colour group. We will use three sliders to change the amount of red, green, and blue.

Softmax just uses some very simple maths to decide which colour is the most likely based on the predictions for all three colours.



The index.html

A reminder to add the ml5.js line of code if you haven't already.

```
<!DOCTYPE html>
<html lang="en"><head>
    <script src="https://cdn.jsdelivr.net/npm/p5@2.2.3/lib/p5.js"></script>
    <script src="https://unpkg.com/ml5@1/dist/ml5.min.js"></script>
    <link rel="stylesheet" type="text/css" href="style.css">
    <meta charset="utf-8">
  </head>
  <body>
    <main>
    </main>
    <script src="sketch.js"></script>
</body></html>
```



Sketch H2.1 creating the data

We are going to create three labels: `red-ish`, `green-ish`, and `blue-ish`. For each of these three labels, we are going to give them the `RGB` numbers that are pretty close. For example, `red-ish` is `(255, 0, 0)`, `(254, 0, 0)`, and `(253, 0, 0)`, and do the same for the `green-ish` and `blue-ish`. This is a json type set of data. For AI to work on this data it needs to be a particular format.

```
let data = [
  { r: 255, g: 0, b: 0, colour: "red-ish" },
  { r: 254, g: 0, b: 0, colour: "red-ish" },
  { r: 253, g: 0, b: 0, colour: "red-ish" },
  { r: 0, g: 255, b: 0, colour: "green-ish" },
  { r: 0, g: 254, b: 0, colour: "green-ish" },
  { r: 0, g: 253, b: 0, colour: "green-ish" },
  { r: 0, g: 0, b: 255, colour: "blue-ish" },
  { r: 0, g: 0, b: 254, colour: "blue-ish" },
  { r: 0, g: 0, b: 253, colour: "blue-ish" },
]

function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(220)
}
```

Notes

This is another small dataset; in reality, you would have a much larger dataset based on people's options (labels), but this is simple enough to be useful.

Challenge

You could add more data.

Code Explanation

`let data = [{r:, g:, b:, colour:},]`	Creating a series of objects in an array called data. These will be the input data

Sketch H2.2 Sliders

We are going to create three sliders to control the colour of the background. We will give it an initial colour of red.

```
let data = [
  { r: 255, g: 0, b: 0, colour: "red-ish" },
  { r: 254, g: 0, b: 0, colour: "red-ish" },
  { r: 253, g: 0, b: 0, colour: "red-ish" },
  { r: 0, g: 255, b: 0, colour: "green-ish" },
  { r: 0, g: 254, b: 0, colour: "green-ish" },
  { r: 0, g: 253, b: 0, colour: "green-ish" },
  { r: 0, g: 0, b: 255, colour: "blue-ish" },
  { r: 0, g: 0, b: 254, colour: "blue-ish" },
  { r: 0, g: 0, b: 253, colour: "blue-ish" },
]


let r
let g
let b
let rSlider
let gSlider
let bSlider

function setup()
{
  createCanvas(400, 400)
  rSlider = createSlider(0, 255, 255).position(10, 20)
  gSlider = createSlider(0, 255, 0).position(10, 40)
  bSlider = createSlider(0, 255, 0).position(10, 60)
}

function draw()
{
  r = rSlider.value()
  g = gSlider.value()
  b = bSlider.value()
  background(r, g, b)
}
```

Notes

The slider has three arguments:
A) the first is the minimum value,
B) the second is the maximum value, and
C) the third is the starting value.
We give the red slider `rSlider` a starting value of `255`, which is the maximum.

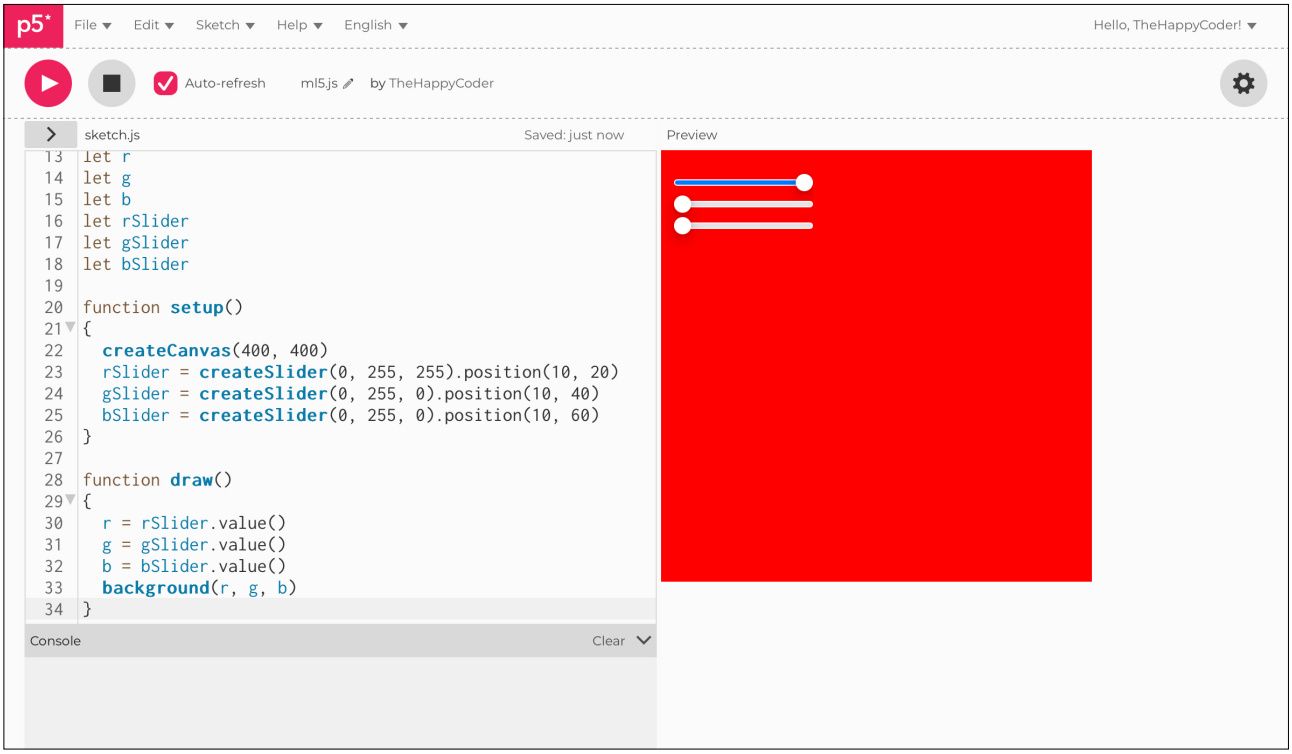
Challenge

Have one of the others start at a different value.

Code Explanation

`createSlider(0, 255, 255)`	The third argument is the value of the slider
`createSlider(...).position(10, 20)`	The position is the x (10 pixels) and y (20 pixels) co-ordinates of the slider

Figure H2.2





Sketch H2.3 adding the neural network

Let's introduce our neural network, we call it nn for short.

```
let data = [
  { r: 255, g: 0, b: 0, colour: "red-ish" },
  { r: 254, g: 0, b: 0, colour: "red-ish" },
  { r: 253, g: 0, b: 0, colour: "red-ish" },
  { r: 0, g: 255, b: 0, colour: "green-ish" },
  { r: 0, g: 254, b: 0, colour: "green-ish" },
  { r: 0, g: 253, b: 0, colour: "green-ish" },
  { r: 0, g: 0, b: 255, colour: "blue-ish" },
  { r: 0, g: 0, b: 254, colour: "blue-ish" },
  { r: 0, g: 0, b: 253, colour: "blue-ish" },
]

let r
let g
let b
let rSlider
let gSlider
let bSlider
let nn

function setup()
{
  createCanvas(400, 400)
  rSlider = createSlider(0, 255, 255).position(10, 20)
  gSlider = createSlider(0, 255, 0).position(10, 40)
  bSlider = createSlider(0, 255, 0).position(10, 60)
  let options = {
    task: 'classification',
    debug: true
  }
  ml5.setBackend("webgl")
  nn = ml5.neuralNetwork(options)
}

function draw()
```

```
{
  r = rSlider.value()
  g = gSlider.value()
  b = bSlider.value()
  background(r, g, b)
}
```

Notes

As you can see, we have a `classification` task and set the `debug` to `true`. We need specify what type of task the model is going to train, it is either a `Classification` task or a `Regression` task. This is explained in far more detail in the series of workbooks on AI. The `debug` shows a loss chart as the model is trained. If you do not want to see the loss chart then set the `debug` to `false`.

Setting the backend means we can specify what hardware is going to run the model. Remember that this is going to do a lot of compute. Well, in our example it isn't because it is a very simple example.

The model we will be using is just the default one, you can specify how many layers, nodes and much more, see the workbooks for more information and explanation.



Sketch H2.4 adding the data

Now, to add the data to the neural network. We cycle through the `data` array, giving the set of inputs and the colour label to a variable called `item`. This holds the value of the `red`, `green`, and `blue` as well as the colour label. However, we only pull out the `r`, `g`, and `b` values. The `colour` values are the outputs for each set of corresponding `r`, `g`, `b` values.

```
let data = [
  { r: 255, g: 0, b: 0, colour: "red-ish" },
  { r: 254, g: 0, b: 0, colour: "red-ish" },
  { r: 253, g: 0, b: 0, colour: "red-ish" },
  { r: 0, g: 255, b: 0, colour: "green-ish" },
  { r: 0, g: 254, b: 0, colour: "green-ish" },
  { r: 0, g: 253, b: 0, colour: "green-ish" },
  { r: 0, g: 0, b: 255, colour: "blue-ish" },
  { r: 0, g: 0, b: 254, colour: "blue-ish" },
  { r: 0, g: 0, b: 253, colour: "blue-ish" },
]

let r
let g
let b
let rSlider
let gSlider
let bSlider
let nn

function setup()
{
  createCanvas(400, 400)
  rSlider = createSlider(0, 255, 255).position(10, 20)
  gSlider = createSlider(0, 255, 0).position(10, 40)
  bSlider = createSlider(0, 255, 0).position(10, 60)
  let options = {
    task: 'classification',
    debug: true
  }
  ml5.setBackend("webgl")
  nn = ml5.neuralNetwork(options)
```

```
  for (let i = 0; i < data.length; i++)
  {
    let item = data[i]
    let inputs = [item.r, item.g, item.b]
    let outputs = [item.colour]
    nn.addData(inputs, outputs)
  }
}

function draw()
{
  r = rSlider.value()
  g = gSlider.value()
  b = bSlider.value()
  background(r, g, b)
}
```

Notes

Using the `item` variable seems a bit redundant, but you are taking them from the complete array. It means you don't accidentally change any values in the dataset. This repeats for the length of the dataset.

Challenge

You could replace `item` with `data[i]` and remove all references to the `item` variable.

Code Explanation

`let item = data[i]`	For each index if the dataset array push it into the item variable
`let inputs = [item.r, item.g, item.b]`	Each set of inputs are added
`let outputs = [item.colour]`	The corresponding output is added
`nn.addData(inputs, outputs)`	Both the inputs and outputs are added to the neural network



Sketch H2.5 normalising the data

Next, we will normalise the data because we don't want very large and very small numbers in the neural network calculations.

```
let data = [
  { r: 255, g: 0, b: 0, colour: "red-ish" },
  { r: 254, g: 0, b: 0, colour: "red-ish" },
  { r: 253, g: 0, b: 0, colour: "red-ish" },
  { r: 0, g: 255, b: 0, colour: "green-ish" },
  { r: 0, g: 254, b: 0, colour: "green-ish" },
  { r: 0, g: 253, b: 0, colour: "green-ish" },
  { r: 0, g: 0, b: 255, colour: "blue-ish" },
  { r: 0, g: 0, b: 254, colour: "blue-ish" },
  { r: 0, g: 0, b: 253, colour: "blue-ish" },
]

let r
let g
let b
let rSlider
let gSlider
let bSlider
let nn

function setup()
{
  createCanvas(400, 400)
  rSlider = createSlider(0, 255, 255).position(10, 20)
  gSlider = createSlider(0, 255, 0).position(10, 40)
  bSlider = createSlider(0, 255, 0).position(10, 60)
  let options = {
    task: 'classification',
    debug: true
  }
  ml5.setBackend("webgl")
  nn = ml5.neuralNetwork(options)
  for (let i = 0; i < data.length; i++)
  {
```

```
    let item = data[i]
    let inputs = [item.r, item.g, item.b]
    let outputs = [item.colour]
    nn.addData(inputs, outputs)
  }
  nn.normalizeData()
}

function draw()
{
  r = rSlider.value()
  g = gSlider.value()
  b = bSlider.value()
  background(r, g, b)
}
```



Sketch H2.6 training the neural network

We will now train our neural network, setting the batch size to 16 and the epochs to 128.

```
let data = [
  { r: 255, g: 0, b: 0, colour: "red-ish" },
  { r: 254, g: 0, b: 0, colour: "red-ish" },
  { r: 253, g: 0, b: 0, colour: "red-ish" },
  { r: 0, g: 255, b: 0, colour: "green-ish" },
  { r: 0, g: 254, b: 0, colour: "green-ish" },
  { r: 0, g: 253, b: 0, colour: "green-ish" },
  { r: 0, g: 0, b: 255, colour: "blue-ish" },
  { r: 0, g: 0, b: 254, colour: "blue-ish" },
  { r: 0, g: 0, b: 253, colour: "blue-ish" },
]

let r
let g
let b
let rSlider
let gSlider
let bSlider
let nn

function setup()
{
  createCanvas(400, 400)
  rSlider = createSlider(0, 255, 255).position(10, 20)
  gSlider = createSlider(0, 255, 0).position(10, 40)
  bSlider = createSlider(0, 255, 0).position(10, 60)
  let options = {
    task: 'classification',
    debug: true
  }
  ml5.setBackend("webgl")
  nn = ml5.neuralNetwork(options)
  for (let i = 0; i < data.length; i++)
  {
```

```
    let item = data[i]
    let inputs = [item.r, item.g, item.b]
    let outputs = [item.colour]
    nn.addData(inputs, outputs)
  }
  nn.normalizeData()
  const trainingOptions = {
    batchSize: 16,
    epochs: 128
  }
  nn.train(trainingOptions, finishedTraining)
}

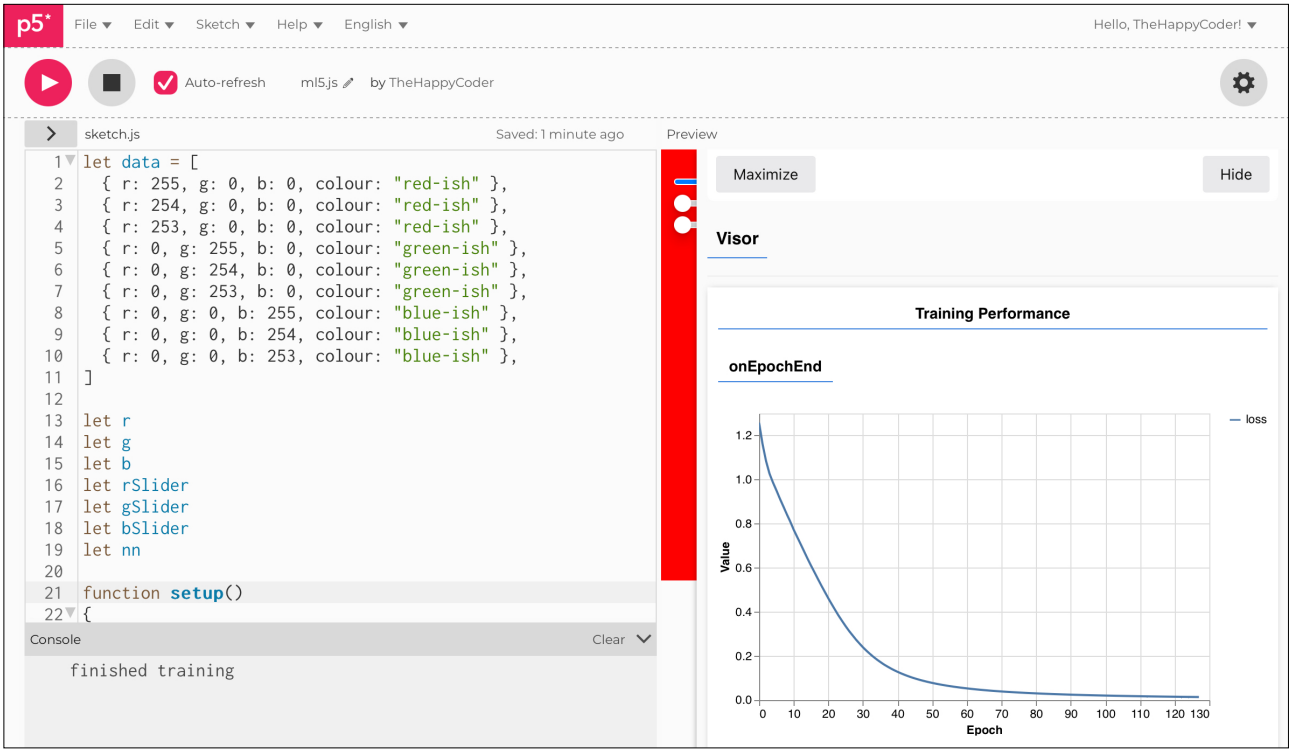
function finishedTraining()
{
  console.log("finished training")
}

function draw()
{
  r = rSlider.value()
  g = gSlider.value()
  b = bSlider.value()
  background(r, g, b)
}
```

Notes

We do get a very nice curved loss function. You can hide the loss curve by clicking on the `Hide` button. Aldo you can hide it permanently by setting debug to `false`.

Figure H2.6


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
ml5.js
by TheHappyCoder
sketch.js
Saved: 1 minute ago
Preview
let data = [
{ r: 255, g: 0, b: 0, colour: "red-ish" },
{ r: 254, g: 0, b: 0, colour: "red-ish" },
{ r: 253, g: 0, b: 0, colour: "red-ish" },
{ r: 0, g: 255, b: 0, colour: "green-ish" },
{ r: 0, g: 254, b: 0, colour: "green-ish" },
{ r: 0, g: 253, b: 0, colour: "green-ish" },
{ r: 0, g: 0, b: 255, colour: "blue-ish" },
{ r: 0, g: 0, b: 254, colour: "blue-ish" },
{ r: 0, g: 0, b: 253, colour: "blue-ish" },
]
let r
let g
let b
let rSlider
let gSlider
let bSlider
let nn
function setup()
{
Console
Clear
finished training
Maximize
Hide
Visor
Training Performance
onEpochEnd
loss
Value
Epoch
0
10
20
30
40
50
60
70
80
90
100
110
120
130
0.0
0.2
0.4
0.6
0.8
1.0
1.2

Sketch H2.7 classifying the colour

We can now classify the colour as red-ish, green-ish, or blue-ish. We create a classify() function where we input the new slider values and classify the output, which is then sent to another function called gotResults(). There we can have a look at the results to understand what data we want. To look inside the data, we console.log the results.

```
let data = [
  { r: 255, g: 0, b: 0, colour: "red-ish" },
  { r: 254, g: 0, b: 0, colour: "red-ish" },
  { r: 253, g: 0, b: 0, colour: "red-ish" },
  { r: 0, g: 255, b: 0, colour: "green-ish" },
  { r: 0, g: 254, b: 0, colour: "green-ish" },
  { r: 0, g: 253, b: 0, colour: "green-ish" },
  { r: 0, g: 0, b: 255, colour: "blue-ish" },
  { r: 0, g: 0, b: 254, colour: "blue-ish" },
  { r: 0, g: 0, b: 253, colour: "blue-ish" },
]

let r
let g
let b
let rSlider
let gSlider
let bSlider
let nn

function setup()
{
  createCanvas(400, 400)
  rSlider = createSlider(0, 255, 255).position(10, 20)
  gSlider = createSlider(0, 255, 0).position(10, 40)
  bSlider = createSlider(0, 255, 0).position(10, 60)
  let options = {
    task: 'classification',
    debug: true
  }
  ml5.setBackend("webgl")
  nn = ml5.neuralNetwork(options)
```

```
  for (let i = 0; i < data.length; i++)
  {
    let item = data[i]
    let inputs = [item.r, item.g, item.b]
    let outputs = [item.colour]
    nn.addData(inputs, outputs)
  }
  nn.normalizeData()
  const trainingOptions = {
    batchSize: 16,
    epochs: 128
  }
  nn.train(trainingOptions, finishedTraining)
}

function finishedTraining()
{
  console.log("finished training")
  classify()
}

function classify()
{
  const input = [r, g, b]
  nn.classify(input, gotResults)
}

function gotResults(results)
{
  console.log(results)
}

function draw()
{
  r = rSlider.value()
  g = gSlider.value()
  b = bSlider.value()
  background(r, g, b)
```

```
}
```

Notes

In the console, you get the following (see image below):

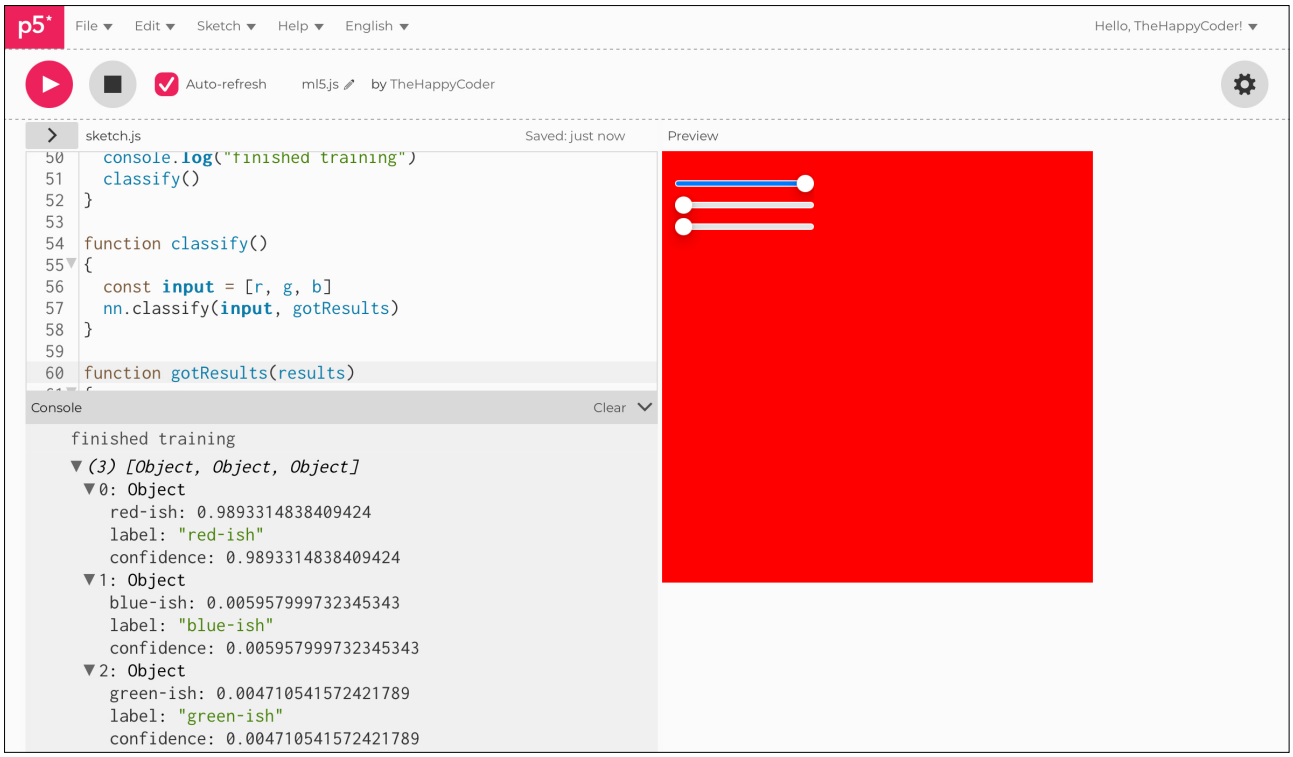
A) An array with three objects
B) Each object has a label and confidence score
C) The highest confidence score is the top one `result[0]`
D) The next highest is `result[1]`
E) The third is `result[2]`
F) We want the top one at `index [0]`

This means we only get the default RGB values; next, we need to get the new values when we move the sliders

Code Explanation

`const input = [r, g, b]`	We use the default settings for the initial slider which is red

Figure H2.7





Sketch H2.8 the best prediction

! Remove the `console.log()` in the `gotResults()` function as we don't need that information now.

Here we will be extracting the top result (`index[0]`) with the highest confidence score and then printing the result as text on the canvas. We initialise the `label` as a string (`let label = " "`). We add the `classify()` function in the `gotResults()` function to check for updates to the slider, so that every time you move the sliders you get an instant update. We only train it once but classify repeatedly. If that is unclear, work through the logic!

! We set the debug to `false` so we don't see the loss curve.

```
let data = [
  { r: 255, g: 0, b: 0, colour: "red-ish" },
  { r: 254, g: 0, b: 0, colour: "red-ish" },
  { r: 253, g: 0, b: 0, colour: "red-ish" },
  { r: 0, g: 255, b: 0, colour: "green-ish" },
  { r: 0, g: 254, b: 0, colour: "green-ish" },
  { r: 0, g: 253, b: 0, colour: "green-ish" },
  { r: 0, g: 0, b: 255, colour: "blue-ish" },
  { r: 0, g: 0, b: 254, colour: "blue-ish" },
  { r: 0, g: 0, b: 253, colour: "blue-ish" },
]

let r
let g
let b
let rSlider
let gSlider
let bSlider
let nn
let label = " "

function setup()
{
  createCanvas(400, 400)
  rSlider = createSlider(0, 255, 255).position(10, 20)
  gSlider = createSlider(0, 255, 0).position(10, 40)
  bSlider = createSlider(0, 255, 0).position(10, 60)
```

```
  let options = {
    task: 'classification',
    debug: false
  }
  ml5.setBackend("webgl")
  nn = ml5.neuralNetwork(options)
  for (let i = 0; i < data.length; i++)
  {
    let item = data[i]
    let inputs = [item.r, item.g, item.b]
    let outputs = [item.colour]
    nn.addData(inputs, outputs)
  }
  nn.normalizeData()
  const trainingOptions = {
    batchSize: 16,
    epochs: 128
  }
  nn.train(trainingOptions, finishedTraining)
}

function finishedTraining()
{
  console.log("finished training")
  classify()
}

function classify()
{
  const input = [r, g, b]
  nn.classify(input, gotResults)
}

function gotResults(results)
{
  // console.log(results)
  label = results[0].label
  classify()
```

```
}

function draw()
{
  r = rSlider.value()
  g = gSlider.value()
  b = bSlider.value()
  background(r, g, b)
  textSize(64)
  text(label, width/4, height/2)
}
```

Notes

You will perhaps notice that the results aren't perfect despite the loss function looking pretty good. We have only provided a small amount of data, but also we haven't offered any change to most of the other hyperparameters, which is something we could do.

Challenges

1. Change the number of hidden layers and nodes,
2. Change the batch size,
3. Change the learning rate
4. Create a JSON. File for all the named colours

Code Explanation

`let label = " "`	Define the label as a string
`label = results[0].label`	Pulling the label from the classified results

Figure H2.8a

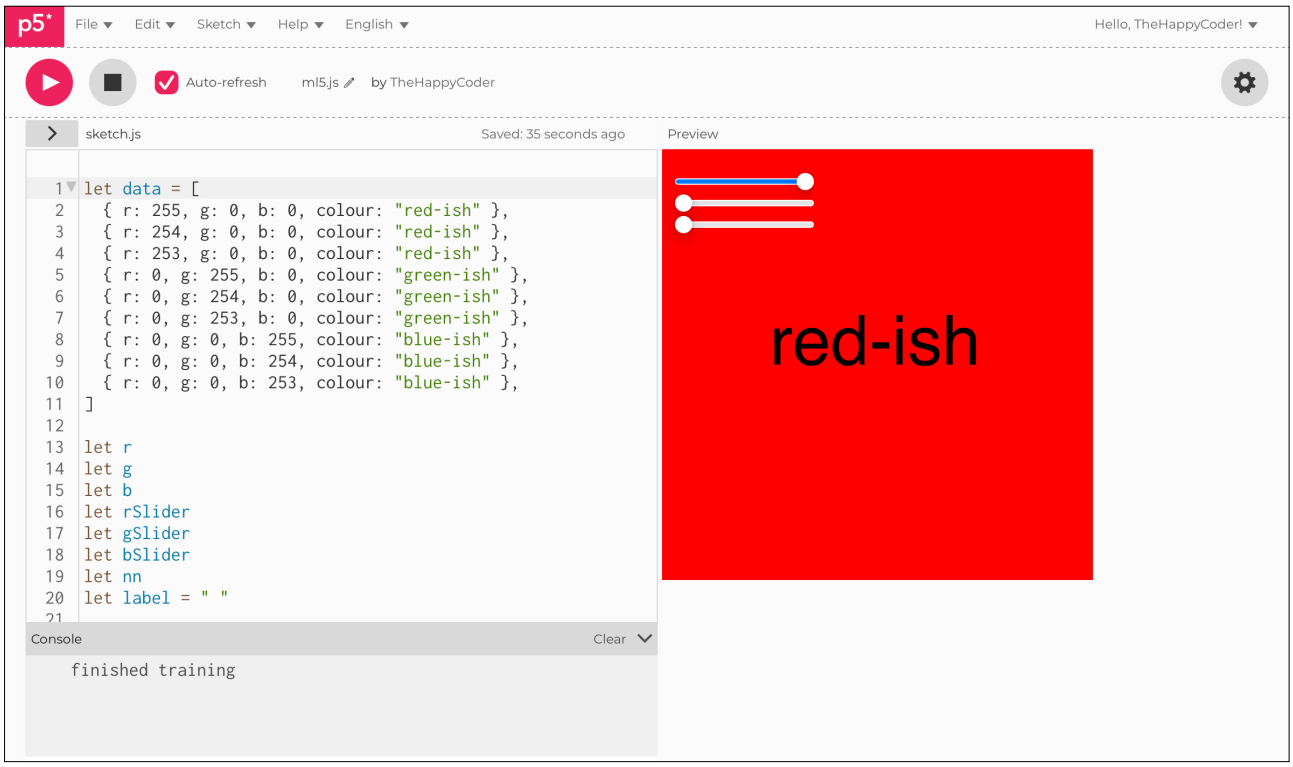


Figure H2.8b

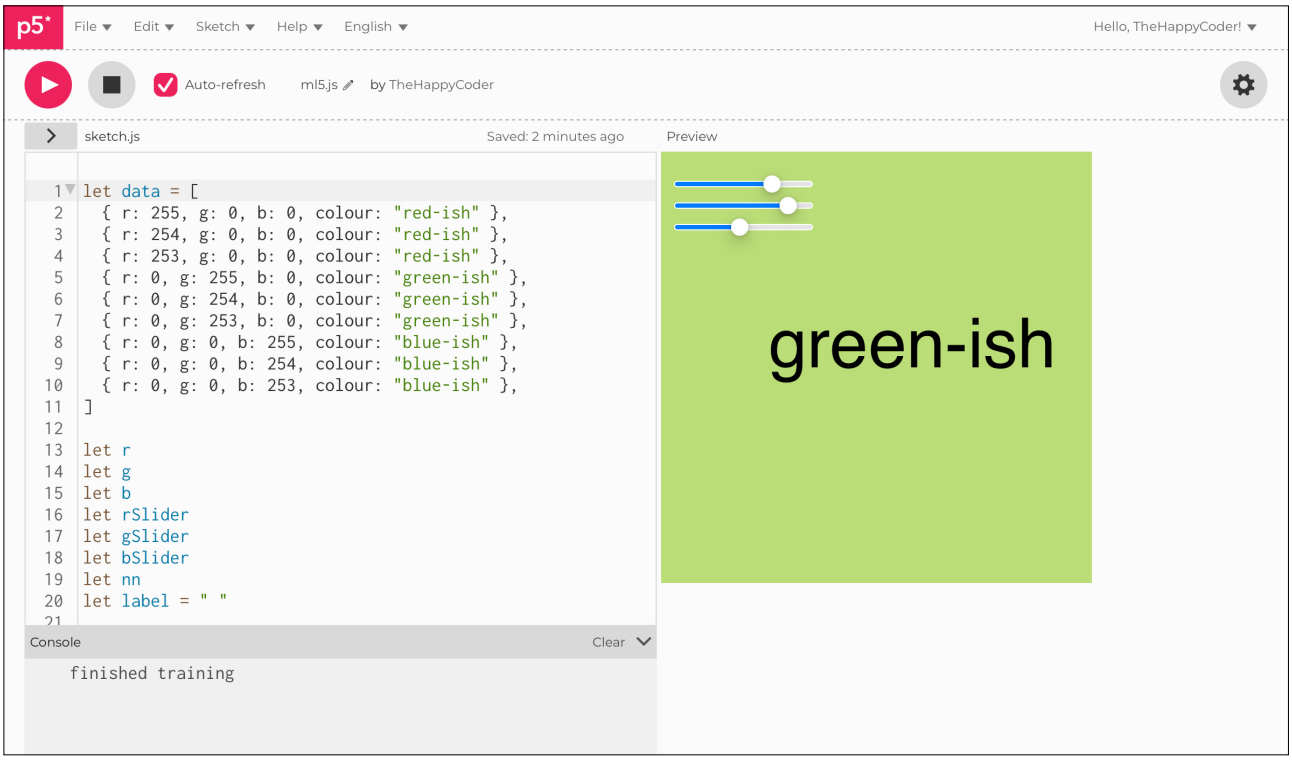
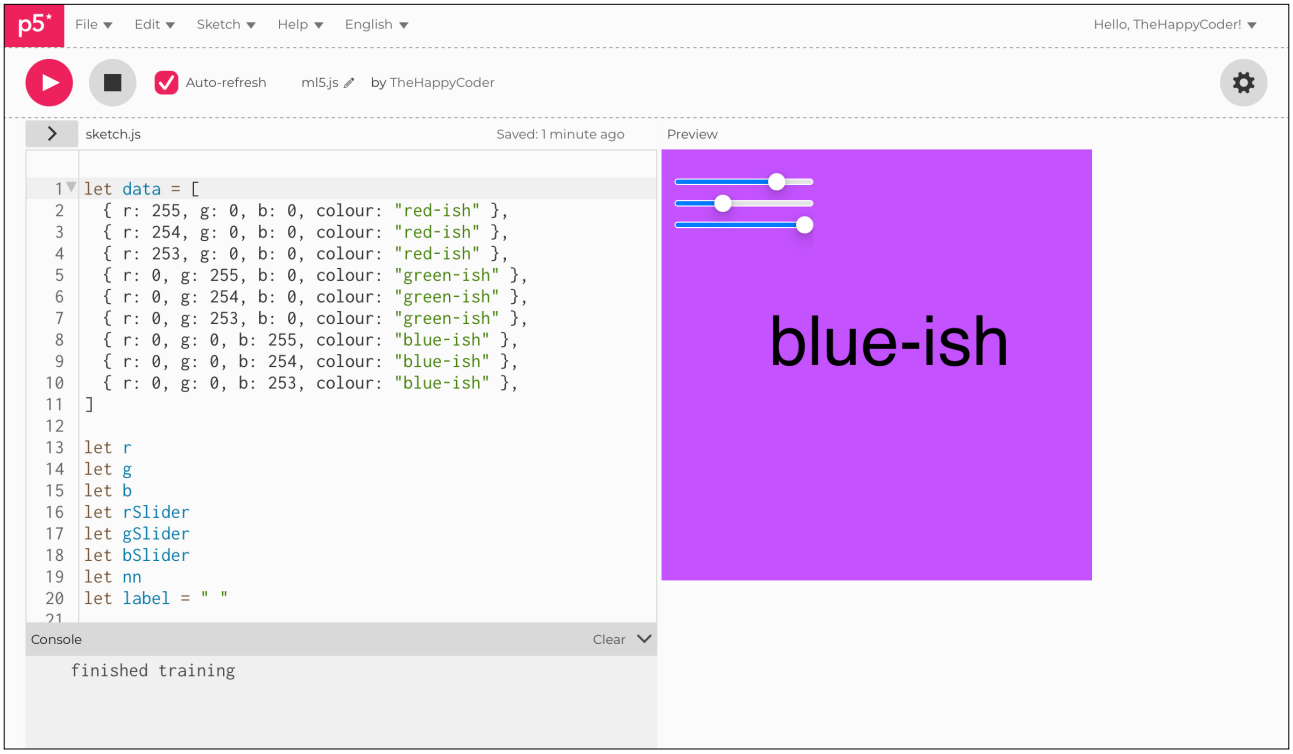


Figure H2.8c


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
ml5.js
by TheHappyCoder
sketch.js
Saved: 1 minute ago
Preview
let data = [
{ r: 255, g: 0, b: 0, colour: "red-ish" },
{ r: 254, g: 0, b: 0, colour: "red-ish" },
{ r: 253, g: 0, b: 0, colour: "red-ish" },
{ r: 0, g: 255, b: 0, colour: "green-ish" },
{ r: 0, g: 254, b: 0, colour: "green-ish" },
{ r: 0, g: 253, b: 0, colour: "green-ish" },
{ r: 0, g: 0, b: 255, colour: "blue-ish" },
{ r: 0, g: 0, b: 254, colour: "blue-ish" },
{ r: 0, g: 0, b: 253, colour: "blue-ish" },
]
let r
let g
let b
let rSlider
let gSlider
let bSlider
let nn
let label = " "
blue-ish
Console
Clear
finished training