Dynamic Identities

National Arboretum Canberra

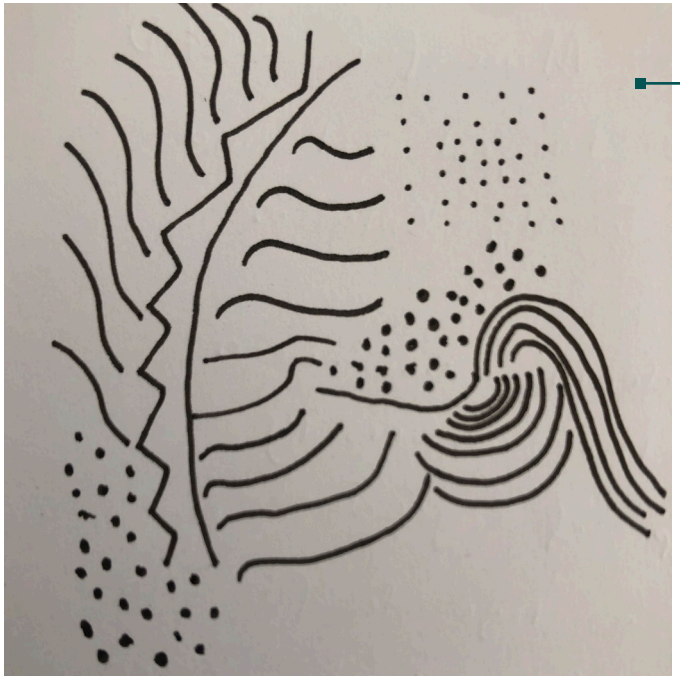
Upon beginning this task, we set out to create a dynamic identity for the National Arboretum Canberra, that reflects its status of one of the world's largest living collections of rare, endangered and significant trees, an important site for botanical arks, recreation and education and one of the most significant works of Australian landscape architecture for urban biodiversity.

The National Arboretum features 94 forests of rare, endangered and symbolic trees from around Australia and the world, with over 44,000 trees growing from 100 countries across the 250 hectare site. The trees within the arboretum forests were selected from the International Union for Conservation of Nature's Red list of Threatened Species. However, only trees that could manage Canberra's climate were planted. Criterion also included striking seasonal colour, whether the tree was a national tree and provided habitat for native wildlife. The intent behind the establishment of the arboretum was to symbolise the local Canberra community's process of healing and recovery from the upheaval and grief of the catastrophic 2003 bushfires. Eventually, the site will house 104 forests. After considering these important aspects of the arboretum's identity, we focussed upon creating a dynamic visual identity which spoke to country, regeneration and seasonality. We aimed to establish a connection between one's conciousness and the land through this. The images above helped to inform the basis of our design, as we were intrigued by the patterns the forrests of the arboretum began to form in the overall landscape.

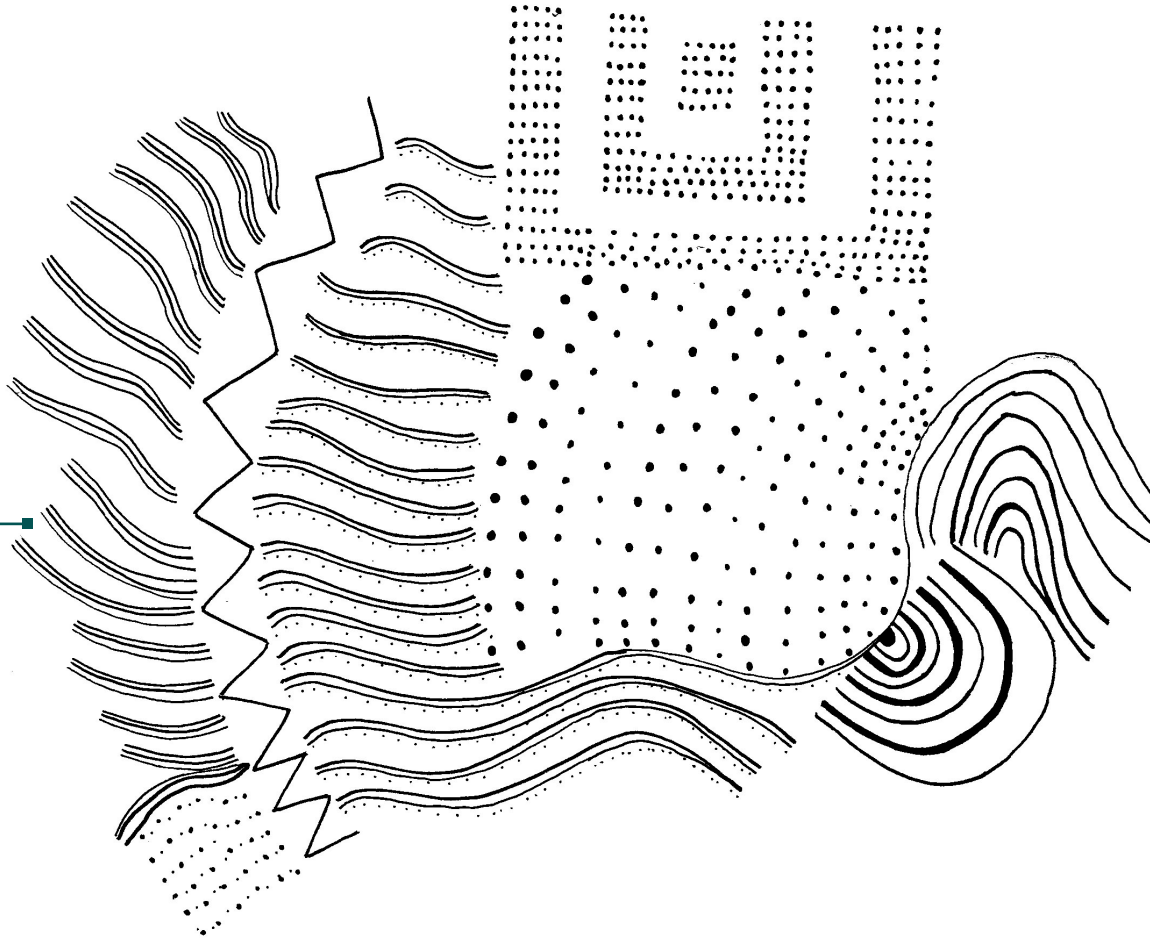
Our Group:
Kyle: Coding, Code Planning, mock-ups and research
Ahmaz: Coding, implementing the p5.svg.js library to manipulate SVGs and research
Yasmin: Coding, stylistic choices, research, hand generated sketches and mock-ups.

What our group found helpful was having a version of the code open while on group chat and talking about it as we worked through the code collectively.

Kyle, Yasmin & Ahmaz



Initial sketch



The revised sketch - this was hand drawn and later adjusted in Adobe Illustrator. Before bringing the sketch into p5.js, we initially converted it into a png file. This compromised the resolution so we later converted it into a svg file.

We have utilised this sketch to extract individual elements which we have used in our id to create dynamism. The various curves, zigzag line and dots are key elements extracted using the aerial sketch above form part of our dynamic ID design.

Stylistic decisions

As the arboretum is a significant cultural, educational and government funded institution visited by a broad spectrum of people of all ages, including tourists, school groups, families, horticulturalists and conservationists, we established there was a need for a design that is playful but legible across all generations. A seasonal colour palette has been combined with hand generated marks that act as a recording of the landscape in it's earlier days. Reducing the landscape of the arboretum to a series of expressive marks also produces imagery that is akin to the fissures found on bark that come with age. Code adds variation to the way this landscape can be realised.

As most trees in the arboretum were selected because of their striking seasonal colour, we decided to reflect this in our design through our seasonal colour palette. Colours were placed into arrays as autumn, winter, spring and summer colours based on Canberra's climate. The code checks which month it is according to the users location and changes the colour respectively. The summer palette is perhaps the most playful of all to appeal to the influx of visitors of all ages the arboretum experiences at this time.

Seasonal colour palette

National Arboretum Canberra

Title font: Latin Modern Bold

Parana Pine
Araucaria angustifolia
average lifespan: **500 years**
average height: **35m**
conservation status:
critically endangered

Signage fonts: Futura Medium, Italic and Heavy

P5.js implementation

In implementing our dynamic ID idea, we began experimenting using white PNG images of the different geometric elements with a transparent background, on p5.js. Using the tint function in p5.js, we were able to get our Dynamic ID operational using PNGs. However, we were constrained by the limited resolution it produced. Although in principle, we could use extremely high resolution PNGs in our code, the larger file sizes meant that processing was poor in p5. To overcome this issue, we have used the p5.svg.js library (https://github.com/zenozeng/p5.js-svg), which provides an SVG runtime for p5.js. Using this SVG library were able to successfully manipulate SVG images of the aerial geometry (created using Adobe Illustrator) in p5.js. Having the ability to manipulate SVGs meant that we could produce outcomes with excellent resolution. In general, processing SVGs were much faster in p5 compared to using PNGs. Since the p5.svg.js library is only compatible with older versions of p5.js (we have used v0.4.13), some of the features available in the latest version of p5.js are not accessible.

Our sketches can be found at:
https://editor.p5js.org/yasroussidis/sketches/3V2GmZw1E
https://editor.p5js.org/kylerobertson98/sketches/yve-wpy6j
https://editor.p5js.org/yasroussidis/sketches/Pdnwginzs

```
var svg, path;
var element_id = 0; // tracks elements in the SVG canvas
//every time the image function is called the number of paths in the SVG canvas increments by one. To ensure that the correct svg
image is modified in terms of colour and position, we must refer to the correct element in the canvas.
var colorautumn = ["#b54a43","#a75d67", "#e86800","#a2242f"]; //array
var colorwinter = ["#adc3b7", "#5e7878", "#035453", "#194b46"];
var colorspring = ["#aedb9f", "#91a658", "#8d9440", "#88b04b"];
var colorsummer = ["#f4e87a", "#cce00b", "#dbbfdd", "#f99584"];
var colours;
```

This code sets variables that will be called throughout the program. It also sets the arrays for the colour that will be used in the program.

P5.js implementation

```
function preload() {
    // Load outer wavy lines
  wavyLO1 = loadSVG('Wavy Line Outside/Grouprow1.svg')
  wavyLO2 = loadSVG('Wavy Line Outside/Grouprow2.svg')
  wavyLO3 = loadSVG('Wavy Line Outside/Grouprow3.svg')
  wavyLO4 = loadSVG('Wavy Line Outside/Grouprow4.svg')
  wavyLO5 = loadSVG('Wavy Line Outside/Grouprow5.svg')

  // Load left group dots
  leftGDot = loadSVG('Left Group Dots/leftGDot.svg')

  // Load zigzag lines
  zizzyTop = loadSVG('Zizzy Lines/Top.svg')
  zizzyMiddle = loadSVG('Zizzy Lines/Middle.svg')
  zizzyBottom = loadSVG('Zizzy Lines/Bottom.svg')

  // Load wavy dotted lines
  wavyLD1 = loadSVG('Wavy Lines Dot/1st.svg')
  wavyLD2 = loadSVG('Wavy Lines Dot/2nd.svg')
  wavyLD3 = loadSVG('Wavy Lines Dot/3rd.svg')
  wavyLD4 = loadSVG('Wavy Lines Dot/4th.svg')
  wavyLD5 = loadSVG('Wavy Lines Dot/5th.svg')

  // Load rectangular array of dots
  CentreDot = loadSVG('Top Org Dots/centregroup.svg');
  middle2 = loadSVG('Top Org Dots/2ndmiddle.svg')
  left2 = loadSVG('Top Org Dots/2ndleft.svg')
  right2 = loadSVG('Top Org Dots/2ndright.svg')
  middle3 = loadSVG('Top Org Dots/3rdmiddle.svg')
```

This section of code loads all of the SVGS and fonts that will be used throughout the code. We grouped the loading of SVGs to their group so other coders can easily see what belong with what. We also used similar naming conventions for the same reason. We used the LoadSVG function from p5.svg.js which loads an SVG and you can assign it a variable name to call in the code.

```
function setup() {
  createCanvas(640, 360,SVG);
  wavyLOArray = [wavyLO1, wavyLO2, wavyLO3, wavyLO4, wavyLO5];
  wavyLOArraySize = [[65,76],[82,101],[84,112],[84,55],[73,55]];

  leftGDotArray = [leftGDot];
  leftGDotSize = [73,65];

  zizzyArray = [zizzyTop, zizzyMiddle, zizzyBottom];
  zizzyArraySize = [[58,114],[38,130],[64,96]];


  wavyLDArray = [wavyLD1,wavyLD2,wavyLD3,wavyLD4,wavyLD5];
  wavyLDArraySize = [[64,73],[101,64],[100,59],[102,47],[111,54]];


  topOrgDot = [CentreDot,left2,middle2,right2,left3,middle3,right3]
  topOrgDotSize = [[27,30],[20,55],[92,25],[24,60],[27,94],[190,28],
[32,93]];

  middleDots = [row1,row2,row3,row4]
  middleDotsSize = [[90,179],[109,171],[108,173],[69,149]];

  WavyLLOArray = [WavyLLO1,WavyLLO2,WavyLLO3,WavyLLO4];
  WavyLLOArraySize = [[401,164],[117,81],[81,95],[55,84]];

  groupArcArray = [GA1,GA2,GA3,GA4]
  groupArcArraySize = [[28,30],[53,54],[87,89],[109,109]];
```

This code create the array's that hold all of the SVG variable name, so you can loop through the array and place each SVG on the screen. The second the array ending in size holds the size of the SVG in the same position but in the other array. It is an array that holds tuples instead of a single value.

P5.js implementation

```
119   switch(month()){
120     case 1:
121       colours = colorsummer; //Changing the var colours to the
    associated colour array
122       break; //leaves the switch cases so code doesn't stack.
123     case 2:
124       colours = colorsummer;
125       break;
126     case 3:
127       colours = colorautumn;
128       break;
129     case 4:
130       colours = colorautumn;
131       break;
132     case 5:
133       colours = colorautumn;
134       break;
135     case 6:
136       colours = colorwinter;
137       break;
138     case 7:
139       colours = colorwinter;
140       break;
141     case 8:
142       colours = colorwinter;
143       break;
144     case 9:
145       colours = colorspring;
146       break;
```

Here is a switch case statement used to set the colour palette according to which season Canberra is in. The switch statement works by looking at condition in the bracket, in this case, the month() function. Then looks at the cases below and if it finds a match it runs the code in the case. We used a switch statement because We think they look cleaner than 4 if statements with a bunch of conditions, We also think they are easier to follow.

```
function draw() {
  // setting background colour
  background(250);
  push(); //Saving the screen
  //Moving and scaling the drawSVG which is seen as a large image/ it does it to each element individually in the function.
  scale(0.7)
  translate(300,75)
  //Bigger Map
  drawSVG();

  pop(); // reset the screen

  push();
  //Again moves and scales
  scale(0.4)
  translate(150,850)
  //Smaller Map
  drawSVG();

  pop();

  strokeWeight(5)
  noFill();
  //Large Rectangle
  rect(275,125,300,250)
  fill(250)
  noStroke();
  //Set the rects around the map, so parts of the map that don't fill in the shell are not seen outside the shell
  //this is done by making four rectangles one each side, and matching the colour to the background.
  rect(275,73,300,50);
  rect(275,378,300,50);
  rect(253,125,20,250);
  rect(578,75,100,350);


  fill(0);
  textSize(50);
  textFont(myFont);
textAlign(RIGHT);  text("National\nArboretum\nCanberra",250,230);

  //Smaller Rectangle
  //Same as above just for the smaller rectangle now.
  stroke(1);//had to called stroke otherwise the contain rectangle would be strokeless
  noFill();
  rect(125,375,150,125)
  fill(250);
  noStroke();//Set strokeless for the covering rectangles so you cannot see the edges, so it blends in better.
  rect(125,358,148,15)
  rect(75,503,250,50)
  rect(73,375,50,128)
  rect(278,378,50,128)
```

The code here is the function that draws everything on the screen, all the rectangle creation are there to contain the map, so it doesn't spill out of the box. The scale and translate function are used to set the location of each random maps. The use of the push and pops are to reset the screen so everything is in the right location.

Kyle, Yasmin & Ahmaz

P5.js implementation

```
function drawSVG(){
 /* These for loop function are here to place each svg/image on the canvas in a random way, the reason for this is that it is the
 most efficient way to place numerous images/svg down by recursing through a array of variables of the svgs/images.
 */
  for ( i = 0; i<wavyLOArray.length;i++){
    /*Have to this form of randomisation because random() wouldn't work on the array for some unknown reason.
    This works by picking a random number below 4 or the length of the array and then floors it thus rounding it down to a whole number.
    Becuase picking from array only works on whole numbers.
    */
    colourRand = random(colours.length);
    colourRand2 = Math.floor(colourRand);
    rand = Math.floor(Math.random() * 25) //Math.random works by picking a random decimal between 0 & 1, so it multiple by 25
    // giving me a random number between 0 and 25. Then I floor to make it a whole number otherwise it would vary by a small amount
   //each time. It also makes it a natural number.
    rand2 = Math.floor(Math.random() * 200)
      // extract width and height of the geometrical element
   size = wavyLOArraySize[i];
   // display image at random location
   image(wavyLOArray[i],100 - (rand), 100 + (rand2),size[0],size[1]);
   // returns an array of SVG elements of current SVG graphics       matching given selector
   path = querySVG('path')[element_id]
    // change stroke width to zero
   path.attribute('stroke-width',0);
   // change colour
   path.attribute('fill',color(colours[colourRand2]));//Then uses the number to pick the position in the array, thus
   //being a different colour each time due to the position being a random number.
   // increment element id
   element_id++;

  }
  // for the remaining for loops the same approach as above is taken
  for (i =0; i <zizzyArray.length;i++){
    //tint('white');
    colourRand = random(colours.length);
    colourRand2 = Math.floor(colourRand);
    rand = Math.floor(Math.random() * 30) //(width/20)
    rand2 = Math.floor(Math.random() * 250) //(width/2.4)
    // extract width and height of the geometrical element
    size = zizzyArraySize[i]; //(width/4)
    image(zizzyArray[i], 150 -(rand),100 + (rand2),size[0],size[1]);
    path = querySVG('path')[element_id]
    path.attribute('stroke-width',0);
    path.attribute('fill',color(colours[colourRand2]));
    element_id++;

  }

  for (i = 0; i<wavyLDArray.length; i ++){
    colourRand = random(colours.length);
    colourRand2 = Math.floor(colourRand);
    rand = Math.floor(Math.random() * 55)
    rand2 = Math.floor(Math.random() * 250)
    // extract width and height of the geometrical element
    size = wavyLDArraySize[i];
    image(wavyLDArray[i], 200-(rand),110+(rand2),size[0],size[1]);
    path = querySVG('path')[element_id]
    path.attribute('stroke-width',0);
    path.attribute('fill',color(colours[colourRand2]));
    element_id++;
  }

  for (i = 0; i <topOrgDot.length; i ++){
    colourRand = random(colours.length);
    colourRand2 = Math.floor(colourRand);
    cRand = Math.floor(Math.random() * 50)
    cRand2 = Math.floor(Math.random() * 50)
```

This code is about half of the drawSVG function, but the remaining code follow the same code structure just with different variable names, so we will only be talking about this part. It works by recursing through a for loop the length of the array and places each SVG on the canvas at random position in between set values. This is achieved by picking a random number than adding or subtracting it from a starting position.

P5.js implementation

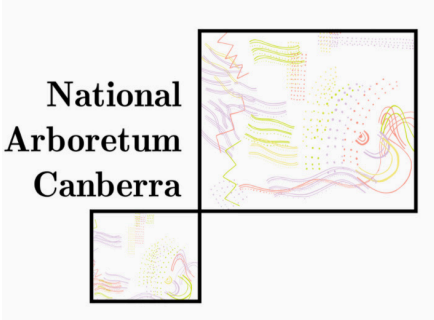
```
}
//no loop on these as there is only one each in the design/map.
colourRand = random(colours.length);
colourRand2 = Math.floor(colourRand);
image(leftGDot,random(100,150),random(380,400),leftGDotSize[0],leftGDotSize[1]);
path = querySVG('path')[element_id]
path.attribute('stroke-width',0);
path.attribute('fill',color(colours[colourRand2]));
element_id++;

colourRand = random(colours.length);
colourRand2 = Math.floor(colourRand);
image(LLWavyLD,random(170,250),random(340,400),LLWavyLDSize[0],LLWavyLDSize[1]);
path = querySVG('path')[element_id]
path.attribute('stroke-width',0);
path.attribute('fill',color(colours[colourRand2]));
```

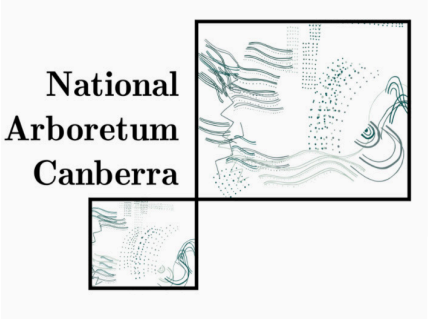
The only code that is different in the drawSVG function is this part. The only difference is that there is no for loop because there is only one of each element/SVG in the map design/program.

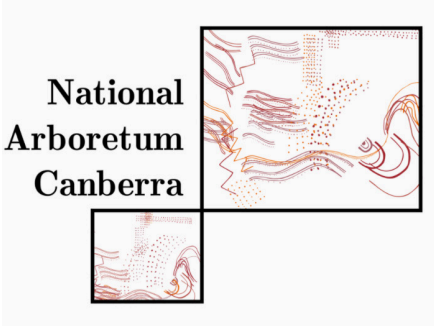
Outputs

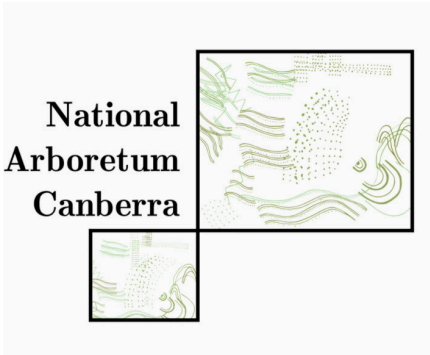
The key creative outcome is the dynamic logo generator. These logos could then be utilised in social media pages such as Facebook, Instagram and twitter. The logo could also be utilised the National Arboretum website. Illustrations of how these dynamic logos could be utilised is provided in the following pages. The aerial geometric elements that form part of this logo could also be used to produce banners. The patterns produces have also been applied to signage and clothing.

Seasonal logo variations:

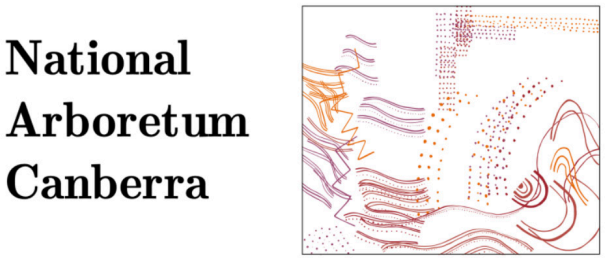


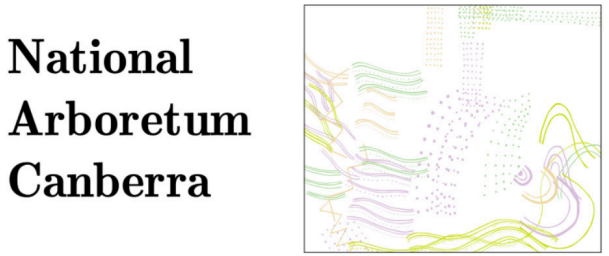




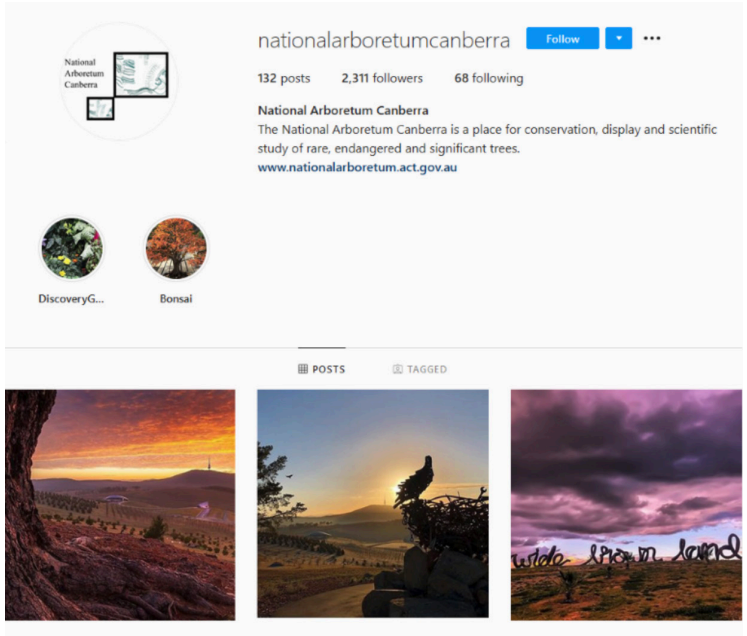


Social media banner variations:

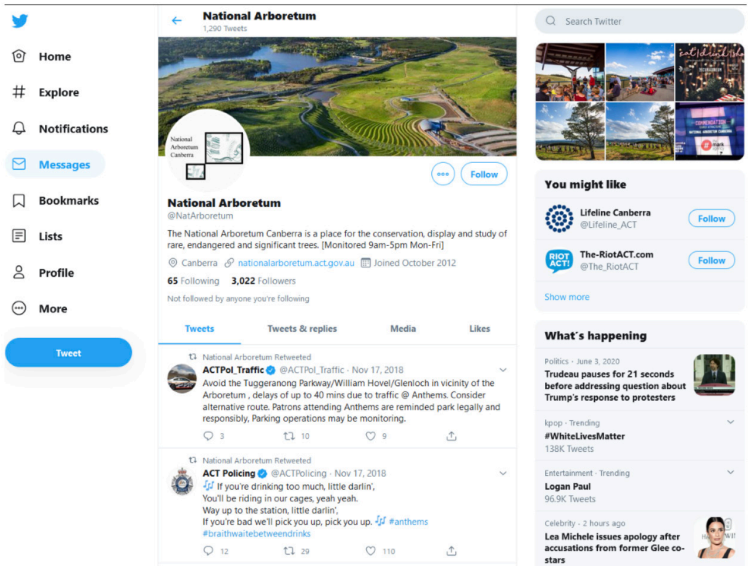




Outputs

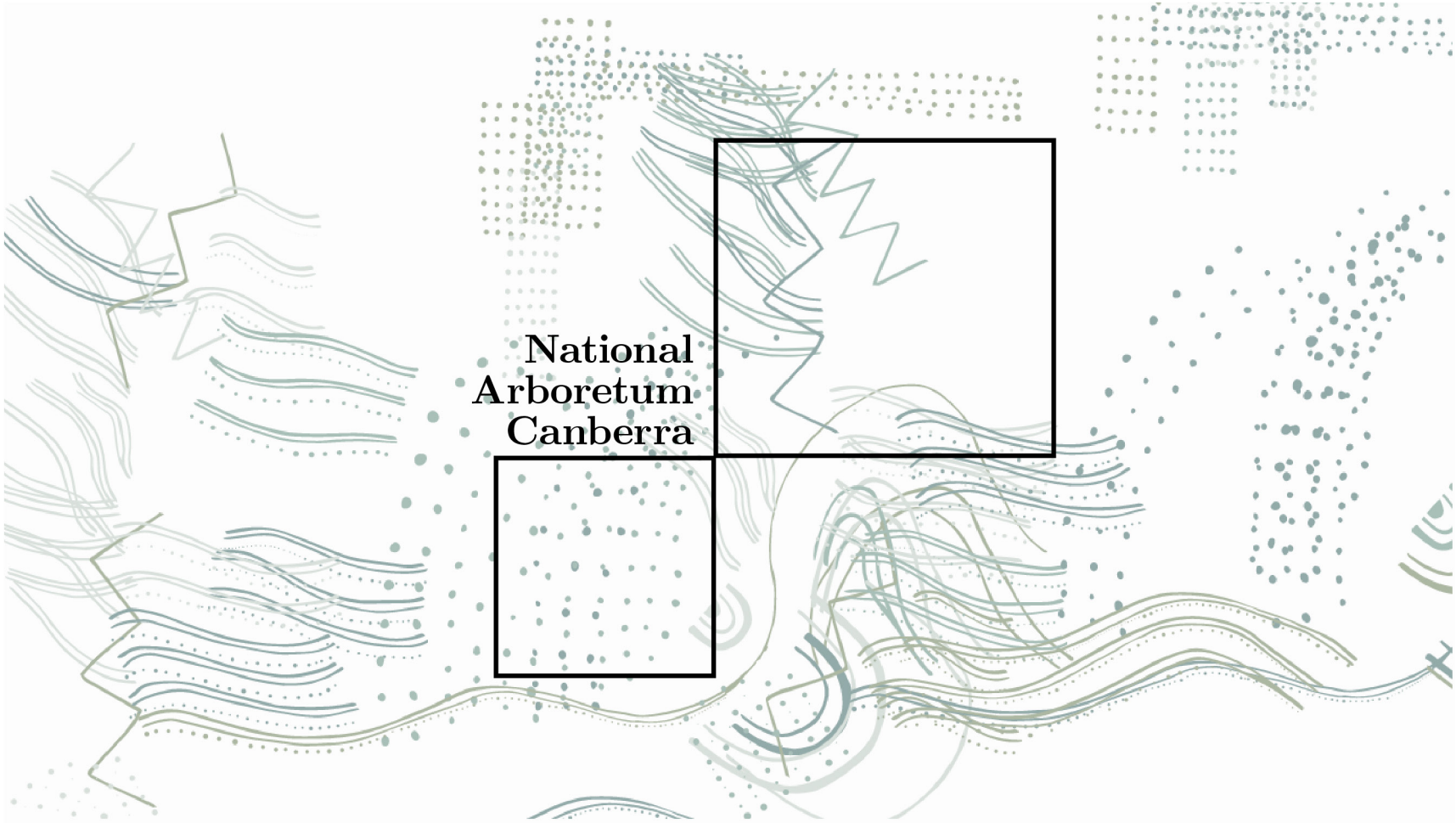


Logo applied to Instagram mockup



Logo applied to Twitter mockup

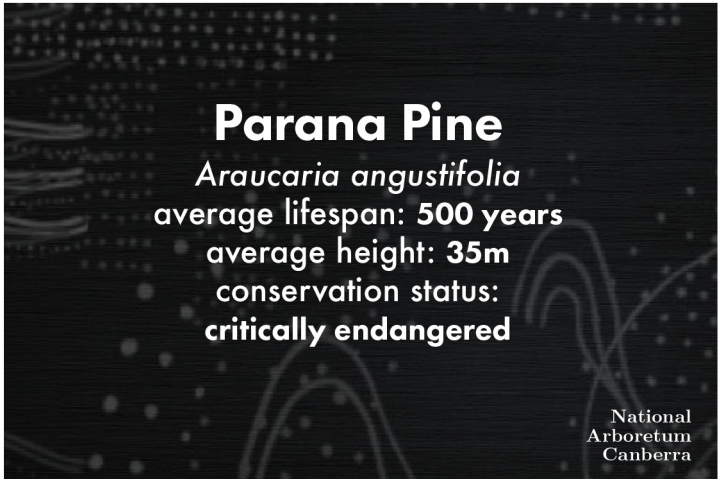
Outputs



Facebook cover photo (optimised for mobile, saved as 640 px by 360 px

Outputs

Summer edition t-shirt



Signage, pattern to be etched into brushed steel with laser cut text