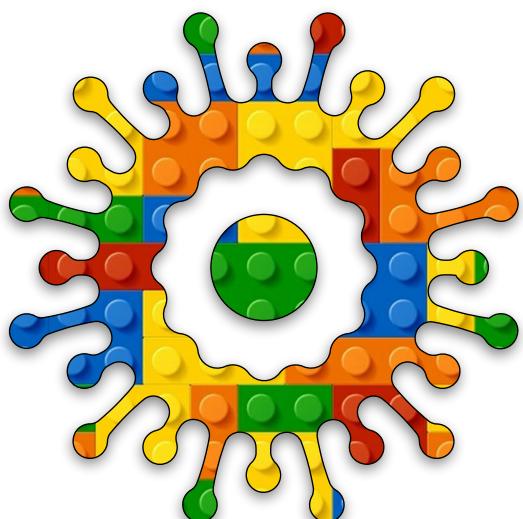


Creative
Coding
Module C
Unit #6

cube wave





Module C Unit #6 Cube Wave

- Sketch C6.1 starting sketch
- Sketch C6.2 rectangle
- Sketch C6.3 breathing
- Sketch C6.4 a row of rectangles
- Sketch C6.5 wavy pattern
- Sketch C6.6 incremental offset
- Sketch C6.7 tidy up
- Sketch C6.8 adding WEBGL
- Sketch C6.9 adding a box
- Sketch C6.10 rotate
- Sketch C6.11 orthographic
- Sketch C6.12 the magic number
- Sketch C6.13 ortho() parameters
- Sketch C6.14 ripples
- Sketch C6.15 a gif



Introduction to the cube wave

This is a coding challenge from the Coding Train YouTube channel. It is a good illustration of what you can do with WebGL. We will add to what we have already learned, including creating a nice little GIF.



Sketch C6.1 starting sketch

This is our starting sketch. We are not WebGL just yet.

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

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



Sketch C6.2 rectangle

We are adding an **angle** variable and incrementing it by **0.1**. We have a rectangle translated to the centre with a height variable **h** set to **100**.

```
let angle = 0
let h

function setup()
{
    createCanvas(400, 400)
    rectMode(CENTER)

}

function draw()
{
    background(220)
    translate(width/2, height/2)
    h = 100
    rect(0, 0, 10, h)
    angle += 0.1
}
```

Notes

Nothing happens; this is just the build-up.

Figure C6.2

The screenshot shows the p5.js IDE interface. At the top, there's a toolbar with icons for play, stop, auto-refresh (which is checked), and help. The title bar says "sketch.js" and "Saved: just now". On the right, it says "Hello, TheHappyCoder! ▾". Below the toolbar is a code editor with the following JavaScript code:

```
1 let angle = 0
2 let h
3
4 function setup()
5{
6  createCanvas(400, 400)
7  rectMode(CENTER)
8 }
9
10 function draw()
11{
12  background(220)
13  translate(width/2, height/2)
14  h = 100
15  rect(0, 0, 10, h)
16  angle += 0.1
17 }
```

To the right of the code editor is a preview window showing a single vertical white rectangle centered on a gray background. At the bottom left is a "Console" tab, and at the bottom center is a "Clear" button.



Sketch C6.3 breathing

We are going to add a sine wave motion to the vertical dimensions of the rectangle. The output from a sine wave is **+1** to **-1**. To increase the visibility of the movement, we will **map()** it to **100**.

```
let angle = 0
let h

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

function draw()
{
    background(220)
    translate(width/2, height/2)
    h = map(sin(angle), -1, 1, 0, 100)
    rect(0, 0, 10, h)
    angle += 0.1
}
```

Notes

What you should see is the rectangle 'breathing' as it expands and contracts vertically. Called Simple Harmonic Motion (SHM).

Code Explanation

<code>h = map(sin(angle), -1, 1, 0, 100)</code>	The value of h is mapped from the sin() output (-1, 1) to (0, 100)
---	--

Figure C6.3

The screenshot shows the p5.js IDE interface. At the top, there's a toolbar with icons for play, stop, refresh, and settings, along with the text "Hello, TheHappyCoder! ▾". Below the toolbar, the file name "sketch.js" and the status "Saved: just now" are displayed. The code editor contains the following JavaScript code:

```
1 let angle = 0
2 let h
3
4 function setup()
5{
6  createCanvas(400, 400)
7  rectMode(CENTER)
8 }
9
10 function draw()
11{
12  background(220)
13  translate(width/2, height/2)
14  h = map(sin(angle), -1, 1, 0, 100)
15  rect(0, 0, 10, h)
16  angle += 0.1
17 }
```

To the right of the code editor is a preview window showing a small gray square centered on a light gray background. At the bottom of the interface, there's a "Console" section with a "Clear" button.



Sketch C6.4 a row of rectangles

Now we will make a row of these moving starting at the left-hand edge, adding an `x` variable for the rectangle and looping through every `10` pixels. We use `x - width/2` because we have translated everything to the centre.

```
let angle = 0
let h

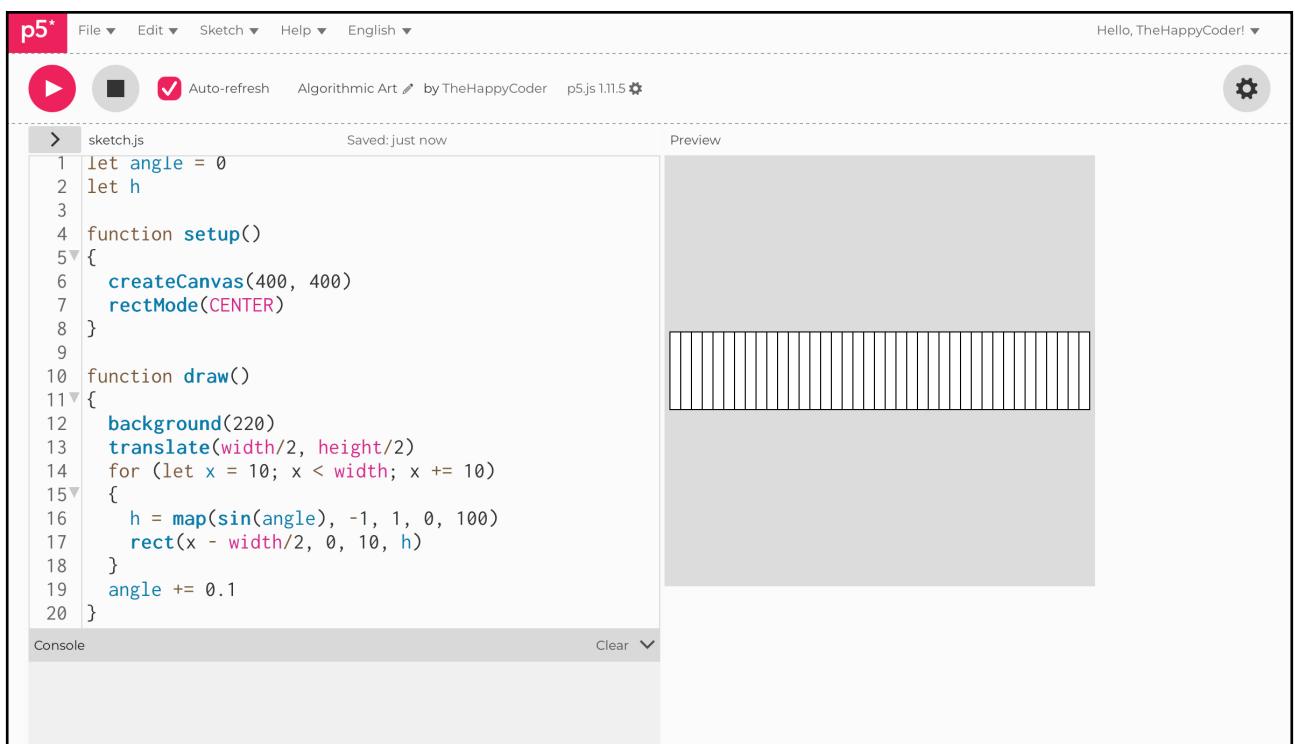
function setup()
{
    createCanvas(400, 400)
    rectMode(CENTER)
}

function draw()
{
    background(220)
    translate(width/2, height/2)
    for (let x = 10; x < width; x += 10)
    {
        h = map(sin(angle), -1, 1, 0, 100)
        rect(x - width/2, 0, 10, h)
    }
    angle += 0.1
}
```

Notes

We have a uniform sine wave, where they are all moving as one.

Figure C6.4



The screenshot shows the p5.js code editor interface. At the top, there's a toolbar with icons for play, stop, auto-refresh (which is checked), and help. The title bar says "Hello, TheHappyCoder! ▾". The main area has tabs for "sketch.js" and "Preview". The code in "sketch.js" is as follows:

```
1 let angle = 0
2 let h
3
4 function setup()
5 {
6   createCanvas(400, 400)
7   rectMode(CENTER)
8 }
9
10 function draw()
11 {
12   background(220)
13   translate(width/2, height/2)
14   for (let x = 10; x < width; x += 10)
15   {
16     h = map(sin(angle), -1, 1, 0, 100)
17     rect(x - width/2, 0, 10, h)
18   }
19   angle += 0.1
20 }
```

The "Preview" window shows a 400x400 canvas with a series of vertical bars of varying heights, creating a sine wave pattern.



Sketch C6.5 wavy pattern

Now we can add an **offset** so that we get a nice wave pattern rather than all together.

```
let angle = 0
let h
let newAngle
let offset

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

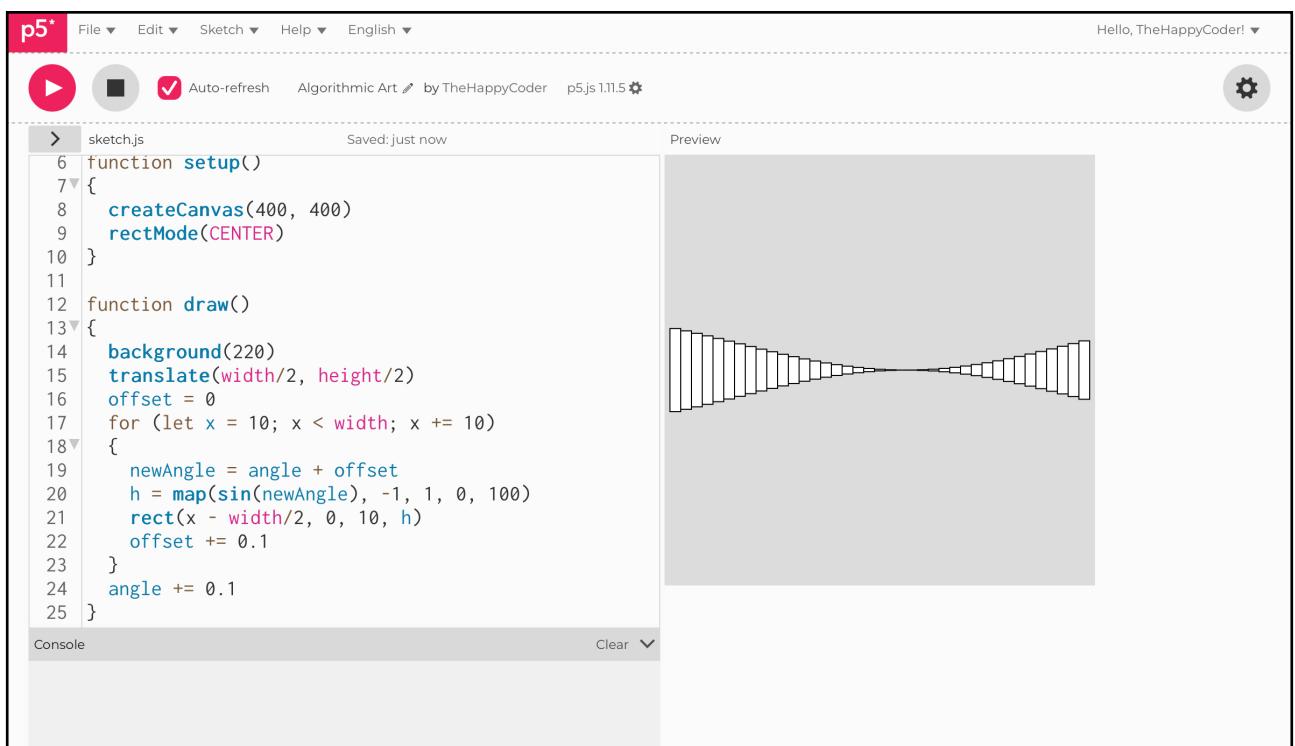
function draw()
{
    background(220)
    translate(width/2, height/2)
    offset = 0
    for (let x = 10; x < width; x += 10)
    {
        newAngle = angle + offset
        h = map(sin(newAngle), -1, 1, 0, 100)
        rect(x - width/2, 0, 10, h)
        offset += 0.1
    }
    angle += 0.1
}
```



Notes

For this, we needed a new variable for the angle, `newAngle`, as the original is being added to all the time.

Figure C6.5



The screenshot shows the p5.js code editor interface. At the top, there's a toolbar with icons for play, stop, auto-refresh (which is checked), and help. The title bar says "Hello, TheHappyCoder! ▾". The main area has tabs for "sketch.js" and "Preview". The code in "sketch.js" is as follows:

```
sketch.js
Saved: just now
function setup()
{
  createCanvas(400, 400)
  rectMode(CENTER)
}

function draw()
{
  background(220)
  translate(width/2, height/2)
  offset = 0
  for (let x = 10; x < width; x += 10)
  {
    newAngle = angle + offset
    h = map(sin(newAngle), -1, 1, 0, 100)
    rect(x - width/2, 0, 10, h)
    offset += 0.1
  }
  angle += 0.1
}
```

The "Preview" window shows a gray canvas with a series of white rectangles arranged in a sine wave pattern, radiating from the center. The rectangles are wider at the bottom and narrower at the top, creating a funnel-like effect.



Sketch C6.6 incremental offset

Changing the **offset** increment to **0.25**.

```
let angle = 0
let h
let newAngle
let offset

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

function draw()
{
    background(220)
    translate(width/2, height/2)
    offset = 0
    for (let x = 10; x < width; x += 10)
    {
        newAngle = angle + offset
        h = map(sin(newAngle), -1, 1, 0, 100)
        rect(x - width/2, 0, 10, h)
        offset += 0.25
    }
    angle += 0.1
}
```



Notes

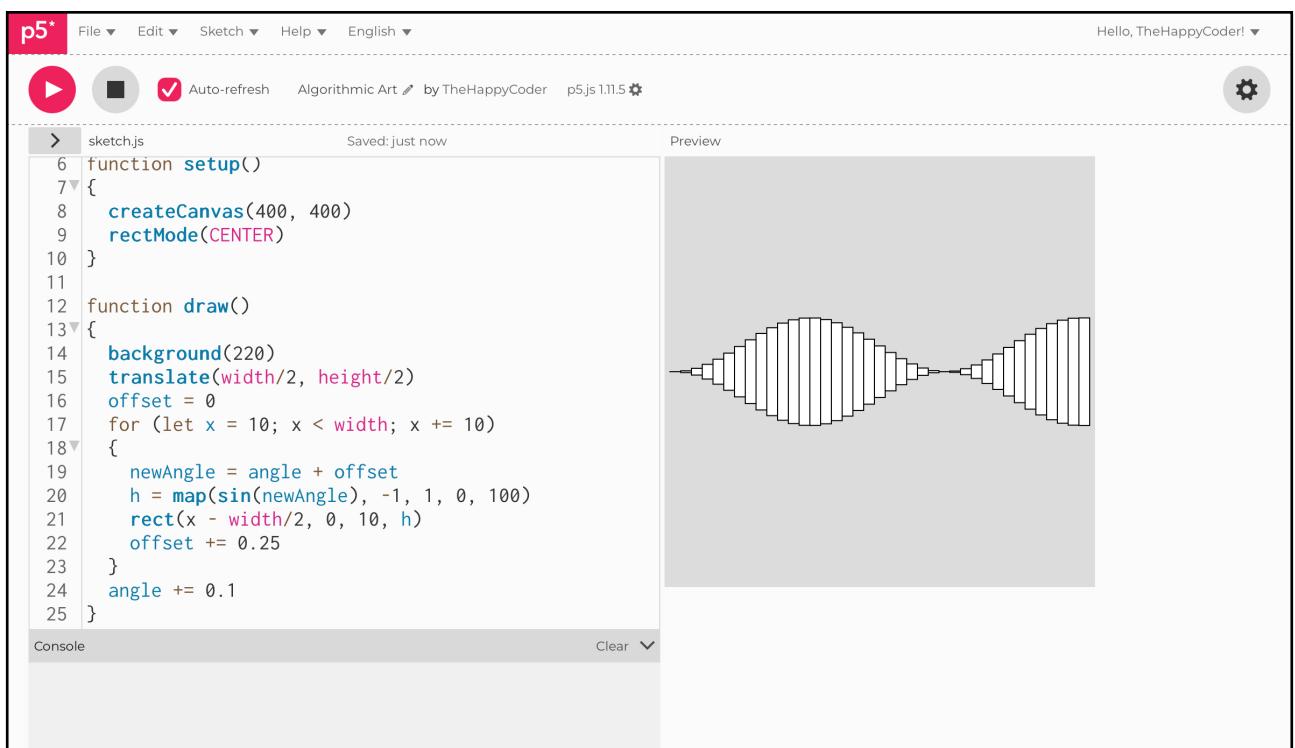
An even nicer sine wave.



Challenge

Try different offsets.

Figure C6.6



The screenshot shows the p5.js code editor interface. At the top, there are buttons for play/pause, stop, auto-refresh (which is checked), and settings. The title bar says "sketch.js" and "Saved: just now". The right side of the interface is labeled "Preview" and shows a visual representation of the code's output. The code itself is as follows:

```
sketch.js
function setup()
{
  createCanvas(400, 400)
  rectMode(CENTER)
}

function draw()
{
  background(220)
  translate(width/2, height/2)
  offset = 0
  for (let x = 10; x < width; x += 10)
  {
    newAngle = angle + offset
    h = map(sin(newAngle), -1, 1, 0, 100)
    rect(x - width/2, 0, 10, h)
    offset += 0.25
  }
  angle += 0.1
}
```

The preview window displays a series of vertical bars forming a sine wave pattern across the center of the canvas.



Sketch C6.7 tidy up

We are going to add a few more features as we prepare to jump from **2D** to **3D**. We create a variable for the width called **w** and set **x** to start at **0**, not **10**, in the **for()** loop. This just tidies things up a bit.

```
let angle = 0
let h
let newAngle
let offset
let w = 20

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

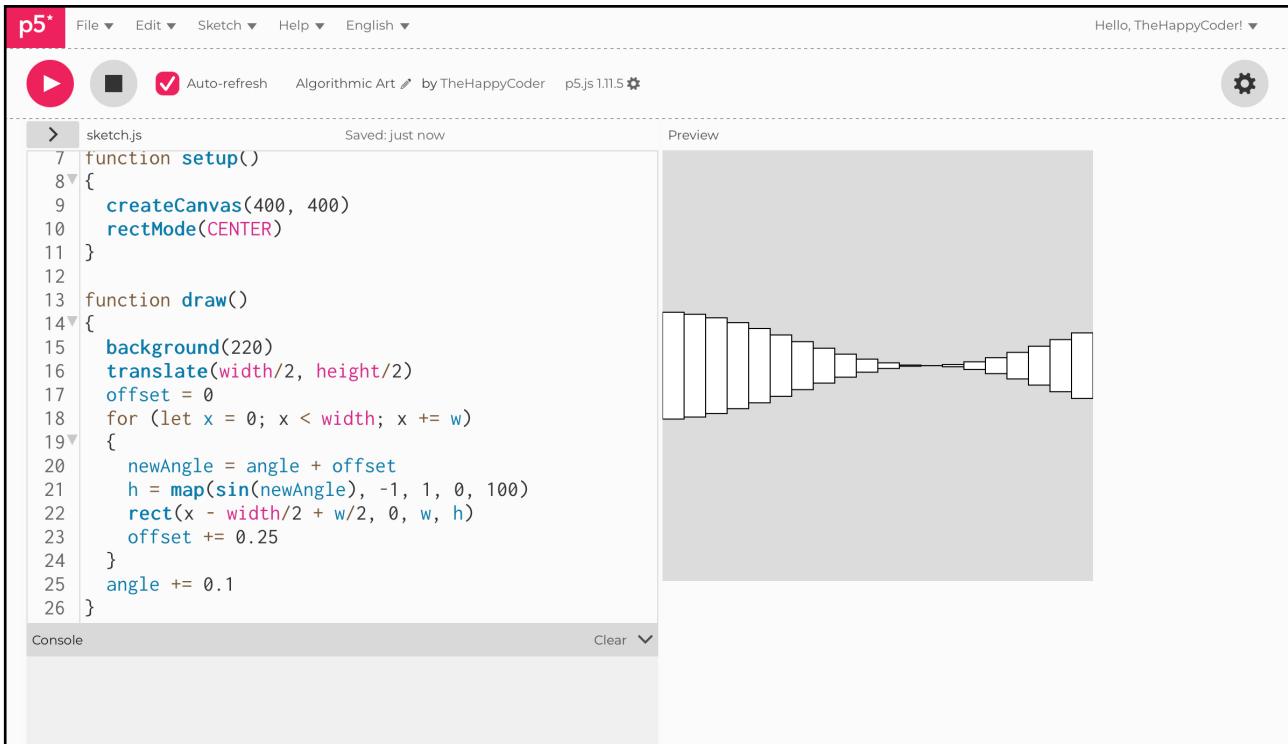
function draw()
{
    background(220)
    translate(width/2, height/2)
    offset = 0
    for (let x = 0; x < width; x += w)
    {
        newAngle = angle + offset
        h = map(sin(newAngle), -1, 1, 0, 100)
        rect(x - width/2 + w/2, 0, w, h)
        offset += 0.25
    }
    angle += 0.1
}
```



Notes

Similar result.

Figure C6.7



The screenshot shows the p5.js code editor interface. At the top, there are tabs for File, Edit, Sketch, Help, and English, along with a user greeting "Hello, TheHappyCoder! ▾". Below the tabs, there are icons for play, stop, and refresh, followed by the text "Auto-refresh Algorithmic Art by TheHappyCoder p5.js 1.11.5". The main area is divided into two sections: "sketch.js" on the left and "Preview" on the right. The "sketch.js" section contains the following JavaScript code:

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

function draw()
{
    background(220)
    translate(width/2, height/2)
    offset = 0
    for (let x = 0; x < width; x += w)
    {
        newAngle = angle + offset
        h = map(sin(newAngle), -1, 1, 0, 100)
        rect(x - width/2 + w/2, 0, w, h)
        offset += 0.25
    }
    angle += 0.1
}
```

The "Preview" section shows a visual representation of the code. It features a light gray background with a series of white rectangles of varying heights arranged in a sine wave pattern. The rectangles are centered around the horizontal axis. The width of each rectangle is approximately one-tenth of the canvas width, and their heights range from about 10 to 90 pixels, corresponding to the mapped sine values.



Sketch C6.8 adding WEBGL

So far everything has been 2D; now we add **WEBGL** and remove the **translate()** (because it automatically puts it in the centre). **WEBGL** is just the render, so we can still draw something in 2D.

```
let angle = 0
let h
let newAngle
let offset
let w = 20

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

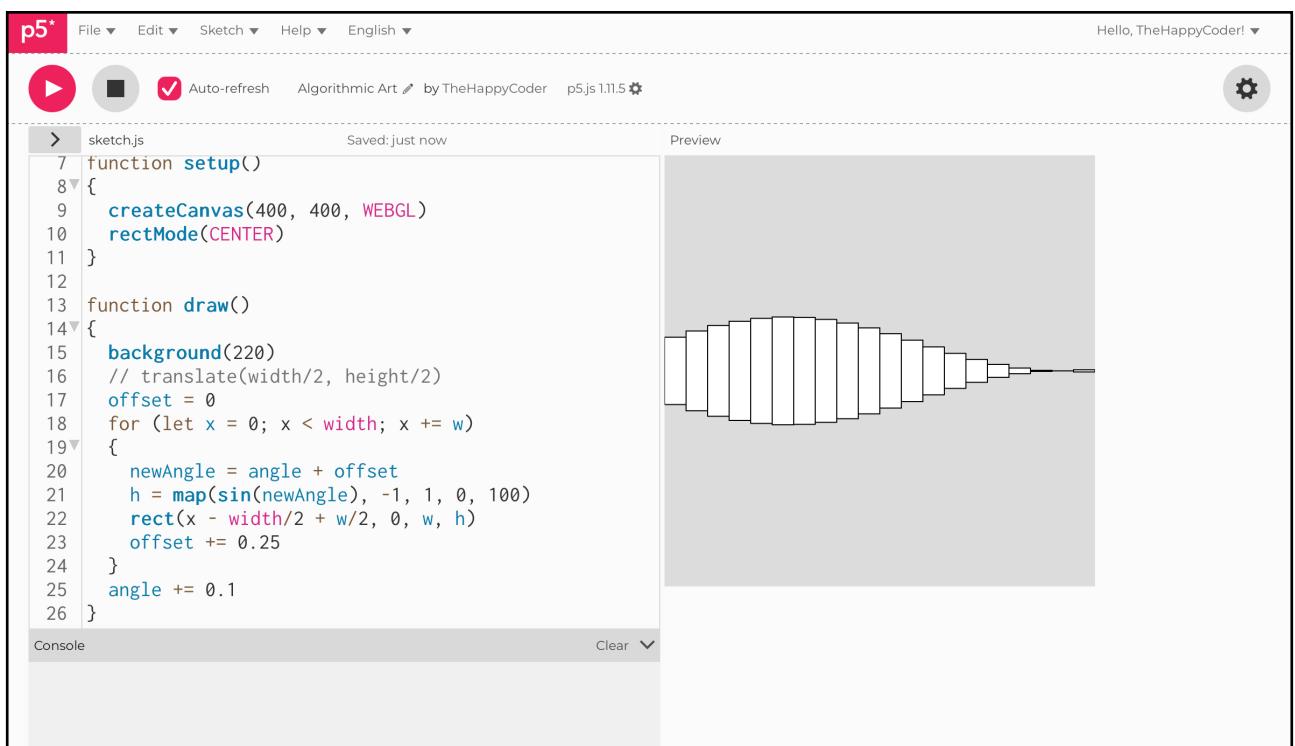
function draw()
{
    background(220)
    // translate(width/2, height/2)
    offset = 0
    for (let x = 0; x < width; x += w)
    {
        newAngle = angle + offset
        h = map(sin(newAngle), -1, 1, 0, 100)
        rect(x - width/2 + w/2, 0, w, h)
        offset += 0.25
    }
    angle += 0.1
}
```



Notes

Pretty much the same result again.

Figure C6.8



The screenshot shows the p5.js code editor interface. At the top, there are buttons for File, Edit, Sketch, Help, and English, along with a user profile "Hello, TheHappyCoder!". Below the menu bar, there are icons for play, stop, and refresh, followed by "Auto-refresh" checked, "Algorithmic Art" by TheHappyCoder, and p5.js 1.11.5. On the right side of the editor is a preview window showing a visual output.

The code in the editor is as follows:

```
sketch.js
function setup()
{
  createCanvas(400, 400, WEBGL)
  rectMode(CENTER)
}

function draw()
{
  background(220)
  // translate(width/2, height/2)
  offset = 0
  for (let x = 0; x < width; x += w)
  {
    newAngle = angle + offset
    h = map(sin(newAngle), -1, 1, 0, 100)
    rect(x - width/2 + w/2, 0, w, h)
    offset += 0.25
  }
  angle += 0.1
}
```

The preview window displays a series of white rectangles on a gray background. The rectangles are arranged in a grid-like pattern, with each row shifted slightly to the right. The height of each rectangle is determined by a sine wave function, creating a wavy effect across the canvas. The rectangles are centered at regular intervals along the horizontal axis.



Sketch C6.9 adding a box

We replace the `rect()` with `box(w)` and have each box drawn at a new `x` position as part of the loop. To make sure we aren't compounding the `translate`, we use `push()` and `pop()` to reset it each iteration of the `for()` loop.

```
let angle = 0
let h
let newAngle
let offset
let w = 20

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

function draw()
{
    background(220)
    offset = 0
    for (let x = 0; x < width; x += w)
    {
        push()
        newAngle = angle + offset
        h = map(sin(newAngle), -1, 1, 0, 100)
        translate(x - width/2, 0, 0)
        box(w)
        offset += 0.25
        pop()
    }
}
```

```
    angle += 0.1  
}
```

Notes

It looks a little off-centre, but we will live with that for now.

Figure C6.9

The screenshot shows the p5.js code editor interface. At the top, there's a toolbar with icons for play, stop, and refresh, followed by the text "Auto-refresh" with a checked checkbox, "Algorithmic Art" by "TheHappyCoder", and "p5.js 1.11.5". On the right, it says "Hello, TheHappyCoder!" with a dropdown arrow. Below the toolbar, the title bar reads "sketch.js" and "Saved: just now". The main area contains the following code:

```
sketch.js
9  createCanvas(400, 400, WEBGL)
10 rectMode(CENTER)
11 }
12
13 function draw()
14 {
15   background(220)
16   offset = 0
17   for (let x = 0; x < width; x += w)
18   {
19     push()
20     newAngle = angle + offset
21     h = map(sin(newAngle), -1, 1, 0, 100)
22     translate(x - width/2, 0, 0)
23     box(w)
24     offset += 0.25
25     pop()
26   }
27   angle += 0.1
28 }
```

The preview window on the right shows a light gray background with a series of small, evenly spaced horizontal rectangles extending across the width of the canvas.



Sketch C6.10 rotate

To prove they are cubes, we will rotate by **-1**, but we want the sine wave to affect the height, so we will put that back in and also have the **z** dimension as **w**.

```
let angle = 0
let h
let newAngle
let offset
let w = 20

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

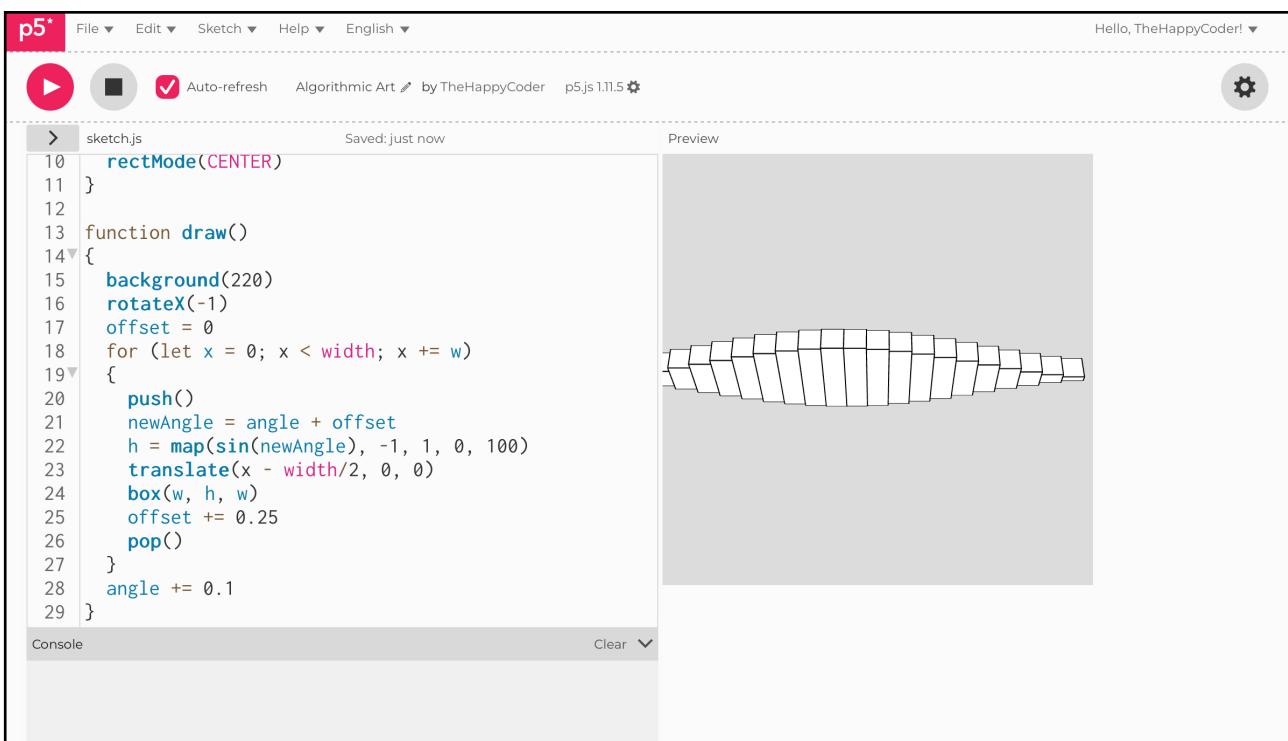
function draw()
{
    background(220)
    rotateX(-1)
    offset = 0
    for (let x = 0; x < width; x += w)
    {
        push()
        newAngle = angle + offset
        h = map(sin(newAngle), -1, 1, 0, 100)
        translate(x - width/2, 0, 0)
        box(w, h, w)
        offset += 0.25
    }
}
```

```
    pop()  
}  
  
angle += 0.1  
}
```

Notes

You can see the motion of the cubes.

Figure C6.10



The screenshot shows the p5.js code editor interface. The top bar includes the p5 logo, file navigation, and a user profile. The main area has tabs for 'sketch.js' and 'Preview'. The code in 'sketch.js' is as follows:

```
sketch.js
10 rectMode(CENTER)
11 }
12
13 function draw()
14 {
15   background(220)
16   rotateX(-1)
17   offset = 0
18   for (let x = 0; x < width; x += w)
19   {
20     push()
21     newAngle = angle + offset
22     h = map(sin(newAngle), -1, 1, 0, 100)
23     translate(x - width/2, 0, 0)
24     box(w, h, w)
25     offset += 0.25
26     pop()
27   }
28   angle += 0.1
29 }
```

The 'Preview' window shows a series of white rectangles of varying heights, creating a wavy, undulating pattern against a light gray background.



Sketch C6.11 orthographic

We are going to introduce a different perspective called **orthographic**. The default is **perspective projection**, which looks more realistic, but we will introduce **orthographic projection**. It makes the boxes look a bit flatter.

```
let angle = 0
let h
let newAngle
let offset
let w = 20

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

function draw()
{
    background(220)
    ortho()
    rotateX(-1)
    offset = 0
    for (let x = 0; x < width; x += w)
    {
        push()
        newAngle = angle + offset
        h = map(sin(newAngle), -1, 1, 0, 100)
        translate(x - width/2, 0, 0)
        box(w, h, w)
        offset += 0.1
    }
}
```

```
    pop()
}
angle += 0.1
}
```

Notes

The difference between the projections is something I will leave you to research if you don't already know.

Code Explanation

ortho()	Changes from the default projection to orthographic
---------	---

Figure C6.11

The screenshot shows the p5.js IDE interface. At the top, there's a toolbar with icons for play, stop, and refresh, followed by the text "Auto-refresh" and "Algorithmic Art" by TheHappyCoder. The version "p5.js 1.11.5" is also visible. On the right, there's a user profile "Hello, TheHappyCoder!" with a gear icon.

The main area has tabs for "sketch.js" and "Preview". The "sketch.js" tab contains the following code:

```
11 }
12
13 function draw()
14{
15   background(220)
16   ortho()
17   rotateX(-1)
18   offset = 0
19   for (let x = 0; x < width; x += w)
20   {
21     push()
22     newAngle = angle + offset
23     h = map(sin(newAngle), -1, 1, 0, 100)
24     translate(x - width/2, 0, 0)
25     box(w, h, w)
26     offset += 0.1
27     pop()
28   }
29   angle += 0.1
30 }
```

The "Preview" window shows a 3D perspective drawing of a series of rectangular blocks. The blocks are arranged in a curve, with their height varying sinusoidally. The background is a light gray gradient.



Sketch C6.12 the magic number

We are now going to rotate along the **x-axis** by **45** degrees ($\pi/4$) and rotate along the **y** axis by a magic number (**ma**), which is arctan of one divided by the **square root of two**. This is a number found from those experimenting with different projections, so just accept it. Also, we have a new loop around the **x for()** loop to have a **z** value and move the **offset** into that loop.

```
let angle = 0
let h
let newAngle
let offset
let w = 20
let ma

function setup()
{
    createCanvas(400, 400, WEBGL)
    rectMode(CENTER)
    ma = atan(1/sqrt(2))
}

function draw()
{
    background(220)
    ortho()
    rotateX(-PI/4)
    rotateY(ma)
    offset = 0
    for (let z = 0; z < height; z += w)
    {
        for (let x = 0; x < width; x += w)
```

```

{
  push()
  newAngle = angle + offset
  h = map(sin(newAngle), -1, 1, 0, 100)
  translate(x - width/2, 0, z - height/2)
  box(w, h, w)
  pop()
}
offset += 0.25
}
angle += 0.1
}

```

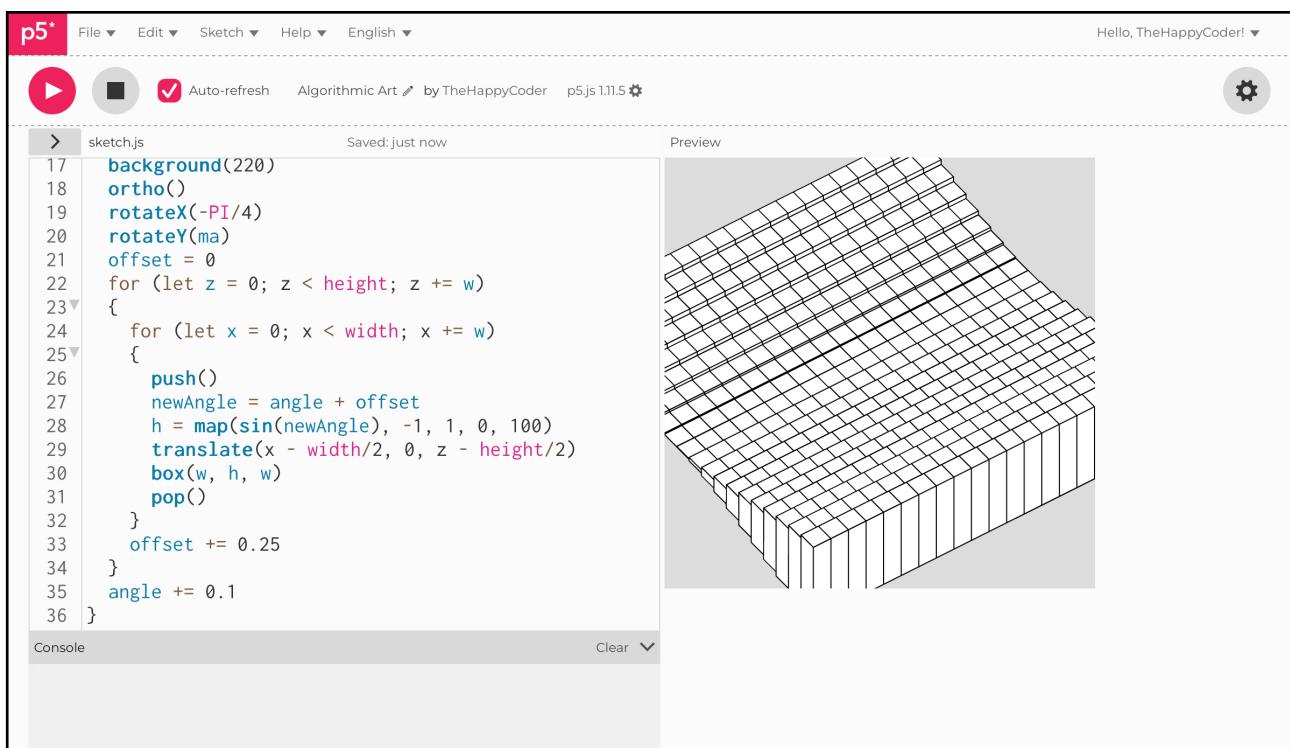
Notes

In p5.js, you can use the constant π by using **PI**. This is useful if you are using radians and don't want to work out the actual number. The atan() function is currently not supported with v2.x.

Code Explanation

ma = atan(1/sqrt(2))	The magic number
rotateX(-PI/4)	Rotating x by $\pi/4$ or 45°

Figure C6.12



The figure shows a screenshot of the p5.js code editor. The top bar includes the p5 logo, file navigation, and a user profile. The main area has tabs for 'sketch.js' and 'Preview'. The code in 'sketch.js' is as follows:

```
sketch.js
17 background(220)
18 ortho()
19 rotateX(-PI/4)
20 rotateY(ma)
21 offset = 0
22 for (let z = 0; z < height; z += w)
23 {
24   for (let x = 0; x < width; x += w)
25   {
26     push()
27     newAngle = angle + offset
28     h = map(sin(newAngle), -1, 1, 0, 100)
29     translate(x - width/2, 0, z - height/2)
30     box(w, h, w)
31     pop()
32   }
33   offset += 0.25
34 }
35 angle += 0.1
36 }
```

The 'Preview' window displays a 3D perspective grid where each vertical column of boxes is slightly rotated around its vertical axis, creating a sense of depth and motion.



Sketch C6.13 ortho() parameters

We can add more parameters to `ortho()` to allow what you can see. There is a lot involved in these parameters to explain, but it is to do with the depth and width of the viewing angle, etc. I've chosen these values as they put the boxes in a nice view. I suggest twiddling the numbers to get what you want.

```
let angle = 0
let h
let newAngle
let offset
let w = 20
let ma

function setup()
{
    createCanvas(400, 400, WEBGL)
    rectMode(CENTER)
    ma = atan(1/sqrt(2))
}

function draw()
{
    background(220)
    ortho(-320, 300, -300, 300)
    rotateX(-PI/4)
    rotateY(ma)
    offset = 0
    for (let z = 0; z < height; z += w)
    {
```

```

for (let x = 0; x < width; x += w)
{
    push()
    newAngle = angle + offset
    h = map(sin(newAngle), -1, 1, 0, 100)
    translate(x - width/2, 0, z - height/2)
    box(w, h, w)

    pop()
}
offset += 0.25
}
angle += 0.1
}

```

Notes

You can have up to six arguments for the `ortho()` projection; the fifth and sixth are near and far. This is all to do with a camera's view of the object shape.

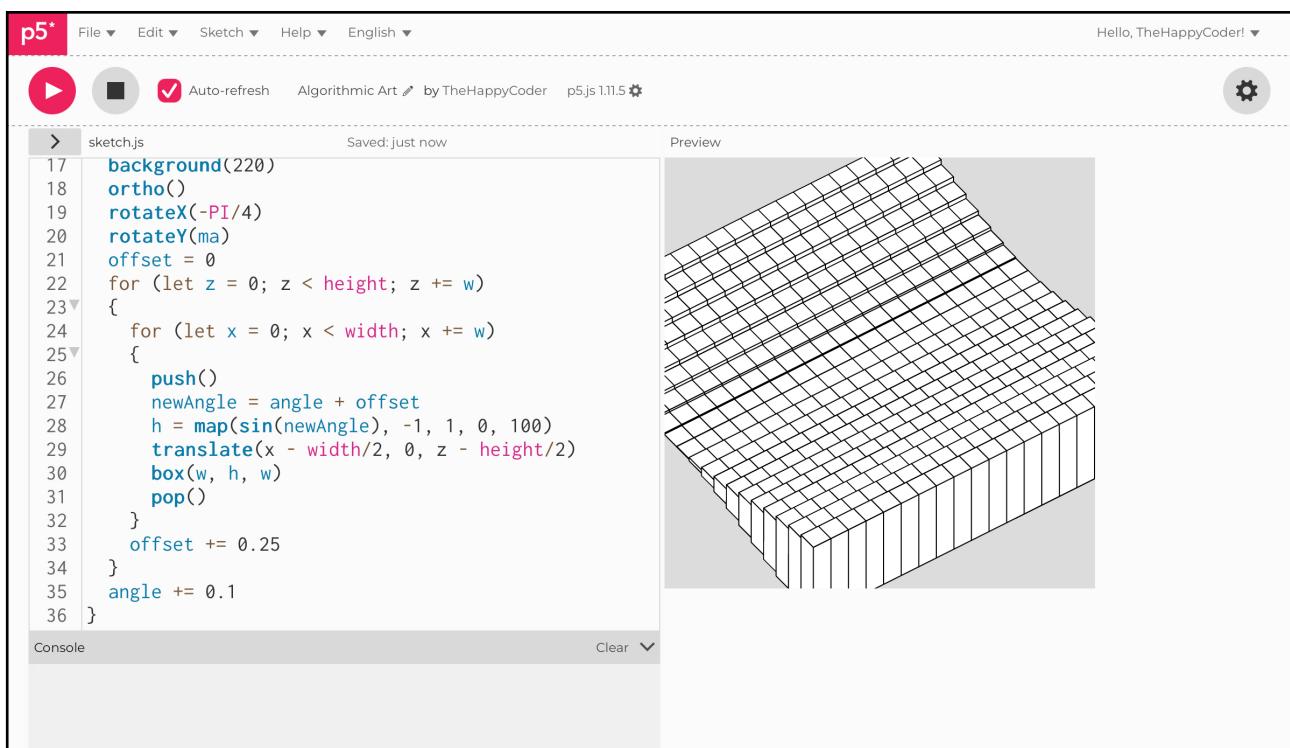
Challenge

Have a play with the values to see what difference they make.

Code Explanation

<code>ortho(-320, 300, -300, 300)</code>	They indicate frustum values for left, right, bottom top
--	--

Figure C6.13



The figure shows a screenshot of the p5.js code editor. The top bar includes the p5 logo, file navigation, and a preview button. The main area has tabs for 'sketch.js' and 'Preview'. The code in 'sketch.js' is as follows:

```
background(220)
ortho()
rotateX(-PI/4)
rotateY(ma)
offset = 0
for (let z = 0; z < height; z += w)
{
  for (let x = 0; x < width; x += w)
  {
    push()
    newAngle = angle + offset
    h = map(sin(newAngle), -1, 1, 0, 100)
    translate(x - width/2, 0, z - height/2)
    box(w, h, w)
    pop()
  }
  offset += 0.25
}
angle += 0.1
```

The 'Preview' window displays a 3D perspective grid composed of small rectangular blocks, creating a stepped or terraced effect. The grid is oriented diagonally, with the perspective centered on a central point.



Sketch C6.14 ripples

At the moment, we have the whole thing moving as a wave, but what we want is a ripple-type effect across the surface. Where the individual boxes move. So we need to calculate the distance between the **x**, **z**, and the **centre**.

We create a **maxDistance** in the **x** and **z** directions, then move the **offset** into the loop so we get the ripple effect across the loop. Also, a few tweaks and adding **normalMaterial**.

```
let angle = 0
let h
let newAngle
let offset
let w = 20
let ma
let maxDistance
let d

function setup()
{
    createCanvas(400, 400, WEBGL)
    rectMode(CENTER)
    ma = atan(1/sqrt(2))
    maxDistance = dist(0, 0, 200, 200)
}

function draw()
{
    background(220)
    normalMaterial()
    ortho(-320, 300, -300, 300)
```

```

rotateX(-PI/4)
rotateY(ma)
offset = 0
for (let z = 0; z < height; z += w)
{
  for (let x = 0; x < width; x += w)
  {
    push()
    d = dist(x, z, width/2, height/2)
    offset = map(d, 0, maxDistance, -3, 3)
    newAngle = angle + offset
    h = map(sin(newAngle), -1, 1, 50, 250)
    translate(x - width/2, 0, z - height/2)
    box(w, h, w)
    pop()
  }
  // offset += 0.25
}
angle += 0.1
}

```

Notes

Pretty neat, eh!

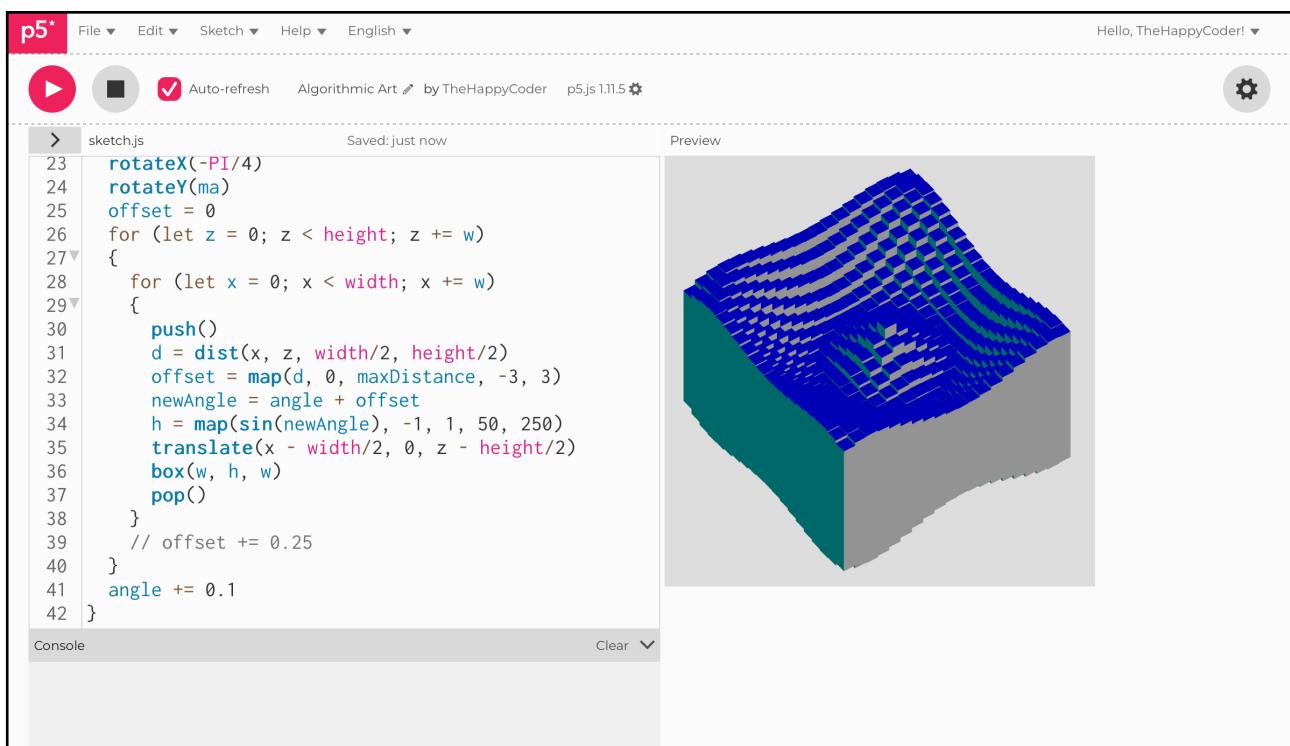
Challenge

Try other materials and lights.

Code Explanation

d = dist(x, z, width/2, height/2)	Measures the distance between x, z and the centre of the space
-----------------------------------	--

Figure C6.14



The screenshot shows the p5.js code editor interface. At the top, there are menu items: File, Edit, Sketch, Help, and English. On the right, it says "Hello, TheHappyCoder!" with a gear icon. The main area has tabs for "sketch.js" and "Preview". The code in "sketch.js" is as follows:

```
sketch.js
23  rotateX(-PI/4)
24  rotateY(ma)
25  offset = 0
26  for (let z = 0; z < height; z += w)
27  {
28    for (let x = 0; x < width; x += w)
29    {
30      push()
31      d = dist(x, z, width/2, height/2)
32      offset = map(d, 0, maxDistance, -3, 3)
33      newAngle = angle + offset
34      h = map(sin(newAngle), -1, 1, 50, 250)
35      translate(x - width/2, 0, z - height/2)
36      box(w, h, w)
37      pop()
38    }
39    // offset += 0.25
40  }
41  angle += 0.1
42 }
```

The "Preview" window shows a 3D perspective view of the generated artwork. It features a series of blue, diamond-shaped blocks arranged in a grid-like pattern on a grey rectangular base. The blocks are tilted at an angle, creating a sense of depth. The overall effect is a digital representation of a geometric or architectural structure.



Sketch C6.15 a gif

Now to create a **GIF**. A GIF is a short video that repeats endlessly. We can do this using our knowledge of the number of frames it takes to do a complete cycle through the sine wave so it will look as if it is seamlessly continuous.

We will use **60** frames and when you press the space bar (**key**), which is what the (" ") means, it will generate a .gif file according to the name you have given it.

I have also swapped the **rotateX** and **rotateY** around to give a different (better) effect.

! Please note that: If you are doing this on a tablet (iPad, etc.), I found that there wasn't enough memory for 60 frames, but less than 60 frames doesn't really work. Also, you might want to slow it down a bit for the .gif, maybe 0.05 rather than 0.1.

```
let angle = 0
let h
let newAngle
let offset
let w = 20
let ma
let maxDistance
let d
let frames = 60

function setup()
{
  createCanvas(400, 400, WEBGL)
  rectMode(CENTER)
  ma = atan(1/sqrt(2))
  maxDistance = dist(0, 0, 200, 200)
```

```

}

function keyPressed()
{
  if (key == " ")
  {
    const options = {
      units: "frames",
      delay: 0
    }
    saveGif("cubewave.gif", frames, options)
  }
}

function draw()
{
  background(220)
  normalMaterial()
  ortho(-320, 300, -300, 300)
  rotateY(ma)
  rotateX(-PI/4)

  offset = 0
  for (let z = 0; z < height; z += w)
  {
    for (let x = 0; x < width; x += w)
    {
      push()
      d = dist(x, z, width/2, height/2)
      offset = map(d, 0, maxDistance, -3, 3)
      newAngle = angle + offset
      h = map(sin(newAngle), -1, 1, 50, 250)
      translate(x - width/2, 0, z - height/2)
      box(w, h, w)
    }
  }
}

```

```
    pop()
}
}

angle += 0.1
}
```

Notes

Play with the values regarding the GIF and see if you can get a better response. Remember that to save the GIF, you need to press the space bar.

Challenge

1. Use lights, colours and other materials.
2. Use other shapes, e.g. cylinders.
3. Expand the ripple effect.
4. Rotate the whole thing.

Code Explanation

saveGif("cubewave.gif", frames, options)	We give the name of the file to saved, the type of file, how many frames and any options stated
---	--