Coding Challenge Guide - 3D Terrain Generation

Step 1 - Setting up the canvas

```
void setup () {
  size (600, 600, P3D);
}

Void draw () {
  background(0);
}
```

Step 2 - Making the variables for columns, rows, and scale

```
Int cols, rows;
Int scl = 20;

void setup () {
  size (600, 600, P3D);
}

Void draw () {
  background(0);
}
```

Step 3 - Making a Nested Loop to create an array of Columns and Rows

```
Int cols, rows;
Int scl = 20;

void setup () {
  size (600, 600, P3D);
}

Void draw () {
  background(0);
```

```
  For (int x = 0; x<cols; x++) {
     For (int y=0; y<rows; y++) {

   }
     }

}
```

Step 4 - making sure things are working

```
int cols, rows;
int scl = 20;
  int w = 600; //Setting the width and the height to random variables (which we will change later)
  int h =600;

void setup () {
  size (600, 600, P3D);
  cols=w/scl; //These variables set the number of columns and rows we will be using in
  rows=h/scl; //making the grid (temporary, currently only set for testing purposes)
}

void draw () {
  background(0);

  for (int x=0; x<cols; x++) {
    for (int y=0; y<rows; y++) {
       stroke(255); //Making it so that the grid is full of rectangles with a white stroke and no fill
       noFill();
       rect(x*scl, y*scl, scl, scl); //Setting the rectangle's dimensions(x, y, width, height)

    }
  }
}
```

Step 5 - Switching X & Y for loops and misplacing fill & stroke

A. Change X & Y for loops (around the beginShape, instead of X on top it should be Y)
B. Take the stroke and fill settings out of the for loops and place them just above

```
int cols, rows;
```

```
int scl = 20;
  int w = 600;
  int h =600;

void setup () {
  size (600, 600, P3D);
  cols=w/scl;
  rows=h/scl;
}

void draw () {
  background(0);
  stroke(255); //STEP 2: We place the stroke and fill settings above the for loops, just to make
  noFill();          //life easy

 for (int y=0; y<rows; y++) {

   for (int x=0; x<cols; x++) {

      rect(x*scl, y*scl, scl, scl);


   }

 }
}
```

Step 6 - Putting in the beginShape stuff and Showing audience what it actually is

A. Show the audience the following link: https://processing.org/reference/beginShape_.html
   We will be using TRIANGLE_STRIP.
B. Insert the beginShape and endShape into our code
C. Comment out the rect and replace it with vertex
D. Run, and explain that it did not work, and that's ok. What we did was set all the vertex's to be at the top, so we only set vertex(see illustration 1). What we need to do is set Vertex up and down and up and down. (see illustration 2).
E. Adding another vertex (the second one

```
int cols, rows;
```

```
int scl = 20;
int w = 600;
int h =600;

void setup () {
  size (600, 600, P3D);
  cols=w/scl;
  rows=h/scl;
}

void draw () {
  background(0);
  stroke(255);
  noFill();

  for (int y=0; y<rows; y++) {
    beginShape(TRIANGLE_STRIP);
    for (int x=0; x<cols; x++) {

        vertex(x*scl, y*scl);
        vertex(x*scl, (y+1) *scl); //DON'T ADD UNTIL STEP E
        //rect(x*scl, y*scl, scl, scl);


    }
    endShape();
  }
}
```

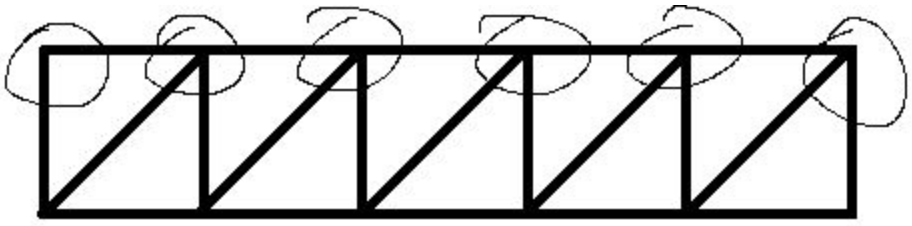

Illustration 1

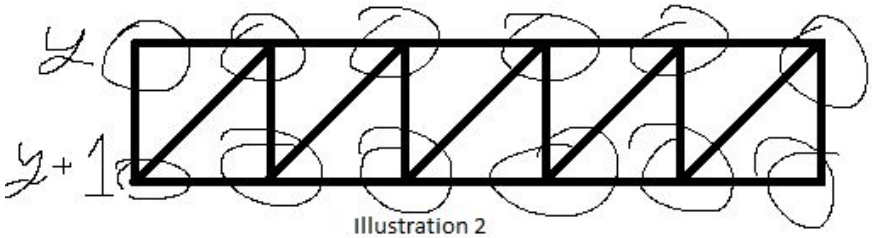

Illustration 2

Step 7 - Rotating along x-axis

A. Insert step A as seen below. “Now we want to rotate this grid to make it 3D, so to do that we’re gonna first set it to begin drawing at the center of the screen, as opposed to at the top left, and then we will rotate it by PI/3 (a.k.a 60 degrees)...This, however, looks pretty weird, and that’s because it *Begins* drawing in the middle of the screen, and then draws to the left from there.”
B. Insert step B as seen below. “To fix that we’re just going to translate the already rotated collection of triangles by *see below*

.

```
int cols, rows;
int scl = 20;
int w = 600;
int h =600;
```

```
void setup () {
  size (600, 600, P3D);
  cols=w/scl;
  rows=h/scl;
}

void draw () {
  background(0);
  stroke(255);
  noFill();

  translate(width/2, height/2);//STEP A
  rotateX(PI/3); //STEP A
  translate(-w/2, -h/2); //STEP B

  for (int y=0; y<rows; y++) {
    beginShape(TRIANGLE_STRIP);
    for (int x=0; x<cols; x++) {

        vertex(x*scl, y*scl);
        vertex(x*scl, (y+1)*scl);
       //rect(x*scl, y*scl, scl, scl);

    }
    endShape();
  }
}
```

Step 7 - Adding Z-Values

A. Adding random z-values to the vertex. “Now we’re gonna add z-values to make our vertices stand out vertically on our 3D plane. We’re just gonna test and see what happens when you randomise it from -10 to 10 (you can experiment with your own values, the higher the more crazy it gets).”

B. “This is pretty great terrain, but to get some nice saucy terrain generated we’re gonna wanna store the Z-values in some sort of data structure. The reason for this is that the vertices X and Y are static, they will always generate in those positions with the parameters we gave them. However the Z-values, which will be what determines how our terrain generates, will need to be generated with some sort of algorithm, and thus we’re gonna wanna keep track of those points and that algorithm. To do so, we’re gonna create a 2D array.

C. Create the array as seen below, and then initialise it.

```
int cols, rows;
int scl = 20;
int w = 600;
int h =600;

Float[] [] terrain; //STEP C

void setup () {
  size (600, 600, P3D);
  cols=w/scl;
  rows=h/scl;
  terrain = new float [cols] [rows];//STEP C
  for (int y=0; y<rows; y++) {//STEP C
    for (int x=0; x<cols; x++) {//STEP C
        terrain[x][y] = random(-10, 10);//STEP C
}
}

void draw () {
  background(0);
  stroke(255);
  noFill();

  translate(width/2, height/2);
  rotateX(PI/3);
  translate(-w/2, -h/2);

  for (int y=0; y<rows; y++) {
    beginShape(TRIANGLE_STRIP);
    for (int x=0; x<cols; x++) {

        vertex(x*scl, y*scl, random(-10, 10));
        vertex(x*scl, (y+1)*scl, random(-10, 10));


    }
    endShape();
  }
}
```

Step 8 - Working the array into the variables

A. *replace the randoms at the bottom with the lines written* “Now we’re gonna want to put that array into use, and so we’re going to replace those random z-values at the bottom with values from the array. We’re also going to tell the for-loop responsible for the y-axis to not generate the last row, as otherwise you will have an error because we are generating every row and the row below it, and once you get to the very bottom there is no row below it, so we’re gonna wanna tell it to just not make that last row.
B. *Run the code*. “So now we should have a terrain. It’s not yet generating, and we’ll get to that, but at the moment it’s not even covering the whole view, there’s a ton of extra space. Well we can change that by simply increasing the W & H variables at the top as they are the ones that control how big the generation is. *increase W to 1200 and H to 900*. “Now it covers our view.”

```
int cols, rows;
int scl = 20;
int w = 1200;
int h =900;

float [] [] terrain;

void setup () {
  size (600, 600, P3D);
  cols=w/scl;
  rows=h/scl;
  terrain = new float [cols][rows];
  for (int y=0; y<rows; y++) {
    for (int x=0; x<cols; x++) {
 terrain[x][y] = random(-10, 10);
}
}

}

void draw () {
  background(0);
  stroke(255);
  noFill();

  translate(width/2, height/2);
  rotateX(PI/3);
```

```
  translate(-w/2, -h/2);

        //Add the -1 to the rows
  for (int y=0; y<rows-1; y++) {
    beginShape(TRIANGLE_STRIP);
    for (int x=0; x<cols; x++) {

      vertex(x*scl, y*scl, terrain[x][y]); //Replace the randoms with these
      vertex(x*scl, (y+1)*scl, terrain[x][y+1]);

    }
    endShape();
  }
}
```

Step 9 - Incorporating Perlin Noise

A. “You may have noticed our terrain is kind of all over the place, it isn’t very smooth and it doesn't really look natural, like mountains. But that’s what we want to make, we want to make realistic terrain, the only issue is that the random function quite literally will choose at random. To combat this, we need to use perlin noise, which will make random numbers but that will generally be similar to those directly beside them.”

B. “Using perlin noise will allow for a smooth terrain generation. To implement it, we’re gonna take out the random generation portion and add in a “noise” portion, which is a built in system in processing which will do the noise generation for us.” *Insert changes seen below*. “We’re gonna make it generate numbers with perlin noise for every x and y, that goes from 0 to 1, -50 to 50.” *Run program*

C. “Now, it still doesn’t look very random, because the variables that are passed into then noise function can have a bit of differentiation beyond the amount the noise function likes, so to fix that we’re going to make our own variables and pass them in. *code step C in*

D. “Now it looks much more like real terrain, and we can play around with the values to suit our fancy.” *mess around with values that we changed in steps B and C (expect the 0 and the 1 in the map function, but the 50’s are ok).

```
int cols, rows;
int scl = 20;
int w = 1200;
int h =900;

float [] [] terrain;
```

```
void setup () {
  size (600, 600, P3D);
  cols=w/scl;
  rows=h/scl;
  terrain = new float [cols][rows];
  Float yoff=0;//STEP C
  for (int y=0; y<rows; y++) {
    Float xoff=0;//STEP C
    for (int x=0; x<cols; x++) {
      terrain[x][y] = map(noise(x,y),0,1,-50, 50); //STEP B
      xoff+=0.1;//STEP C
  }
 yoff+=0.1;//STEP C
 }

}

void draw () {
  background(0);
  stroke(255);
  noFill();

  translate(width/2, height/2);
  rotateX(PI/3);
  translate(-w/2, -h/2);

  for (int y=0; y<rows-1; y++) {
    beginShape(TRIANGLE_STRIP);
    for (int x=0; x<cols; x++) {

        vertex(x*scl, y*scl, terrain[x][y]);
        vertex(x*scl, (y+1)*scl, terrain[x][y+1]);

    }
    endShape();
  }
}
```

Step 10- Animating movement

A. “Now, there are lots of ways to animate us moving forward along a 3D plane. We, of course, are going to use the simplest way. To begin, cut that entire section just below

where you declare the terrain array in your setup function, down to just under the yoff increases, and paste it into the draw function, just at the very top."

B. "Now if you run the code it will look exactly the same, so to begin animating we will make a new global variable and call it "flying", and set it to 0". *create new global variable and call it flying and set it to 0* .-"Now we're going to make it decrease by 0.1 every time the draw function runs" *program step B* "And now we're going to set the yoff to flying".
C. "And now we have animated it, and it looks like we're flying! All we really did in this step was manipulate where yoff began generating. SInce we want to move forward or backwards, and since we have rotated the 2D plane into 3D, we will need to deal with the Y axis. All we're really doing here is changing where the noise in the Y axis starts, so the noise looks like it continuously moves forward, but we're really just changing where it starts. You can get some really cool results if you tinker with some variable values, especially with W & H variables."

```
int cols, rows;
int scl = 20;
int w = 2000; //Tinker with these for step C
int h =1600;

float flying =0;

float [] [] terrain;

void setup () {
  size (600, 600, P3D);
  cols=w/scl;
  rows=h/scl;
  terrain = new float [cols][rows];


}

void draw () {

 flying -=0.01; //STEP B

  float yoff=flying;        /*STEP A
   for (int y=0; y<rows; y++) {
     float xoff=0;
     for (int x=0; x<cols; x++) {
  terrain[x][y] = map(noise(xoff,yoff),0,1,-150, 150);
  xoff+=0.1;
}
```

```
yoff+=0.1;
}         */STEP A


  background(0);
  stroke(255);
  noFill();

  translate(width/2, height/2);
  rotateX(PI/3);
  translate(-w/2, -h/2);

  for (int y=0; y<rows-1; y++) {
    beginShape(TRIANGLE_STRIP);
    for (int x=0; x<cols; x++) {

        vertex(x*scl, y*scl, terrain[x][y]);
        vertex(x*scl, (y+1)*scl, terrain[x][y+1]);
       //rect(x*scl, y*scl, scl, scl);


    }
    endShape();
  }
}
```

Step 11 - our own spin on things

A. “At this point we’ve programmed it like shiffman, but here’s where we add a spin to it. First, we were able to make it so that you can alter your angle based on the the Y coordinate of your mouse” *Program step A*
B. “Then we were also able to make it change colour depending on where your mouse was on the screen” *Program step B*. “This is especially cool as you don’t really need to copy our values at all, play around with them yourself for some cool results.”

```
int cols, rows;
int scl = 20;
int w = 2000;
int h =1600;

float flying =0;

float [] [] terrain;
```

```
void setup () {
   size (600, 600, P3D);
   cols=w/scl;
   rows=h/scl;
   terrain = new float [cols][rows];


}

void draw () {

  flying -=0.1;

  float yoff=flying;
   for (int y=0; y<rows; y++) {
     float xoff=0;
     for (int x=0; x<cols; x++) {
  terrain[x][y] = map(noise(xoff,yoff),0,1,-150, 150);
  xoff+=0.1;
}
yoff+=0.1;
}

   background(0);
   stroke(mouseX, mouseY, 150); //STEP B
   noFill();

   translate(width/2, height/2);
   rotateX(PI/(3+(mouseY/100))); //STEP A
   translate(-w/2, -h/2);

   for (int y=0; y<rows-1; y++) {
     beginShape(TRIANGLE_STRIP);
     for (int x=0; x<cols; x++) {

          vertex(x*scl, y*scl, terrain[x][y]);
          vertex(x*scl, (y+1)*scl, terrain[x][y+1]);

     }
     endShape();
   }
}
```

Finished Code:

```
int cols, rows;
int scl = 20;
int w = 2000;
int h =1600;

float flying =0;

float [] [] terrain;

void setup () {
   size (600, 600, P3D);
   cols=w/scl;
   rows=h/scl;
   terrain = new float [cols][rows];


}

void draw () {

  flying -=0.1;

  float yoff=flying;
   for (int y=0; y<rows; y++) {
     float xoff=0;
     for (int x=0; x<cols; x++) {
  terrain[x][y] = map(noise(xoff,yoff),0,1,-150, 150);
  xoff+=0.1;
}
yoff+=0.1;
}


   background(0);
   stroke(mouseX, mouseY, 150);
   noFill();

   translate(width/2, height/2);
   rotateX(PI/(3+(mouseY/100)));
   translate(-w/2, -h/2);
```

```
  for (int y=0; y<rows-1; y++) {
    beginShape(TRIANGLE_STRIP);
    for (int x=0; x<cols; x++) {

        vertex(x*scl, y*scl, terrain[x][y]);
        vertex(x*scl, (y+1)*scl, terrain[x][y+1]);
       //rect(x*scl, y*scl, scl, scl);


    }
    endShape();
  }
}
```