Generating 3D Shapes

So we have a cube now, but what if we want to change its position or size? Or what if we want a rectangular cuboid or many cuboids? With our current code, we would have to change the nodes one-by-one, which would be nuisance. What we would like is a simple method to create a cuboid with a certain position and dimensions. In other words, we want a function that maps a position and dimensions into an array of nodes and an array of edges.

Defining a cuboid

A cuboid has three dimensions: width, height and depth:

It also has a position in 3D space, giving us six parameters. There are a couple of ways we could define the position of the cube: we could define its center or we could define one corner. The former is probably more common, but I think the latter is easier to use.

Our function needs to return both the nodes and edges array. One way to return two variables is to package the variables into an object, with a key for `nodes` and a key for `edges`. Note, you can use any string to refer to the variable, I just find it easier to use the same word.

```
// Create a cuboid with a vertex at (x, y, z)
// with width, w, height, h, and depth, d.
var createCuboid = function(x, y, z, w, h, d) {
   var nodes = [];
   var edges = [];
   var shape = { 'nodes': nodes, 'edges': edges };
   return shape;
};
```

If we used that function to create a cuboid, we'd access the first node like this:

```
var object = createCuboid(0, 0, 0, 100, 160, 50);
var node0 = shape.nodes[0];
```

This will set `node0` to the first value in the `nodes` array. At the moment, however, there are no values in the nodes or edges arrays.

We define the nodes as being every combination of position with or without the corresponding dimension. Edges are defined the same way as before (except rather than define each of the edges individually first, I define them all at once). Note that this function allows you to specify negative dimensions for the cuboid.

```
var createCuboid = function(x, y, z, w, h, d) {
   var nodes = [[x, y, z ], [x, y, z+d], [x, y+h, z ], [x, y+h, z+d], [x+w, y, z ],
[x+w, y, z+d], [x+w, y+h, z ], [x+w, y+h, z+d]];

   var edges = [[0, 1], [1, 3], [3, 2], [2, 0], [4, 5], [5, 7], [7, 6], [6, 4], [0, 4],
[1, 5], [2, 6], [3, 7]];

   return { 'nodes': nodes, 'edges': edges};
};
```

We can then create a cuboid with width 100, height 160, depth 50 and one node on the origin like this:

```
var shape = createCuboid(0, 0, 0, 100, 160, 50);
```

Since our previous code just references global `nodes` and `edges` variables, we need to store our object's properties into those globals:

```
var nodes = shape.nodes; var edges = shape.edges;
```

You can see the complete code below.

```
 var backgroundColour = color(255, 255, 255);
var nodeColour = color(40, 168, 107);
var edgeColour = color(34, 68, 204);
var nodeSize = 8;


var createCuboid = function(x, y, z, w, h, d) {
    var nodes = [[x,   y,   z  ],
                 [x,   y,   z+d],
                 [x,   y+h, z  ],
                 [x,   y+h, z+d],
                 [x+w, y,   z  ],
                 [x+w, y,   z+d],
                 [x+w, y+h, z  ],
                 [x+w, y+h, z+d]];
    var edges = [[0, 1], [1, 3], [3, 2], [2, 0],
                 [4, 5], [5, 7], [7, 6], [6, 4],
                 [0, 4], [1, 5], [2, 6], [3, 7]];
    return { 'nodes': nodes, 'edges': edges };
};
var object = createCuboid(0, 0, 0, 100, 160, 50);
var nodes = object.nodes;
var edges = object.edges;


// Rotate shape around the z-axis
var rotateZ3D = function(theta) {
    var sinTheta = sin(theta);
```

```
    var cosTheta = cos(theta);

    for (var n=0; n<nodes.length; n++) {
        var node = nodes[n];
        var x = node[0];
        var y = node[1];
        node[0] = x * cosTheta - y * sinTheta;
        node[1] = y * cosTheta + x * sinTheta;
    }
};

var rotateY3D = function(theta) {
    var sinTheta = sin(theta);
    var cosTheta = cos(theta);

    for (var n=0; n<nodes.length; n++) {
        var node = nodes[n];
        var x = node[0];
        var z = node[2];
        node[0] = x * cosTheta - z * sinTheta;
        node[2] = z * cosTheta + x * sinTheta;
    }
};

var rotateX3D = function(theta) {
    var sinTheta = sin(theta);
    var cosTheta = cos(theta);
```

```
    for (var n=0; n<nodes.length; n++) {
        var node = nodes[n];
        var y = node[1];
        var z = node[2];
        node[1] = y * cosTheta - z * sinTheta;
        node[2] = z * cosTheta + y * sinTheta;
    }
};

rotateZ3D(30);
rotateY3D(30);
rotateX3D(30);

var draw= function() {
    background(backgroundColour);
        // Draw edges
    stroke(edgeColour);
    for (var e=0; e<edges.length; e++) {
        var n0 = edges[e][0];
        var n1 = edges[e][1];
        var node0 = nodes[n0];
        var node1 = nodes[n1];
        line(node0[0], node0[1], node1[0], node1[1]);
    }

    // Draw nodes
    fill(nodeColour);
    noStroke();
```

```
    for (var n=0; n<nodes.length; n++) {
        var node = nodes[n];
        ellipse(node[0], node[1], nodeSize, nodeSize);
    }

};

var mouseDragged = function() {
    rotateY3D(mouseX - pmouseX);
    rotateX3D(mouseY - pmouseY);
};

translate(200, 200);
```

Working with multiple shapes

We can create shapes with different dimensions, what if we want more than one? Whenever we want a variable number of things, an array is useful, so lets create an array of shapes.

```
var shape1 = createCuboid(-120, -20, -20, 240, 40, 40);
var shape2 = createCuboid(-120, -50, -30, -20, 100, 60);
var shape3 = createCuboid( 120, -50, -30, 20, 100, 60);
var shapes = [shape1, shape2, shape3];
```

Now we need to change the display and rotate functions to work with an array of objects. Start by wrapping the code to display edges in a for loop that loops through all the shapes:

```
// Draw edges
stroke(edgeColor);
for (var shapeNum = 0; shapeNum < shapes.length; shapeNum++) {
    var nodes = shapes[shapeNum].nodes;
    var edges = shapes[shapeNum].edges;
    for (var e = 0; e < edges.length; e++) {
        var n0 = edges[e][0];
        var n1 = edges[e][1];
        var node0 = nodes[n0];
        var node1 = nodes[n1];
        line(node0[0], node0[1], node1[0], node1[1]);
    }
}
```

And a similar for loop for displaying nodes:

```
// Draw nodes
```

```
fill(nodeColor);
noStroke();
for (var shapeNum = 0; shapeNum < shapes.length; shapeNum++) {
   var nodes = shapes[shapeNum].nodes;
   for (var n = 0; n < nodes.length; n++) {
      var node = nodes[n]; ellipse(node[0], node[1], nodeSize, nodeSize);
   }
}
```

We could add a similar for loop to each of the rotate functions, but I think it's more flexible to pass the array of nodes to each function - that way we can rotate shapes independently of one another. For example, the rotateZ3D() function would look like this:

```
var rotateZ3D = function(theta, nodes) { ... };
```

Now when we use the mouse to rotate, we have to loop through the shapes and call the function for each one:

```
mouseDragged = function() {
   var dx = mouseX - pmouseX;
   var dy = mouseY - pmouseY;
   for (var shapeNum = 0; shapeNum < shapes.length; shapeNum++) {
      var nodes = shapes[shapeNum].nodes;
      rotateY3D(dx, nodes);
      rotateX3D(dy, nodes);
   }
};
```

Make sure you remove any other calls to the rotate functions that do not pass in any nodes. You can see the complete code below.

```
var backgroundColour = color(255, 255, 255);

var nodeColour = color(40, 168, 107);

var edgeColour = color(34, 68, 204);

var nodeSize = 8;



var createCuboid = function(x, y, z, w, h, d) {

   var nodes = [[x,   y,   z  ],

            [x,   y,   z+d],

            [x,   y+h, z  ],

            [x,   y+h, z+d],

            [x+w, y,   z  ],

            [x+w, y,   z+d],

            [x+w, y+h, z  ],
```

```
            [x+w, y+h, z+d]];
    var edges = [[0, 1], [1, 3], [3, 2], [2, 0],
            [4, 5], [5, 7], [7, 6], [6, 4],
            [0, 4], [1, 5], [2, 6], [3, 7]];
    return { 'nodes': nodes, 'edges': edges };
};

var shape1 = createCuboid(-120, -20, -20, 240, 40, 40);
var shape2 = createCuboid(-120, -50, -30, -20, 100, 60);
var shape3 = createCuboid( 120, -50, -30,  20, 100, 60);
var shapes = [shape1, shape2, shape3];

// Rotate shape around the z-axis
var rotateZ3D = function(theta, nodes) {
    var sinTheta = sin(theta);
    var cosTheta = cos(theta);

    for (var n = 0; n < nodes.length; n++) {
        var node = nodes[n];
        var x = node[0];
        var y = node[1];
        node[0] = x * cosTheta - y * sinTheta;
        node[1] = y * cosTheta + x * sinTheta;
    }
};

var rotateY3D = function(theta, nodes) {
    var sinTheta = sin(theta);
```

```
    var cosTheta = cos(theta);

    for (var n = 0; n < nodes.length; n++) {
        var node = nodes[n];
        var x = node[0];
        var z = node[2];
        node[0] = x * cosTheta - z * sinTheta;
        node[2] = z * cosTheta + x * sinTheta;
    }
};

var rotateX3D = function(theta, nodes) {
    var sinTheta = sin(theta);
    var cosTheta = cos(theta);

    for (var n = 0; n < nodes.length; n++) {
        var node = nodes[n];
        var y = node[1];
        var z = node[2];
        node[1] = y * cosTheta - z * sinTheta;
        node[2] = z * cosTheta + y * sinTheta;
    }
};

//rotateZ3D(30);
//rotateY3D(30);
//rotateX3D(30);
```

```
var draw= function() {
    background(backgroundColour);
    var nodes, edges;

    // Draw edges
    stroke(edgeColour);

    for (var shapeNum = 0; shapeNum < shapes.length; shapeNum++) {
        nodes = shapes[shapeNum].nodes;
        edges = shapes[shapeNum].edges;

        for (var e = 0; e < edges.length; e++) {
            var n0 = edges[e][0];
            var n1 = edges[e][1];
            var node0 = nodes[n0];
            var node1 = nodes[n1];
            line(node0[0], node0[1], node1[0], node1[1]);
        }
    }

    // Draw nodes
    fill(nodeColour);
    noStroke();
    for (var shapeNum = 0; shapeNum < shapes.length; shapeNum++) {
        nodes = shapes[shapeNum].nodes;

        for (var n = 0; n < nodes.length; n++) {
            var node = nodes[n];
```

```
            ellipse(node[0], node[1], nodeSize, nodeSize);
        }
    }

};

mouseDragged = function() {
    var dx = mouseX - pmouseX;
    var dy = mouseY - pmouseY;

    for (var shapeNum = 0; shapeNum < shapes.length; shapeNum++) {
        var nodes = shapes[shapeNum].nodes;
        rotateY3D(dx, nodes);
        rotateX3D(dy, nodes);
    }
};

translate(200, 200);
```