```
// Openscad, animatronic dragon, work in progress, see http://github.com/zzorn/dragon

$fn = 40;
cutCurveFn = 60;

holeMargin = 1;


//testSpine(xNum=2, yNum=2);

//dragon();

neckSegment(30,
            20,
            30,
            10.3,
            3,
            spikeSize=1);


// Slice outline for a neck / spine piece.
// Use a polygon editor to define this, e.g. http://daid.mine.nu/~daid/3d/
spineOutline = [[0,18],[1,16],[5,13],[10,11],[14,12],[13,9],[14,5],[16,2],[18,0],[16,-1],[14,-5],[12,-11],[11,-
12],[6,-15],[0,-16],[-6,-15],[-11,-12],[-12,-11],[-14,-5],[-16,-1],[-18,0],[-16,2],[-14,5],[-13,9],[-14,12],[-
10,11],[-5,13],[-1,16]];
spineScaleX = 1.0/32.0;
spineScaleZ = 1.0/32.0;


// Profile for a neck / spline segment
// Small: spineProfile = [[15,32],[16,31],[16,30],[15,28],[14,23],[14,21],[15,21],[15,19],[13,16],[12,13],
[12,11],[13,11],[13,9],[11,6],[10,2],[10,0]];
spineProfile = [[13,32/*1:0,0,0,0*/] ,[14,31.88] ,[14.88,31.39],[15,31/*1:0,1,1,-3*/] ,[15.13,29.95] ,
[14.98,28.9] ,[14.69,27.92] ,[14.32,26.93] ,[13.95,25.96] ,[13.61,24.93] ,[13.43,23.9] ,[13.63,22.89] ,
[14.37,22.22],[15,22/*1:-4,1,-1,-6*/] ,[14.79,20.95] ,[14.53,19.97] ,[14.17,18.89] ,[13.79,17.89] ,
[13.39,16.97] ,[12.95,16] ,[12.5,15] ,[12.12,14.01] ,[11.91,12.96] ,[12.09,11.97] ,[12.75,11.18],[13,11/*1:-
3,2,-1,-5*/] ,[12.78,10.01] ,[12.49,8.99] ,[12.15,7.98] ,[11.75,6.97] ,[11.35,6.04] ,[10.94,5.05] ,[10.58,4.03] ,
[10.3,3.02] ,[10.12,2.01] ,[10.03,0.99],[10,0/*1:0,5,-4,0*/] ,[8.88,0] ,[7.83,0] ,[6.75,0] ,[5.67,0] ,[4.62,0] ,
[3.62,0] ,[2.58,0] ,[1.58,0] ,[0.54,0]];
spineProfileScaleX = 0.5*1.0/15.0;
spineProfileScaleY = 1.0/32.0;
spineProfileLen = 46; // Works in later openscad versions only? len(spineProfile);


// Test setup for spines
module testSpine(xNum = 3, yNum = 3, step = 10, startSize = 20) {
    spacing = max(xNum,yNum)*step*1.8+startSize+10;
    w = xNum*spacing;
    h = yNum*spacing;

    for (x = [0 : xNum-1], y = [0 : yNum-1])
        translate([(x-(xNum-1)/2)*spacing, (y-(yNum-1)/2) * spacing, 0]) {
            neckSegment(x*step + startSize,
```

```
                    y*step + startSize,
                    (x+y) * 0.5 *step + startSize,
                    10.3,
                    3);

            translate([0,0,(x+y) * 0.5 *step + startSize])
                neckSegment(x*step + startSize,
                        y*step + startSize,
                        (x+y) * 0.5 *step + startSize,
                        10.3,
                        3);
        }
}


module dragon(width = 100, length = 700, wingspan = 500, tubeDiam = 8, tendonDiam = 3) {

    spacing = 0;


    // Head
    headWidth = width * 0.6;
    headLength = length * 0.125;
    translate([-headWidth/2, bodyLength/2 + spacing + neckLength + spacing*neckSegments, 0]) {
        head(headWidth, headWidth*0.7, headLength);
    }

    // Neck
    neckWidth = width * 0.3;
    neckLength = length * 0.2;
    neckSegments = 4;
    translate([0, bodyLength/2 + spacing, 0]) {
        spine(neckWidth*1.1, neckWidth, neckLength, 1.2, 0.8, tubeDiam, tendonDiam, neckSegments,
spacing, startSpikeSize=1, startSpikeAngle=28, endSpikeSize=0.9, endSpikeAngle=26);
    }

    // Body
    bodyWidth = width*1;
    bodyLength = length * 0.175;
    translate([0,-bodyLength/2,0]) {
        body(bodyWidth, bodyLength, wingspan);
    }

    // Tail
    tailWidth = width * 0.25;
    tailLength = length * 0.5;
    tailSegments = 8;
    translate([0, -tailLength - spacing*tailSegments-bodyLength/2, 0]) {
        spine(tailWidth*1.2, tailWidth, tailLength, 0.6, 1.2, tubeDiam, tendonDiam, tailSegments, spacing,
startSpikeSize=0.5, startSpikeAngle=30);
    }
}
```

```
module body(width, length, wingspan) {

    height = width * 0.5;

    wingBoneSize = 10;

    translate([-width/2,0,0])
        cube([width,length,height]);

    translate([width/2, 0, height/2])
        wing((wingspan-width)/2, length, wingBoneSize);
    translate([-width/2, 0, height/2])
        scale([-1, 1, 1])
            wing((wingspan-width)/2, length, wingBoneSize);

}

module wing(span, width, wingBoneSize) {
    cube([span,width,wingBoneSize]);
}

module spine(width, height, length, startSize, endSize, tubeDiam, tendonDiam, neckSegments, spacing,
startSpikeSize=1, endSpikeSize=1, startSpikeAngle=28, endSpikeAngle=28) {
    segmentLength = length / neckSegments;
    for (segment = [1 : neckSegments]) {
        translate([0, segmentLength + (segment - 1) * (segmentLength+spacing), width/2]) {
            rotate([90, 0,0])
                neckSegment(width * mix(segment/neckSegments, startSize, endSize),
                        height * mix(segment/neckSegments, startSize, endSize),
                        segmentLength,
                        tubeDiam,
                        tendonDiam,
                        spikeSize=mix(segment/neckSegments, startSpikeSize, endSpikeSize),
                        spikeAngle=mix(segment/neckSegments, startSpikeAngle, endSpikeAngle));
        }
    }
}

function mix(pos, start, end) = start + pos * (end -start);

function profileScale(relativePos, scale, startScale, endScale) = mix(relativePos, startScale, endScale) * scale
* (spineProfile[spineProfileLen - 1 - relativePos * (spineProfileLen - 1)][0]);

module neckSegment(sizeX, sizeY, length, tubeDiam, tendonDiam, spikeSize = 1, spikeAngle = 40,
bendAmount = 0.3, supportSpacing = 0.75) {
    tendonX = (sizeX/2 - tendonDiam/2) - holeMargin;
    tendonY = (sizeY/2 - tendonDiam/2) - holeMargin;
    sliceCount = spineProfileLen;
    sliceH = length / sliceCount;


    depressionRelativePos = 0.5;
```

```
    depressionDepth = length * depressionRelativePos;
    bumpHeight = 0.5 * max(sizeX, sizeY) * bendAmount;

    connectorLength = depressionDepth*0.5;

    //bumpWidth = tubeDiam;// + 2 * holeMargin*2;
    //bumpLen = (width - bumpWidth) * bendAmount;
    difference() {
        union() {

            // Basic body
            difference() {
                translate([0,0,connectorLength])
                    carvedSegment(sizeX, sizeY, length, spikeSize, spikeAngle = spikeAngle);

                // Depression for connecting to the previous segment
                translate([0,0,length - bumpHeight])
                    scale([sizeX, sizeY, 1])
                        cylinder(r1=0.5, r2= 0.6 + bendAmount * depressionRelativePos,
h=depressionDepth+bumpHeight);
            }

            // Bump to pivot previous segment on
            translate([0,0,length - bumpHeight])
                scale([sizeX, sizeY, 1])
                    cylinder(r1=0.5, r2=0, h=bumpHeight);

            // Socket to connect to next segment
            translate([0,0,0])
                scale([sizeX, sizeY, length - bumpHeight])
                    cylinder(r=0.5, h=1);
        }



        // Hole for central tube carrying electronic wires
        translate([0,0,-1])
            cylinder(r=tubeDiam/2, h=length+connectorLength + 2);

        // Holes for tendons along each edge
        cylinderCircle(tendonDiam, length+connectorLength, tendonX, angleNum=2, startAngle=0);
        cylinderCircle(tendonDiam, length+connectorLength, tendonY, angleNum=2, startAngle=90);

    }
}


function polarR(a, x, y) = sqrt(pow(sin(a) * y * 0.5, 2) + pow(cos(a) * x * 0.5, 2));

module carvedSegment(origX,
                     origY,
                     sizeZ,
                     spikeSize = 1,
```

```
            spikeAngle = 28,
            padding = 1.7,
            bottomCarveDepth = 0.4,
            bumpCount = 1) {

extraSideCarveAngle = 15;
carveAreaAngle = 180.0+extraSideCarveAngle*2;

carvePadding = padding * 5;
carveDepth = 1.0 - 1.0 / carvePadding;


maxSizeX = origX * padding;
maxSizeY = origY * padding;

cutSharpness = 5;
cutLength = 7;
cutOffset = -6;

spikeRootSize = 0.4;


bottomCutAspect = 1.1;
bottomCutRadius = 0.9;
bottomCutDist = sizeZ * 1.45;

difference() {
   union() {
      scale([origX, origY, sizeZ]) {
         translate([0,0,-0.2])
            cylinder(r1=0.5, r2=padding/2*0.65, h=0.25);
         translate([0,0,-0.1])
            cylinder(r1=0.5, r2=padding/2*0.74, h=0.25);
         translate([0,0,1])
            scale([1,1,1.5])
               sphere(r=padding/2);

         // Spike
         translate([0,0.5,0.6])
            scale([spikeSize, spikeSize, spikeSize])
               rotate([270+spikeAngle,0,0])
                  cylinder(r1 = spikeRootSize, r2 = 0.05, h=1);
      }

   }

   scale([origX, origY, sizeZ]) {
      // Cut behind spike
      translate([0,0.5,1.2])
         scale([spikeSize, spikeSize, spikeSize])
            rotate([270,0,0])
               cylinder(r1 = spikeRootSize, r2 = 0.05, h=1);
```

```
        // Carve sides
        sideCarver(padding-0.35,
                cutRadiusFactor = 0.85,
                cutRootScale = 0.05,
                cutAngle=25,
                carveStartAngle = -extraSideCarveAngle,
                carvingCount=4,
                carveAreaAngle = carveAreaAngle);

        sideCarver(padding-0.04,
                cutRadiusFactor = 1.20,
                cutAngle=24,
                cutSharpness = 1,
                cutRootScale=0.75,
                carveStartAngle = -extraSideCarveAngle,
                carvingCount=2,
                carveAreaAngle = carveAreaAngle);
    }

    // Carve bottom
    translate([0,0,bottomCutDist])
        scale([origX*bottomCutAspect, origY/bottomCutAspect, sizeZ])
            sphere(r=bottomCutRadius, $fn=cutCurveFn);
    translate([0,0,bottomCutDist*1.35])
        scale([origX*bottomCutAspect*1.4, origY/bottomCutAspect*1.4, sizeZ])
            rotate([0,90,0])
                cylinder(r=1, h=2, center=true, $fn=cutCurveFn);
  }

}


module sideCarver(cutDistanceFactor,
                cutRadiusFactor = 0.9,
                cutAngle = 25,
                cutRootScale = 0.1,
                cutSharpness = 1.5,
                carveStartAngle = 90,
                carveAreaAngle = 360.0,
                carvingCount = 8) {

  carveStep = carveAreaAngle / carvingCount;

  for (ca = [carveStep*0.5 : carveStep : carveAreaAngle + 0.01 - carveStep*0.5]) {
    rotate([0,0,ca + carveStartAngle]) {
        translate([cutDistanceFactor, 0, 0]) {
            rotate([0, cutAngle, 0])
                scale([cutSharpness, 1/cutSharpness, 1])
                    cylinder(r1=cutRootScale*cutRadiusFactor, r2=cutRadiusFactor, h=2, center=true,
$fn=cutCurveFn);
        }
    }
  }
```

```
}

module cylinderCircle(diameter, length, centerDistance, angleNum=4, startAngle = 0, extend = 1) {
    angleStep = 360.0 / angleNum;
    for (ai = [0 : angleNum - 1]) {
        rotate([0,0,startAngle + angleStep * ai])
            translate([centerDistance, 0, -extend])
                cylinder(r = diameter/2, h = length + extend*2);
    }
}

module neckEnd(radius, thickness) {
    cylinder(r=radius, h=thickness);
}

module rib(radius, thickness) {
    cylinder(r=radius, h=thickness);
}

module head(headWidth, headHeight, headLength) {
    cube([headWidth, headLength, headHeight]);
}

// A box rounded along the ground plane only, with rounding equal to the size of the smallest side
module stretchedCylinder(width, depth, height, centerHeight=false) {
    heightOffset = centerHeight ? -height/2 : 0;
    if (depth > width) {
        union() {
            translate([0, -0.5*(depth-width), heightOffset])
                cylinder(r=width/2, h = height);
            translate([0,  0.5*(depth-width), heightOffset])
                cylinder(r=width/2, h = height);
            translate([0, 0, height/2 + heightOffset])
                cube([width, depth-width, height], center=true);
        }
    }
    else if (width == depth) {
        translate([0,0, heightOffset])
            cylinder(r=width/2, h = height);
    }
    else {
        union() {
            translate([-0.5*(width-depth), 0, heightOffset])
                cylinder(r=depth/2, h = height);
            translate([ 0.5*(width-depth), 0, heightOffset])
                cylinder(r=depth/2, h = height);
            translate([0, 0, height/2 + heightOffset])
                cube([width-depth, depth, height], center=true);
        }
    }

}
```