```
// save file
boolean doSave;

// new variables for controlP5
int normalsX = 0;       // control normals Length based on blow element
int looseness = 0;    // sensitivity to re-sizing itself, 0-100
float bounce = 0;    // how much bounce is in the re-shaping of itself, .005f
float crunch = 0;   // how much in the re-shaping of itself, it contracts .00
float inflater = .001;  // working on the shape
//  determine frameCount based on some pinwheel input value and that let's an
boolean blowIs = false;

// Sam's addeed time controls
float timeStamp;
float duration = 1;
boolean inflateStarted;

float rX, rY, rZ;
float frameCounter;


// inflate 3D mesh
import toxi.geom.*;
import toxi.geom.mesh.subdiv.*;
import toxi.geom.mesh.*;
import toxi.physics.*;
import toxi.physics.behaviors.*;
import toxi.physics.constraints.*;
import toxi.processing.*;

// spherical harmonics
import java.util.Iterator;
import toxi.math.waves.*;

VerletPhysics physics;
AttractionBehavior inflate;
WETriangleMesh box;

ToxiclibsSupport gfx;

// screensaver text


/////  Arduino
```

```
import processing.serial.*;
Serial myPort;
short portIndex = 5;  // select the com port, 0 is the first port

//variables to measure breath data
float fixFloatInts;      // conversion variable for floats to ints
boolean blow = false;  // are they blowing or not
boolean blow2 = false;  //check when they are not blowing
boolean timer = false;  // more ways to track time

int inst = 0;
int blowPower;           // total blow power
int blowTotal;           // store final total number of blows read in one ins
int blowRead = 0;        // total times blow strength is read
int strength = 0;        // how strong is the breath (high / low)
int runningAvg = 0;      // running average of the breath
float highest = 0;         // find higest strength read

float[] blows = new float[500];      // array stores the number of blowing in
float[] blowValues = new float[500];   // array stores all the strength value



void setup() {
  size(680, 382, P3D);
  gfx = new ToxiclibsSupport(this);
  initPhysics();


  String portName = Serial.list()[portIndex];
  myPort = new Serial(this, portName, 9600);
  myPort.bufferUntil('\n');
}

void draw() {
  pinWheelControl();

  background(200);
  fill(192);

  lights();
  directionalLight(255, 255, 255, -200, 1000, 500);
  specular(255);
  shininess(150);  //  higher value = smoother
```

```
// NEW! Inflate state control
if (inflateStarted) {
  inflateShape(); // Keep applying physics to the shape until duration time
  // take pics every 100 frames
  int takePic = frameCount % 100;
  frameCounter = 0;
  frameCounter++;
  if (takePic == 0) {
    doSave=true;
  }

  if (frameCount > timeStamp+duration) {
    println("Done!");
    physics.removeBehavior(inflate);
    inflateStarted = false;

    doSave=true;
    println("snap a pic cause we are done.");
    println("looseness  =  "+looseness+" |  duration  =  "+duration+"  |  b
  }
}

physics.update();
for (Vertex v : box.vertices.values()) {
  v.set(physics.particles.get(v.id));
}
box.center(null);
for (Vertex v : box.vertices.values()) {
  physics.particles.get(v.id).set(v);
}

box.computeFaceNormals();
box.faceOutwards();
box.computeVertexNormals();
translate(width / 2, height / 2, inflater);    // maybe you control "Camera
//rotateX((height / 2 - mouseY) * 0.01);  // mouseX and mouseY ... can also

// If you're interested in making everything spin automatically...
rotateX(rX);
rotateY(rY);
rotateZ(rZ);
rX += .0025;
rY += .0025;
rZ += .0025;
```

```
  // gfx.origin(new Vec3D(), 20);  // never see the Normals again!!!
  noStroke();
  gfx.mesh(box, true, normalsX);      // never no Normals!!!

  if (doSave) {
    saveFrame("exported_images/image-"+month()+day()+hour()+minute()+millis()
    doSave=false;
  }
}

void pinWheelControl() {
  if (strength > 5) {  // is someone blowing
    blow = true;
    blowValues[blowRead] = strength;   // STORE blowing instance data
    blowRead++;

    println(" ");
    print("Blowing ... ");
    print("(read number:   ");
    print(blowRead);
    println(")");
    print("blow strength =   ");
    println(strength);
  }
  else if (strength < 6) {  //has the blowing stopped
    blow = false;  // no one is blowing
  }

  if (blow == true && blow2 == false) {        // this is the first instance
    println(" ... blowing situation has begun ...   ");
  }

  else if (blow == false && blow2 == true) {    // this is the first instance
    timer = false;             // STOP timer

    println(" ");
    println("Blowing has stopped!!! ");
    println(" ");
    print("total blows read   =      ");
    println(blowRead);

    // now store array values for this blow instance
    for (int i = 0; i < 50; i++) {
      blowPower += blowValues[i];  // keep adding these blow values to calcul
  }

  // do some math and give me some values
  runningAvg = (blowPower/blowRead);

  print("total blow power   =   ");
  println(blowPower);
  print("average blow strength   =   ");
  println(runningAvg);


  blowValues = sort(blowValues);   // find the array you need ...
  float storage = blowValues[199];     // push it up !
  print("your strongest blow was   =   ");
  println(storage);                 // store highest blowRead to get averag
  //println(sort(blowValues));        // find highest read



  if (inst < 200) { //  if we haven't filled the STORED blowing arrays
    inst++;  //  keep adding stuff
    println(" ");
    print("Total stored blows:    ");
    println(inst);
    println(" ");
  }
  else { // if we have filled the STORED blowing array
    inst = 0;  // reset data stored for blows
    println("BLOW INSTANCE ARRAY FULL MUST RESET !!!");
  }

  //// Input from the Arduino
  duration = blowRead;
  looseness = int(map(storage, 0, 200, 0, 400));
  bounce = constrain(blowPower, 1, 200);
  bounce = map(blowPower, 1, 200, -.01, -.99);
  crunch = map(runningAvg, 0, 200, .01, 6);
  normalsX = (int)map(storage, 0, 200, 0, 100);
  startInflate();

  ////
  blowRead = 0;  // reset blowRead values to story in next blow instance ar
  blowPower = 0; // reset value as well
  blowValues = new float[200];   // clear blowValues ready for next instanc
}
```

```
  blow2 = blow;  // keep checking for blowing period equal
  if (strength > 5) {
  }
}

void serialEvent(Serial myPort) {
  fixFloatInts = float(myPort.readStringUntil('\n'));
  strength = int(fixFloatInts);
}

// Resets shape
void initPhysics() {
  box = new WETriangleMesh();
  box.addMesh(new AABB(new Vec3D(0, 1, 0), 50).toMesh());
  for (int i = 0; i < 4; ++i) {
    box.subdivide();
  }

  physics = new VerletPhysics();
  physics.setWorldBounds(new AABB(new Vec3D(), 180));
  // turn mesh vertices into physics particles
  for (Vertex v : box.vertices.values()) {
    physics.addParticle(new VerletParticle(v));
  }
  // turn mesh edges into springs
  for (WingedEdge e : box.edges.values()) {
    VerletParticle a = physics.particles.get(((WEVertex) e.a).id);
    VerletParticle b = physics.particles.get(((WEVertex) e.b).id);
    physics.addSpring(new VerletSpring(a, b, a.distanceTo(b), .01f));
  }
}

// Two functions to deform the shape
void startInflate() {
  // Trigger this state and then wait for it to finish to reset
  if (!inflateStarted) {
    print("Inflating... ");
    initPhysics();          // 1st, reset shape (otherwise things get out of
    timeStamp = frameCount; // Keep track of the frame number when the inflat
    inflateStarted = true;  // Only do these things when the state is switche
  }
}

// Add physics to the shape
```

```
void inflateShape() {
  inflate = new AttractionBehavior(new Vec3D(), looseness, bounce, crunch);
  physics.addBehavior(inflate);
}


// Debugging
void keyPressed() {
  if (key == 'r') {
    initPhysics();
  }
  else if (key == ' ') {
    doSave=true;
  }
  else if (key == 'f') {
    inflate = new AttractionBehavior(new Vec3D(), looseness, bounce, crunch);
    physics.addBehavior(inflate);
  }
  else if (key == 's') {
    box.saveAsSTL(sketchPath("exported_images/moment-"+(System.currentTimeMil
  }
}
```