DESIGNING A 3D VIDEO GAME WITH P5.js

Paul Bouthellier
Department of Computer Science and Mathematics
University of Pittsburgh-Titusville
Titusville, PA 16354
pbouthe@pitt.edu

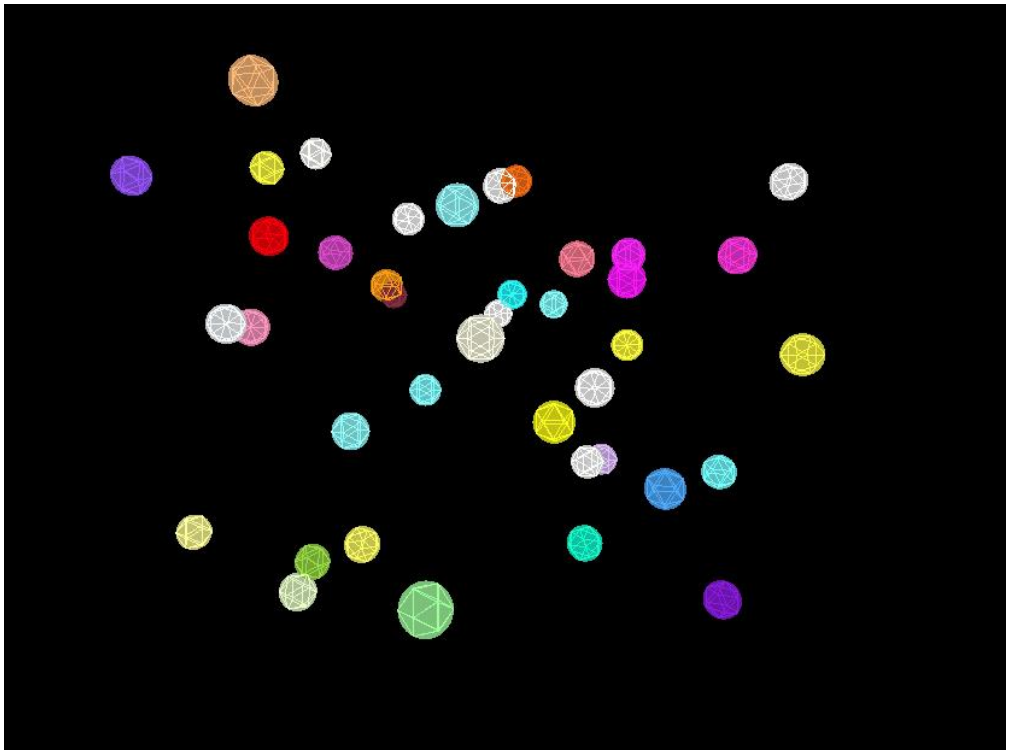

Figure 1-A 3D Game Designed with p5.js

In this paper we will discuss using the freeware package P5.js-based on the Javascript programming language-to create a 3-dimensional video game. We will use geometry, linear algebra, quaternions, calculus, and probability to create our game.

Our game is a simple 3D shooter where a person clicks (or touches the screen) on spheres rotating in 3-space to explode them. The objective is to destroy all the spheres either: as quickly as possible or using the fewest number of shots possible. One aspect of the game which adds strategy is that when a sphere explodes, it does so into twenty pieces-which can hit and destroy other spheres.

The spheres will be created using icospheres consisting of 12 vertices and 20 faces. They will be placed in 3-space using random x, y, and z coordinates for their centers.

The size, translations, and rotations of the spheres about the origin in 3-space will be accomplished by using matrices. The rotation the spheres about their own local axes will use quaternions.

The spheres actual appearance, location and size on the screen will be based on projecting the spheres onto a 2D screen taking into account that the package P5.js uses a 3D point of infinity when creating geometric objects.

We then need to discuss two problems of collision detection. First, when our mouse (or finger on a touch screen device) "hits" a sphere we need to register a hit. After a sphere is hit we need to create the normal vector for each face of the corresponding icosphere in order to be able to simulate the icosphere exploding. The second collision problem is that of figuring out if any of the faces of the exploding sphere hit any of the other spheres-causing them to explode-possibly causing a chain reaction. This second problem can be studied either as a geometric problem or as a probability problem.

At the end of this paper, instructions and code are given which will allow the reader to create the game discussed in this paper.

This paper is geared towards anyone teaching undergraduate mathematics.

What is P5.js?

P5 stands for Processing 5.

- A set freeware Javascript files for designing 2D and 3D animation in web pages
- A work in progress
- User needs to write a lot of their own code to create 3D games-hence can have an initial learning curve which is somewhat steep and requires a strong background in mathematics
- Can download the current javascript files from the site https://p5js.org/download/ (more on this latter).
- A very useful site for learning p5.js can be found at https://p5js.org/

Needed to Create The Game

- Approximating spheres by creating 12 vertex 20 face icospheres
- Randomly placing 40 such icospheres in 3D space
- Projecting the icospheres onto the xy-plane (the computer screen) using a point of infinity
- Using the projections to figure out how to "score a hit"

- After hitting a sphere-explode them using normal vectors
- Creating chain explosions using collision detection/probability models

It will be noted that at the time this game was created, none of the above were built into p5.js and hence had to be created by mathematical modeling and converting the models into code.

Creating the Icospheres

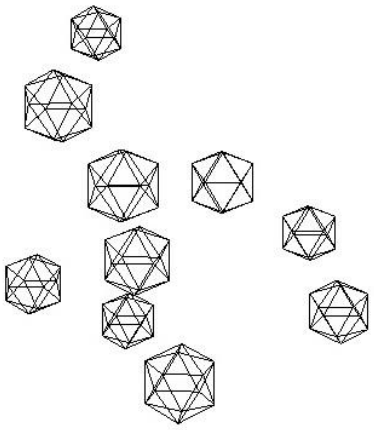

Figure 2-Basic Icospheres

Icospheres are wireframes models of spheres that are easy to construct and simple to iterate to allow more detail [1].

Our starting icospheres each shall have:

- 12 vertices
- 20 faces-Each face an equilateral triangle of the same size
- Can be easily subdivided to increase the number of faces by a factor of 4 for more detail and have
- Easy to find normal vectors

The code to create the icospheres is given as following. After listing the code I will explain the key aspects.

(1) this.faces=function(i,j,k){

(2) beginShape();

```
vertex(v[i].x,v[i].y,v[i].z);
vertex(v[j].x,v[j].y,v[j].z);
vertex(v[k].x,v[k].y,v[k].z);

(3) endShape(CLOSE);

}

(4) var t=this.scale*(1+sqrt(5))/2.0;

(5) v=[];
v[0]=createVector(-this.scale,t,0);
v[1]=createVector(this.scale,t,0);
v[2]=createVector(-this.scale,-t,0);
v[3]=createVector(this.scale,-t,0);

v[4]=createVector(0,-this.scale,t);
v[5]=createVector(0,this.scale,t);
v[6]=createVector(0,-this.scale,-t);
v[7]=createVector(0,this.scale,-t);

v[8]=createVector(t,0,-this.scale);
v[9]=createVector(t,0,this.scale);
v[10]=createVector(-t,0,-this.scale);
v[11]=createVector(-t,0,this.scale);

(6) this.display=function() {

this.faces(0,11,5);
this.faces(0,5,1);
this.faces(0,1,7);
this.faces(0,7,10);
this.faces(0,10,11);
this.faces(1,5,9);
this.faces(5,11,4);
this.faces(11,10,2);
this.faces(10,7,6);
this.faces(7,1,8);
this.faces(3,9,4);
this.faces(3,4,2);
this.faces(3,2,6);
this.faces(3,6,8);
this.faces(3,8,9);
this.faces(4,9,5);
this.faces(2,4,11);
```

```
this.faces(6,2,10);
this.faces(8,6,7);
this.faces(9,8,1);

}
```

(1) this.faces is a method that will create the 20 faces for each icosphere
(2) beginShape() will start the process of drawing and connecting the 12 vertices
(3) endShape() will actually draw the 20 faces corresponding to the 12 vertices
(4) this.scale() will scale the size of each of the icospheres-orginally 15 pixels in radius
(5) v is an array which lists the coordinates of the 12 vertices which make up each icosphere
(6) this.display is a method which will draw the icospheres onto the computer screen.

It will be noted that the 12 vertices given create a mesh of the surface of a sphere consisting of 20 equally sized equilateral triangles. One can increase the detail of the mesh by breaking each triangle (face) into 4 equal parts by using the midpoints of each edge of a given face.

Placing the Icospheres Randomly in 3-Space

First we need to understand the orientation of the axes in P5.js. They are illustrated as follows:

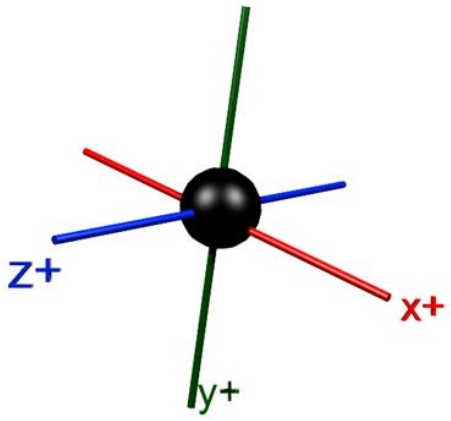

Figure 3

Two important notes:

1) The y axis is inverted
2) The origin is at the center to the screen-not the upper left corner of the screen (as is the usual location in most art packages)

The x, y, and z coordinates of the center of each icosphere are placed randomly inside a 800 by 800 by 800 pixel box as follows:

```
this.x=int(this.scale*(random()-.5));
this.y=int(this.scale*(random()-.5));
this.z=int(this.scale*(random()-.5));}
while(dist(this.x,this.y,this.z,0,0,0)<100);
this.thetastart=atan2(this.z,this.x);
```

- Where this.x, this.y, and this.z are the x, y, and z coordinates of the center of each icosphere.
- this.scale() is equal to 400 pixels
- random() is the P5.js random number generator creating a pseudo random number between 0 and 1 (including 0 but not including 1).
- The distance function is used to ensure no icospherre is too close to the origin.
- Finally, the arctan function (in radians) is used to locate the initial position of the icosphere in the xz-plane. This is useful when we want to rotate the icospheres and to locate them when "shooting" at them.

Rotations in R^3

The command to rotate each of the 40 icospheres is given by the simple p5.js command

```
rotate(frameCount*.04*speeds[b],createVector(0,-1,0));
```

where:

- The rotate method takes two arguments: the angle in radians and the vector about which we rotate. Note: we rotate about the y-axis-which in p5.js is defined by <0,-1,0>. The rotation corresponds to a right-handed system.
- speed[] is an array of 40 randomly assigned speeds-a separate rotation speed for each icosphere.
- frameCount is the number of frames that have been displayed since the animation (game) has started. By default, p5.js attempts to run at 60 frames per second. However, due to the computational complexity of this game, the game is run at 30 frames per second.

Rotations in 3-space can be accomplished by: angle-axis, rotation matrices, or by quaternions.

And yes, the program can be gimbaled locked.

Using quaternions it is also possible to rotate each icosphere about a locally defined axis.

Projection onto R^2 and "Hitting" an Icosphere

The icospheres we have created so far are mathematically placed in R^3, however have to be projected onto a 2-dimensional computer screen

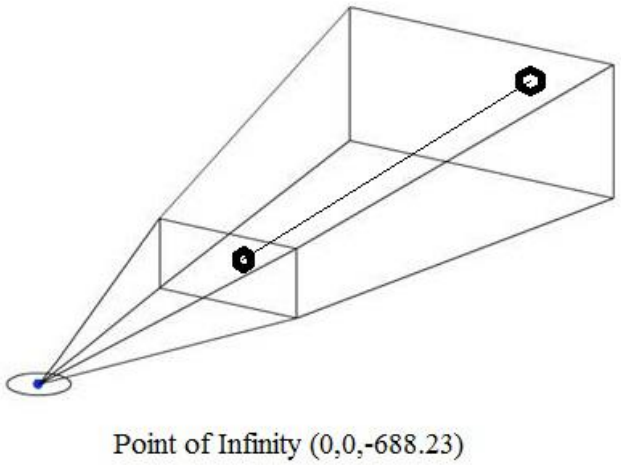

Figure 4-Projecting onto the computer screen

P5.js uses the point (0,0,-688.23) as its point of infinity. Hence it projects an icosphere in 3-space (shown as the object on the right in Figure 4) onto the computer screen. This will cause the x and y-coordinates of the icosphere to be altered. In order for us to be able to "shoot" the original icosphere, we need the new x and y-coordinates. These are found by using homogenous coordinates as follows:

Given:

```
xpos=width/2.0+cos(this.thetastart+frameCount*.04*speeds[d])*this.totaldist;
ypos=height/2.0+this.y;
zpos=sin(this.thetastart+frameCount*.04*speeds[d])*this.totaldist;
```

where xpos, ypos, and zpos are the original x, y, and z-coordinates of a given icosphere in 3-space

xproj=xpos-(zpos/(688.23-zpos))*(width/2-xpos);
yproj=ypos-(zpos/(688.23-zpos))*(height/2-ypos);

where xproj and yproj are the new x and y-coordinates of the icosphere, as projected onto the computer screen.

One aspect of this projection is that different icospheres, in different locations in R^3, are projected onto the same location on the computer screen

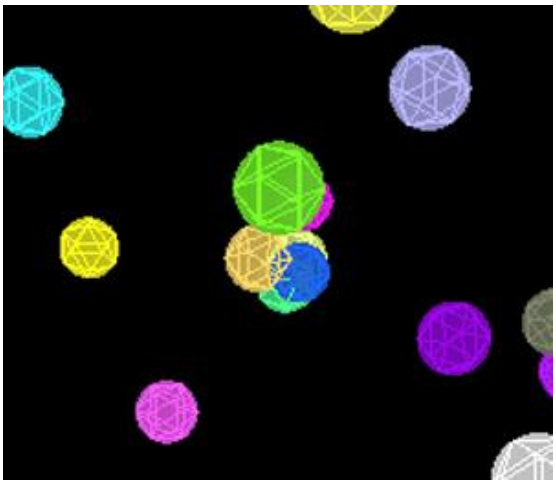

Figure 5-Overlapping icospheres

These icospheres are actually in totally different location in R^3, however, due to projection, appear to be in the same position on the computer screen. Hence, when we click on these icospheres all are "hit" even though they are in different locations in space.

Looking back on Figure 4, we see that when we click on an object on the screen, we are actually hitting every icosphere lying on a ray from the point of infinity.

To "hit" an icosphere with our mouse (or use a touch screen) we simply require that the coordinates of the mouse (when clicked) are within 20 pixels of the center of the projected icosphere. This is given by the following code:

```
if (dist(xproj,yproj,mouseX,mouseY)<20) {

        this.numhits+=1;}
```

where (mouseX, mouseY) is p5.js notation for the coordinates of the mouse when clicked. As numhits is initialized to 0 for each icosphere, any value greater than 0 indicates it has been hit.

Explosions via Cross Products

If a given icosphere is “hit” we want its faces to “blow apart”.

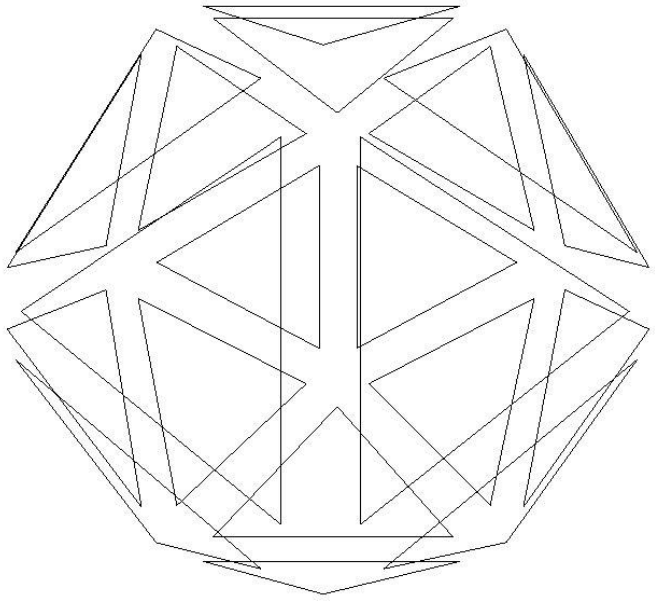

Figure 6-An Application of Cross Products

Where v[i], v[j], and v[k] are the vertices of a given face of an icosphere, the following code creates a (properly oriented) unit normal vector to the given face.

```
cp=p5.Vector.cross(p5.Vector.sub(v[i],v[j]),p5.Vector.sub(v[j],v[k])).normalize();
```

This normal vector is added to each of the vertices of the face to create the new coordinates of the face

```
bavi=p5.Vector.add(v[i],cp*apart);
bavj=p5.Vector.add(v[j],cp*apart);
bavk=p5.Vector.add(v[k],cp*apart);
```

where the ba in bavi, bavj, and bavk stands for “break-apart” and apart is how fast we want the icosphere to blow apart. These coordinates are now used to create the new faces.

Figure 7-Icospheres Blown Apart by Cross Products

Do the Faces Hit Other Spheres?

In order to cause a chain reaction-the faces of one exploding icosphere hitting another icosphere causing it to explode, we need to do some basic collision detection.

The mathematics is as follows: for each exploded icosphere we need to check if any of its 20 exploding faces hit any of the remaining faces of any of the other icospheres.

Whereas this in theory is easy-with 40 icospheres, up to 800 faces and running at 30 frames per second, it takes too much time to compute and slows the game down to molasses (and this is on a new machine with 32GB RAM). Hence we need another way:

1. Compute the surface area of the hit sphere
2. Compute the distance of the center of the sphere to the center of each other sphere
3. Compute surface area of this sphere
4. The ratio of these two surface areas will give the probability of a hit

Is this Perfect?

No.

However, it is much less computationally complex then trying to detect if each exploding face of a hit sphere actually hits another sphere.

This method has one other main advantage: The actual chance the faces of one icosphere with hit another icosphere is actually quite small-which makes for a boring game. All we need to do is to increase the chance of a collision by a factor of 6. Yes, it is not mathematically accurate, but makes the game much more fun.

It will be noted that if two icospheres intersect on the computer screen they will not explode without being hit by the user.

Future Directions

There are many ways to extend the results given here. Several are as follows:

- Create meshes for shapes other than just spheres. Examples could be chess pieces, pyramids, tori, …. The possibilities are only limited by one's imagination and the speed and amount of memory of the user's computer or phone.
- Exploding the objects into patterns other than just their faces. Fireworks would be one example.

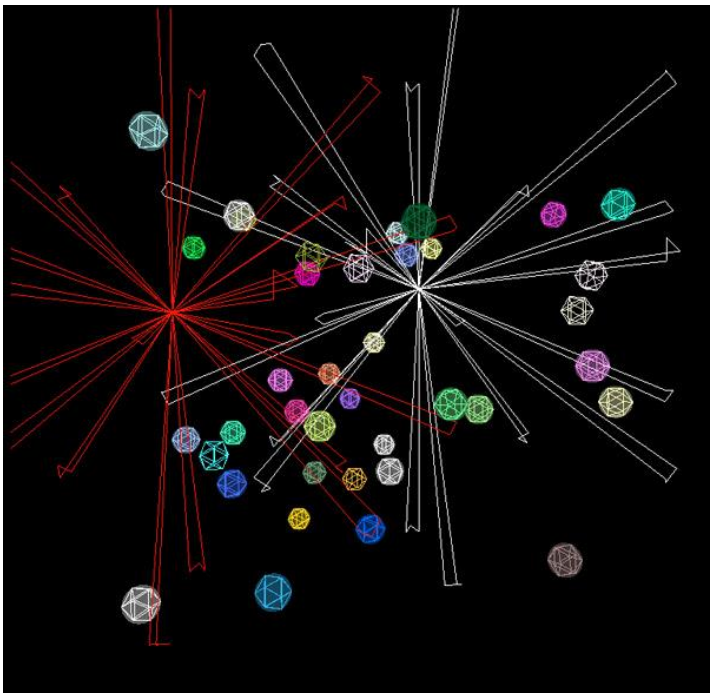

Figure 8-A Mathematical Fireworks Display

Summary

The mathematics needed to create this basic game are as follows:

- Creating meshes for the surfaces of spheres
- Using random functions to place the objects in 3-space
- Using rotation matrices and quaternions to rotate the objects in space
- Understanding gimbal lock to get out of gimbal lock.
- Projecting 3-dimensional objects onto a two dimensional computer screen using homogenous coordinates
- Using cross products to create normal vectors for the faces of the icosphere
- Probability to compute the chance of the faces of an exploding icosphere hitting another icosphere.

The above is just a small subset of the mathematics that can be illustrated with computer graphics and game design.

Working Code for the Game

Below are the five files needed to create the game.

In order to create the game:

a. Go the https://github.com/processing/p5.js/releases/tag/0.5.11 and download the file p5.min.js
b. Go to https://github.com/processing/p5.js/releases/tag/0.5.4 and download the file p5.dom.js

2. Create a folder called libraries and put both the above files in the libraries folder
3. Copy the following code for Spheregame and save it as Spheregame.html
4. Code the following code for index17 and save it as index17.js
5. Code the following code for workingsphere17 and save it as workingsphere.js
6. Place the three files from steps 3, 4,and 5 into the same folder as the libraries folder (but not inside the libraries folder)
7. Double click on Spheregame.html and enjoy (hopefully)

Spheregame.html

```
<html>
<head>
<meta charset="utf-8">
<script language="javascript" type="text/javascript" src="libraries/p5.min.js"></script>
<script language="javascript" src="libraries/p5.dom.js"></script>
```

```
<script language="javascript" type="text/javascript" src="index17.js"></script>
<script language="javascript" type="text/javascript" src="workingsphere17.js"></script>

<style>
body { background-color: #000;}
.hithere {
position: relative;
margin-left: auto;
margin-right: auto;
height: 800px;
width: 800px;
}
</style>
</head>
<body>
<div class="hithere" id="hithere">
</div>
</body>
</html>
```

index17.js

```
var n, s,counter,currentpos,i,j,k, rx,ry,rz,rapart=0,prob=0,carray=[], distexplode,
spheres=[],speeds=[],axes_rot=[],forcos=0,forsin=0,forrotations=[];

function setup() {
//create canvas
createCanvas(800, 800,WEBGL).parent('hithere');
frameRate(30);
ambientLight(255,255,255);
ambientLight(255,255,255);
n=40;
for (var a=0;a<n;a++) {
spheres.push(new Meshicosphere());}
rx=ry=rz=0;
i=j=k=0;
counter=-1;
distexplode=10;
currentpos=[];

for (var q=0;q<n;q++) {
carray[q]=0;}

for (var u=0;u<n;u++) {
```

```
speeds[u]=1+.5*random();}

for (var w=0;w<n;w++) {
axes_rot[w]=createVector(.5-random(),.5-random(),.5-random()).normalize();}}

function draw() {
background(0,0,0);
normalMaterial();


for (var b=0;b<n;b++) {
counter++;
if (spheres[b].apart_done==false) {
push();
{rotate(frameCount*.04*speeds[b],createVector(0,-1,0));}
spheres[b].move();
spheres[b].display(b);
forcos=cos(frameCount*.02*speeds[b]);
forsin=sin(frameCount*.02*speeds[b]);

currentpos[counter%n]=[width/2.0+cos(spheres[b].thetastart+frameCount*.04*speeds[b])
*spheres[b].totaldist,
height/2.0+spheres[b].y,
sin(spheres[b].thetastart+frameCount*.04*speeds[b])*spheres[b].totaldist];
pop();

if (spheres[b].hit==true && spheres[b].numhits>1) {
carray[b]+=1;

for (var c=0;c<n;c++) {
rapart=dist(currentpos[b][0],currentpos[b][1],
currentpos[b][2],currentpos[c][0],currentpos[c][1], currentpos[c][2])/spheres[b].spsize;

if (rapart<=2) {prob=1;}

else prob=5*speeds[b]/(pow(rapart-1,2)-1);

if (b!=c && random()<prob && carray[b]==1)


{spheres[c].numhits+=1;

if (spheres[c].numhits==2)
{spheres[c].hit=true;}}}}}}}
```

```
//mouseMoved

function mousePressed() {
for (var d=0;d<spheres.length;d++) {
spheres[d].clicked(d);}}
```

workingsphere17.js

```
//Create the spheres

function Meshicosphere() {
this.scale=400;
this.spsize=15;
do {
this.x=int(this.scale*(random()-.5));
this.y=int(this.scale*(random()-.5));
this.z=int(this.scale*(random()-.5));}
while(dist(this.x,this.y,this.z,0,0,0)<100);
this.thetastart=atan2(this.z,this.x);
this.totaldist=dist(this.x,this.z,0,0);
this.i=0;
this.j=0;
this.k=0;
this.hit=false;
this.apart=0;
this.apart_done=false;
this.numhits=0;


var forcolor=[];

for (var v=0;v<n;v++) {
forcolor[v]=createVector(255*random(),255*random(),255*random());}


var
thefaces=[createVector(0,11,5),createVector(0,5,1),createVector(0,1,7),createVector(0,7,
10),createVector(0,10,11),createVector(1,5,9),createVector(5,11,4),createVector(11,10,2)
,createVector(10,7,6),createVector(7,1,8),createVector(3,9,4),createVector(3,4,2),createV
ector(3,2,6),createVector(3,6,8),createVector(3,8,9),createVector(4,9,5),createVector(2,4,
11),createVector(6,2,10),createVector(8,6,7),createVector(9,8,1)];
```

```
var t=(1+sqrt(5))/2.0;

v=[];
v[0]=createVector(-1,t,0),v[1]=createVector(1,t,0),v[2]=createVector(-1,-
t,0),v[3]=createVector(1,-t,0),v[4]=createVector(0,-
1,t),v[5]=createVector(0,1,t),v[6]=createVector(0,-1,-t),v[7]=createVector(0,1,-
t),v[8]=createVector(t,0,-1),v[9]=createVector(t,0,1),v[10]=createVector(-t,0,-
1),v[11]=createVector(-t,0,1);

for (var i=0;i<12;i++) {
        v[i]=v[i].normalize();
        v[i]=p5.Vector.mult(v[i],this.spsize);}

var cp;
var bavi, bavj,bavk;
var face_counter=-1;

////////////////////////////////////////////////////////////////////////////////////////////////////////////////

var fnarray=[];
var cpnormalized;


Meshicosphere.prototype.face_normals=function(i,j,k,w) {
cpnormalized=p5.Vector.cross(p5.Vector.sub(v[i],v[j]),p5.Vector.sub(v[j],v[k])).normali
ze();

fnarray[w]=cpnormalized;}

for (var i=0;i<20;i++) {
this.face_normals(thefaces[i].x,thefaces[i].y,thefaces[i].z,i);}


////////////////////////////////////////////////////////////////////////////////////////////////////////////////

Meshicosphere.prototype.faces=function(i,j,k,w,b){

face_counter++;
cp=p5.Vector.mult(fnarray[w],this.apart);

bavi=p5.Vector.add(v[i],cp),bavj=p5.Vector.add(v[j],cp),bavk=p5.Vector.add(v[k],cp);

if (this.spsize==0) {
bavi=p5.Vector.mult(bavi,0),bavj=p5.Vector.mult(bavj,0),bavk=p5.Vector.mult(bavk,0);
}
```

```
beginShape();
push();ambientMaterial(forcolor[b].x,forcolor[b].y,forcolor[b].z);

vertex(bavi.x,bavi.y,bavi.z),vertex(bavj.x,bavj.y,bavj.z),vertex(bavk.x,bavk.y,bavk.z);
/*if (this.hit==true  && random()<.5)
{vertex(10,10,10);}*/
pop();
endShape(CLOSE);

face_center[(face_counter)%20]=p5.Vector.div(p5.Vector.add(p5.Vector.add(bavi,bavj),
bavk),3.0);

return face_center;}

Meshicosphere.prototype.move=function() {

if (keyIsDown(LEFT_ARROW))
{this.i++;}

if (keyIsDown(DOWN_ARROW))
{this.j++;}

if (keyIsDown(UP_ARROW))
{this.k++;}

translate(this.x,this.y,this.z);

rotate(this.i*.05,createVector(1,0,0));
rotate(this.j*.05,createVector(0,-1,0));
rotate(this.k*.05,createVector(0,0,1));}

var xpos,ypos, xproj,yproj;

Meshicosphere.prototype.clicked=function(d) {

xpos=width/2.0+cos(this.thetastart+frameCount*.04*speeds[d])*this.totaldist;
ypos=height/2.0+this.y;
zpos=sin(this.thetastart+frameCount*.04*speeds[d])*this.totaldist;

xproj=xpos-(zpos/(688.23-zpos))*(width/2-xpos);
yproj=ypos-(zpos/(688.23-zpos))*(height/2-ypos);

if (dist(xproj,yproj,mouseX,mouseY)<20) {
this.tester();}}
```

```
Meshicosphere.prototype.tester=function(){

this.numhits+=1;

if (this.numhits>1) {this.hit=true;}}

var face_center=[];

Meshicosphere.prototype.display=function(b) {

//After hit blow apart

if (this.hit==true && this.apart<1000 && this.apart_done==false){
this.apart+=10;}

if (this.hit==true && this.apart>=1000){
this.apart=0;
this.spsize=0;
this.apart_done=true;}

if (this.numhits==0) {
push();

ambientMaterial(forcolor[b].x,forcolor[b].y,forcolor[b].z,200);

sphere(15);
pop();}


for (var i=0;i<20;i++) {

   this.faces(thefaces[i].x,thefaces[i].y,thefaces[i].z,i,b);}}}
```

References:

[1] Creating an icosphere mesh in code http://blog.andreaskahler.com/2009/06/creating-icosphere-mesh-in-code.html.