# In Search of a Beautiful 3D Mandelbrot Set

by Rudy Rucker
Started Sept 5 - 14, 1988, while working at Autodesk with Mathematica 1.0
Upgraded in March, 2009, working with Mathematica 7.
Most recent update: September 24, 2009.

## Background

The Mandelbrot set M is defined in terms of a plane map mandelmap[z,c] which takes z into z*z + c.  We add plane points by adding their cartesian components---this is easy.  Multiplying plane points (in particular computing z times z) is done by pretending the plane points represent complex numbers.   A problem in defining a 3D Mandelbrot set is that there is no good 3D analogue of complex numbers, and therefore there's no obvious best way to "square" a 3D point (x, y, z).

Various other ideas have been tried for finding the platonic 3D bulb, such as using quaternions. The quaterion approach gives taffy-like structures lacking the warty smooth quality we seek.  Daniel White discusses the quest and some of the approaches in his web page http://www.skytopia.com/project/fractal/mandelbrot.html

What do we want?  A "Mandelbulb," a root - like object that' s like a big sphere with a dimple in the bottom and with  bulbs on it, and further warts on the bulbs.  I wrote about this object in my 1987 science-fiction short story "As Above, So Below,"
www.rudyrucker.com / pdf / 3 dmandelbrotsetstory.pdf

## The Cubic Connectedness Map

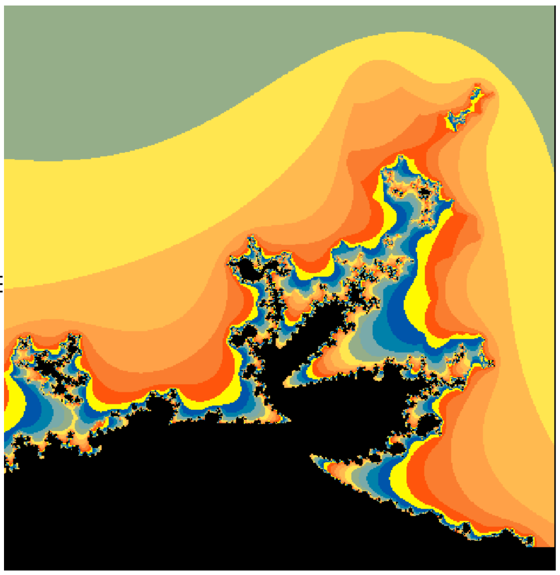
I'll mention here as an aside a completely different approach to looking for a 3D Mandelbrot set. The idea is to take a 3D cross section of a 4D fractal called the Cubic Connectedness Map, which was first investigated by Adrian Douady, John Hubbard and John Milnor.

In a nutshell, if k is a complex number, we define Mk to be the set of complex numbers c such that both k and -k, when plugged in for z, lead to a finitely large sequence of iterates in the map z = z^3 - 3*k*z + c. The set of all Mk in k-c space forms the four-dimensional Cubic Connectedness map or CCM.

I created some software for finding 2D cross-sections of the Cubic Connectedness map in the Autodesk CHAOS program, see my remarks at http://www.cs.sjsu.edu/faculty/rucker/cubic_mandel.htm Note that you can download the (old) Windows version of this program at this same site.

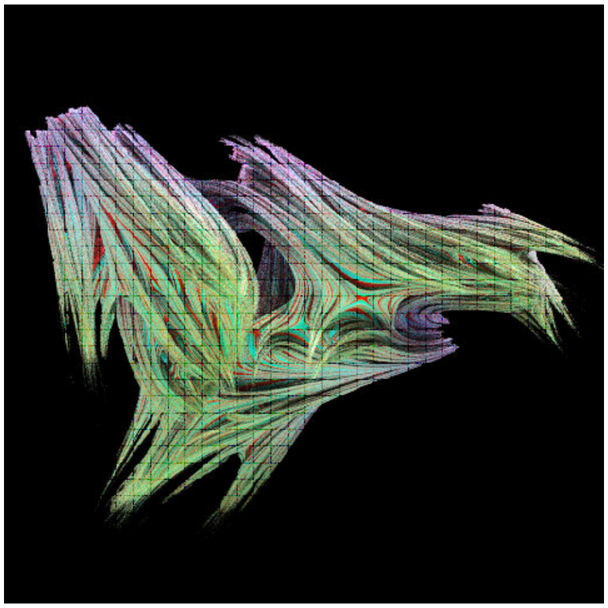
I think I got the idea for the CCM fractal from the technical papers:
Adrian Douady and John Hubbard, "On the Dynamics of Polynomial-like Mappings" Bodil Branner and John Hubbard,
"The iteration of cubic polynomials Part I: The global topology of parameter space",
and "The iteration of cubic polynomials Part II: Patterns and parapatterns"

Another of the surprisingly few links on this rich topic is an article by the amateur mathematician Ingvar Kullberg, "The Cubic Tutorial," concerning a plug-in that generates these images with the popular Ultra Fractal software.



Shown above is a detail of the Mk you get when k = -0.4055 -0.1135 i. The detail is the image is in the area centered on (0.12, 0.87) in the c plane.

By incrementing up and down through the k parameters, I had gotten the impression that by "stacking" some of these 2D slices while varying a single real-number parameter, such as the real part of k, we could get a nice 3D Mandelbulb, and I still can't quite believe this isn't true.  But Paul Nylander recently rendered a run that looks discouraging.



I still have some slight hope that Paul's rendering algorithm might be in some way not quite what we want, or that, perhaps we need to do something other than simple stacking of 2D cross-sections in order to find the Mandelbulb within the CCM.

## A Spherical Coordinates Algorithm

A different way of thinking of complex number multiplication is to express (x,y) in polar coordinates (norm,theta). In these coordinates, multiplying two complex numbers means adding their norm-values and adding their thetas. This is readily generalized to traditional 3D spherical coordinates. A point can be represented by a triple (norm,theta,phi),
where phi is the azimuth elevation above the xy plane, and theta is the polar theta of the point's projection onto the xy plane. And then multoplying two of these numbers means multiplying their norm-values, adding their thetas, and adding their phis.

So squaring a complex number in the plane passes from (r, theta) to (r*r, 2 theta). And in spherical 3D coordinates, (r, theta, phi) becomes (r*r, 2 theta, 2 phi).

One encouraging thing about this approach is that we can be sure that the set we get will have a famililar Mandelbrot set as its interesections with both the xy plane and the xz plane...for in the xy plane, phi is identically zero; and in the zy plane, theta is identically zero.

In the code below, think of p as a point with cartesian coordinates, and ap as a point with spherical coordinates. Re. the *Mathematica* foibles, recall that 0. is simply the real number constant for 0, that N is an operator that turns an algebraic experession into a number, and that the ArcTan[ x, y] gives the angle whose tangent is y / x, taking into account which quadrant (x, y) lies in. I feel slightly paranoid about ArcTan[ 0. 0.], so I define a safearctan that returns a 0.0 in the intedeterminate case.

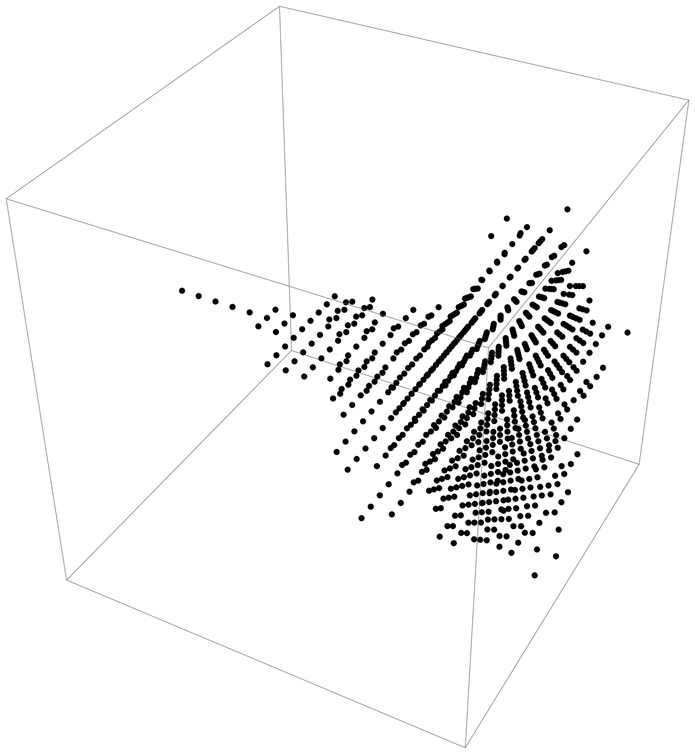
For this try at the Mandelbulb, I "square" a cartesian-coordinate vector (x, y, z) by creating, on the fly, the corresponding spherical-coordinate vector (norm, theta, phi), passing to (norm norm, 2 theta, 2 phi), and then converting this back into cartesian-coordinate form. This is inefficient, I should write a pure cartesian version like in White's algorithm shown in the next section.

I call my spherical coordinate-based squaring method scsquare, and I call the map scmandelmap and the resulting set scmandel.

```
In[79]:= norm[p_] := N[Sqrt[p . p]];
         safearctan[x_, y_] := If[x ≠ 0. || y ≠ 0., ArcTan[x, y], 0.0]
         xynorm[p_] := N[Sqrt[p[[1]] + p[[2]] p[[2]]]]

         theta[{0, 0, 0}] = theta[{0., 0., 0.}] =  0
         theta[p_] :=  N[safearctan[p[[1]], p[[2]]]]
         phi[{0, 0, 0}] = phi[{0., 0., 0.}] = 0
         phi[p_] := N[safearctan[xynorm[p],  p[[3]]]]
         spherical[p_] := N[{norm[p], theta[p], phi[p]}];

         xc[ap_] := N[ap[[1]] Cos[ap[[3]]] Cos[ap[[2]]]];
         yc[ap_] := N[ap[[1]] Cos[ap[[3]]] Sin[ap[[2]]]];
         zc[ap_] := N[ap[[1]] Sin[ap[[3]]]];
         cartesian[ap_] := N[{xc[ap], yc[ap], zc[ap]}];

         scsquare[p_] := N[cartesian[{p.p, 2 theta[p], 2 phi[p]}]];
         scmandelmap[p_, c_] := N[scsquare[p] + c];

         scmandel[maxnorm_, maxiterate_, gridsteps_] :=
          (set = {PointSize[.009]};
           Do[Do[Do[
              test = {-2. + (3.25 i) / gridsteps,
                -1. + (2. j) / gridsteps, -1. + (2. k) / gridsteps};
              hit = test;
              Do[hit = scmandelmap[hit, test];, {m, 1, maxiterate, 1}];
              If[norm[hit] < maxnorm, AppendTo[set, Point[test]]],
              {i, gridsteps}], {j, gridsteps}], {k, gridsteps}];
           Show[Graphics3D[set], BoxRatios → {1, 1, 1}])

Out[82]= 0

Out[84]= 0
```

The next cells  contain commands to generate views of a run gotten with scmandel[maxnorm, maxiterate, gridsteps]. To quickly check if it's working, I might run it with (2, 4, 3).   I've successfully run it with 2, 10, 20 and with 2, 20, 30.
Note that you can click and spin the graphics box.

```
scmandel[2, 20, 30]
```



## White's Algorithm

In 2007, a similar approach was used by Daniel White, who posted about his search for a nice 3D Mandelbort shape on his website mentioned above,
http://www.skytopia.com/project/fractal/mandelbrot.html
and on the fractalforums site at
http://www.fractalforums.com/3d-fractal-generation/true-3d-mandlebrot-type-fractal/msg4109/?PHPSESSID=e16569e0fdbf61cb290d61d04a9c0896#msg4109

White uses the world "Mandelbulb" for the image we seek, although he's not sure who first used the word.

White's method of "squaring" a point (x, y, z) is shown below. Note that it's in C language, where atan2 (y, x) is the arc tangent of y/x, expressed in radians, and trying to evalutate atan2 (0.0, 0.0) crashes the program. White's method of squaring seems much like mine, I think he's adding those multiples of pi because he measures phi from the z axis rather than from the xy-plane as I was doing.

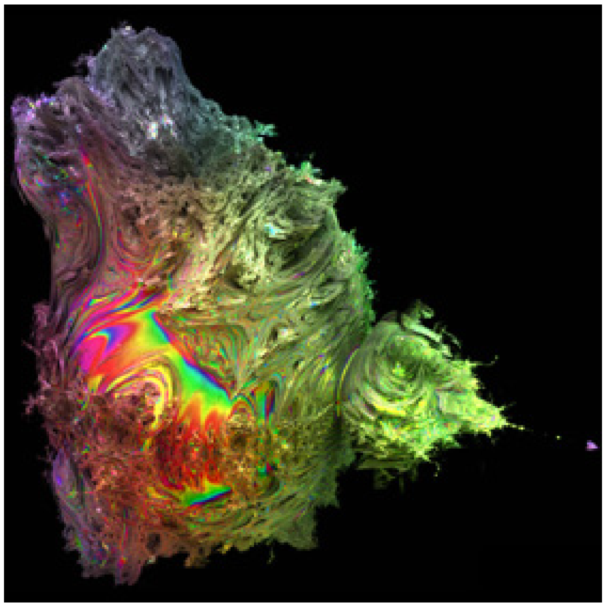
```
double pi = 3.14159265;
double r    = sqrt (x*x + y*y + z*z );
double phi = atan2 (sqrt (x*x + y*y) , z  )  ;
double theta = atan2 (y , x);
newx = (r*r) * sin ( phi*2 + 0.5*pi ) * cos (theta*2 + pi);
newy = (r*r) * sin ( phi*2 + 0.5*pi ) * sin (theta*2 + pi);
newz = (r*r) * cos ( phi*2 + 0.5*pi)
```

Paul Nylander blogged about "hypercomplex fractals" at
http://www.bugman123.com/Hypercomplex/index.html

A guy who call himself Karl131058 pointed out on fractalforum that we can simplify Whit'es algorithm as follows, with an initial condition to avoid dividing by zero in the main three formulae at the end (without this condition all the points run away and you get an empty set). The point is that you don't really need to look at the arctan of an angle if you're planning to take the sine or cosine of it, and even if you're doubling the angle, you can use the angle doubling formulae cos(2a)=cos^2 a - sin^ a  and sin(2a)=2cos a sin a.

```
if( abs(y) < really_small_value )
{
        newx = x*x-z*z
        newy = 0
        newz = -2*z*sqrt(x*x)
        }
else
{
        newx = ( x*x + y*y - z*z )*( x*x - y*y) / ( x*x + y*y )
        newy = 2 * ( x*x + y*y - z*z )*x*y / ( x*x + y*y )
        newz = - 2 * z * sqrt( x*x + y*y )
}
```

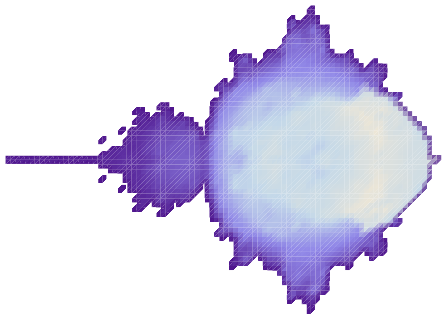
Nylander shows a rendered image of a White Mandelbulb that, as White already noted, doesn't  look like the smooth bulbous shape we'd dreamed of.  It's close, but it has the taffy/whipped-cream/spun-glass excrescenses that we want to sand away.



Nylander also shows a scrap of  Mathematica code to quickly display
White' s Mandelbulb as a depth map by slowly marching tiny cubes (voxels)
toward the boundary.

```
(*runtime:1 minute,increase n for higher resolution*)
n = 100; Norm[x_] := x.x;
Square[{x_, y_, z_}] := If[x == y == 0, {-z^2, 0, 0},
    Module[
     {a = 1 - z^2 / (x^2 + y^2)},
     {(x^2 - y^2) a, 2 x y a, -2 z Sqrt[x^2 + y^2]}]];
Mandelbrot3D[pc_] :=
  Module[{p = {0, 0, 0}, i = 0},
   While[i < 24 && Norm[p] < 4, p = Square[p] + pc; i++]; i];
image = Table[z = 1.5; While[z >= -0.1 &&
     Mandelbrot3D[{x, y, z}] < 24, z -= 3 / n];
   z, {y, -1.5, 1.5, 3 / n}, {x, -2, 1, 3 / n}];
ListDensityPlot[image, Mesh -> False, Frame -> False,
 PlotRange -> {-0.1, 1.5}]
```

```
SetDelayed::write : Tag Norm in Norm[x_] is Protected. >>
```



## Higher-Degree White's Algorithm

Reasoning by analogy, Daniel White proposed that we look at higher-degree versions of the argument-azimuth approach. He uses this formula for raising a vector to the power n:

He converts (x, y, z) into (r, theta, phi) as before

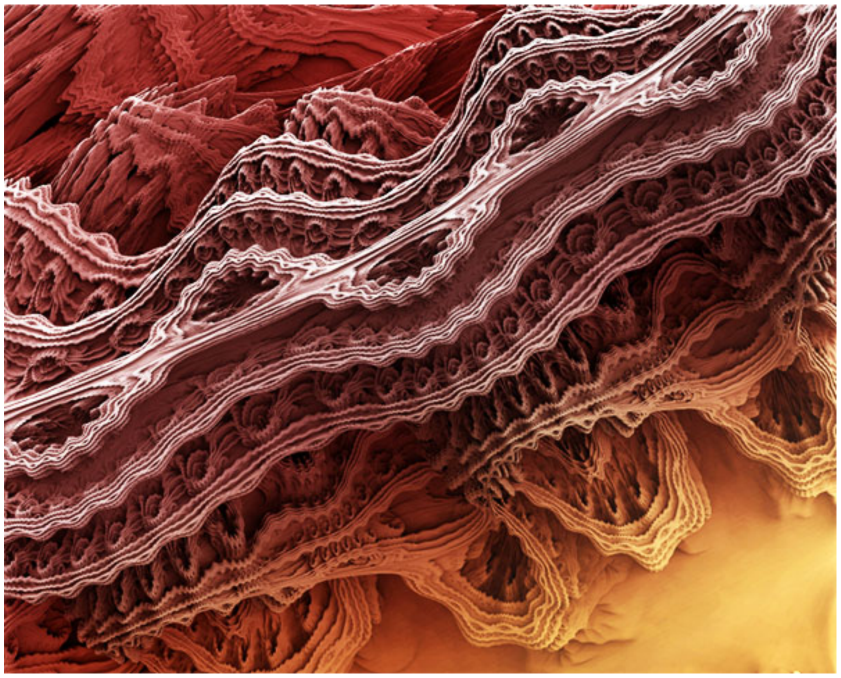
r=sqrt($x^2+y^2+z^2$)
theta=atan(y/x)
phi=atan(z/sqrt($x^2+y^2$))

And then he computes (x, y, z) to the n_th power as

newx = r^n * cos(n*theta)cos(n*phi)
newy = r^n * sin(n*theta)cos(n*phi)
newz = r^n * -sin(n*phi))

I'm not sure what he does to avoid problems along the z-axis, where x and y are both zero, I suppose he just sets phi equal to the sign of z times pi over 2 there.  More information about this approach can be found on the Fractal Forums discussion page.

I'll include one image, a detail of a rendering by Daniel White, of the degree 8 agument-azimuth 3D Mandelbrot set, with only four iterations of the map.



## Why Doesn't the Argument-Azimuth 3d Mandelbrot Fully Work?

Although we get some good images One problem might be that there' s some asymmetry in how we get the two angles theta and phi in the spherical coordinates, in that theta is an argument angle the xy plane and the other is an azimuth elevation above it.  So maybe it' s not surprising that simply doubling those angles doesn' t produce a smooth bulbous 3 d mandelbulb.  It would be nice if the angles were more similar in origin.

Another problem could be that, when we double two angles we're in some sense moving the point too violently. Perhaps you should be multiplying the angles not by two, but by some weighting factors mul1 and mul2 which might depend on whether, say the point is closer to the xy plane or to the xz plane.

### A Two - Azimuth Version

One idea is to define both angles as azimuths like phi.
An alternate idea, which I look at in the next subsection, is to define both angles as arguments like theta.

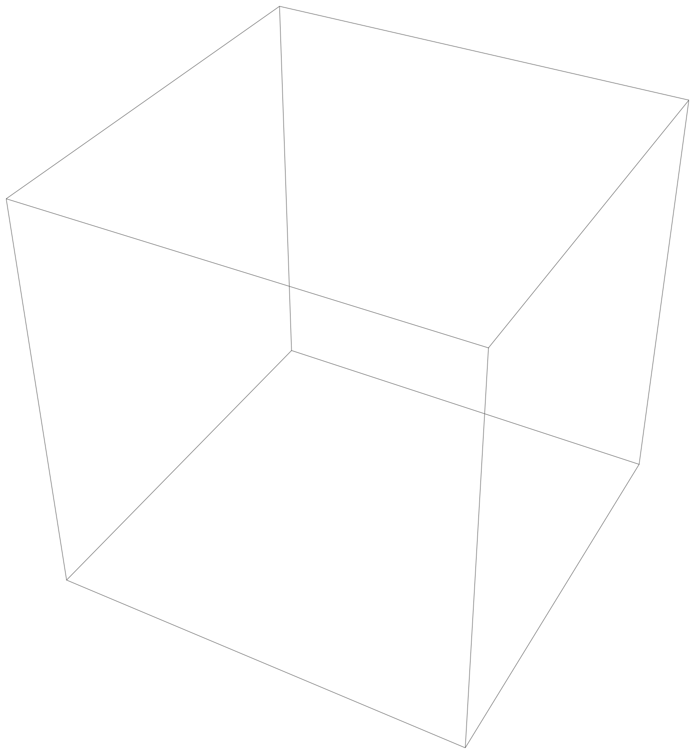
For the two - azimuth version, we let phi can be the elevation above the xy plane as before, and now we suppose that eta is the azimuth above, say, xz plane.  If the point lies in the xy plane, note that theta and eta are the same, for in the xy plane, the xznorm is the same as x.  But if the point is off the xy plane then they' re different.  In converting from
(r, eta, phi) to (x, y, z),  note that
xznorm = r cos (eta); xynorm = r cos (phi);  xznorm*xznorm + y*y = r*r; and xynorm*xynorm + z*z = r*r
So I build a 3 D mandelbrot algorithm as before, again not worrying initially about making the rules fast.

To distinguish this from my intial "sc" approach in this notebook, I write "taz" in front of the fucntion names, such as tazsquare, tazmandelmap and tazmandel.

This doesn't seem to work---as I continue tweaking the code below, I seem to get  blank graphical boxes which suggests that maybe all the points run away to infinity under this iteration and we don't get any set.  Probably I need to be checking for division by zero somewhere.  It would be a good idea to convert this into x, y, z form and get rid of the angles and then see what I really have.

```
norm[p_] := N[Sqrt[p . p]];
safearctan[x_, y_] := If[x ≠ 0. || y ≠ 0., ArcTan[x, y], 0.0]
xynorm[p_] := N[Sqrt[p[[1]] p[[1]] + p[[2]] p[[2]]]]
xznorm[p_] := N[Sqrt[p[[1]] p[[1]] + p[[3]] p[[3]]]]

eta[{0, 0, 0}] = eta[{0., 0., 0.}] = 0
eta[p_] := N[safearctan[xznorm[p], p[[2]]] ]
phi[{0, 0, 0}] = phi[{0., 0., 0.}] = 0
phi[p_] := N[safearctan[xynorm[p], p[[3]]]]
tazspherical[p_] := N[{norm[p], eta[p], phi[p]}];

tazsphxznorm[ap_] := ap[[1]] Cos[ap[[2]]]
tazsphxynorm[ap_] := ap[[1]] Cos[ap[[3]]]
tazyc[ap_] :=
  N[Sqrt[ap[[1]] ap[[1]] - tazsphxznorm[ap] tazsphxznorm[ap]]];
tazzc[ap_] :=
  N[Sqrt[ap[[1]] ap[[1]] - tazsphxynorm[ap] tazsphxynorm[ap]]];
tazxc[ap_] := N[Sqrt[ap[[1]] ap[[1]] -
     tazyc[ap] ytazc[ap] - tazzc[ap] tazzc[ap] ]];
tazcartesian[ap_] := N[{tazxc[ap], tazyc[ap], tazzc[ap]}];

tazsquare[p_] := N[tazcartesian[{p.p, 2 eta[p], 2 phi[p]}]];
tazmandelmap[p_, c_] := N[tazsquare[p] + c];
tazmandel[maxnorm_, maxiterate_, gridsteps_] :=
 (set = {PointSize[.009]};
  Do[Do[Do[
     test = {-2. + 3.25 i/gridsteps, -1. + 2. j/gridsteps, 1. + 2. k/gridsteps};
     hit = test;
     Do[hit = tazmandelmap[hit, test];, {m, 1, maxiterate, 1}];
     If[norm[hit] < maxnorm, AppendTo[set, Point[test]]],
     {i, gridsteps}], {j, gridsteps}], {k, gridsteps}];
  Show[Graphics3D[set], BoxRatios → {1, 1, 1}])
```

```
tazmandel[2, 4, 3]
```



## A Two - Argument Version

So now let's try something that's perhaps more natural than using two azimuths.  What if we use two argument-style angles.

Maybe we'll use the same angle theta that measures the angle between the xy projection and the x axis in the xy plane.  And we'll use a new angle that we'll call the nuphi, which measures the angle between the xz projection and the x axis in the xz lane.  And then we view a 3D point as a triple (r, theta, nuphi), and we'll square it by passing to (r*r, 2 theta, 2 nuphi).

As with the spherical coordinates approach, we can again be sure that the set we get will have a famililar Mandelbrot set as its interesections with both the xy plane and the xz plane.

In principle, this coordinate system should work, as specifying the theta puts the point on a certain plane perpendicular to the xy axis, and specifing the nuphi puts the point in a plane prependiuclar to the xz axis, so we have a line as the intersection of these two planes, and the r coordinate picks the point.

My friend Michael Beeson helped me figure out some formulae to pass from (r, theta, nuphi) to (x, y, z).

denom = sqrt[ cos^2 nuphi +  cos^2 theta sin^2 nuphi ]
x  =  r cos theta  cos nuphi /  denom
y =   r sin theta cos nuphi  /  denom
z  =  r cos theta sin nuphi   /  denom

In the algorithm below, I'll prefix some of the variables with targ to stand for two arg.

```
norm[p_] := N[Sqrt[p . p]];
safearctan[x_, y_] := If[x ≠ 0. || y ≠ 0., ArcTan[x, y], 0.0]

theta[{0, 0, 0}] = theta[{0., 0., 0.}] =  0
theta[p_] :=  N[safearctan[p[[1]], p[[2]]]]
nuphi[{0, 0, 0}] = phi[{0., 0., 0.}] = 0
nuphi[p_] := N[safearctan[p[[1]],  p[[3]]]]
denom[ap_] := N[sqrt[ Cos[ap[[3]]] Cos[ap[[3]]] +
    Cos[ap[[2]]] Cos[ap[[2]]] Sin[ap[[3]]] Sin[ap[[3]]] ]]
targx[ap_] :=  N[ap[[1]] Cos[ap[[2]]] Cos[ap[[3]]] / denom[ap]]
targy[ap_]  :=  N[ ap[[1]] Sin[ap[[2]]] Cos[ap[[3]]] / denom[ap]]
targz[ap_] :=  N[ ap[[1]] Cos[ap[[2]]] Sin[ap[[3]]] / denom[ap]]
targcartesian[ap_] := N[{targx[ap], targy[ap], targz[ap]}];

targsquare[p_] := N[targcartesian[{p.p, 2 theta[p], 2 nuphi[p]}]];
targmandelmap[p_, c_] := N[targsquare[p] + c];

targmandel[maxnorm_, maxiterate_, gridsteps_] :=
 (set = {PointSize[.009]};
  Do[Do[Do[
     test = {-2. + (3.25 i) / gridsteps, -1. + (2. j) / gridsteps, -1. + (2. k) / gridsteps};
     hit = test;
     Do[hit = targmandelmap[hit, test];, {m, 1, maxiterate, 1}];
     If[norm[hit] < maxnorm, AppendTo[set, Point[test]]],
     {i, gridsteps}], {j, gridsteps}], {k, gridsteps}];
  Show[Graphics3D[set], BoxRatios → {1, 1, 1}])
```

```
0

0
```

I tried this with a very small grid ... and no points survive.  They all escape.  So I don't think this approach looks good either.

```
targmandel[3, 5, 5]
```

```
$Aborted
```

```
targmandel[2, 20, 30]
```

It's possible that the ArcTan is killing this.  So I had a shot at eliminating the trig functions with this kind of approach for converting (x, y, z) into a squared (newx, newy, newz). I'm using a bunch of temporary variables like "costheta" and "cos2theta"...the latter stands for the square of the cosein of theta, for instance.  I could eliminate the temporary variables, but if I do, I get a really nasty expression in the denom variable, something with a square root of an order six polynomial in x, y, and z.

```
double rsquared = x*x + y*y + z*z
double rxy = sqrt (x*x + y*y);
double rxz = sqrt (x*x + z*z);

double costheta = x/rxy;
double sintheta = y/rxy;
double cos2theta = costheta*costheta - sintheta*sintheta;
double sin2thetha = 2*costheta*sintheta;
double cosnuphi = x/rxz;
double sinnuhpi = z/rxz;
double cos2nuphi = cosnuphi*cosnuphi - sinnuphi*sinnuphi;
double sin2nuphi = 2*cosnuphi*sinnuphi;

double rsquared_over _denom = rsquared/sqrt (pow (cos2nuphi, 2.0) + pow (cos2theta*sin2nuphi, 2.0));

newx = rsquared_over _denom*cos2theta*cos2nuphi;
newy = rsquared_over _denom*sin2theta*cos2nuphi;
newz = rsquared_over _denom*cos2theta*sin2theta;
```

I tried to implement this as *Mathematica* code, but for some dumb reason, I can't get the code to work, *Mathematica* complains about p[[1]], p[[2]], and p[[3]],which seemed fine in the earlier algorithms.  For now, I'm giving up, although if I can ever get this to "compile", I'd want to add a check for divising by zero in the rsquaredoverdenom term.

In[43]:=
```
rsquared[p_] := N[p.p];
x[p_] := p[[1]]
x[p_] := p[[2]]
z[p_] := p[[3]]
rxy[p_] := N[Sqrt[(x[p])^2 + (y[p])^2]];
rxz[p_] := N[Sqrt[(x[p])^2 + (z[p])^2]];

costheta[p_]  := N[x[p] / rxy[p]];
sintheta[p_]  := N[y[p] / rxy[p]];
cos2theta[p_] := N[costheta[p] costheta[p] - sintheta[p] sintheta[p]];
sin2theta[p_] := N[2 costheta[p] sintheta[p]];
cosnuphi[p_]  := N[x[p] / rxz[p]];
sinnuphi[p_]  := N[z[p] / rxz[p]];
cos2nuphi[p_] := N[cosnuphi[p] cosnuphi[p] - sinnuphi[p] sinnuphi[p]];
sin2nuphi[p_] := N[2 cosnuphi[p] sinnuphi[p]];
rsquaredoverdenom[p_] =
  N[rsquared[p] / Sqrt[(cos2nuphi[p])^2 + (cos2theta[p] sin2nuphi[p])^2]];

newx[p_] = N[rsquaredoverdenom[p] cos2theta[p] cos2nuphi[p]];
newy[p_] = N[rsquaredoverdenom[p] sin2theta[p] cos2nuphi[p]];
newz[p_] = N[rsquaredoverdenom[p] cos2theta[p] sin2theta[p]];

targsquare[p_] := N[{newx[p], newx[p], newz[p]}];

targmandelmap[p_, c_] := N[targsquare[p] + c];

targmandel[maxnorm_, maxiterate_, gridsteps_] :=
 (set = {PointSize[.009]};
  Do[Do[Do[
     test = {-2. + (3.25 i) / gridsteps, -1. + (2. j) / gridsteps, -1. + (2. k) / gridsteps};
     hit = test;
     Do[hit = targmandelmap[hit, test];, {m, 1, maxiterate, 1}];
     If[norm[hit] < maxnorm, AppendTo[set, Point[test]]],
     {i, gridsteps}], {j, gridsteps}], {k, gridsteps}];
  Show[Graphics3D[set], BoxRatios → {1, 1, 1}])
```