Cube Polywell Wiffleball Modeling Using Method of Images

Indrek Mandre <indrek@mare.ee>
Icarus @ http://talk-polywell.org/

July 29, 2008

Abstract

It has been speculated that the wiffleball effect in the cube polywell could be modeled as a superconducting sphere in the middle of the system. To do that the method of images is a perfect solution.

1 Method of Images

From http://en.wikipedia.org/wiki/Method_of_images:

> *Method of images (or mirror images) is a mathematical tool for solving differential equations in which the domain of the sought function is extended by the addition of its mirror image with respect to a symmetry hyperplane, with the purpose of facilitating the solution of the original problem. It is used in electrostatics to simply calculate or visualize the distribution of the electric field of a charge in the vicinity of the conducting surface. It is based on the fact that the tangential component of the electrical field to the surface of a conductor is zero, and that some field E with* $\nabla \times \mathbf{E} = 0$ *and* $\nabla \cdot \mathbf{E} = 0$ *in some region is uniquely defined by its normal component over the surface which confines this region (the uniqueness theorem).*
>
> *The method may also be used in magneto-statics for calculating the magnetic field of a magnet which is close to a superconducting surface. Here, the component of the magnetic field normal to the superconductor is zero. Another use is in fluid dynamics: the movement of a vortex near a wall may be calculated using an image vortex.*

To apply the method of images we shall place inverted coils within the system so that the resulting field lines on the surface of the superconducting sphere would be tangential to it. This method is known to be correct for regions exterior and upon the sphere itself, integral quantities of physical properties for the domain interior to the sphere may also be related to the image system physics.

2 The Image Coils

The size and shape of the image coils can be determined by using inversive geometry. From http://en.wikipedia.org/wiki/Inversive_geometry:

> *In the plane, the inverse of a point P with respect to a circle of center O and radius r is a point P' such that P and P' are on the same ray going from O, and OP times OP' equals the radius squared, $|OP||OP'| = r^2$.*

Nomenclature:
a - radius of the spherical inversion boundary
r_p - planar radius of physical coils
s_p - spacing of physical coils
c_p - conical radius to physical coil from center of Polywell
r_i - planar radius of image coils
s_i - spacing of image coils
c_i - conical radius to image coil from center of Polywell
l - half-length of side of the Polywell cube

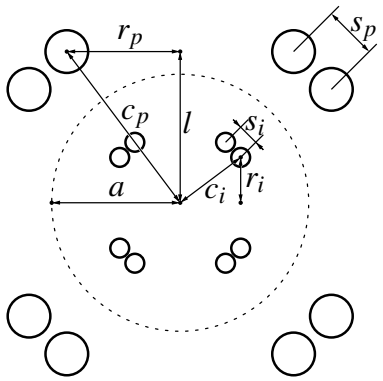

Figure 1: Cross-section of the cube polywell with image coils

The spherical inversion has the property that all points exterior to the sphere are reflected to points interior to the sphere by the relation

$$r_{int} = \frac{a^2}{r_{ext}}$$

where r_{ext} is the radius of the point exterior to the sphere and r_{int} is the radius of the point interior to the sphere. Using the spherical inversion property for the points on the physical coil circular axis we get

$$c_i = \frac{a^2}{c_p} \tag{1}$$

and by extension

$$\frac{c_i}{c_p} = \left(\frac{a}{c_p}\right)^2 \tag{2}$$

The half-length of a side of the cubical framework that Polywell coils lie upon is given by

$$l = r_p + \frac{s_p}{\sqrt{2}}$$

and by Pythagoras it then follows that

$$c_p^2 = r_p^2 + l^2 = r_p^2 + \left(r_p + \frac{s_p}{\sqrt{2}}\right)^2 \tag{3}$$

and further, by similar triangles we can show that

$$\frac{c_i}{c_p} = \frac{r_i}{r_p} = \frac{s_i}{s_p}$$

So taking equation 2 and equation 3 we get

$$\frac{c_i}{c_p} = \frac{a^2}{r_p^2 + \left(r_p + \frac{s_p}{\sqrt{2}}\right)^2} = \zeta$$

where we shall call ζ our spherical inversion factor.

Now to find the location of our image coils that will generate the spherical boundary, given a specified wiffleball radius, we use

$$r_i = r_p\zeta \quad \text{and} \quad s_i = s_p\zeta$$

To get the correct solution the field generated by the image coils has to be corrected. We can do this through modifying the current I_i running in the image coils:

$$I_i = -I_p\sqrt{\frac{r_p}{r_i}} = -\frac{I_p}{\sqrt{\zeta}}$$

How this equation arises is currently unclear but it gives us a perfect solution where the field lines are tangential to the sphere. By speculation this could come from the Kelvin transformation. These equations are also described in [sezginer].

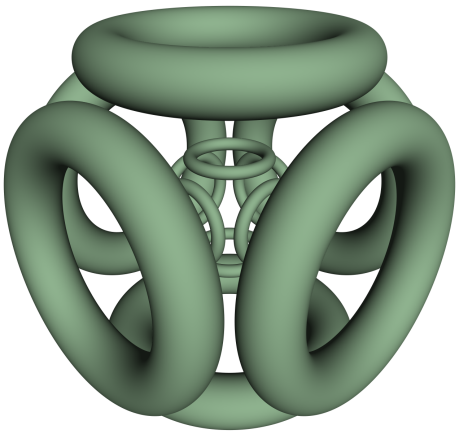

Figure 2: Cube polywell with image coils

3 Field visualizations

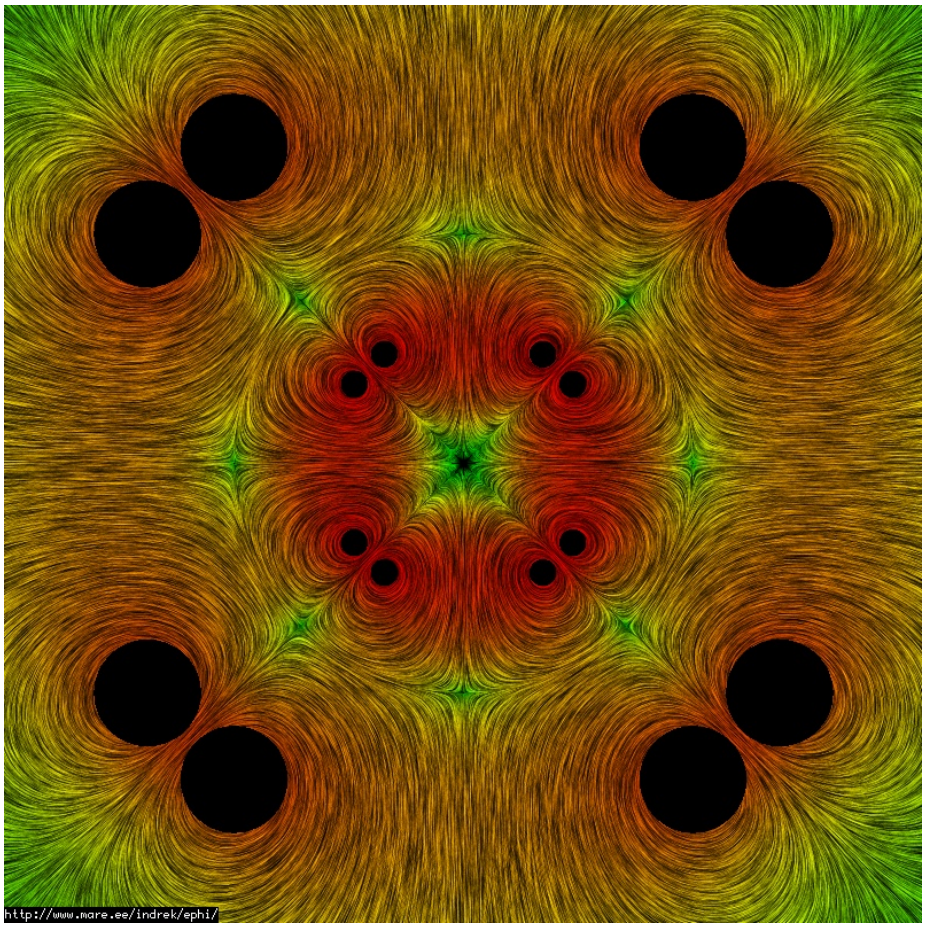

Figure 3: Field lines at cube polywell cross-section (meridian 0)

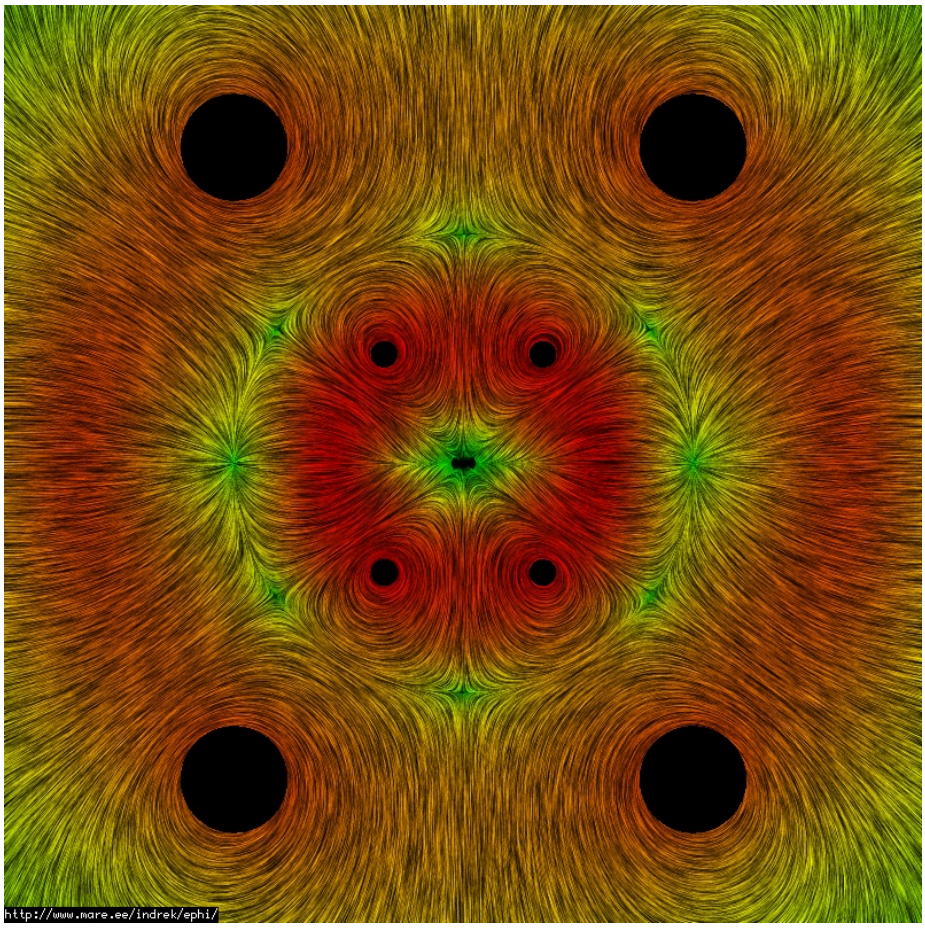

Figure 4: Field lines at cube polywell cross-section (meridian 45)

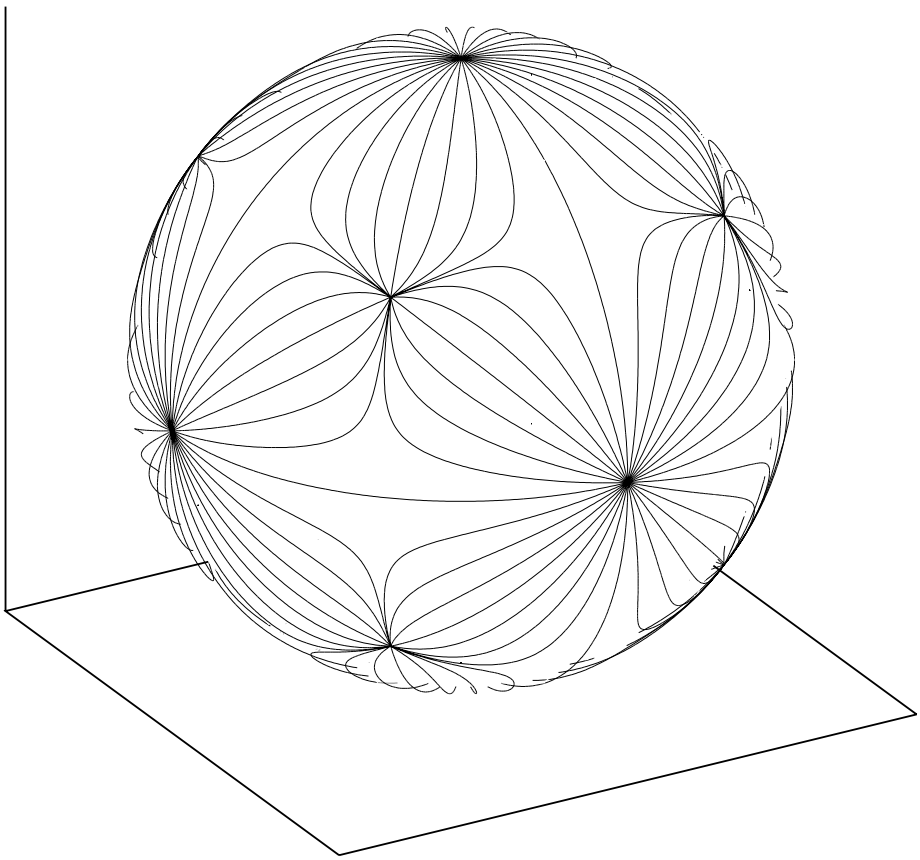

Figure 5: Field lines on the wiffleball sphere

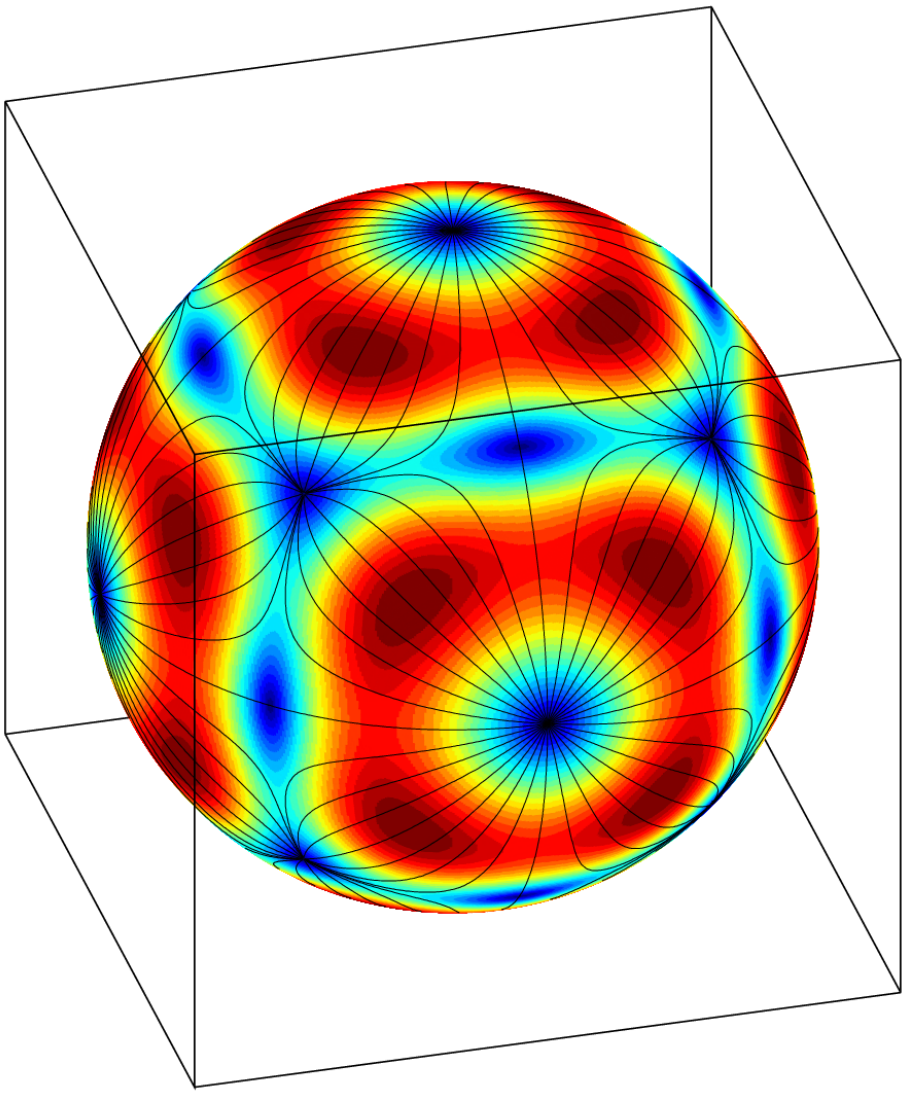

Figure 6: Field magnitude on the wiffleball sphere

4 Cusps

If we define cusps for the wiffleball sphere as points on the sphere where field lines are perpendicular to the sphere surface, we can identify three types of cusps:

- 6 cusps at the faces
- 8 cusps at the corners
- 12 singularities where coils are the closest

It appears there are no line cusps. Cusps are the places where the electrons can more easily leak out of the wiffleball.

References

[sezginer] Apo Sezginer and Weng Cho Chew, "Image of a Static Current Loop over a Superconducting Sphere", IEEE TRANSACTIONS ON MAGNETICS, VOL. 26, NO. 3, MAY 1990.

A Octave code for calculating the fields

A.1 loop_bfield.m - off-axis magnetic field radial and axial components of a loop of current

```
function [Br, Ba] = loop_bfield(radius, current, r, a)
  f1 = 2 * 10^-7 * current;
  R = radius;
  q = r .* r + R .* R + a .* a + 2 .* r .* R;
  m = 4 .* r .* R ./ q;

  [K,E] = ellipke(m);
  sqrt_q = sqrt(q);
  f2 = q - 4 .* r .* R;
  Br = f1 .* a .* (E .* (q - 2 .* r .* R) ./ f2 - K) ./ (sqrt_q .* r);
  Ba = f1 .* (E .* (R .* R - r .* r - a .* a) ./ f2 + K) ./ sqrt_q;
  % recalculate the Br for very small m values
  idx = find(m < 5e-7);
  if (!isempty(idx))
    a = a(idx);
    rr = r(idx);
    m = repmat (5e-7, size(a));
    m2 = m .^ 2;
    % assuming given m and a, find cutoff r
    r = -(sqrt((4 .- m2) .* R^2 - 2 * R * a .* m2 - a .^ 2 .* m2) .- 2 * R) ./ m;
    q = r .* r + R .* R + a .* a + 2 .* r .* R;
    m = 4 .* r .* R ./ q;
    [K,E] = ellipke(m);
    sqrt_q = sqrt(q);
    f2 = q - 4 .* r .* R;
    % use linear interpolation from the cutoff radius
    Br(idx) = (rr ./ r) .* f1 .* a .* (E .* (q - 2 .* r .* R) ./ f2 - K) ./ (sqrt_q .* r);
  endif
endfunction
```

A.2 loop_bfield3d.m - off-axis magnetic field vector components of a loop of current

```
function [B] = loop_bfield3d(center, normal, radius, current, pos)
  len = size(pos, 1);
  v = pos - repmat(center, len, 1); % vector from the center of a coil to the point
  nvs = repmat(normal, len, 1); % replicate the normal vectors
  a = dot(v, nvs, 2); % axial distance, axial vector is the normal vector
  radial = v - nvs .* repmat(a, 1, 3);
  r = sqrt(dot(radial, radial, 2));
  radialn = radial ./ repmat(r, 1, 3);
  idx = find(any(r, 2) == 0);
  if (!isempty(idx))
    radialn(idx, :) = zeros(size(idx, 1), 3);
  endif
  [Br,Ba] = loop_bfield (radius, current, r, a);
  B = radialn .* repmat(Br,1,3) + nvs .* repmat(Ba,1,3);
endfunction
```

A.3 polywell_bfield.m - magnetic field vector components for the entire polywell system

```
function [B] = polywell_bfield(radius, spacing, current, pos)
  rr = radius + spacing / sqrt(2);
  B = loop_bfield3d([-rr,   0,   0], [ 1,   0,   0], radius, current, pos) + \
      loop_bfield3d([ rr,   0,   0], [-1,   0,   0], radius, current, pos) + \
      loop_bfield3d([  0, -rr,   0], [ 0,   1,   0], radius, current, pos) + \
      loop_bfield3d([  0,  rr,   0], [ 0,  -1,   0], radius, current, pos) + \
      loop_bfield3d([  0,   0, -rr], [ 0,   0,   1], radius, current, pos) + \
      loop_bfield3d([  0,   0,  rr], [ 0,   0,  -1], radius, current, pos);
endfunction
```

A.4 camera3dn.m - octave camera view angle

```
function [normal] = camera3dn()
  [theta,phi]=view();
  phi = 90 - phi;
  theta = theta + 270;
  phi = phi * pi / 180;
  theta = theta * pi / 180;
  normal = [cos(theta)*sin(phi), sin(theta)*sin(phi), cos(phi)];
endfunction
```

A.5 ball.mat - visualize the field on the modeled wiffleball sphere

```
% 24-sector symmetry reflections for the cube polywell
sectors = cat (3,
  [ 1  0  0; 0  1  0; 0  0  1 ], [ 1  0  0; 0 -1  0; 0  0 -1 ],
  [ 1  0  0; 0  0 -1; 0  1  0 ], [ 1  0  0; 0  0  1; 0 -1  0 ],
  [-1  0  0; 0  1  0; 0  0 -1 ], [-1  0  0; 0 -1  0; 0  0  1 ],
  [-1  0  0; 0  0  1; 0  1  0 ], [-1  0  0; 0  0 -1; 0 -1  0 ],
  [ 0  1  0; 0  0  1; 1  0  0 ], [ 0 -1  0; 0  0 -1; 1  0  0 ],
  [ 0  0 -1; 0  1  0; 1  0  0 ], [ 0  0  1; 0 -1  0; 1  0  0 ],
  [ 0  1  0; 0  0 -1; -1  0  0 ], [ 0 -1  0; 0  0  1; -1  0  0 ],
  [ 0  0  1; 0  1  0; -1  0  0 ], [ 0  0 -1; 0 -1  0; -1  0  0 ],
  [ 0  0  1; 1  0  0; 0  1  0 ], [ 0  0 -1; 1  0  0; 0 -1  0 ],
  [ 0  1  0; 1  0  0; 0  0 -1 ], [ 0 -1  0; 1  0  0; 0  0  1 ],
  [ 0  0 -1; -1  0  0; 0  1  0 ], [ 0  0  1; -1  0  0; 0 -1  0 ],
  [ 0  1  0; -1  0  0; 0  0  1 ], [ 0 -1  0; -1  0  0; 0  0 -1 ]);

% set up polywel size and the image coil size
global R S CURRENT iR iS iCURRENT
R = 0.15; S = 0.08; CURRENT = 1;
A = 0.15; % wiffleball radius
inv_f = (A^2)/(R^2 + (R + S/sqrt(2))^2);
iR = R*inv_f; iS = S*inv_f; iCURRENT = -CURRENT / sqrt(inv_f);

% half-length of the polywell cube side
L = sqrt(R^2 + (R + S/sqrt(2))^2);

% calculate field magnitude on the sphere
[X,Y,Z] = sphere(1000);
X = X .* A; Y = Y .* A; Z = Z .* A;
POS = cat(2, X(:), Y(:), Z(:));
B = polywell_bfield(R, S, CURRENT, POS);
B = B + polywell_bfield(iR, iS, iCURRENT, POS);
C = reshape (sqrt(dot(B, B, 2)), size(X));

% direction function within the field for our ode
function [d] = dirf(t, x)
  global R S CURRENT iR iS iCURRENT
  pos = reshape(x, size(x, 1)/3, 3);
  B = polywell_bfield (R, S, CURRENT, pos);
  B = B + polywell_bfield (iR, iS, iCURRENT, pos);
  B = B ./ repmat(sqrt(dot(B, B, 2)), 1, 3);
  d = reshape(B, size(x));
endfunction

% starting points for lines for the ode
% assumption: lines go radially at equal spacing
sp = zeros(8, 3);
for i=1:8
  phi = (i - 1) * pi / 16;
  sp(i,:) = [-A + 0.01, 1e-5 * cos(phi), 1e-5 * sin(phi)];
endfor

% calculate the field lines on the sphere
options = odeset ('MaxStep', 0.001, 'InitialStep', 0.001,
  'AbsTol', 1e-6, 'RelTol', 1e-6);
[t, x] = ode45(@dirf, [0 0.22], sp(:)', options);
POS = [0 0 0];
n = size(x,2)/3;
for i=1:n
  pos = cat(2, x(:, i), x(:, i + n), x(:, i + 2 * n));
  % erase the cusp lines
  idx=find(dot(pos,pos,2) < (A-0.001)^2);
  pos(idx,:) = 0;
  % map the positions into 24 sectors
  for j=1:size(sectors, 3)
    POS = cat(1, POS, (sectors(:,:,j) * pos')');
    POS = cat(1, POS, [0 0 0]);
  endfor
endfor

hold on
% draw the sphere with field magnitudes
set(gca, 'XTick',[], 'YTick',[], 'ZTick',[]);
axis([-A A -A A -A A]);
axis square;
view(-15, 30);
surf(X,Y,Z,C);
view(-15, 30);
shading interp
colormap(jet(24));

% draw the field lines
cn = camera3dn();
cn = repmat(cn, size(POS)(1), 1);
% erase points behind the view
POS(find(dot(POS, cn, 2) < 0), :) = 0;
idx=find(any(POS,2)==0);
start = 0;
```

```
for i=1:size(POS,1)
  if any(POS(i,:))
    if start == 0
      start = i;
    endif
  else
    if start != 0
      data = POS(start:i-1, :);
      x = data(:, 1);
      y = data(:, 2);
      z = data(:, 3);
      plot3(x, y, z, 'color', [0 0 0]);
      view(-15, 30);
    endif
    start = 0;
  endif
endfor

hold off

print ("ball.png", "-S1200,1200");
print ("ball.eps", "-depsc");

clf
hold on
set(gca, 'XTick',[], 'YTick',[], 'ZTick',[]);
view(-15, 30);
axis([-A A -A A -A A]);
axis square;

% draw the field lines
cn = camera3dn();
cn = repmat(cn, size(POS)(1), 1);
% erase points behind the view
POS(find(dot(POS, cn, 2) < -0.005), :) = 0;
idx=find(any(POS,2)==0);
start = 0;
for i=1:size(POS,1)
  if any(POS(i,:))
    if start == 0
      start = i;
    endif
  else
    if start != 0
      data = POS(start:i-1, :);
      x = data(:, 1);
      y = data(:, 2);
      z = data(:, 3);
      plot3(x, y, z, 'color', [0 0 0]);
      view(-15, 30);
    endif
    start = 0;
  endif
endfor

hold off

print ("ball2.png", "-S1200,1200");
print ("ball3.eps", "-deps");
```