PSCF Visualization

Table of Contents

1 Initializing Data

1.1 Reading the PCSF Output File

The program begins by converting the contents of the rho_rgrid file into text, and storing each line as a cell in Matlab, using the following line of code.

```
C = textread(filename, '%s','delimiter', '\n');
```

The parameters in the file are preceded by 'keywords'or abbreviations that indicate their identity; for example, the cell parameters are preceded by 'cell_param' and the number of monomers is preceded by 'N_monomer'. All markers are summarized in Table 1.1.1.

Table 1.1.1 | Parameter Markers

Marker	Description	Type	Name
`crystal_system`	The crystal system	Text	`type`
`N_monomer`	The number of monomer units	Scalar	`n_mnr`
`dim`	The number of grid dimensions	Scalar	`dim`
`ngrid`	The grid dimensions	Vector	`grid`
`cell_param`	The cell parameters	Vector	`param`

The following code searches for these markers in the cell `C` using a while loop, and then converts and stores the data in the subsequent line based on its type using relevant Matlab functions, as described in Table 1.1.2.

```
while ndata <= required_rep
    ic = ic +1;

    if round(sum (sscanf(C{ic},'%f')),2)== 1.00 && round(sum
sscanf(C{ic+1},'%f')),2)== 1.00
        ndata = ndata+1;
    else
        ndata = 0;
    end

    if strcmp(strrep(char(C(ic)),' ', ''),'dim')==1
        dim = str2double(C{ic+1});            % Reads the grid dimensions
    elseif strcmp(strrep(char(C(ic)),' ', ''),'crystal_system')==1
        type = strrep(C(ic+1), '''', '');    % Reads the system type
    elseif strcmp(strrep(char(C(ic)),' ', ''),'cell_param')==1
        param = sscanf(C{ic+1},'%f')';       % Reads the cell parameters
    elseif strcmp(strrep(char(C(ic)),' ', ''),'N_monomer')==1
        n_mnr = str2double(C{ic+1});      % Reads the number of monomers
    elseif strcmp(strrep(char(C(ic)),' ', ''),'ngrid')==1
        grid = sscanf(C{ic+1},'%f')';        % Reads the grid size
    end
end
```

Table 1.1.2| Data Conversion Functions

Data Type	Function
Text	`char`
Scalar	`str2double`
Vector	`sscanf`

The while loop terminates after reading five consecutive cells whose values each sum to unity that is, after reaching the density grid points. The number five is the value of the parameter `required_rep`. The following values are stored for later use.

```
end_info = ic - required_rep - 1;
% The row in which the supplementary information ends
start_row = ic - required_rep;
% The row in which the density values start
```

The density values at each grid point are then read into a matrix row by row.

```
A = zeros(length(C) - end_info,n_mnr);

for i =start_row:length(C)
    A(i - end_info,:) = sscanf(C{i},'%f')';
end
```

1.2 Reading Additional User inputs

Additional inputs are read from a text file in the same manner as that from the PSCF output file. A summary is presented in Table 1.2.1.

Table 1.2.1| User Input Markers

Marker/Name		Description	Type
`isovalue`		The minimum volume fraction present in the display	Vector/ Text
`opacity`	`row 1`	The desired opacity of the solid colours	Vector
	`row 2`	The desired opacity of the interpolated colours	Vector
`thickness`		The thickness of the unit cell outline	Scalar
`box_clr`		The color of the unit cell outline (RGB)	Vector
`map_choice`		The desired map coluors	Vector
`mono_disp`		The desired monomers to be visualized independently	Vector
`comp_disp`		The desired monomers to be visualized in a composite display	Vector
`weight`		The relative weight of each monomer in the composite display	Vector
`scatterdraw`		The monomers through which scattering must be analyzed	Vector
`h_set`		Miller index range, h	Vector
`k_set`		Miller index range, k	Vector
`l_set`		Miller index range, l	Vector
`inputvec`		Direction through which density will be output[a]	Vector

[a]See Section 6.

It should be noted that if `isovalue = 'auto'`, the program will determine appropriate isovalues for the system. This is discussed in Section 3.

1.3 Fixed Inputs

The following are fixed inputs into the program that need not be changed unless the program is rewritten to include more colourmaps, or to increase visual precision.

```
ncolour = 8; %Number of stored colourmaps (7 maps + 1 fill)
n_dp = 3; %Number of significant decimal places for colourmapping
```

2 Translating the Inputs into Useable Form

2.1 Unit Cell Properties

The `param` vector contains the unit cell dimensions followed by the unit cell angles. The following code separates the two parameters and stores them in separate vectors, `cell_d` and `angle`.

```
if strcmp(type,'hexagonal') == 1
    angle = [pi/2 pi/2 (2*pi)/3];
    cell_d = param;
elseif strcmp(type,'triclinic') == 1
    angle = [param(4) param(5) param(6)];
    cell_d = [param(1) param(2) param(3)];
elseif strcmp(type,'monoclinic') == 1
    angle = [pi/2 param(4) pi/2];
    cell_d = [param(1) param(2) param(3)];
elseif strcmp(type,'trigonal') == 1
    angle = [param(2) param(2) param(2)];
    cell_d = [param(1)];
else
    angle = [pi/2 pi/2 pi/2];
    cell_d = param;
end
```

The following code then ensures that cell_d has three values (one to describe each spatial dimension of the visualized unit cell).

```
if(length(cell_d)==1)
    new_cell = ones(1,3)*cell_d;            % Cubic crystals
elseif(length(param)==2)
    new_cell(1:2) = cell_d(1);              % Tetragonal crystals
    new_cell(3)   = cell_d(2);
else
    new_cell = cell_d;                      % Orthorhombic crystals
end

clear cell_d;cell_d = new_cell;
```

The following code expands the grid to three dimensions in case the calculations were done in one or two dimensions.

```
if(length(grid)==1)
```

```
    grid(2) = grid(1);                        % 3D grid for 1D crystals
    grid(3) = grid(1);
elseif(length(grid)==2)
    grid(3) = grid(1);                       % 3D grid for 2D crystals
end
```

2.2 Formulating the Volume Fraction Data into 4-D arrays

The following code creates three 3-D arrays containing the grid points (`x`, `y`, `z`) converts the matrix `A` into a 4-D array, `R`, that contains the density data at each grid point (`x`, `y`, `z`) for each monomer (`in`).

```
for iz=1:grid(3)+1,
    for iy=1:grid(2)+1,
        for ix=1:grid(1)+1,
            counter = counter + 1;
            x(ix,iy,iz) = cell_d(1) * (ix-1)/grid(1) +
(cos(angle(3)))*( cell_d(2) * (iy-1)/grid(2)) + ((iz-
1)/grid(3))*(cos(angle(1))*cell_d(3));
            y(ix,iy,iz) = cell_d(2) * (iy-1)/grid(2) * sin(angle(3)) +
((iz-1)/grid(3))*cos(angle(2))*cell_d(3);
            z(ix,iy,iz) = cell_d(3) * (iz-1)/grid(3) * sin(angle(1)) *
sin(angle(2));
            for in = 1:n_mnr
                if ix == grid(1)+1
                    R(grid(1)+1,:,:,in) = R(1,:,:,in);
                    counter = counter - (1/n_mnr);
                elseif iy == grid(2) + 1
                    R(:,grid(2)+1,:,in) = R(:,1,:,in);
                    counter = counter - (1/n_mnr);
                elseif iz == grid(3) + 1
                    R(:,:,grid(3)+1,in) = R(:,:,1,in);
                    counter = counter - (1/n_mnr);
                else
                    R(ix,iy,iz,in) = A(round(counter),in);
                end
            end
        end
        if(dim==1)
            counter = 0;
        end
    end
    if(dim==2)
        counter=0;
    end
end
```

Note that if the data were only calculated for one or two dimensions, the density values are repeated along the other dimensions, thus creating a 3-D grid of density values for visualization regardless of the dimensions for which the data were calculated.

3 Isovalue Calculation

The isovalue for each monomer is the lower bound of the density value to which each density profile is drawn. These are hard to appropriately determine by hand such that there is no overlap or white space in the composite density profile.

The algorithm in this code determines a set of isovalues such that the most dominant portion of each monomer can be seen in the composite density profile. This is done by drawing a line that passes through the maximum density value of each monomer, setting the isovalue to that at the highest at the intersections, and then adjusting to fill in any gaps. This is broken down in the next sections.

3.1 Rescaling the Density

In this section the `R` matrix is rescaled such that the density values for each monomer range from zero to one. These new values are then stored in the matrix `S`.

```
for in = 1:n_mnr
        polmax(in) = max(max(max((R(:,:,:,in)))));
        polmin(in) = min(min(min((R(:,:,:,in)))));
        l_length(in) = polmax(in) -polmin(in);
    end


    %Creating the scaled matrix, S
    S = R;
    for in = 1:n_mnr
        for iz=1:grid(3)+1,
            for iy=1:grid(2)+1,
                for ix=1:grid(1)+1,
                    S (ix,iy,iz,in) = weight(in)*(R(ix,iy,iz,in)-
polmin(in))/l_length(in); % Matrix with scaled values
                end
            end
        end
    end
```

The location of the first `n_lines` points where the rescaled value is equal to one is then found using the following code. `n_lines` is set by default to a value of one, as only one value is usually sufficient for the operation of this program.

```
for in =1:n_mnr
        [si,sj,sk] = ind2sub([plot_grid n_mnr],find(S(:,:,:,in) ==
weight(in),n_lines));
        pmax_loc(:,:,in) = [si,sj,sk];
end
```

3.2 Drawing the Line

The points that the line must pass through are those that were just stored in `pmax_loc`. These must be interleaved to guarantee intersections between density values for different monomers. This is done as follows.

```
for ir = 1:n_lines
        for in = 1:n_mnr
```

```
                point_series(n_mnr*(ir-1)+in,:) = pmax_loc(ir,:,in);
%Series of points to be plotted
            end
        end
```

The density values at the nearest grid points along the line are found and stored in the ‘*n* × *m*’ matrix `line`, where *n* is the number of monomers and *m* is the number of grid points that the line passes through. This is done as follows.

```
for ir = 1:(n_lines*n_mnr)-1 %Number of lines
        start_coord = point_series(ir,:);
        end_coord = point_series(ir+1,:);
        dir_vec(ir,:) = end_coord-start_coord;
        step_length(ir) = max(abs(dir_vec(ir,:)));
        for il = 1:step_length(ir)
            ix(ir,il) = start_coord(1)+ round((il-
1)*(dir_vec(ir,1)/step_length(ir)));
            iy(ir,il) = start_coord(2)+ round((il-
1)*(dir_vec(ir,2)/step_length(ir)));
            iz(ir,il) = start_coord(3)+ round((il-
1)*(dir_vec(ir,3)/step_length(ir)));
            x_plot(il+l_size) = il+l_size;
            for in= 1:n_mnr
                line (in,il+l_size) = R
(ix(ir,il),iy(ir,il),iz(ir,il),in);

            end
        end
        l_size = l_size + step_length(ir);
 end

%The final point

for in= 1:n_mnr
    line (in,l_size+1)= R (end_coord(1),end_coord(2),end_coord(3),in);
end
x_plot(l_size+1) = l_size+1;
```

The density values along this line are then rescaled.

```
for in = 1:n_mnr
        for ip = 1:length(x_plot)
            line_new(in,ip) = weight(in)*(line(in,ip)-
polmin(in))/l_length(in);
        end
end
```

Figure 3.2.1 displays the information contained within `line_new`. The label on the x-axis ‘Line-Index’ is simply the number of grid points that the line has passed by, whose data are contained within `line_new`.

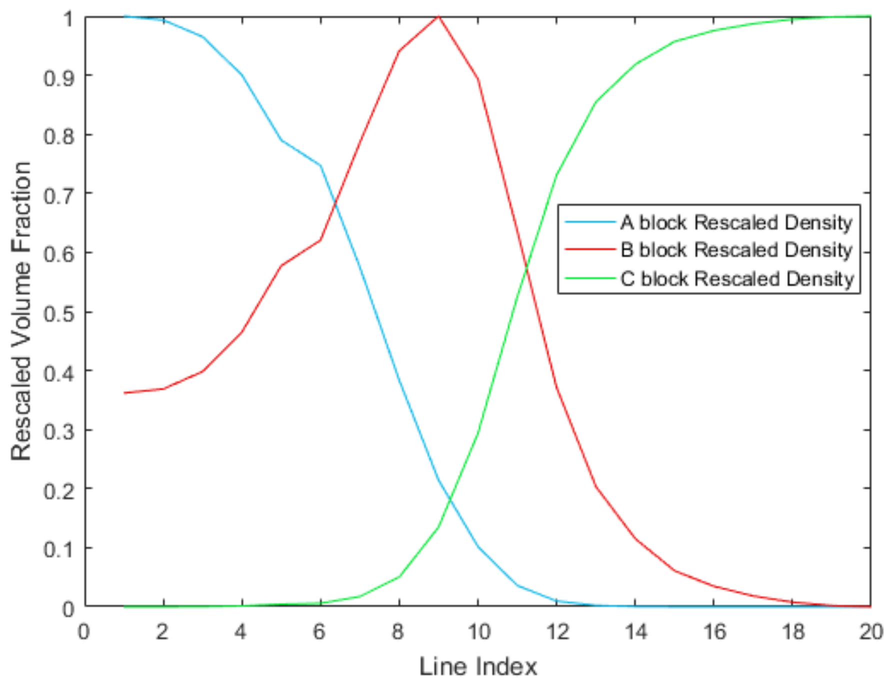


Figure 3.2.1 | Density Values Along `line_new`

3.3 Finding the Intersections

The values of the grid points between which the rescaled density values of monomers intersect are found using and stored using the following line of code.

```
x_inter_store{k} = find(diff(sign(line_new(loop_round+j,:)-
line_new(loop_round,:))));
```

Linear interpolation is then used to find and store the rescaled density value for each monomer at each intersection.

```
  for i = 1:length(cell2mat(x_inter_store(k)))
                     x_inter = cell2mat(x_inter_store(k));
                     p(1,1)= line_new(loop_round,x_inter(i));
                     p(1,2)= line_new(loop_round,1+x_inter(i));
                     p(2,1)= line_new(loop_round+j,x_inter(i));
                     p(2,2)= line_new(loop_round+j,1+ x_inter(i));
                     inter_point(k,i) = (p(1,1)*(p(2,2)-p(2,1))-
p(2,1)*(p(1,2)-p(1,1)))/(p(2,2)-p(2,1)-p(1,2)+p(1,1));
                 end
```

The highest rescaled density value for each type of intersection (i.e. AB, AC, BC) is then found and stored in the vector `intervalue`.

```
ipts = inter_point(k,:);
                 if ~isempty(ipts(ipts > 0.05 & ipts< 0.95))
                 intervalue(k) = max(ipts(ipts< 0.95));
                 else
```

```
                intervalue(k) = max(inter_point(k,:));
```

An initial rescaled isovalue for each monomer is then found by taking the highest `intervalue` that the monomer was involved in. So, for a three monomer system, the three intersection 'types' are AB, CA (which is the same as AC) and BC, which are described by the `involved` matrix,

`involved = [1,2;3,1;2,3].`

The following code stored the initial rescaled isovalue in a vector `isovalue_s`.

```
    for k = 1:n_mnr-1
        for in = 1:n_mnr
            in_mat(in,k)= intervalue(involved(in,k));
            isovalue_s(in) = max(in_mat(in,:));

        end
    end
```

3.4 Filling the Gaps

Close inspection of Figure 3.2.1 will reveal that there is a gap in continuity between the current rescaled isovalue of the B block and that of the C block.

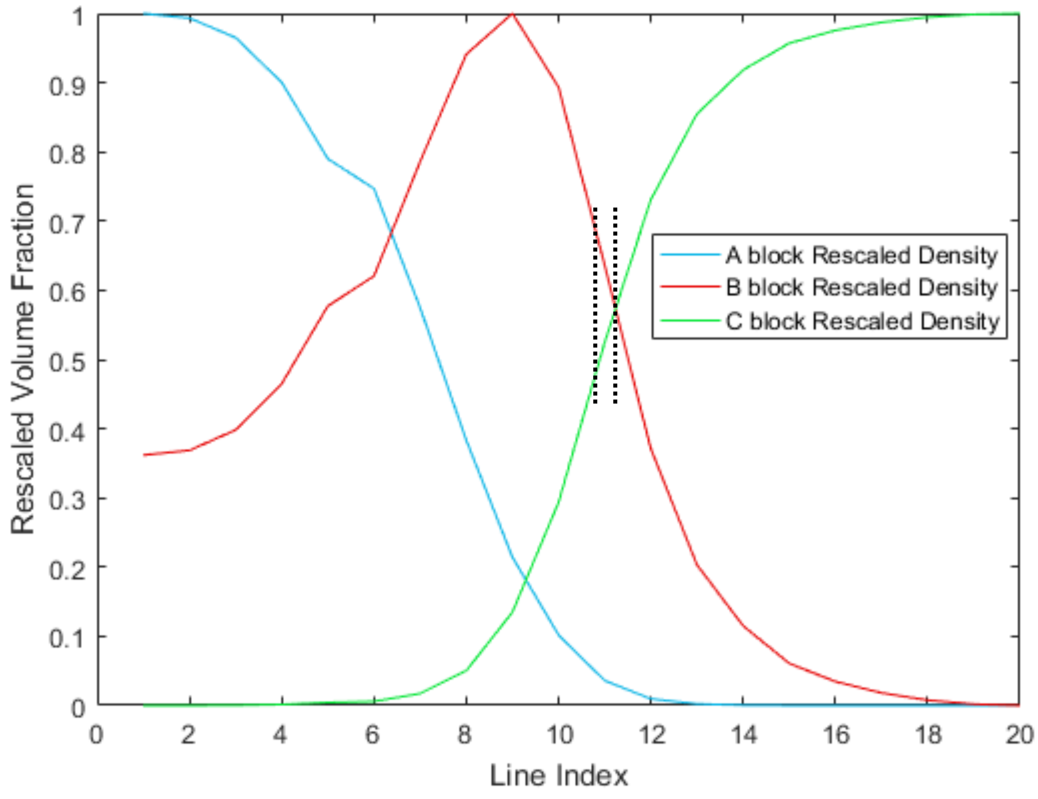


Figure 3.2.1 | Density Values Along `line_new`
The dotted lines represent the discontinuity in rescaled isovalue.

To bridge this gap, the following is a sample of the code that searches for discontinuities and adjusts the appropriate isovalue accordingly. The vector `no_2` contains the second highest intersection value for each monomer, which is required to locate gaps. The program contains variations of this code to account for all permutations for three monomers.

```
        for in = 1:n_mnr
            r_row = in_mat(in,:);
            no_2(in) = max(r_row(r_row <max(r_row)));
        end

        for in = 1:n_mnr-1
            if isovalue_s(in)== no_2(in+1)
                start_mat = find(diff(sign(isovalue_s(in+1)-
line_new(in+1,:))));
                intersect_val = (find(diff(sign(line_new(in,:)-
line_new(in+1,:))), 1));
                start_ind = start_mat(abs(start_mat -
intersect_val)==min(abs(start_mat - intersect_val)));
                py1= line_new(in+1,start_ind);
                py2= line_new(in+1,1+start_ind);
                px1= start_ind;
                px2= start_ind+1;
                y_int = isovalue_s(in+1);
                x_int = px1 + ((y_int-py1)*(px2-px1))/(py2-py1);


                py1= line_new(in,start_ind);
                py2= line_new(in,1+start_ind);
                px1= start_ind;
                px2= start_ind+1;
                y_new = py1 + ((x_int-px1)*(py2-py1))/(px2-px1);
                isovalue_s(in) = y_new;
            end
        end
```

3.5 The 'Weight' Feature

The 'weight' feature allows the user to use an available degree of freedom to increase/decrease the visual presence of any monomer. Due to the nature of the aforementioned isovalue finding algorithm, it is not usually necessary. However, as illustrated in the example below, it might be desired in some cases.

```
weight = [1   1   1]
```

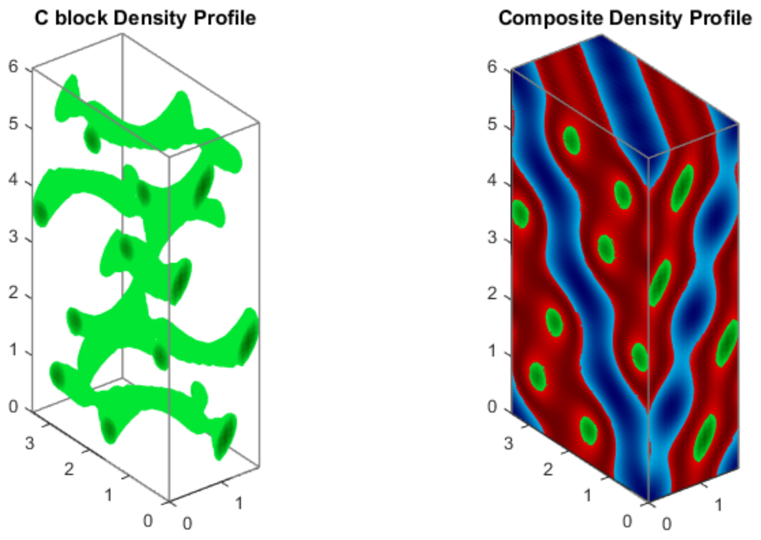


Figure 3.5.1 | Unweighted Gyroid Visualization
File Source: pscf-examples/triblock/o70

An easy way to fix the issue of the relatively sparse representation of the C block is to increase its 'weight' by a small increment, and not to decrease that of the others (though the effect would be the same). Note that if the 'weight' deviates too far from 1, then the program will either run into an error, produce a graphically inaccurate representation, or completely neglect to display one of the monomers. Therefore, modifying the weight must be exhibited with caution, it should only be increased by small increments at a time. Here is the same system, with an increased weight for the 'C' block.

```
weight = [1   1   1.2]
```

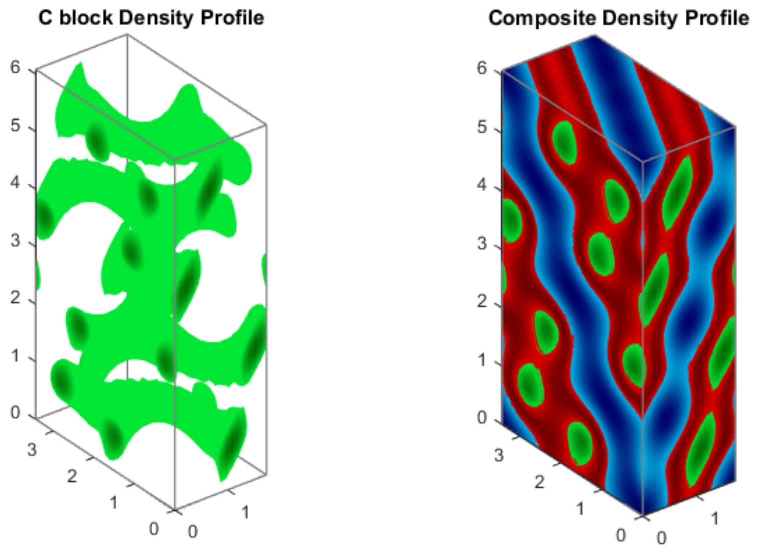


Figure 3.5.2 | Weighted Gyroid Visualization
File Source: pscf-examples/triblock/o70

3.6 Storing the Isovalue

Finally, the true isovalues to be used for visualization are converted from the `isovalue_s` vector that contains the final rescaled isovalues. These true isovalues are stored in the vector `isovalue`.

```
for in=1:n_mnr
    isovalue(in) = (isovalue_s(in)*l_length(in))/weight(in) +polmin(in);
end
```

4 Visualization

4.1 Drawing the Colour Maps

The colour maps are drawn such that there is one colour for every 0.1% of the density for each monomer. This value is stored in the vector `cn`. So, for example, if a polymer is drawn from a maximum density of 0.7748 to an isovalue of 0.2934, there will be 1000 * (0.7748 – 0.2934) + 1 = 483 colours (round up) to display the variation of density in space for that monomer.

```
for in = 1:n_mnr
    face1(:,in) = reshape(squeeze(R(1,:,:,in)),[],1);
    face2(:,in) = reshape(squeeze(R(:,1,:,in)),[],1);
    face3(:,in) = reshape(squeeze(R(:,:,1,in)),[],1);
    face4(:,in) = reshape(squeeze(R(grid(1)+1,:,:,in)),[],1);
    face5(:,in) = reshape(squeeze(R(:,grid(2)+1,:,in)),[],1);
    face6(:,in) = reshape(squeeze(R(:,:,grid(3)+1,in)),[],1);
    face_data(:,in) = [face1(:,in); face2(:,in); face3(:,in);
face4(:,in); face5(:,in); face6(:,in)];
    polmax(in) = max(face_data(:,in)); %Max Face Density value
    cn(map_choice(in)) = 1+ceil((10^n_dp)*polmax(in)-
((10^n_dp)*isovalue(in))); %Effective colourmap range (+1 to buffer)
    newisovalue(in) = in + isovalue(in) - 1;
    mono_label(in) = char(in+'A'-1);
    titles(in) = {[mono_label(in) ' block Density Profile']};
end
```

Note that this maximum density is that on the faces of the unit cell and not that of the entire unit cell, for that is the relevant portion for the colourmap. If the density of any monomer is less than its isovalue on every face, the map is neglected in the composite density profile. The `newisovalue` vector contains unique density values for each monomer (stored from 0 to 1 for monomer A, 1 to 2 for monomer B, etc.), which is required for the colour map assignment in the composite image.

The colours for these maps range from pre-determined RGB values as follows.

```
% low = low fraction, i.e. light

colour_low = zeros (ncolour,3);
colour_low(1,:) = [0,0.7,0.9]; %blue
colour_low(2,:) = [0.9,0,0]; %red
colour_low(3,:) = [0,0.9,0.2]; %green
colour_low(4,:) = [1,1,0]; %yellow
colour_low(5,:) = [0.5,0,1]; %purple
colour_low(6,:) = [1,0,1]; %pink
colour_low(7,:) = [0.75,0.75,0.75]; %grey

% high = high fraction, i.e. dark

colour_high = zeros (ncolour,3);
colour_high(1,:) = [0,0,0.4]; %blue
colour_high(2,:) = [0.4,0,0]; %red
colour_high(3,:) = [0,0.4,0]; %green
colour_high(4,:) = [0.4,0.4,0]; %yellow
colour_high(5,:) = [0.15,0,0.30]; %purple
colour_high(6,:) = [0.25,0,0.25]; %pink
colour_high(7,:) = [0.3,0.3,0.3]; %grey
```

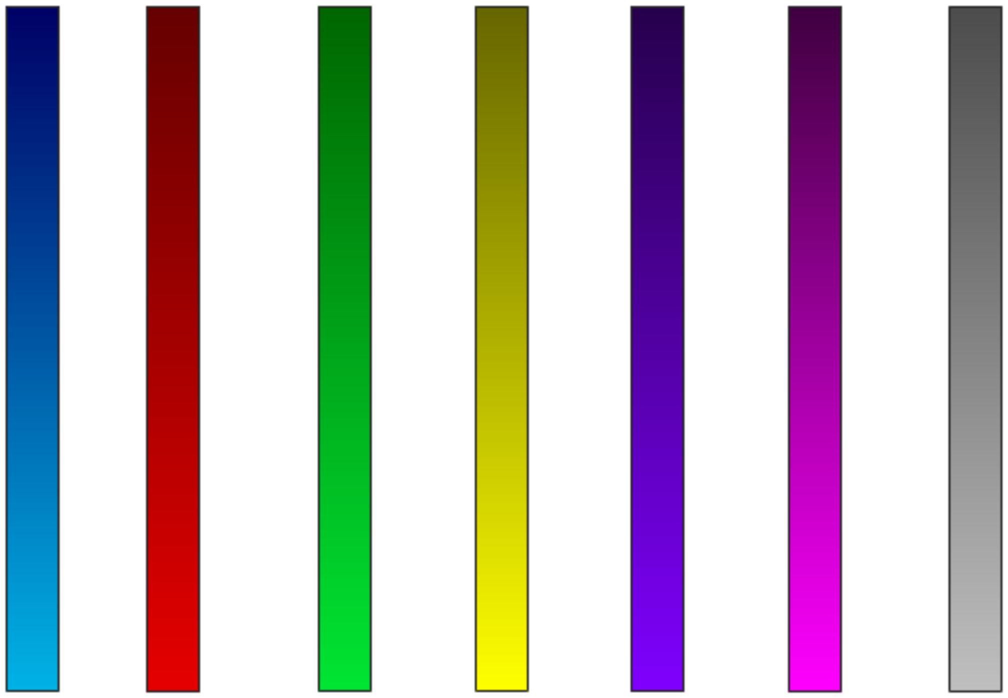


Figure 4.1.1 | Colour Bars Representing Available Colour Ranges

The maps are created using the `linspace` function to create a map with `cn` colours between `colour_low` and `colour_high` for each set colour.

```
for in = 1:ncolour
    temp_map = zeros(cn(in),3);
    temp_map(:,1) = 
linspace(colourpad(in,1,1),colourpad(in,1,2),cn(in)); %Red
    temp_map(:,2) = 
linspace(colourpad(in,2,1),colourpad(in,2,2),cn(in)); %Green
    temp_map(:,3) = 
linspace(colourpad(in,3,1),colourpad(in,3,2),cn(in)); %Blue
    map_store{in}=temp_map;
end
```

4.2 Drawing Discrete Density Profiles

The following are the default axis properties, along with comments that explain their function.

```
if dim == 3
    view(3);                %Sets the view to 3-D
else
    view(2);                %Sets the view to 2-D
end
axis equal;                 %Equates the aspect ratio for each axis
axis vis3d;                 %Freezes aspect ratio (allowing rotation)
axis tight;                 %Snaps the axis to the data set
```

The density profiles are visualized by drawing two separate patches using the `patch` function in Matlab. The first is an isosurface that connects all points of equal isovalue.

```
p1 = patch(isosurface(x,y,z,data,isovalue(in)), ...

'FaceColor',outcolor(map_choice(in),:),'EdgeColor','none','FaceAlpha',opacity
(in,1));
```

These are independently drawn for a BCC structure with three monomer types in Figure 4.2.1.

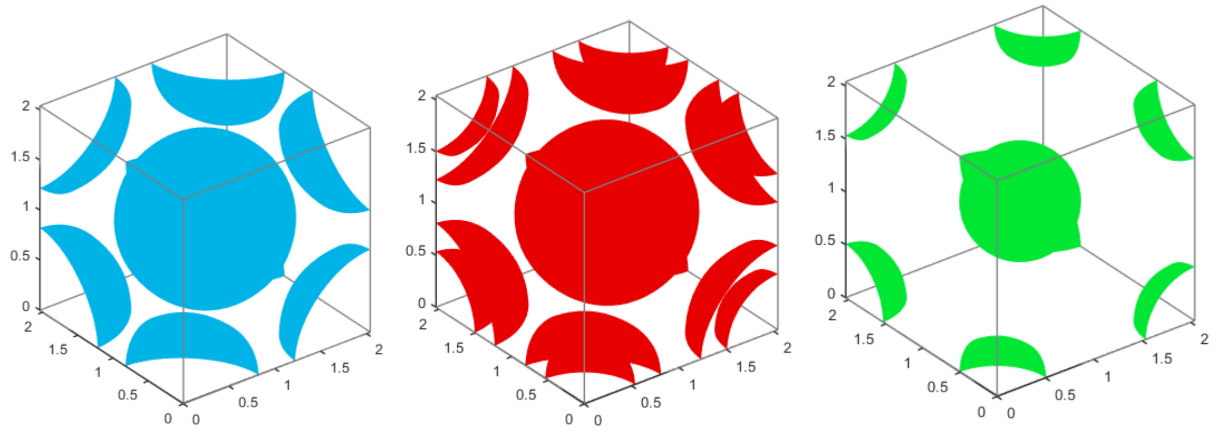


Figure 4.2.1 | `Isosurface` Examples
File Source: http://pscf.cems.umn.edu/tetrablocks/core-shell-spheres

The second is a patch drawn on the faces of the unit cell whose darkness increase with increasing density.

```
p2 = patch(isocaps(x,y,z,data,isovalue(in)), ...
        'FaceColor','interp','EdgeColor','none','FaceAlpha',opacity(in,2));
```

These are independently drawn for the same BCC structure in Figure 4.2.2.

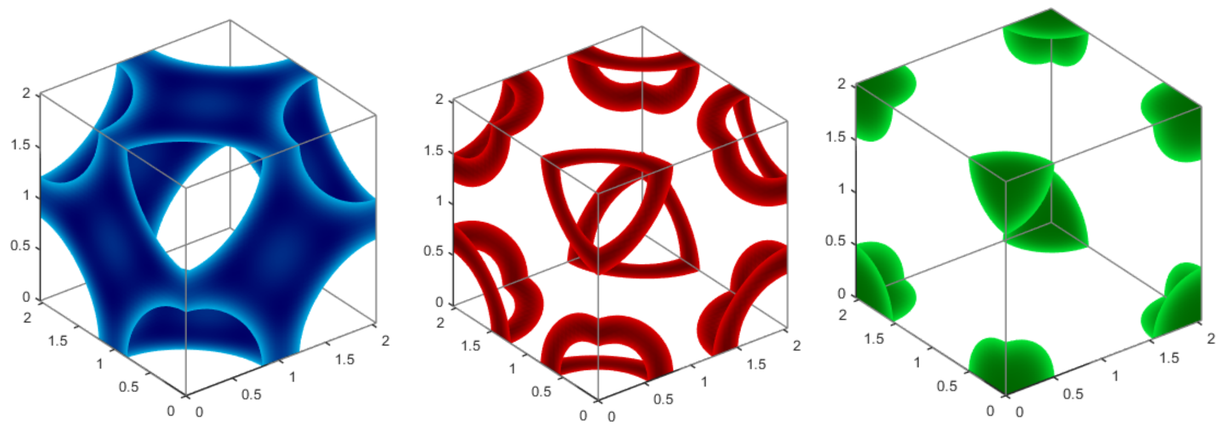


Figure 4.2.2 | `Isocaps` Examples
File Source: http://pscf.cems.umn.edu/tetrablocks/core-shell-spheres

When combined, the outputs look like this.

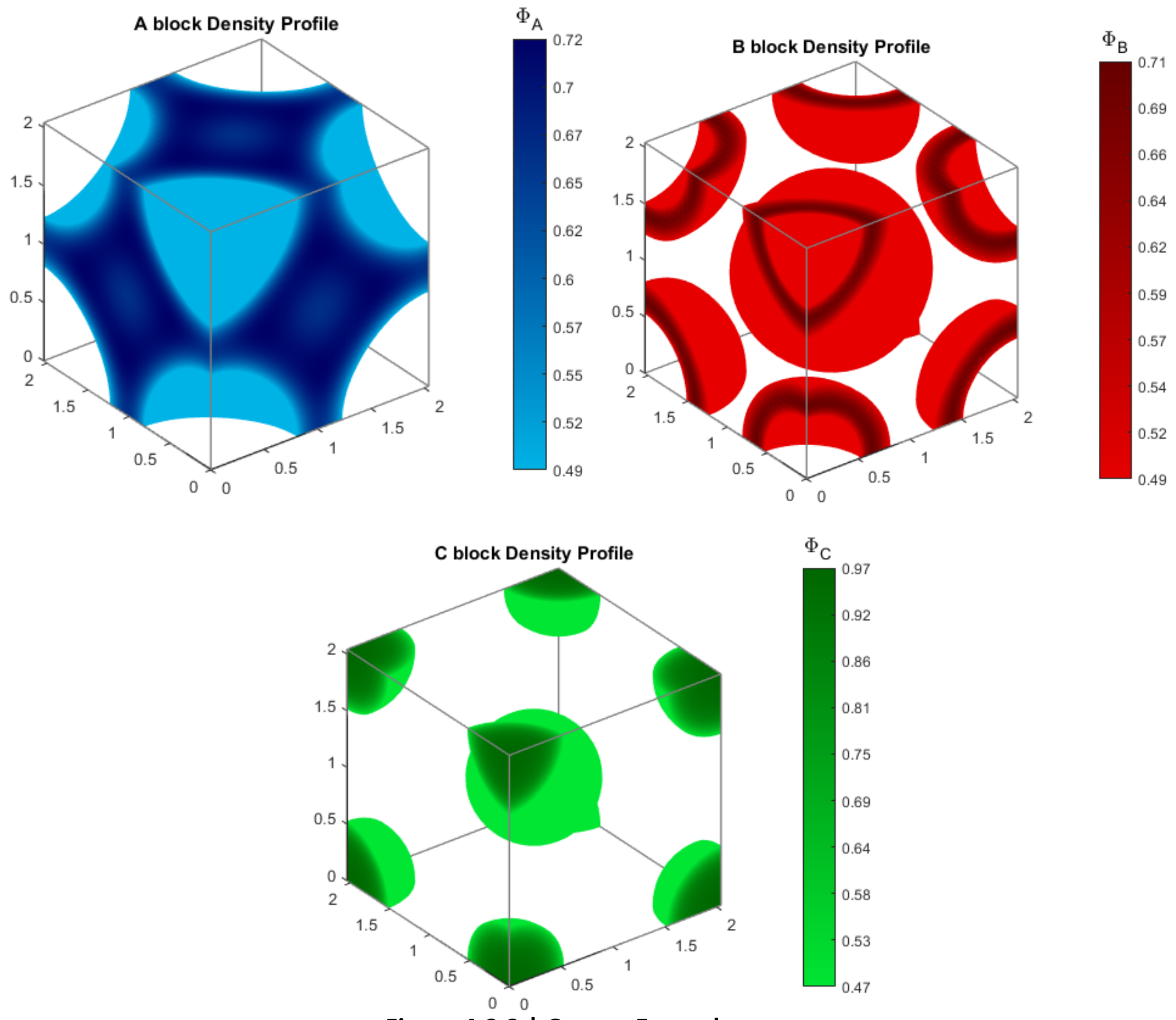


Figure 4.2.3 | Output Examples
File Source: http://pscf.cems.umn.edu/tetrablocks/core-shell-spheres

For hexagonal crystal systems, the calculations are done in the primitive unit cell (1/3 of the hexagon), which is not ideal for visualization. The following code rotates and redraws the patches (twice) so that a full hexagon may be visualized.

```
if strcmp(type,'hexagonal') == 1
        for i =1:2
            size = (grid(1)+1)*(grid(2)+1)*(grid(3)+1);
            coord_set = zeros(size,3);
            counter = 0;
            rotangle = 2*pi/3;

            for iz = 1:grid(3)+1
                for iy = 1:grid(2)+1
                    for ix = 1:grid(1)+1
                        counter = counter +1;
                        coord_set(counter,1) = x(ix,iy,iz) ;
                        coord_set(counter,2) = y(ix,iy,iz) ;
                        coord_set(counter,3) = z(ix,iy,iz) ;
                    end
                end
            end

            coord_set = coord_set*[cos(rotangle),sin(rotangle),0;-
sin(rotangle),cos(rotangle),0;0,0,1];

            counter = 0;
            for iz = 1:grid(3)+1
                for iy = 1:grid(2)+1
                    for ix = 1:grid(1)+1
                        counter = counter +1;
                        x(ix,iy,iz) = coord_set(counter,1) ;
                        y(ix,iy,iz) = coord_set(counter,2) ;
                        z(ix,iy,iz) = coord_set(counter,3) ;
                    end
                end
            end

            figure(in)
            data = R(:,:,:,in);
            p1 = patch(isosurface(x,y,z,data,isovalue(in)), ...

'FaceColor',outcolor(map_choice(in),:),'EdgeColor','none','FaceAlpha',opacity
(in,1));
            p2 = patch(isocaps(x,y,z,data,isovalue(in)), ...

'FaceColor','interp','EdgeColor','none','FaceAlpha',opacity(in,2));

        end
    end
```

Figure 4.2.4 contains outputs for a simple hexagonal system.

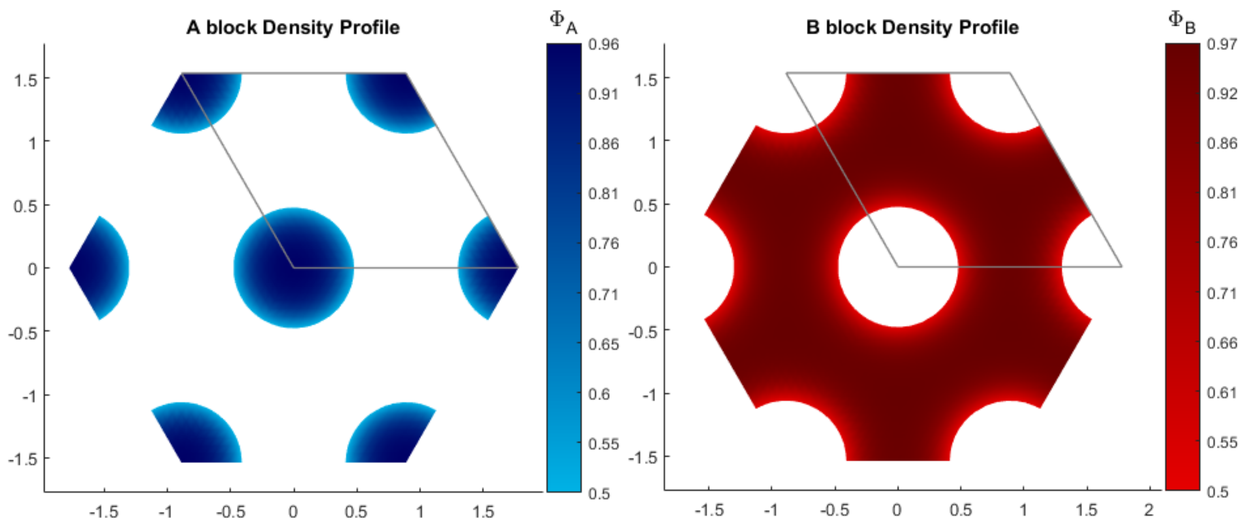


Figure 4.2.3 | Hexagonal Output Example
File Source: pscf-examples/diblock/hex

4.3 Drawing the Composite Density Profile

The composite density profile is drawn using the same basic principles as that of the discrete density profile. However, to accommodate for the differing monomer colours, new array must be created, `D`.

```
      D(:,:,:,in) = R(:,:,:,in) +in -1;
```

In this array, the (density) values from zero to one correspond to that of the A block, as usual. The (density) values from one to two correspond to that of the B block, those from two to three, that of the C block, and so on. In addition to this, a new colour map will have to be generated. Because the last patched value for any given monomer is very unlikely to be adjacent in value to that of the next monomer to be patched, some portion of the map must contain unused filler colours, whose size is determined by the matrix `fill`.

```
c = 0;
for in = comp_disp(1:end-1)
    c = c +1;

    if in~= n_mnr
        fill(in) = (newisovalue(comp_disp(c+1))-newisovalue(in))*(10^n_dp) -
cn(map_choice(in));
    end

end
```

This fill matrix is then used to create the filler sections of the colour map, which spans from the darkest colour for the first monomer to the lightest colour of the next.

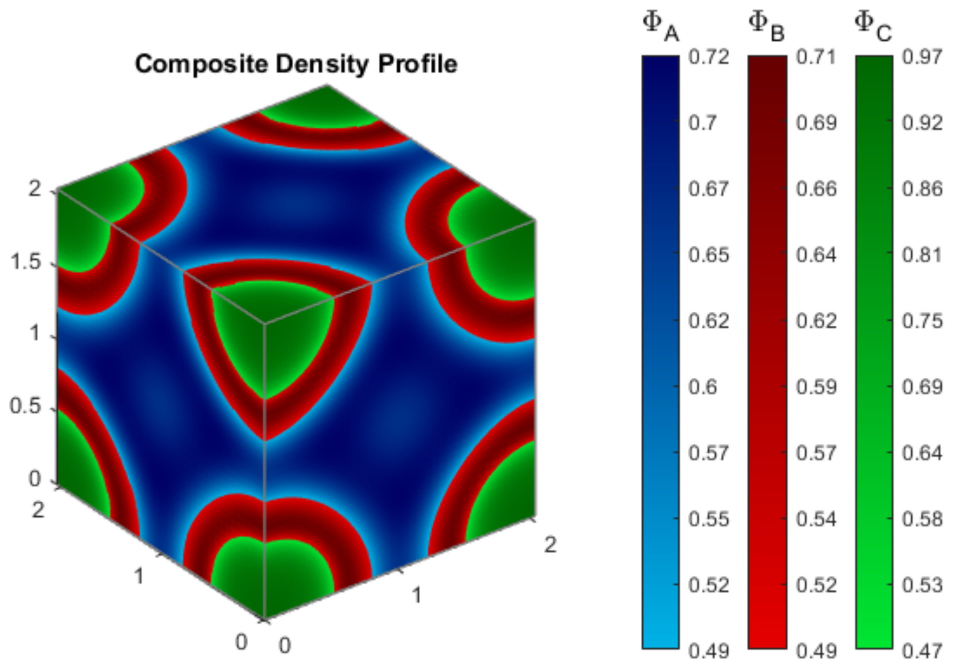
```
      c = 0;
```

```
        for in = comp_disp(1:end-1)
            c = c +1;
            temp_fillmap = zeros(round(fill(in)),3);
            temp_fillmap(:,1) =
        linspace(colourpad(map_choice(comp_disp(c)),1,2),colourpad(map_choice(c
        omp_disp(c+1)),1,1),round(fill(in))); %Red
            temp_fillmap(:,2) =
        linspace(colourpad(map_choice(comp_disp(c)),2,2),colourpad(map_choice(c
        omp_disp(c+1)),2,1),round(fill(in))); %Green
            temp_fillmap(:,3) =
        linspace(colourpad(map_choice(comp_disp(c)),3,2),colourpad(map_choice(c
        omp_disp(c+1)),3,1),round(fill(in))); %Blue
            fillmap_store{in} = temp_fillmap;
        end
```

The composite colour map, `newmap` is then created by the following code. If the isovalue for any monomer is greater than the maximum density value at any unit cell face for that monomer, then the corresponding colour map is not necessary for visualization and will be neglected.

```
newmap = [];
for in = comp_disp
    if isovalue(in) < max(face_data(:,in))
        if in == n_mnr
            newmap = [newmap;map_store(map_choice(in))];
        else
            newmap = [newmap;map_store(map_choice(in));fillmap_store(in)];
        end
    end
end
```

Two examples of composite density profiles are shown in Figure 4.3.1.



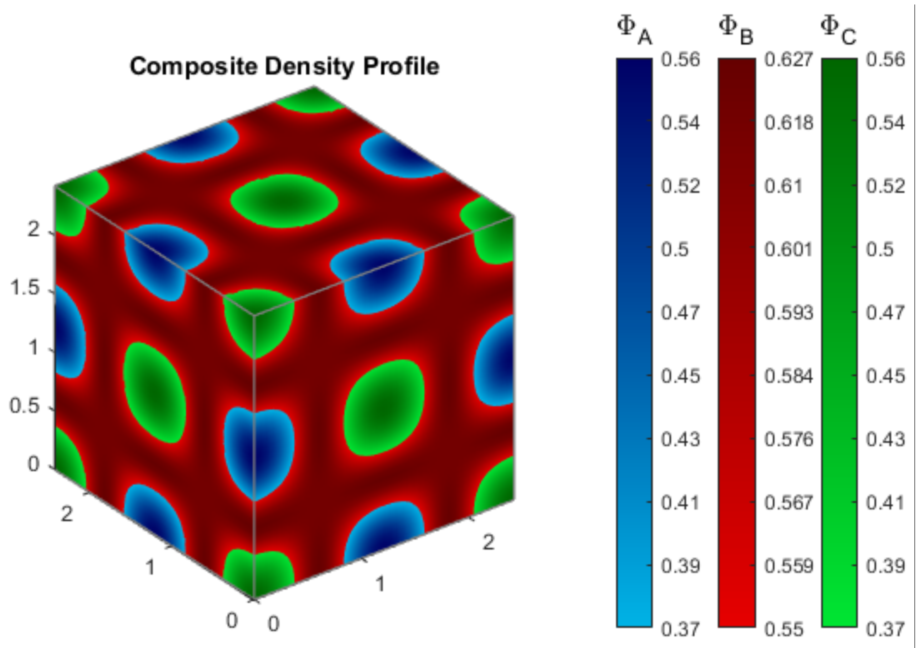


Figure 4.3.1 | Composite Density Profiles

~~File Source (top): http://pscf.cems.umn.edu/tetrablocks/core-shell-spheres~~

File Source (bottom): pscf-examples/triblock/NaCl

It should be noted that the opacity of each patch of each monomer can be modified. This is one of the user inputs, `opacity` and is described in Section 1.2. For the example on the top, if the opacity is set as `opacity` = [1,1;0,0.65;1,1], both the A and C blocks will be fully opaque, and the B block (red) will have invisible isosurfaces and isocaps with 65% opacity. This is illustrated in Figure 4.3.2.

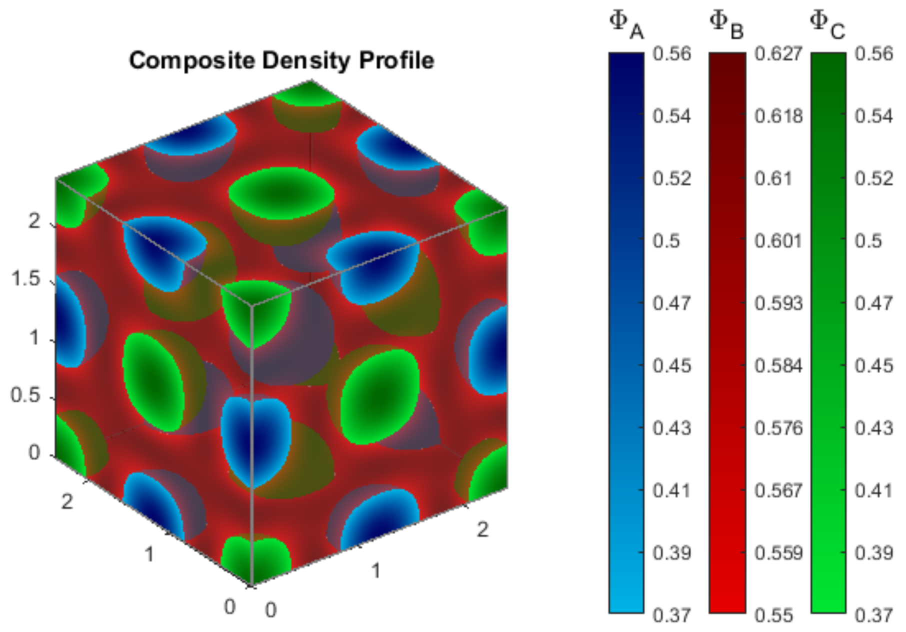


Figure 4.3.2 | Partially Transparent Composite Density Profile

File Source: pscf-examples/triblock/NaCl

4.4 Drawing the Unit Cell Outline

The subfunction `draw_lattice` draws the unit cell outline given the parameters `cell_d`, `angle`, `box_clr` and `thick`, whose meaning have been defined in Section 1. The default colour for the unit cell outline is set to grey and this can be seen in the images in Figure 4.2.1 through to Figure 4.3.2.

5 Scattering

The following code loops through each reflection, or set of Miller indices, as set by the user, to take the Fourier transform of the electron density in the grid and give the structure factor. The intensity, `I` is calculated as the square of this structure factor.

```
    for i_f = 1:length(F_store)
            h = F_store(i_f,1);
            k = F_store(i_f,2);
            l = F_store(i_f,3);
            F_sum = 0;
            for in = drawscatter
                x_s = zeros(grid);
                y_s = zeros(grid);
                z_s = zeros(grid);

                for iz=1:grid(3)
                    for iy=1:grid(2)
                        for ix=1:grid(1)
                            x_s(ix,iy,iz) = x(ix,iy,iz)/cell_d(1);
                            y_s(ix,iy,iz) = y(ix,iy,iz)/cell_d(2);
                            z_s(ix,iy,iz) = z(ix,iy,iz)/cell_d(3);
                            F_sum = F_sum +
R(ix,iy,iz,in)*exp(2*1i*pi*((h*x_s(ix,iy,iz))+(k*y_s(ix,iy,iz))+(l*z_s(
ix,iy,iz))));

                        end
                    end
                end

            end
            F_store(i_f,4) = F_sum * cell_d(1)*cell_d(2)*cell_d(3);
            F_store(i_f,5) = abs(F_sum)^2;
            I(i_f) = F_store(i_f,5);
            x_index(i_f) = i_f;
            x_index_label(i_f) = {[ h k l ]};
            x_label{i_f}=mat2str(cell2mat(x_index_label(i_f)));
            q(i_f) = sqrt((b(1)*h)^2 + (b(2)*k)^2 + (b(3)*l)^2);
        end
```

5.1 Intensity vs Scattering Vector

The following code finds duplicate values for `q`, takes the arithmetic mean of their intensities, and places the values back into a sorted matrix.

```
    plotmat = sortrows([q;I]');

        [q1,~,q_ind] = uniquetol(plotmat(:,1));
        plotmat_avg = [q1,accumarray(q_ind,plotmat(:,2),[],@mean)];

        q_sort = plotmat_avg(:,1);
        I_sort = plotmat_avg(:,2);
```

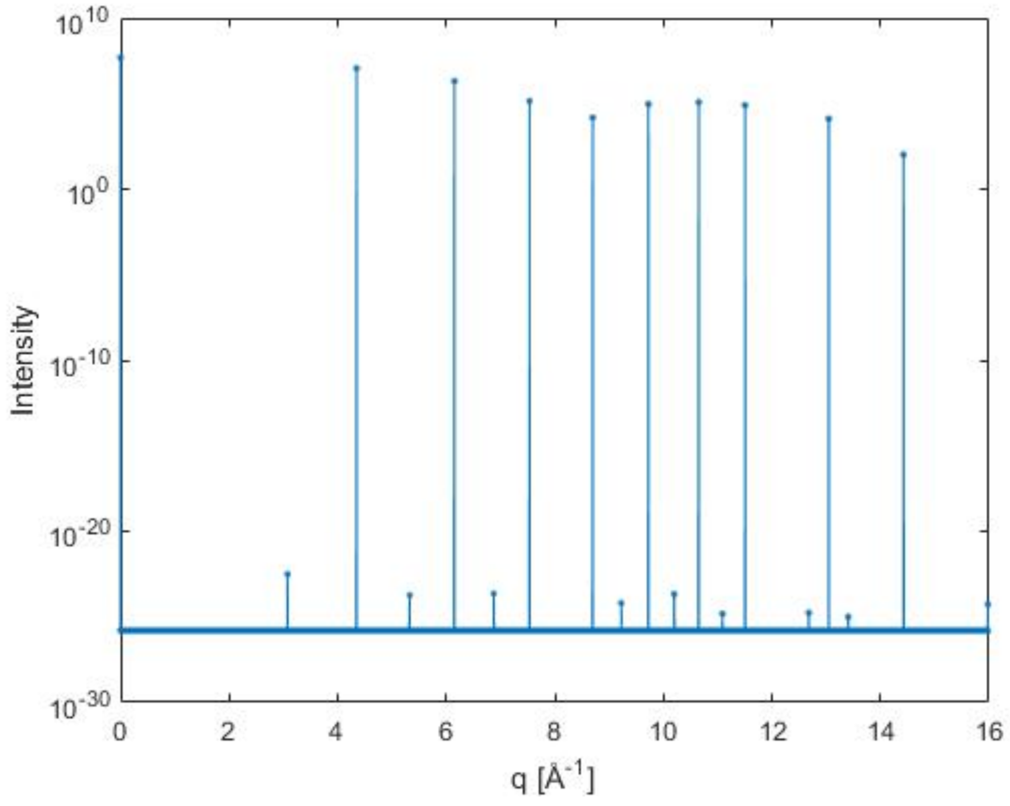
Many points are then added in between those calculated so that the function plotted will approximate a delta function.

```
    t_fcr = 100;
    q2 = linspace(0,max(q_sort),length(q_sort)*t_fcr);
    q_plot = sort([q2 q_sort']);
    I_plot = zeros(length(q_plot),1)+min(I);

    step = 1;
    for i = 1:length(q_plot)
        if q_plot(i)== q_sort(step)

            I_plot(i) = I_sort(step);
            step = step + 1;
            if step > length(q_sort)
                break
            end
        end
    end
```

Two output examples are provided in Figure 5.1.1, that of a BCC structure and an FCC structure.



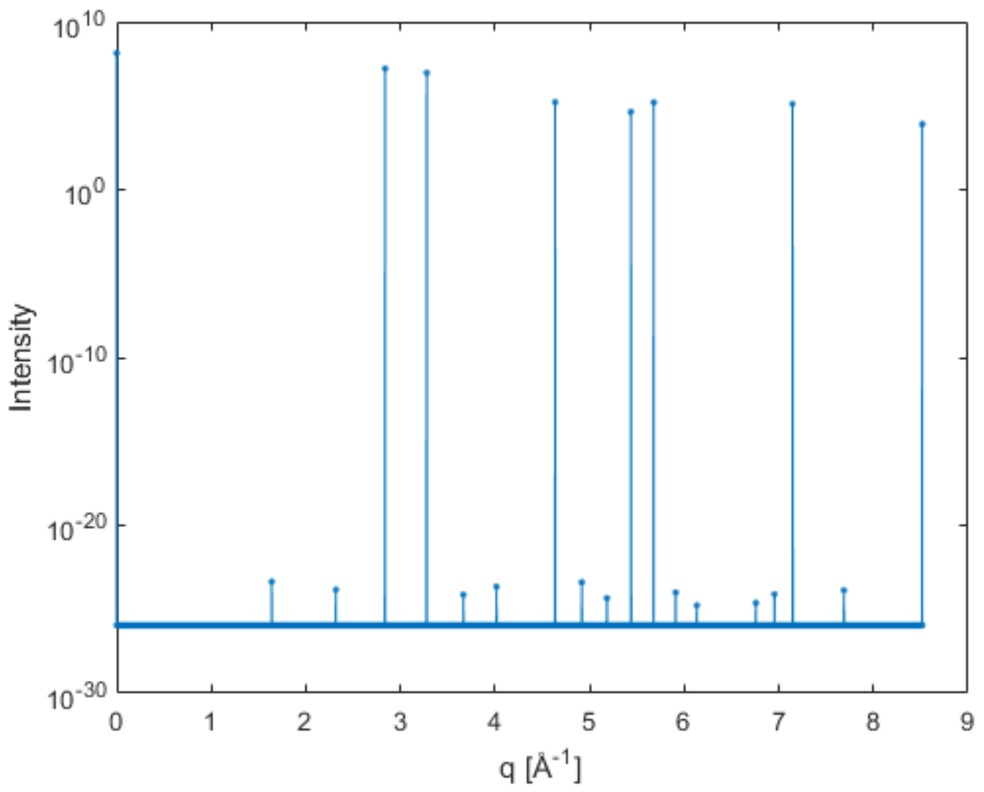
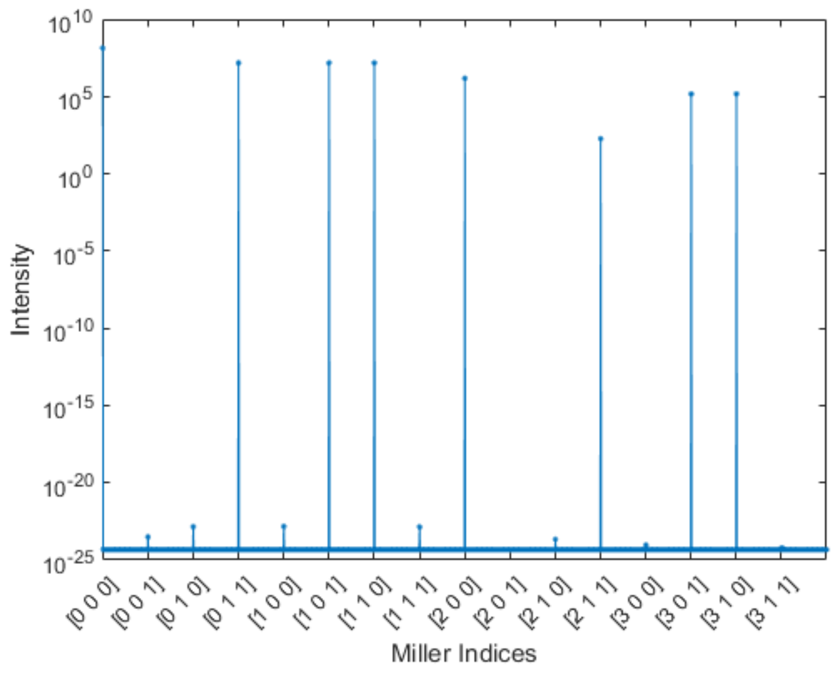


Figure 5.1.1 | I(q) plots for BCC (top) and FCC (bottom)

5.2 Intensity vs Miller Indices

The program also outputs plots of Intensity vs Miller Indices, which are stored in the main loop. Examples are shown in Figure 5.2.1 for a BCC and an FCC structure.



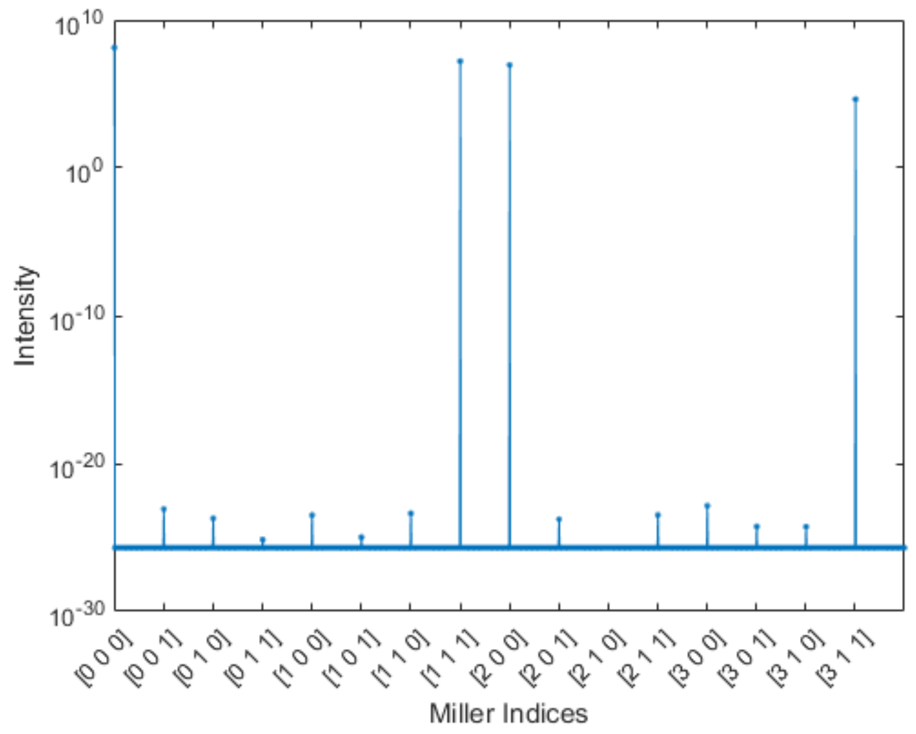


Figure 5.1.2 | Intensity vs Miller Indices plots for BCC (top) and FCC (bottom)

6 Density Along a Line

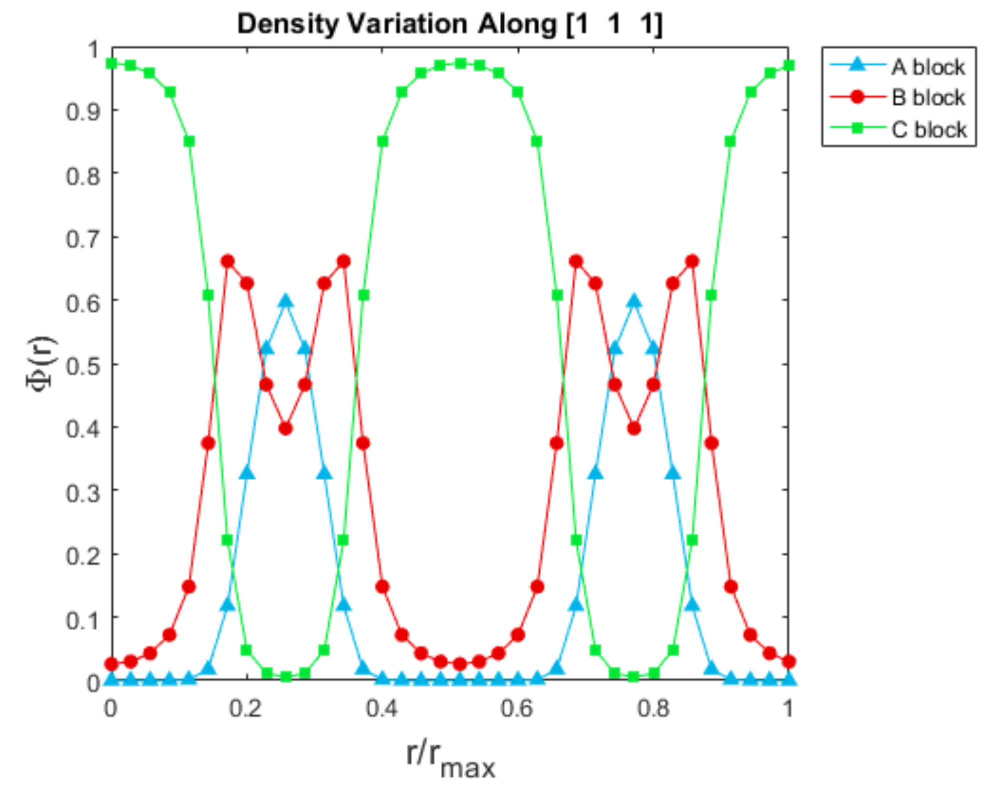
The final output of the program requires the user input `inputvec`. This is the direction vector along which the density values of each monomer will be found. The following code accomplishes this.

```
userinput = inputvec;
    start_coord = [1 1 1];
    end_coord = userinput .* grid + [1 1 1];
    dir_vec = end_coord-start_coord;
    step_length = max(abs(dir_vec));
    clear ix iy iz x_plot

    for il = 1:step_length
        ix(il) = start_coord(1)+ round((il-
1)*(dir_vec(1)/step_length));
        iy(il) = start_coord(2)+ round((il-
1)*(dir_vec(2)/step_length));
        iz(il) = start_coord(3)+ round((il-
1)*(dir_vec(3)/step_length));
        x_plot(il) = (il-1)/(step_length-1);
        for in= 1:n_mnr
            line_plot (in,il) = R (ix(il),iy(il),iz(il),in);

        end
    end
```

Two example outputs can be seen in Figure 6.1.



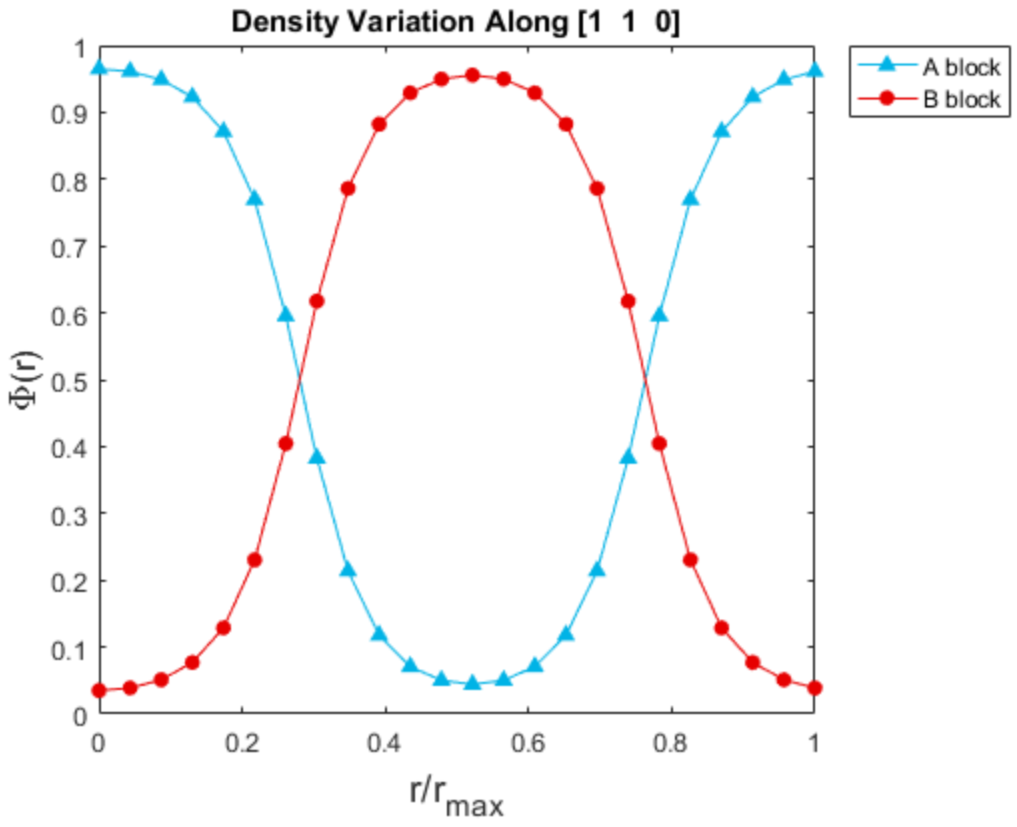


Figure 6.1 | Line Density plots for Core-Shell Spheres (top) and a Cylinder System (bottom)
File Source (top): http://pscf.cems.umn.edu/tetrablocks/core-shell-spheres
File Source (bottom): pscf-examples/diblock/hex