3.016 Problem Set #3.Group, 2010

1. Show that the face normals computed for the tetrahedron in the supplied notebook are correct.

The following data was taken from the Notebook given in the Pset and is listed here for ease of normal verification

```
tetraPoints = 2 / 3 {(*center of mass is 0,0,0*)
    {0, 0, Sqrt[2/3] - 1/(2 Sqrt[6])},
    {-1/(2 Sqrt[3]), -1/2, -1/(2 Sqrt[6])},
    {-1/(2 Sqrt[3]), 1/2, -1/(2 Sqrt[6])},
    {1/Sqrt[3], 0, -1/(2 Sqrt[6])}};
```

```
(**tetraFaces = {
   {2,3,4},(*normal {0,0,-1}*)
   {3,2,1}, (*normal { 2 Sqrt[2]/3,0,1/3}*)
   {4,1,2},(*normal{Sqrt[2]/3,-Sqrt[2/3] ,1/3}*)
   {1,4,3} (*normal {Sqrt[2]/3,Sqrt[2/3] ,1/3} *)
  };**)
```

Verifying the Normal for Face 1:

```
GivenNormal1 = {0, 0, -1}
```

```
{0, 0, -1}
```

This just defines GivenNormal1 to be the normal stated in the problem for the face indexed by {2, 3, 4}.

```
V1a = Simplify[tetraPoints[[2]] - tetraPoints[[4]]]
```

$$\left\{-\frac{1}{\sqrt{3}}, -\frac{1}{3}, 0\right\}$$

V1a represents the vector going from point 4 to point 2.

```
V1b = Simplify[tetraPoints[[3]] - tetraPoints[[4]]]
```

$$\left\{-\frac{1}{\sqrt{3}}, \frac{1}{3}, 0\right\}$$

V1a represents the vector going from point 4 to point 3.

```
OrthV1 = Cross[V1a, V1b]
```

$$\left\{0, 0, -\frac{2}{3\sqrt{3}}\right\}$$

By crossing the two vectors, we obtain the vector in the direction of the plane 1's normal.

```
UnitOrthV1 = Normalize[OrthV1]
```

```
{0, 0, -1}
```

Here, OrthV1 was normalized so that it can make seeing information easier further down the road.

```
Proof1 = Dot[UnitOrthV1, GivenNormal1]
```

```
1
```

By taking the Dot Product for two unit vectors and getting 1, shows that the two vectors are 0 degree apart and identical.

Verifying the Normal for Face 2 similarly:

```
GivenNormal2 = {2 Sqrt[2]/3, 0, 1/3}
```

$$\left\{\frac{2\sqrt{2}}{3}, 0, \frac{1}{3}\right\}$$

This just defines GivenNormal2 to be the normal stated in the problem for the face indexed by {3, 2, 1}.

```
V2a = Simplify[tetraPoints[[1]] - tetraPoints[[3]]]
```

$$\left\{\frac{1}{3\sqrt{3}}, -\frac{1}{3}, \frac{2\sqrt{\frac{2}{3}}}{3}\right\}$$

V2a represents the vector going from point 3 to point 1.

```
V2b = Simplify[tetraPoints[[2]] - tetraPoints[[3]]]
```

$$\left\{0, -\frac{2}{3}, 0\right\}$$

V2b represents the vector going from point 3 to point 2.

```
OrthV2 = Simplify[Cross[V2a, V2b]]
```

$$\left\{\frac{4\sqrt{\frac{2}{3}}}{9}, 0, -\frac{2}{9\sqrt{3}}\right\}$$

By crossing the two vectors, we obtain the vector in the direction of the plane 2's normal.

```
UnitOrthV2 = Normalize[OrthV2]
```

$$\left\{\frac{2\sqrt{2}}{3}, 0, -\frac{1}{3}\right\}$$

Here, OrthV2 was normalized so that it can make seeing information easier further down the road.

```
VerifyNormal = Dot[UnitOrthV2, GivenNormal2]
```

$$\frac{7}{9}$$

By taking the Dot Product for two unit vectors and not getting 1, shows that the two vectors are not 0 degree apart and **not identical**.

Verifying the Normal for Face 3 same as above:

```
GivenNormal3 = {Sqrt[2]/3, -Sqrt[2/3], 1/3}
```

$$\left\{\frac{\sqrt{2}}{3}, -\sqrt{\frac{2}{3}}, \frac{1}{3}\right\}$$

```
V3a = Simplify[tetraPoints[[1]] - tetraPoints[[4]]]
```

$$\left\{-\frac{2}{3\sqrt{3}}, 0, \frac{2\sqrt{\frac{2}{3}}}{3}\right\}$$

```
V3b = Simplify[tetraPoints[[2]] - tetraPoints[[4]]]
```

$$\left\{-\frac{1}{\sqrt{3}}, -\frac{1}{3}, 0\right\}$$

```
OrthV3 = Simplify[Cross[V3a, V3b]]
```

$$\left\{\frac{2\sqrt{\frac{2}{3}}}{9}, -\frac{2\sqrt{2}}{9}, \frac{2}{9\sqrt{3}}\right\}$$

```
UnitOrthV3 = Normalize[OrthV3]
```

$$\left\{\frac{\sqrt{2}}{3}, -\sqrt{\frac{2}{3}}, \frac{1}{3}\right\}$$

```
VerifyNormal3 = Dot[GivenNormal3, UnitOrthV3]
```

1

By taking the Dot Product for two unit vectors and getting 1, shows that the two vectors are 0 degree apart and identical.

Verifying the Normal for Face 4 same as above:

```
GivenNormal4 = {Sqrt[2]/3, Sqrt[2/3], 1/3}
```

$$\left\{\frac{\sqrt{2}}{3}, \sqrt{\frac{2}{3}}, \frac{1}{3}\right\}$$

```
V4a = Simplify[tetraPoints[[4]] - tetraPoints[[1]]]
```

$$\left\{\frac{2}{3\sqrt{3}}, 0, -\frac{2\sqrt{\frac{2}{3}}}{3}\right\}$$

```
V4b = Simplify[tetraPoints[[3]] - tetraPoints[[1]]]
```

$$\left\{-\frac{1}{3\sqrt{3}}, \frac{1}{3}, -\frac{2\sqrt{\frac{2}{3}}}{3}\right\}$$

```
OrthV4 = Simplify[Cross[V4a, V4b]]
```

$$\left\{\frac{2\sqrt{\frac{2}{3}}}{9}, \frac{2\sqrt{2}}{9}, \frac{2}{9\sqrt{3}}\right\}$$

```
UnitOrthV4 = Normalize[OrthV4]
```

$$\left\{\frac{\sqrt{2}}{3}, \sqrt{\frac{2}{3}}, \frac{1}{3}\right\}$$

```
VerifyNorm4 = Dot[UnitOrthV4, GivenNormal4]
```

1

By taking the Dot Product for two unit vectors and getting 1, shows that the two vectors are 0 degree apart and identical.

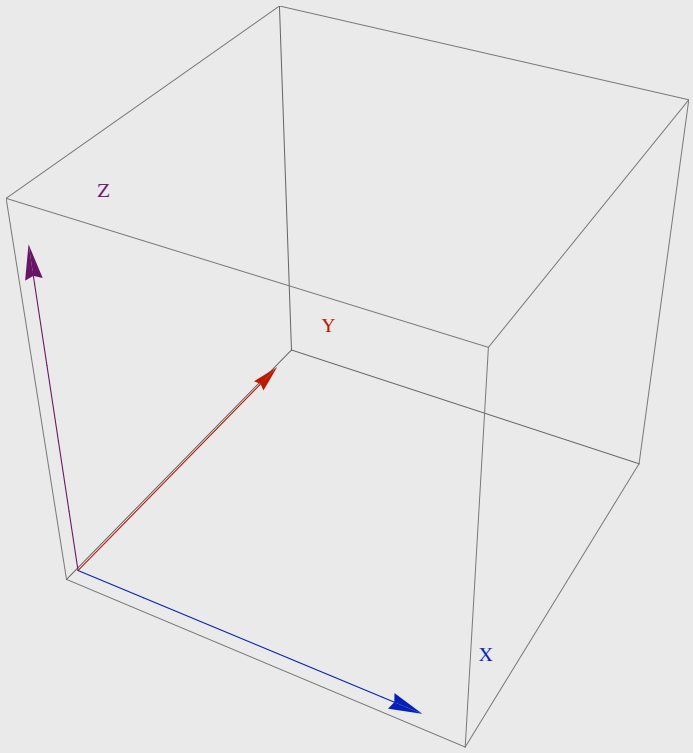
Problems 2 and 4

- **2. By placing Tetrahedra on a primitive Cubic lattice, create a structure with no additional symmetry (i.e., no rotational, mirror, inversion, rotoinversion, rotoreflection or glide symmetry).**

  The three things we have to do are to define functions which will 1) make the x-y-z coordinate marker, 2) make the cubic outline for any array we wish to have AND let us plot the 3D figures, and 3) change the points of our polyhedron and move them to specified coordinates in the cubic lattice.

  1) Making coordinate marker:

```
CoordinateMarker =
 Graphics3D[{{Darker[Blue], Arrow[{{0, 0, 0}, {.5, 0, 0}}],
    Text["X", {.55, .0625, .0625}]},
   {Darker[Red], Arrow[{{0, 0, 0}, {0, .5, 0}}],
    Text["Y", {.0625, .55, .0625}]},
   {Darker[Purple], Arrow[{{0, 0, 0}, {0, 0, .5}}],
    Text["Z", {.0625, .0625, .55}]}}]
```



2) Generating and plotting the cubic outline for any array we wish to have:

```
ArrayMaker[PointsAndColours_] := Show[CoordinateMarker,
  Table[Apply[WeirdTetraGraphics, PointsAndColours[[i]]],
   {i, 1, Length[PointsAndColours]}], Lighting → "Neutral"]
```

Now we can begin with the problem:

3) Function to move points to specified coordinates in the cubic lattice:

```
MoveMePlease[DirVec_, Ptlist_] :=
 Table[DirVec + Ptlist[[i]], {i, 1, Length[Ptlist]}]
```

- The above function translates the points of the object to whichever direction you choose.

- **Note that we can also generate a "weird" tetrahedron shape without symmetry by itself. Applying different colors would also work. Here it is an example of doing both.**

```
WeirdTetraPts = (1 / 10) {{-5, 1 / 5, .125},
    {1 / 2, -1 / 8, -.5}, {0, 3, 1.5}, {-4 / 5, 1 / 5, -.5}}
```

$$\left\{\left\{-\frac{1}{2}, \frac{1}{50}, 0.0125\right\}, \left\{\frac{1}{20}, -\frac{1}{80}, -0.05\right\}, \left\{0, \frac{3}{10}, 0.15\right\}, \left\{-\frac{2}{25}, \frac{1}{50}, -0.05\right\}\right\}$$

- These are asymmetrical points for such a tetrahedron.

```
WeirdTetraFaces = {{2, 1, 3}, {2, 4, 1}, {1, 4, 3}, {3, 2, 4}}
```

```
{{2, 1, 3}, {2, 4, 1}, {1, 4, 3}, {3, 2, 4}}
```

- These will connect points and form faces.

```
WeirdTetraColours = {Pink, LightBlue, Orange, Yellow};
```

- These assign colors for faces.

```
ColourWeirdTetraFaces[colours_] :=
 Riffle[colours, Polygon /@ WeirdTetraFaces]
```

- These riffle the objects to align them into correct GraphicsComplex format.

```
ColourWeirdTetraFaces[WeirdTetraColours]
```

```
{RGBColor[1, 0.5, 0.5], Polygon[{2, 1, 3}], RGBColor[0.87, 0.94, 1], Polygon[{2, 4, 1}],
 RGBColor[1, 0.5, 0], Polygon[{1, 4, 3}], RGBColor[1, 1, 0], Polygon[{3, 2, 4}]}
```

```
{RGBColor[1, 0.5`, 0.5`], Polygon[{2, 1, 3}],
 RGBColor[0.87`, 0.94`, 1], Polygon[{2, 4, 1}], RGBColor[1, 0.5`, 0],
 Polygon[{1, 4, 3}], RGBColor[1, 1, 0], Polygon[{3, 2, 4}]}
```

```
{RGBColor[1, 0.5, 0.5], Polygon[{2, 1, 3}], RGBColor[0.87, 0.94, 1], Polygon[{2, 4, 1}],
 RGBColor[1, 0.5, 0], Polygon[{1, 4, 3}], RGBColor[1, 1, 0], Polygon[{3, 2, 4}]}
```

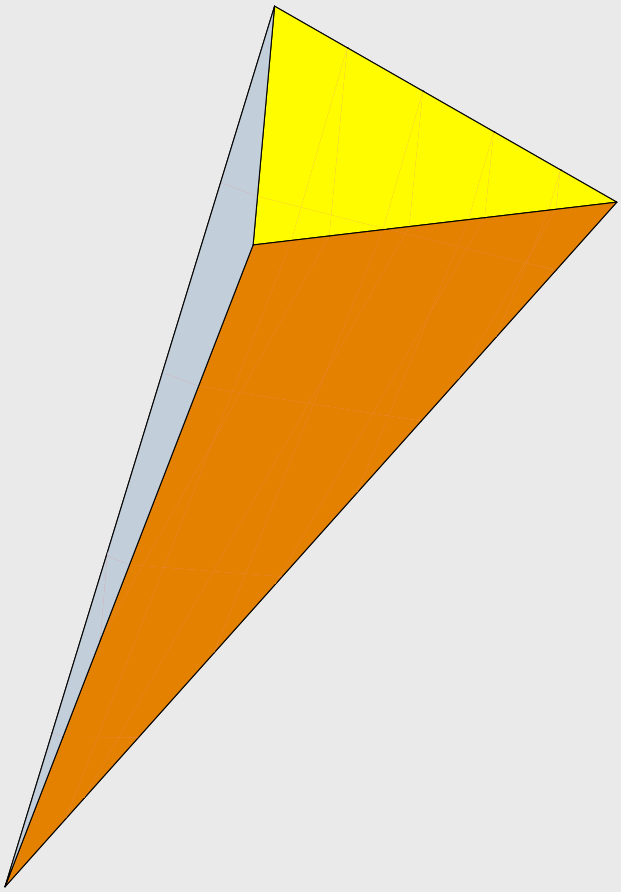
- This is an exaple of the function.

Now let's see our figure:

```
WeirdTetraGraphics[thepoints_, colourset_] := Graphics3D[
  GraphicsComplex[thepoints, ColourWeirdTetraFaces[colourset]],
  Lighting → "Neutral", Boxed → False]
```

This creates a weird tetrahedron given the set of points and the color set/plane assignment you want.

```
WeirdTetraGraphics[WeirdTetraPts, WeirdTetraColours]
```



- This is our strange tetrahedron.

```
ArraySetUp = Flatten[Table[{1, 0, 0} i + {0, 1, 0} j + {0, 0, 1} k,
    {i, -1, 1, 2}, {j, -1, 1, 2}, {k, -1, 1, 2}], 2]
```

```
{{-1, -1, -1}, {-1, -1, 1}, {-1, 1, -1}, {-1, 1, 1}, {1, -1, -1}, {1, -1, 1}, {1, 1, -1}, {1, 1, 1}}
```

- This has created a set of points with the center of mass around the origin.

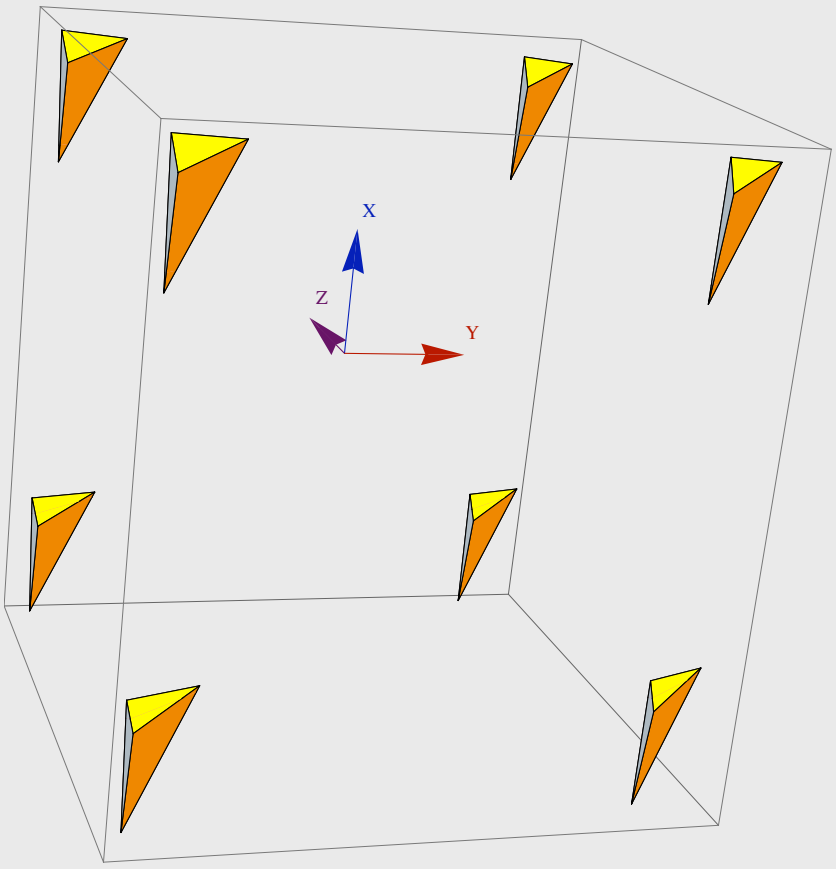
```
TetraSetUp[a_, n_] :=
  Table[{MoveMePlease[ArraySetUp[[i]], WeirdTetraPts],
    WeirdTetraColours}, {i, a, n}];
```

- This is a function that can assign any number of tetrahedrons from a to n to the array of locations mentioned earlier.

```
AWeirdTetraLattice = TetraSetUp[1, Length[ArraySetUp]];
```

- This puts it into correct format

```
ArrayMaker[AWeirdTetraLattice]
```



- This is the function in action.

Problems 3, 5, and 6

**Problem 3 is with Problem 5, 6 AFTER Problem 4

Problem 4: By placing colored tetrahedra onto the primitive cubic lattice, produce a structure with a mirror plane.

We can use the information and equations which we had in problem two. We can even use that tetrahedra, this will save us time.

```
TransformersRobotsInDisguise[Matrix_, PtList_] :=
 Table[Matrix.PtList[[i]], {i, 1, Length[PtList]}]
```

- This function will take make transform a set of points using a matrix, a mirror matrix in this case.

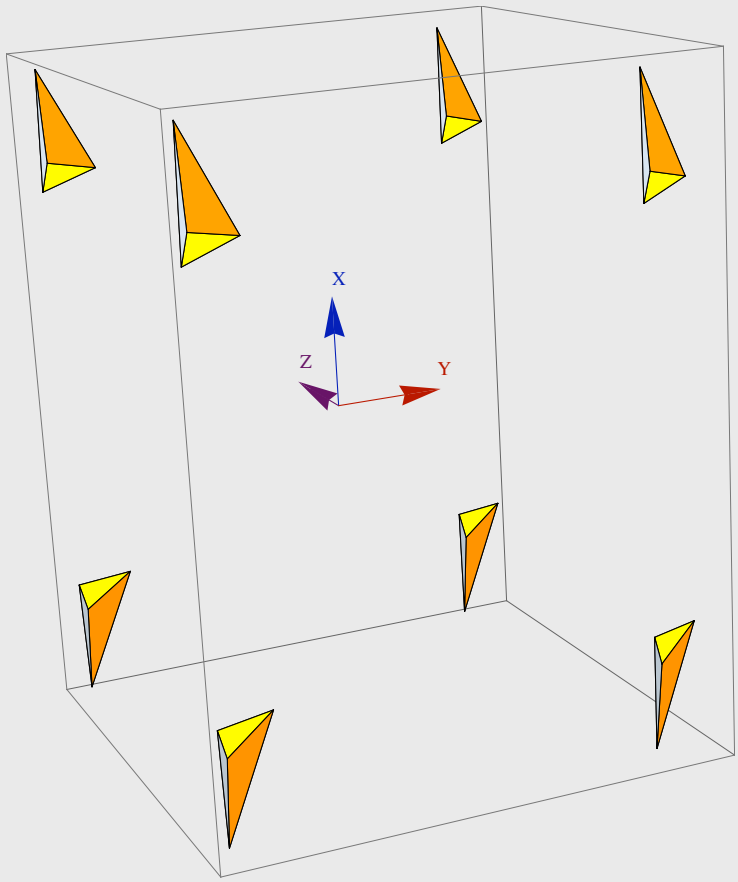
```
TetraSetUpV2[a_, n_] := Table[{MoveMePlease[ArraySetUp[[i]],
     TransformersRobotsInDisguise[MirrorMatrix, WeirdTetraPts]],
    WeirdTetraColours}, {i, a, n}];
AWeirdTetraPlane = TetraSetUp[1, 4];
AWeirdTetraMirroPlane = TetraSetUpV2[5, 8];
MirrorMatrix = {{-1, 0, 0}, {0, 1, 0}, {0, 0, 1}};
```

- This function is similar to the previous placing function from problem 2, however it has the added effect of mirroring whatever it would like along the y-z plane.
  AWeirdTetraPlane is used to pick out the tetrahedrons you want to keep the same
  AWeirdTetraMirrorPlane is used to pick out the tetrahedrons you want to mirror.
  The MirrorMatrix is an operator on the points to mirror them from the y-z plane.

```
NewArray = Join[AWeirdTetraPlane, AWeirdTetraMirroPlane];
```

- This will join the two previous variables to put them into proper GraphicsComplex format.

```
ArrayMaker[NewArray]
```



- And here is a cubic primitive cell with mirror symmetry.

- ***Problems 3, 5 ,6 Set Up/Info needed***

  We can share the info because the cubes used in these problems will have 2 colors only.

```
ArrayMaker2[PointsAndColours_] := Show[CoordinateMarker,
  Table[Apply[CubeGraphics, PointsAndColours[[i]]],
   {i, 1, Length[PointsAndColours]}], Lighting → "Neutral"]
```

- This makes an array for points and colors

```
CubeGraphics[colourset_, thepoints_] :=
 Graphics3D[GraphicsComplex[thepoints, ColorCubeFaces[colourset]],
  Lighting → "Neutral", Boxed → False]
```

- This will form a cube depending on the a given color set and point set.

```
ColorCubeFaces[Colours_] := Riffle[Colours, Polygon /@ CubeFaces]
```

This will put colors and planes into proper GraphicsComplex format

```
PointsForACube = (1 / 5) {
    {0, 0, 0},
    {0, 1, 0},
    {0, 1, 1},
    {0, 0, 1},
    {1, 1, 1},
    {1, 0, 0},
    {1, 1, 0},
    {1, 0, 1}}
```

$$\left\{\{0, 0, 0\}, \left\{0, \frac{1}{5}, 0\right\}, \left\{0, \frac{1}{5}, \frac{1}{5}\right\}, \left\{0, 0, \frac{1}{5}\right\}, \left\{\frac{1}{5}, \frac{1}{5}, \frac{1}{5}\right\}, \left\{\frac{1}{5}, 0, 0\right\}, \left\{\frac{1}{5}, \frac{1}{5}, 0\right\}, \left\{\frac{1}{5}, 0, \frac{1}{5}\right\}\right\}$$

- These are points for a cube with one of its points on the origin, edge size of 1/10, and has all its points on the x, y, and z axis except for the opposite corner from the origin

```
CubeFaces = {{1, 2, 3, 4}, {5, 3, 2, 7},
   {4, 3, 5, 8}, {8, 4, 1, 6}, {8, 5, 7, 6}, {1, 2, 7, 6}}
```

```
{{1, 2, 3, 4}, {5, 3, 2, 7}, {4, 3, 5, 8}, {8, 4, 1, 6}, {8, 5, 7, 6}, {1, 2, 7, 6}}
```

- These assign cube faces

```
ColorsForACube = {Red, Red, Blue, Blue, Blue, Red, Red, Red}
```

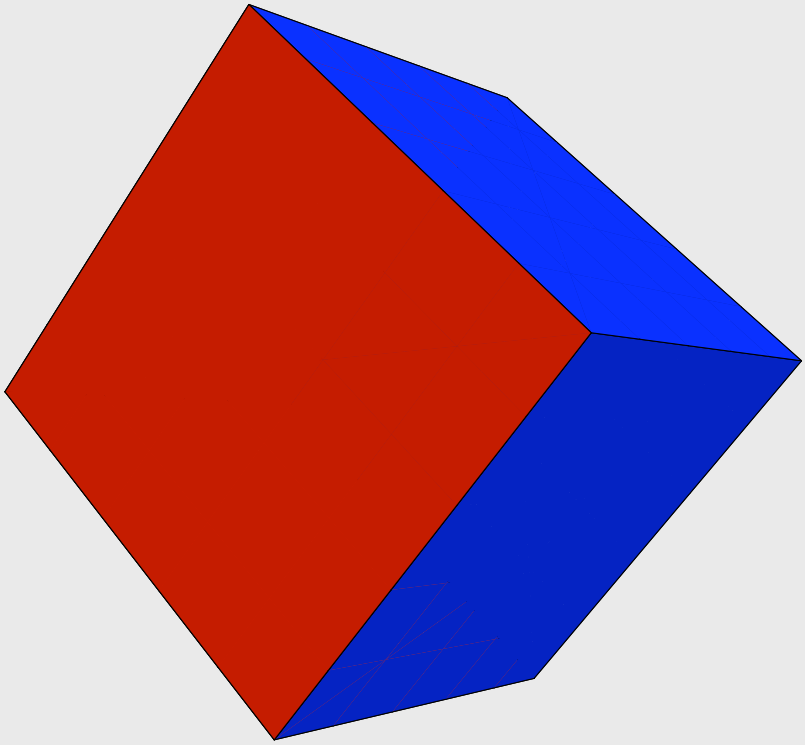
```
{RGBColor[1, 0, 0], RGBColor[1, 0, 0], RGBColor[0, 0, 1], RGBColor[0, 0, 1],
 RGBColor[0, 0, 1], RGBColor[1, 0, 0], RGBColor[1, 0, 0], RGBColor[1, 0, 0]}
```

- These assign cube face colors.

```
CubeGraphics[thepoints_, colourset_] :=
 Graphics3D[GraphicsComplex[thepoints, ColorCubeFaces[colourset]],
  Lighting → "Neutral", Boxed → False]
```

- This will create a cube depending on the proper color set and the points given.

```
CubeGraphics[PointsForACube, ColorsForACube]
```



- This is the cube we will use the the three fold rotational axis.

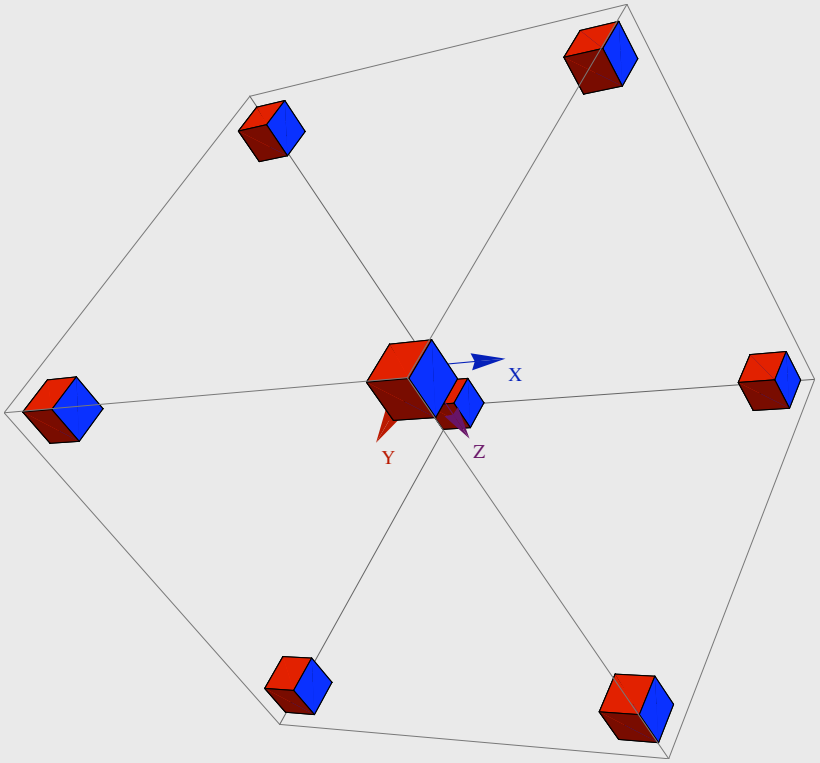
```
CubeSetUp[a_, n_] :=
  Table[{MoveMePlease[ArraySetUp[[i]], PointsForACube],
    ColorsForACube}, {i, a, n}];
```

- This will move cubes to an array of positions near the corners of the cube.

```
ThreeFoldRotation = CubeSetUp[1, Length[ArraySetUp]];
```

- This is the correct format for the array we want

```
ArrayMaker2[ThreeFoldRotation]
```



- This is a primitive cell with a three fold rotational axis

Now that we have all of this basic stuff down, we can go ahead and what we have to make lattices with rotation and symmetry.

```
CubeSetUpV2[a_, n_, radians_, plane_] :=
  Table[{MoveMePlease[ArraySetUp[[i]],
     TransformersRobotsInDisguise[RotationMatrix[radians, plane],
      PointsForACube]], ColorsForACube}, {i, a, n}];
```

- This will rotate the cubes any way we'd like.

```
CubeSetUpV3[a_, n_, matrix_] := Table[
   {MoveMePlease[ArraySetUp[[i]], TransformersRobotsInDisguise[
      matrix, PointsForACube]], ColorsForACube}, {i, a, n}];
```

- This is identical to the tetrahedron transformer function except it will work for cubes now.

```
NewArraySetUp = {{1, 1, 1}, {1, 2, 1}, {1, 1, 2},
  {1, 2, 2}, {2, 1, 1}, {2, 2, 1}, {2, 1, 2}, {2, 2, 2}}
```

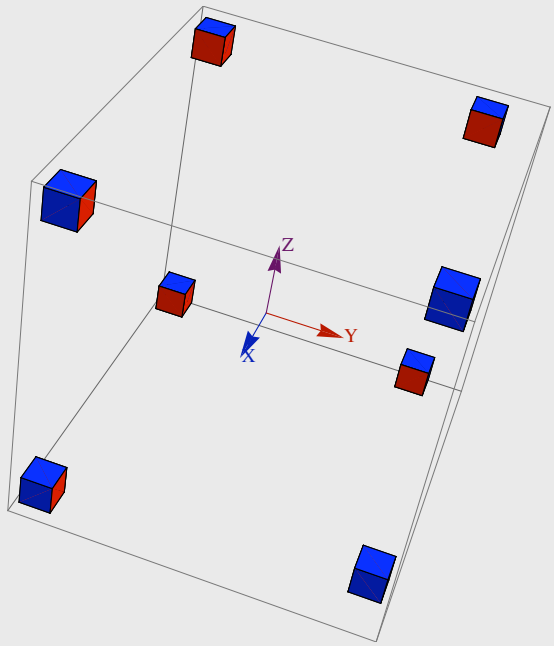
```
{{1, 1, 1}, {1, 2, 1}, {1, 1, 2}, {1, 2, 2}, {2, 1, 1}, {2, 2, 1}, {2, 1, 2}, {2, 2, 2}}
```

- These are new points we want to use

```
FourFoldAxisSetUp = Join[CubeSetUpV3[1, 2, MirrorMatrix],
   CubeSetUpV2[3, 4, Pi, {0, 0, 1}], CubeSetUp[5, 6],
   CubeSetUpV2[7, 8, Pi / 2, {0, 0, 1}]];
```

- To get four fold symmetry we just need to mirror the three fold rotational cubes, so this is what we've done.

```
ArrayMaker2[FourFoldAxisSetUp]
```



- And here it is.

```
MirrorMatrix3 = {{-1, 0, 0}, {0, -1, 0}, {0, 0, -1}}
```

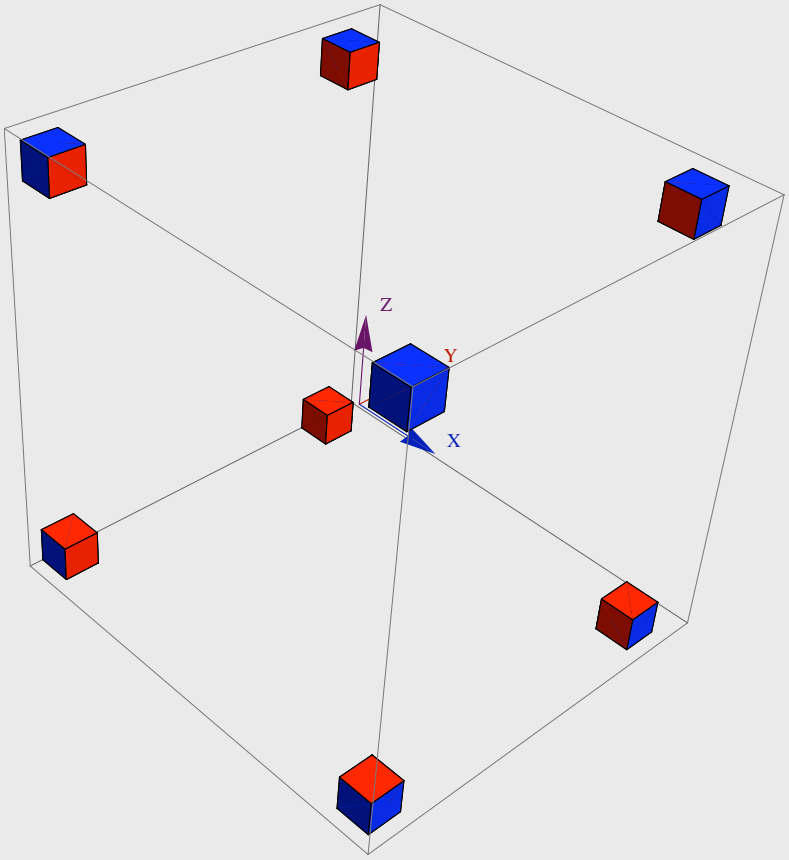
```
{{-1, 0, 0}, {0, -1, 0}, {0, 0, -1}}
```

- Here is a new matrix designed to flip over some of the cubes so that we can individually rotate certain cubes to get the desired three fold rotoinversion we want

```
ThreeFoldRotoinversionAxisSetUp =
  Join[CubeSetUpV2[1, 1, Pi, {0, -1, 0}],
   CubeSetUpV2[2, 2, Pi / 2, {0, -1, 0}],
   CubeSetUpV3[3, 3, {{-1, 0, 0}, {0, -1, 0}, {0, 0, -1}}],
   CubeSetUpV2[4, 4, Pi, {0, 0, 1}],
   CubeSetUpV2[5, 5, Pi / 2, {0, 1, 0}],
   CubeSetUp[6, 6], CubeSetUpV2[7, 7, Pi, {1, 0, 0}],
   CubeSetUpV2[8, 8, Pi / 2, {0, 0, 1}]];
```

- Now we've applied all the necessary operations to get this three fold rotoinversion primitive cell.

```
ArrayMaker2[ThreeFoldRotoinversionAxisSetUp]
```



- And here is our primitive cell with a three fold rotoinversion axis.