Geometry and Symmetry

In[ ]:=



6. Discrete groups of isometries in three space with introduction to crystallography

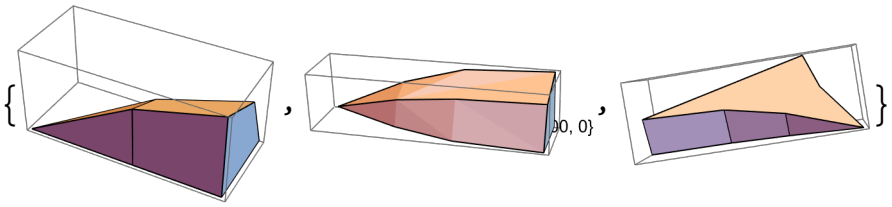
In this chapter we will continue with the idea of dynamic objects in $\mathbb{R}^3$ but use this for placing objects in a lattice rather than movement. By lattice I will mean, in this chapter, filling space with isometric images of a basic object using a discrete group of isometries containing 3 independent transforma-

tions. A finite subset of adjoining objects will be called a crystal such as the illustration above. This illustration is an edited, by AI, of a physical set of 18 *boxes* made by the author.

A main thrust of this chapter is the idea of the *primitive cell* of a tessellation or lattice. These sets form the tessellation or lattice by use of a discrete group of isometries. In classical crystallography these are often chosen to have a large finite group of symmetries. I am suggesting, however that a better choice is to use primitive cells with no symmetries except for the identity. However for convenience I will use compound cells with no symmetries made from other primitive cells

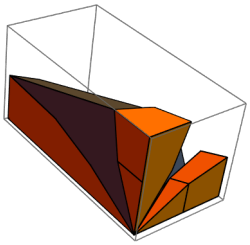
6.1 Barry' s Boxes

My first example I will call Barry's Boxes, named by my wife and daughter. These can be made by a *primitive cell* called the *corner*. A complete discussion can be found in https://barryhdayton.space/TransGeom/Dynamics/FullDocumentationOnLogo3D.pdf
Where these corner pieces are themselves derived from cells with no symmetries

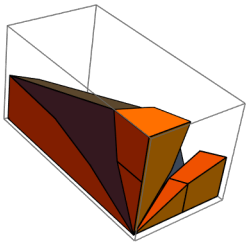


The corner piece is

In[•]:=



Out[•]=



For the purposes of discussion I will replace the boxes in the illustration above with boxes that are somewhat simpler to work with but instead of having diamond shaped windows to the empty insides of boxes I will have only colored diamonds and squares differentiating each side of the box like I did earlier with spindles. The **Mathematica** code is below.

The basic box is

In[1]:=

```
DBbox[α_] := {GrayLevel[.3], EdgeForm[],
  Polyhedron[α /@ {{{-1, -.5, .5}, {-1, .5, .5}, {-1, .5, -.5}, {-1, -.5, -.5}},
```

```
   {{-1, -.5, .5}, {-1, -.5, -.5}, {1, -.5, -.5}, {1, -.5, .5}},
   {{-1, -.5, -.5}, {1, -.5, -.5}, {1, .5, -.5}, {-1, .5, -.5}},
   {{-1, .5, .5}, {1, .5, .5}, {1, .5, -.5}, {-1, .5, -.5}},
   {{1, .5, .5}, {1, .5, -.5}, {1, -.5, -.5}, {1, -.5, .5}},
   {{1, .5, .5}, {1, -.5, .5}, {-1, -.5, .5}, {-1, .5, .5}}}], {Blue, EdgeForm[],
 Polygon[α@{{-.8, 0, .52}, {0, .3, .52}, {.8, 0, .52}, {0, -.3, .52}}]},
(*New polygons*)
{Orange, EdgeForm[], Polygon[α@{{-.8, 0, .52}, {-1, -.5, .52}, {-1, .5, .52}}],
 Polygon[α@{{.8, 0, .52}, {1, -.5, .52}, {1, .5, .52}}],
 Polygon[α@{{-1, .5, .52}, {0, .3, .52}, {0, .5, .52}}],
 Polygon[α@{{1, .5, .52}, {0, .3, .52}, {0, .5, .52}}],
 Polygon[α@{{-1, -.5, .52}, {0, -.3, .52}, {0, -.5, .52}}],
 Polygon[α@{{1, -.5, .52}, {0, -.3, .52}, {0, -.5, .52}}],

 Polygon[α@*σ4x@{{-.8, 0, .52}, {-1, -.5, .52}, {-1, .5, .52}}],
 Polygon[α@*σ4x@{{.8, 0, .52}, {1, -.5, .52}, {1, .5, .52}}],
 Polygon[α@*σ4x@{{-1, .5, .52}, {0, .3, .52}, {0, .5, .52}}],
 Polygon[α@*σ4x@{{1, .5, .52}, {0, .3, .52}, {0, .5, .52}}],
 Polygon[α@*σ4x@{{-1, -.5, .52}, {0, -.3, .52}, {0, -.5, .52}}],
 Polygon[α@*σ4x@{{1, -.5, .52}, {0, -.3, .52}, {0, -.5, .52}}],
 Polygon[α@*ηx@{{-.8, 0, .52}, {-1, -.5, .52}, {-1, .5, .52}}],
 Polygon[α@*ηx@{{.8, 0, .52}, {1, -.5, .52}, {1, .5, .52}}],
 Polygon[α@*ηx@{{-1, .5, .52}, {0, .3, .52}, {0, .5, .52}}],
 Polygon[α@*ηx@{{1, .5, .52}, {0, .3, .52}, {0, .5, .52}}],
 Polygon[α@*ηx@{{-1, -.5, .52}, {0, -.3, .52}, {0, -.5, .52}}],
 Polygon[α@*ηx@{{1, -.5, .52}, {0, -.3, .52}, {0, -.5, .52}}],
 Polygon[α@*ηx@*σ4x@{{-.8, 0, .52}, {-1, -.5, .52}, {-1, .5, .52}}],
 Polygon[α@*ηx@σ4x@{{.8, 0, .52}, {1, -.5, .52}, {1, .5, .52}}],
 Polygon[α@*ηx@*σ4x@{{-1, .5, .52}, {0, .3, .52}, {0, .5, .52}}],
 Polygon[α@*ηx@*σ4x@{{1, .5, .52}, {0, .3, .52}, {0, .5, .52}}],
 Polygon[α@*ηx@*σ4x@{{-1, -.5, .52}, {0, -.3, .52}, {0, -.5, .52}}],
 Polygon[α@*ηx@*σ4x@{{1, -.5, .52}, {0, -.3, .52}, {0, -.5, .52}}],
 Polygon[α@{{-1.02, -.5, .52}, {-1.02, -.5, -.52}, {-1.02, -.3, 0}}],
 Polygon[α@{{-1.02, -.5, .52}, {-1.02, .5, .52}, {-1.02, 0, .3}}],
 Polygon[α@{{-1.02`, 0.5`, -0.52`}, {-1.02`, -0.5`, -0.52`}, {-1.02`, 0.`, -0.3`}}],
 Polygon[α@{{-1.02`, 0.52`, 0.5`}, {-1.02`, 0.52`, -0.5`}, {-1.02`, 0.3`, 0.`}}],
 Polygon[α@*ρ1@{{-1.02, -.5, .52}, {-1.02, -.5, -.52}, {-1.02, -.3, 0}}],
 Polygon[α@*ρ1@{{-1.02, -.5, .52}, {-1.02, .5, .52}, {-1.02, 0, .3}}], Polygon[
  α@*ρ1@{{-1.02`, 0.5`, -0.52`}, {-1.02`, -0.5`, -0.52`}, {-1.02`, 0.`, -0.3`}}],
 Polygon[α@*ρ1@{{-1.02`, 0.52`, 0.5`}, {-1.02`, 0.52`, -0.5`}, {-1.02`, 0.3`, 0.`}}]
},

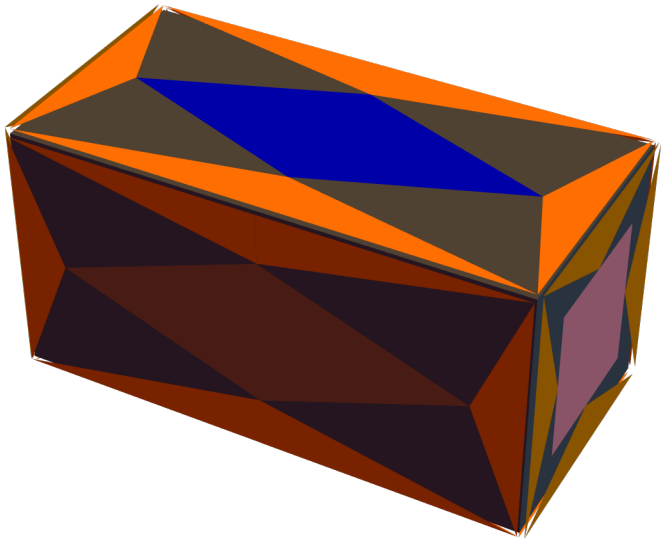
{Brown, EdgeForm[],
 Polygon[α@{{.8, -.52, 0}, {0, -.52, .3}, {-.8, -.52, 0}, {0, -.52, -.3}}]},
{Yellow, Polygon[α@{{-.8, 0, -.52}, {0, .3, -.52}, {.8, 0, -.52}, {0, -.3, -.52}}]},
{Green, Polygon[α@{{.8, .52, 0}, {0, .52, .3}, {-.8, .52, 0}, {0, .52, -.3}}]},
```

```
{Pink, Polygon[α@{{1.02, -.3, .3}, {1.02, .3, .3}, {1.02, .3, -.3}, {1.02, -.3, -.3}}]},
{Cyan,
 Polygon[α@{{-1.02, -.3, .3}, {-1.02, .3, .3}, {-1.02, .3, -.3}, {-1.02, -.3, -.3}}]}
```

DBbox[ι] where ι is the identity isometry .

In[•]:=
```
Graphics3D[{DBbox[ι]}, Boxed → False]
```

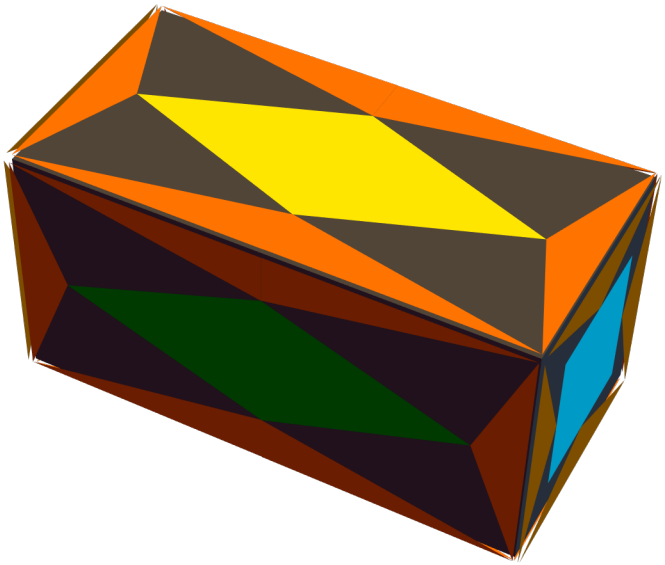
Out[•]=



We can see the rest by applying the inversion ζ

In[•]:=
```
Graphics3D[{DBbox[ζ]}, Boxed → False]
```

Out[•]=

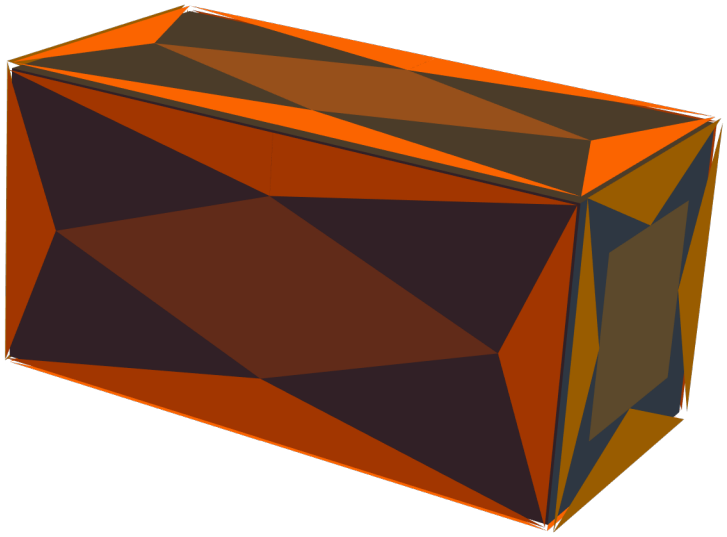


Note also that for physical work odd isometries such as ζ are not possible, but they may be theoretically interesting.

Sometimes I will just think of the box without considering the different colors and then I just have a rectangular box with reflections, rotations and inversions. The code is almost the same so I hide it here.

In[•]:=
```
Graphics3D[DBbox2[ι], Boxed → False]
```

Out[•]=



6.2 Review of 3D isometries

Here we give the theoretically possible isometries . For use in crystals only the even ones and rotations of $\frac{n\pi}{d}$, n, d positive integers, are possible. Isometries ending in TF3D are local defined in *Current-CodeCrystals.nb* others are built in. Here are the possible types with **Mathematica** functions, o,p,q,r ,v represent points or vectors. The type abbreviation is in parentheses.

```
Translation by vector v (T)
Rotation of angle ϴ and axis {p, q} (R)
Half turn about axis {p, q} (H)
Reflection in plane determined by three points {o, p, q} (RF)
Involution about ○ (INV)
Glide reflection in plane {p, q, r} and glide vector v - p (GRF)
Screw Displacement in angle ϴ and translation / axis {p, q} (SD)
Rotary reflection (RRF)
```

Actual code below.

Recall ι is the identity transform TranslationTransform[{0,0,0}]. It is typed as a translation. Also ζ or ζb is the inversion but typed as (INV). Half turns are rotations of angle π but are typed as (H).

Standard definitions in this chapter are

In[3]:=
```
ι := TranslationTransform[{0, 0, 0}]
τbx := TranslationTransform[{2, 0, 0}]
τby := TranslationTransform[{0, 1, 0}]
τbz := TranslationTransform[{0, 0, 1}]
τb[v_] := TranslationTransform[v]
ρ1 := TransformationFunction[{{-1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0}, {0, 0, 0, 1}}]
ρ2 := TransformationFunction[{{1, 0, 0, 0}, {0, -1, 0, 0}, {0, 0, 1, 0}, {0, 0, 0, 1}}]
ρ3 := TransformationFunction[{{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, -1, 0}, {0, 0, 0, 1}}]
ξ := ρ1@*ρ2@*ρ3
ξb[o_] := τb[o]@*ξ@*τb[-o]
InverseTF[λ_] := TransformationFunction[Inverse[TransformationMatrix[λ]]]
orthPart[α_] := Take[TransformationMatrix[α], 3, 3]
rotationTF3D[ang_, axis_] := Module[{ta, wab},
  ta = TranslationTransform[axis[[1]]];
  wab = RotationTransform[ang, InverseTF[ta]@axis[[2]]];
  ta@*wab@*InverseTF[ta]]
σ4 := rotationTF3D[Pi / 2, {{-1., 0, 0}, {0, 0, 1}}]
reflectionTF3D[{o_, p_, q_}] := Module[{v},
  v = Cross[p - o, q - o];
  getTF3D[{o, p, q, o + v}, {o, p, q, o - v}]]
glideReflectionTF3D[{p_, q_, r_, t_}] :=
TranslationTransform[t - p]@*reflectionTF3D[{p, q, r}]
rotationTF3D[ang_, axis_] := Module[{ta, wab},
  ta = TranslationTransform[axis[[1]]];
  wab = RotationTransform[ang, InverseTF[ta]@axis[[2]]];
  ta@*wab@*InverseTF[ta]]
rotaryReflectionTF3D[ang_, {o_, p_, q_}] :=
 rotationTF3D[ang, {o, o + Cross[p - o, q - o]}]@*reflectionTF3D[{o, p, q}]
screwDisplacementTF3D[ang_, {u_, v_}] :=
 TranslationTransform[v - u]@*rotationTF3D[ang, {u, v}]
σ4x := rotationTF3D[Pi / 2, {{0, 0, 0}, {1, 0, 0}}]
σ4y := rotationTF3D[Pi / 2, {{0, 0, 0}, {0, 1, 0}}]
σ4z := rotationTF3D[Pi / 2, {{0, 0, 0}, {0, 0, 1}}]
ηx := rotationTF3D[Pi, {{0, 0, 0}, {1, 0, 0}}]
ηy := rotationTF3D[Pi, {{0, 0, 0}, {0, 1, 0}}]
ηz := rotationTF3D[Pi, {{0, 0, 0}, {0, 0, 1}}]
```

Standard functions from earlier chapters **getTF3D, general position3DQ** are defined in the closed cell below to make this notebook self contained.

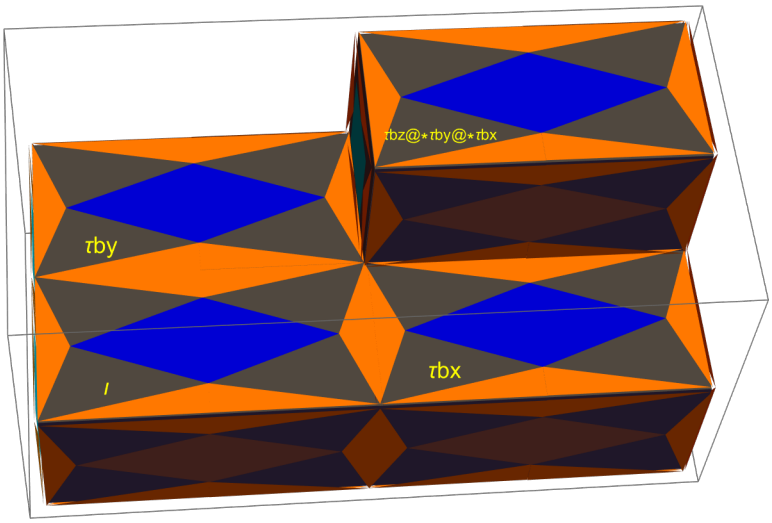
Here are some graphics illustrating these transformation types. Remember, only even transformations are physically possible.

The point **cb** below helps place the names in the graphics below . It also serves as a test point below.

In[31]:=
```
cb := {-.6, -.3, .54}
```

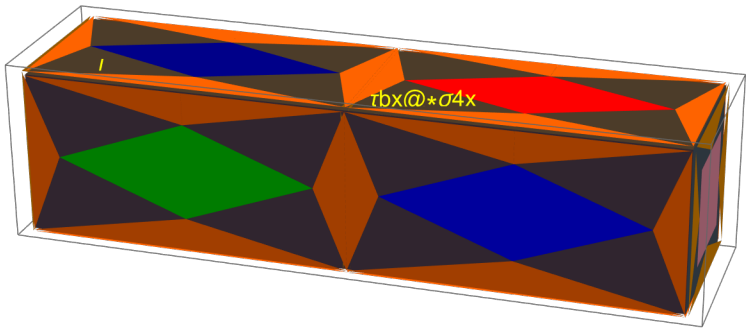
```
In[•]:= Graphics3D[{{{DBbox[ι]}, Inset[Style["ι", Medium, Yellow], cb]},
   {DBbox[τbx], Inset[Style["τbx", Medium, Yellow], τbx@cb]},
   {DBbox[τby], Inset[Style["τby", 12, Yellow], τby@cb]}, {DBbox[τbz@∗τby@∗τbx],
    Inset[Style["τbz@∗τby@∗τbx", 8, Yellow], τbz@∗τby@∗τbx@cb]}}, ImageSize → 400]
```

Out[•]=




```
In[•]:= Graphics3D[{{{DBbox[ι]}, Inset[Style["ι", Medium, Yellow], cb]},
   {DBbox[τbx@∗σ4x], Inset[Style["τbx@∗σ4x", Medium, Yellow], τbx@cb]}}]
```

Out[•]=

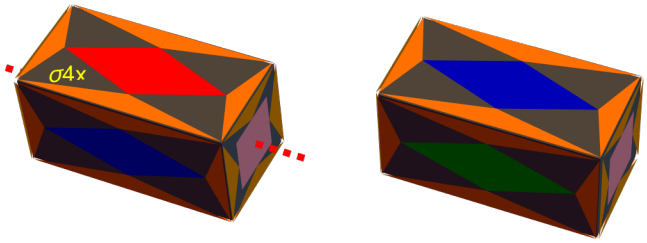


Below I add a translation of the standard box, unlabeled, in the picture as a reference .

Rotation (even transformation)

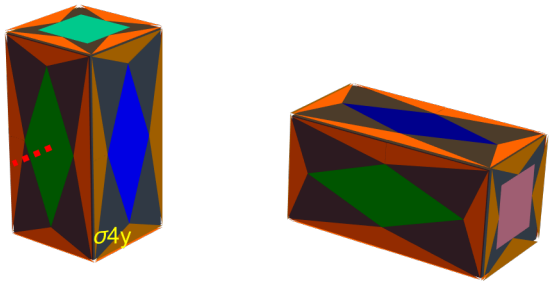
In[•]:=
```
Graphics3D[{{DBbox[σ4x], Inset[Style["σ4x", Medium, Yellow], cb]}, {DBbox[τb[{3, 1, 0}]]},
  {Red, Dashed, Thickness[.01], Line[{{1.5, 0, 0}, {-1.5, 0, 0}}]}}, Boxed → False]
```

Out[•]=



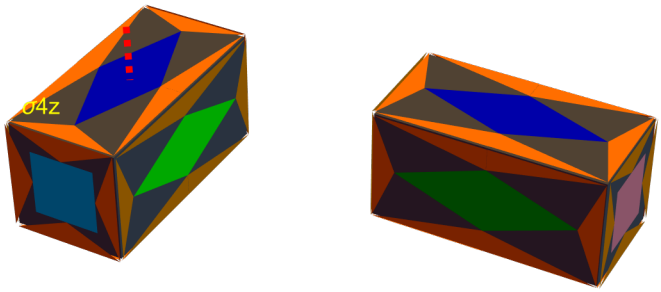
In[•]:=
```
Graphics3D[{{{DBbox[σ4y], Inset[Style["σ4y", Medium, Yellow], τb[{1.5, -.4, -1.1}]@cb]},
   {DBbox[τb[{3, 1, 0}]]}},
  {Red, Dashed, Thickness[.01], Line[{{0, 1, 0}, {0, -1, 0}}]}}, Boxed → False]
```

Out[•]=



In[•]:=
```
Graphics3D[{{{DBbox[σ4z], Inset[Style["σ4z", Medium, Yellow], τb[{.6, -.9, .4}]@cb]},
    {DBbox[τb[{3, 1, 0}]]},
    {Red, Dashed, Thickness[.01], Line[{{0, 0, 1}, {0, 0, -1}}]}}}, Boxed → False]
```

Out[•]=



Here the glide reflection with the original box in its home position showing the motion.

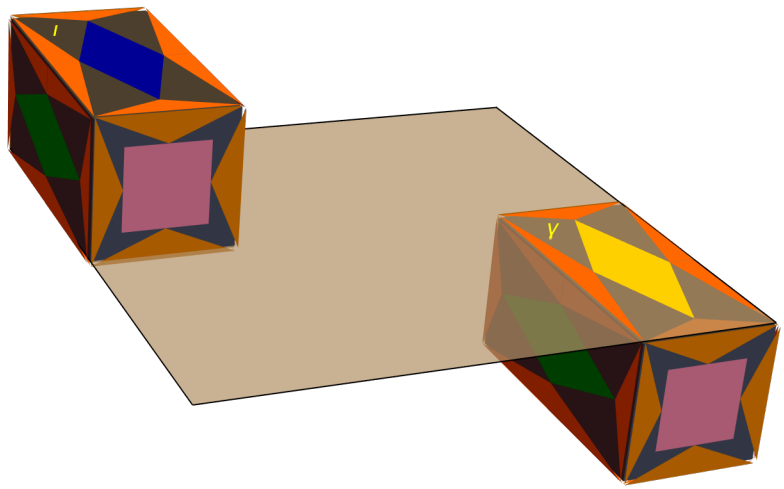
In[•]:=
```
γ = glideReflectionTF3D[{{0, 0, -.5}, {1, 0, -.5}, {0, 1, -.5}, {2, 3, -.5}}]
```

Out[•]=

$$\text{TransformationFunction}\left[\left(\begin{array}{ccc|c} 1. & 0. & 0. & 2. \\ 0. & 1. & 0. & 3. \\ 0. & 0. & -1. & -1. \\ \hline 0. & 0. & 0. & 1. \end{array}\right)\right]$$

```
In[•]:= Graphics3D[{{GrayLevel[.7], Opacity[.7],
     Polygon[{{-1, -.5, -.5}, {-1, 3.5, -.5}, {3, 3.5, -.48}, {3, -.5, -.48}}]},
    {DBbox[ι], Inset[Style["ι", Medium, Yellow], cb]},
    {DBbox[γ], Inset[Style["γ", Medium, Yellow], {0, 0, 1.12} + γ@cb]}}, Boxed → False]
```

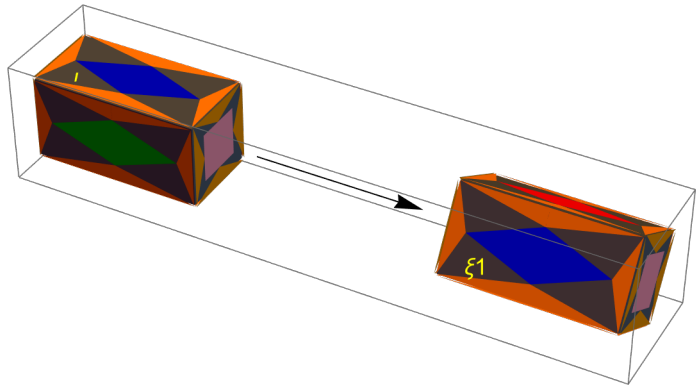
Out[•]=




```
In[•]:= ξ1 := screwDisplacementTF3D[Pi / 3, {{0, 0, 0}, {5, 0, 0}}]
```

```
In[•]:= Graphics3D[{{DBbox[ι], Inset[Style["ι", Medium, Yellow], cb]}, {DBbox[ξ1],
     Inset[Style["ξ1", Medium, Yellow], ξ1@cb]}, Arrow[{{1.5, 0, 0}, {3.5, 0, 0}}]}]
```

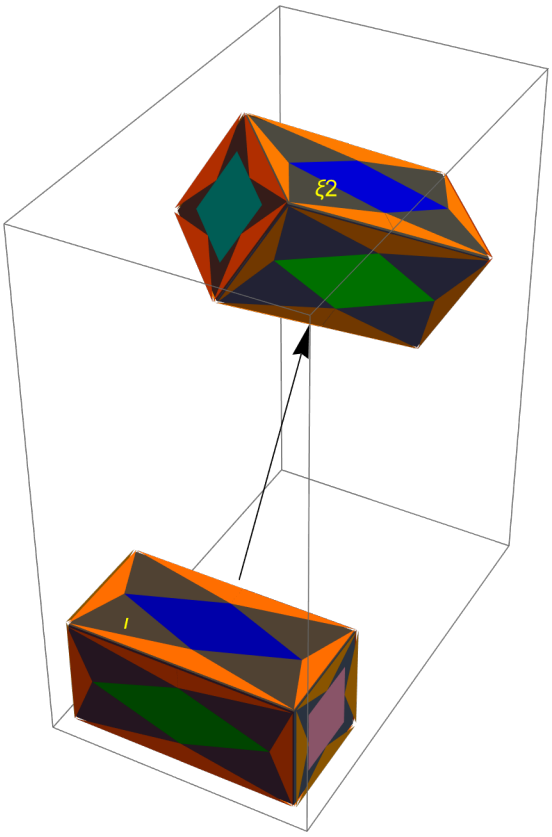
Out[•]=




```
In[•]:= ξ2 := screwDisplacementTF3D[Pi / 3, {{0, 0, 0}, {0, 2, 3}}]
```

```
In[•]:= Graphics3D[{{DBbox[ι], Inset[Style["ι", Medium, Yellow], cb]},
   {DBbox[ξ2], Inset[Style["ξ2", Medium, Yellow], ξ2@cb + {0, .02, .03}]},
   {Arrow[{{0, .44, .66}, {0, 1.6, 2.4}}]}}]
```

Out[•]=

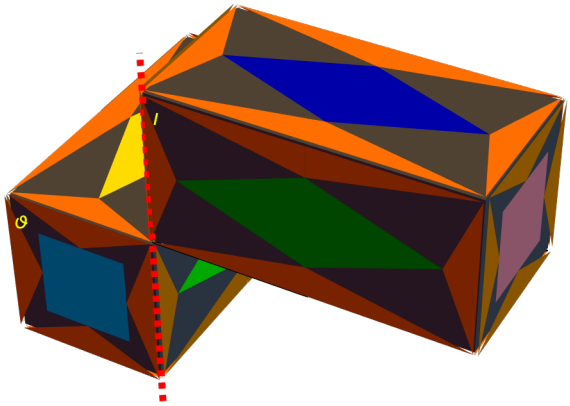


Rotary reflection with axis the dashed red line and angle Pi/2 and plane of reflection the bottom of the original box *ι*.

```
In[•]:= ϑ := rotaryReflectionTF3D[Pi / 2, {{-1, -.5, -.52}, {0, -.5, -.52}, {1, .5, -.52}}]
```

In[•]:=
```
Graphics3D[{{DBbox[ل], Inset[Style["ل", Medium, Yellow], {-.9, -.54, .4}]},
  {DBbox[ℴ], Inset[Style["ℴ", Medium, Yellow], {-1.9, -.54, -.66}]},
  {Polygon[{{-1, -.5, -.52}, {0, -.5, -.52}, {1, .5, -.52}}]},
  {Red, Dashed, Thickness[.01], Line[{{-1, -.5, -1.75}, {-1, -.5, +.75}}]}}, Boxed → False]
```

Out[•]=



It is usually easy to distinguish the type of an isometry and possibly some property such as the order of a rotation, screw displacement . Here is some code

This function is not a proof that all isometries are of the types described and in the case of glide reflection, rotary reflection and screw displacement, which are composite isometries, this does not tell how to decompose them. In fact it may return U, (unknown) in some cases, particularly for isometries close to the inversion which is, in an exact sense, a singular isometry in that arbitrary close isometries are not inversions. Unfortunately a proof requires working exactly.

An example is

In[•]:=
```
ϱ = rotaryReflectionTF3D[2 Pi / 3, {{0, 0, 0}, {1, 0, 0}, {0, 1, 0}}]
```

Out[•]=

$$\text{TransformationFunction}\left[\left(\begin{array}{ccc|c} -\frac{1}{2} & -\frac{\sqrt{3}}{2} & 0 & 0 \\ \frac{\sqrt{3}}{2} & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & -1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right)\right]$$

In[32]:=
```
classTF3D[δ_] := Module[{M, M1, M2, ordM1, K, tp, rr, gp, det},
  M = TransformationMatrix[δ];
  If[ Norm[M - IdentityMatrix[4]] < .001, Return["ι"]];
  M1 = Take[M, 3, 3];
  If[Norm[M1 - IdentityMatrix[3]] < .001 , Return["T"]];
  If[Norm[M1.M1 - IdentityMatrix[3]] < .001,
   (*Start involutions*)
   K = TransformationMatrix[δ@*δ];
   If[Det[M1] < 0 && Norm[K - IdentityMatrix[4]] > .001, Return["GRF"]];
   If[Det[M1] > 0 && Norm[K - IdentityMatrix[4]] > .001, Return["SD2"]];
   If[Norm[K - IdentityMatrix[4]] < .001 && Det[M1] > 0, Return["H"]];
   If[Norm[K - IdentityMatrix[4]] < .001 && Det[M1] < 0 &&
     Norm[M1 + IdentityMatrix[3]] < .001, Return["INVO"], Return["RF"]];
   Return["U"]];
  (*End Involutions*)
  ordM1 = 1;
  M2 = M1;
  tp = {.3924, -.1245, .7738};
  While[ordM1 < 7 && Norm[M2 - IdentityMatrix[3]] > .005, M2 = M2.M1;
   ordM1++];
  rr = {tp, δ@tp, δ@*δ@tp, δ@*δ@*δ@tp};
  gp = generalPosition3DQ[rr];
  det = Det[M1];
  If[gp,
   If[det > 0,
    Return[Switch[ordM1, 2, "SD2", 3, "SD3", 4, "SD4", 5, "TRSD", 6, "SD6", _, "SDU"]]];
   Return[Switch[ordM1, 4, "RRF4", 6, "RRF6", _, "RRFU"]]];
  Return[Switch[ordM1, 3, "R3", 4, "R4", 5, "R5", 6, "R6", _, "RU"]];
  Return["U1"]]
```

In[•]:=
```
classTF3D[ϱ, cb]
```

Out[•]=

```
RRF6
```

This describes ϱ as a rotary reflection of order 6. This may be a surprise to some as the rotation has angle 2 Pi/3 of order 3. But

In[•]:=
```
ϱ@*ϱ@*ϱ@*ϱ@*ϱ@*ϱ@{1, 1, .5}
```

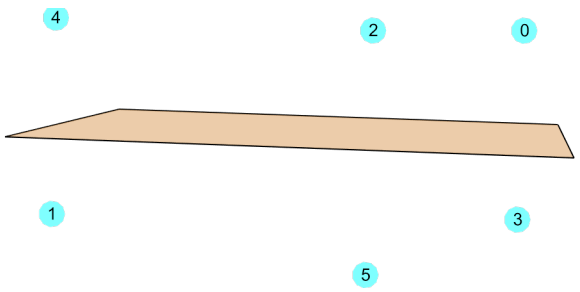
Out[•]=

```
{1., 1., 0.5}
```

while

In[•]:=
```
Graphics3D[{{GrayLevel[8], Opacity[.5],
    Polygon[{{-1.2, 1.2, 0}, {1.2, 1.2, 0}, {1.2, -1.2, 0}, {-1.2, -1.2, 0}}]},
   {Cyan, Opacity[.5], PointSize[.04], Point[{{1, 1, .5}, ϱ@{1, 1, .5}, ϱ@*ϱ@{1, 1, .5},
      ϱ@*ϱ@*ϱ@{1, 1, .5}, ϱ@*ϱ@*ϱ@*ϱ@{1, 1, .5}, ϱ@*ϱ@*ϱ@*ϱ@*ϱ@{1, 1, .5}}]},
   {Black, Inset[0, {1, 1, .5}], Inset[1, ϱ@{1, 1, .5}], Inset[2, ϱ@*ϱ@{1, 1, .5}],
    Inset[3, ϱ@*ϱ@*ϱ@{1, 1, .5}], Inset[4, ϱ@*ϱ@*ϱ@*ϱ@{1, 1, .5}],
    Inset[5, ϱ@*ϱ@*ϱ@*ϱ@*ϱ@{1, 1, .5}]}}, Boxed → False]
```

Out[•]=



ϱ moves each power across the plane so it takes 6 applications of ϱ to get back to the beginning .

---

2. Finite Transformation Groups

These have been discussed in earlier Chapters of this book but for completeness for those just reading this chapter I will summarize this again.

A *finite group* has a finite number of transformations as elements. Since a transformation group contains the identity transformation and is closed under compositions we can define a group by giving a few generators, that is the group is composed of all products and inverses of these generators and products and inverses of these products. It is necessary for finiteness that the generators of a finite group have finite order, that is a composition of a finite number of this transformation gives the identity. The example ϱ above is a good example of a transformation of order 6. In practice we have only rotations of an axis through the origin of order 2,3,4 and 6, that is angles of π, $\pi/2$, $\pi/3$, $\pi/4$.

To get transformation groups from a set of generators, that is, isometries as transformation functions we use a **Mathematica Association** we call a *tas-association.*

If all the generations have finite order this will look like the example for our standard isometries

In[33]:=
```
tasE = <|1 → ρ1, 2 → σ4|>
```

Out[33]=

$$\left\langle\left|\, 1 \to \text{TransformationFunction}\left[\left(\begin{array}{ccc|c} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right)\right],\right.$$

$$\left. 2 \to \text{TransformationFunction}\left[\left(\begin{array}{ccc|c} 0.5 & -0.707107 & 0.5 & -0.5 \\ 0.707107 & 0. & -0.707107 & 0.707107 \\ 0.5 & 0.707107 & 0.5 & 0.5 \\ \hline 0. & 0. & 0. & 1. \end{array}\right)\right] \,\right|\right\rangle$$

The output tends to be long so the definition symbol := will be used in the future to suppress output. Then the following code will produce the finite group.

In[34]:=
```
TasTF[S_, tas_] := Module[{len, t, i}, len = Length[S];
  t = tas[S[[len]]];
  i = 1;
  While[i < len, t = t@*tas[S[[len - i]]]; i++];
  t]
```

In[35]:=
```
Options[nFindKeys] = {tol → 1.*^-5};
nFindKeys[A_, v_, OptionsPattern[]] := Module[{S, R},
  S = Normal[A];
  R = Reap[Do[If[Norm[s[[2]] - v] < OptionValue[tol], Sow[s[[1]]], {s, S}]];
  If[Length[R[[2]]] > 0, Return[R[[2, 1]], Return[{{1}}]]]
```

In[37]:=
```
Options[transGroup] = {tol → .001};
transGroup[tas_, tp_, n_, OptionsPattern[]] := Module[{k, A, An, V, K},
  k = Length[tas];
  A = <| ({#} → tas[#]@tp) & /@ Range[k] |>;
  Do[An = <| (# → TasTF[#, tas]@tp) & /@ Tuples[Range[k], j] |>;
   A = Join[A, An], {j, 2, n}];
  V = DeleteDuplicates[Values[A], Norm[#1 - #2] < OptionValue[tol] &];
  K = Table[First[nFindKeys[A, V[[i]]]], {i, Length[V]}];
  Echo[Length[V], "number of group elements calculated "];
  K]
```

This last function will give a set of ordered tuples of integers from 1 to k where k is the number of generators in the tas association, there will be duplicates in this tuples. These tuples can be read by the first function TasTF as in the example tasE above the tuple {1,2,2}

In[•]:=
```
TasTF[{1, 2, 2}, tasE]
```

Out[•]=

$$\text{TransformationFunction}\left[\left(\begin{array}{ccc|c} 1.11022\times10^{-16} & -2.29935\times10^{-17} & 1. & -1. \\ -2.29935\times10^{-17} & -1. & 2.29935\times10^{-17} & 0. \\ -1. & -2.29935\times10^{-17} & -1.11022\times10^{-16} & 1. \\ \hline 0. & 0. & 0. & 1. \end{array}\right)\right]$$

Note that the output is

In[51]:=
```
σ4@*σ4@*ρ1
```

Out[51]=

$$\text{TransformationFunction}\left[\left(\begin{array}{ccc|c} 1.11022\times10^{-16} & -2.29935\times10^{-17} & 1. & -1. \\ -2.29935\times10^{-17} & -1. & 2.29935\times10^{-17} & 0. \\ -1. & -2.29935\times10^{-17} & -1.11022\times10^{-16} & 1. \\ \hline 0. & 0. & 0. & 1. \end{array}\right)\right]$$

Note that {1,2,2} is read from right to left with the transformations applied in that order where as composition notation goes left to right with the leftmost applied first.

In[•]:=
```
Chop[TasTF[{1, 2, 2}, tasE]]
```

Out[•]=

$$\text{TransformationFunction}\left[\left(\begin{array}{ccc|c} 0. & 0. & 1. & -1. \\ 0. & -1. & 0. & 0. \\ -1. & 0. & 0. & 1. \\ \hline 0. & 0. & 0. & 1. \end{array}\right)\right]$$

In[•]:=
```
Chop[σ4@*σ4@*ρ1]
```

Out[•]=

$$\text{TransformationFunction}\left[\left(\begin{array}{ccc|c} 0. & 0. & 1. & -1. \\ 0. & -1. & 0. & 0. \\ -1. & 0. & 0. & 1. \\ \hline 0. & 0. & 0. & 1. \end{array}\right)\right]$$

so we do get the same thing .

Caution: The the triple {1,2,2} is the order in which the transformations are applied, the reverse of the order written in $\sigma 4@^*\sigma 4@^*\rho 1$.

The second function is just a often used subroutine in the third function which will give some of the group, hopefully all. Our second argument is a random test point, if it is random enough it will work. For the purpose of finding discrete groups later this function has a third argument , n, which limits the length of tuples returned. If the group is finite then all large n will give the full group. Unfortunately unless we already know this group we will not know n until we actually use the algorithm for various n. If two consecutive n give the same output we are done. We may wish to check our work with a different test point.

Our notation below will generally follow the format where we start with a tas-association called, say, tasA, the association of *generators*, then the *finite semigroup of length n* will be called GA[n]. The infinite semi-group generated in this way is simply GA, the union of all GA[n]. This will actually be a group if and only if the semigroup contains the identity at some level and each generator has an inverse in the semigroup at some level. That every two elements has a product is assured by our transGroup algorithm.

Example

In[52]:=
```
tpe = {.2807, 3.421, -1.399};
```

In[39]:=
```
GE[3] = transGroup[tasE, tpe, 3]
```

» number of group elements calculated 11

Out[39]=
```
{{1}, {2}, {1, 1}, {1, 2}, {2, 1}, {2, 2}, {1, 2, 1}, {1, 2, 2}, {2, 1, 2}, {2, 2, 1}, {2, 2, 2}}
```

In[•]:=
```
GE[4] = transGroup[tasE, tpe, 4]
```

» number of group elements calculated 18

Out[•]=
```
{{1}, {2}, {1, 1}, {1, 2}, {2, 1}, {2, 2}, {1, 2, 1}, {1, 2, 2}, {2, 1, 2}, {2, 2, 1}, {2, 2, 2},
 {1, 2, 1, 2}, {1, 2, 2, 1}, {1, 2, 2, 2}, {2, 1, 2, 1}, {2, 1, 2, 2}, {2, 2, 1, 2}, {2, 2, 2, 1}}
```

So in this case n = 3 is enough . This group has 8 elements, it is actually the symmetry group of the colored box in section 1.

If we had been stupid and use test point {0,0,0}

In[•]:=
```
G3stupid = transGroup[tasE, {0, 0, 0}, 3]
```

» number of group elements calculated 4

Out[•]=
```
{{1}, {2}, {2, 1}, {2, 2}}
```

we would not get all the elements .

Groups should have an identity element and inverses . These are automatic here. The identity is {1,1}

In[•]:=
```
TasTF[{1, 1}, tasE]
```

Out[•]=
```
TransformationFunction[( 1 0 0 | 0
                         0 1 0 | 0
                         0 0 1 | 0
                         ------+--
                         0 0 0 | 1 )]
```

and the inverse of $\sigma 4$ is {1,2,1}

In[•]:=
```
Chop[σ4@*TasTF[{1, 2, 1}, tasE]]
```

Out[•]=
```
TransformationFunction[( 1. 0. 0. | 0.
                         0. 1. 0. | 0.
                         0. 0. 1. | 0.
                         ---------+---
                         0. 0. 0. | 1. )]
```

More generally we can find the key giving a specific element of a group .

In[40]:=
```
GroupFindKey[g_, G_, tas_, tp_] := Module[{S, R, gv},
  S = Table[α → TasTF[α, tas]@tp, {α, G}];
  gv = g@tp;
  R = Reap[Do[If[Norm[s[[2]] - gv] < 1.*^-5, Sow[s[[1]]]], {s, S}]];
  If[Length[R[[2]]] > 0, Return[R[[2, 1]][1]], Return[Fail]]]
```

To find the inverse to, say {1,2,1} in G3

In[•]:=
```
g = InverseTF[TasTF[{1, 2, 1}, tasE]]
```

Out[•]=
```
TransformationFunction[(    0.5   -0.707107   -0.5   |    0.5
                         0.707107     0.     0.707107 | -0.707107
                           -0.5    -0.707107    0.5   |    0.5
                         ---------------------------+----------
                            0.        0.        0.    |    1. )]
```

In[•]:=
```
GroupFindKey[g, GE[3], tasE, tpe]
```

Out[•]=
```
Fail
```

Multiplication in the group is just concatenation but you want to reduce this to a key in the list given. For example to multiply {1,2,1} by {1,2}

In[•]:=
```
h = TasTF[{1, 2, 1, 1, 2}, tasE]
```

Out[•]=

$$\text{TransformationFunction}\left[\left(\begin{array}{ccc|c} 1.11022\times10^{-16} & -2.29935\times10^{-17} & 1. & -1. \\ -2.29935\times10^{-17} & -1. & 2.29935\times10^{-17} & 0. \\ -1. & -2.29935\times10^{-17} & -1.11022\times10^{-16} & 1. \\ \hline 0. & 0. & 0. & 1. \end{array}\right)\right]$$

In[•]:=
```
GroupFindKey[h, GE[3], tasE, tpe]
```

Out[•]=
```
{1, 2, 2}
```

Check

In[•]:=
```
TasTF[{1, 2, 2}, tasE]
```

Out[•]=

$$\text{TransformationFunction}\left[\left(\begin{array}{ccc|c} 1.11022\times10^{-16} & -2.29935\times10^{-17} & 1. & -1. \\ -2.29935\times10^{-17} & -1. & 2.29935\times10^{-17} & 0. \\ -1. & -2.29935\times10^{-17} & -1.11022\times10^{-16} & 1. \\ \hline 0. & 0. & 0. & 1. \end{array}\right)\right]$$

We can find out what isometries we have in a group .

In[41]:=
```
classAssoc3D[G_, tas_] := <|Table[g → classTF3D[TasTF[g, tas]], {g, G}]|>
```

In[•]:=
```
classAssoc3D[GE[4], tasE]
```

Out[•]=
```
<|{1} → RF, {2} → R4, {1, 1} → ι, {1, 2} → RRF6, {2, 1} → RRF6,
 {2, 2} → H, {1, 2, 1} → R4, {1, 2, 2} → RRF4, {2, 1, 2} → RRF4, {2, 2, 1} → RRF4,
 {2, 2, 2} → R4, {1, 2, 1, 2} → R3, {1, 2, 2, 1} → H, {1, 2, 2, 2} → RRF6,
 {2, 1, 2, 1} → R3, {2, 1, 2, 2} → RRF6, {2, 2, 1, 2} → RRF6, {2, 2, 2, 1} → RRF6|>
```

As expected we have rotations, reflections and a half turn . Note the identity is classed as a translation. Finally we have a function **classCountG** with hidden code below. For example

In[•]:=
```
classCountG[GE[4], tasE]
```

```
∟  1
RF  1
H  2
RRF4  3
RRF6  6
R3  2
R4  3
```

3. DiscreteTransformation Groups

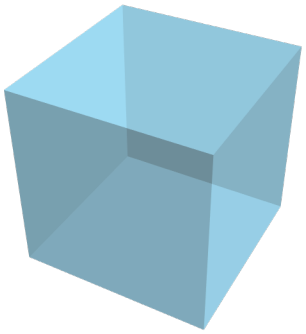
As mentioned above while classical crystallography likes to work with finite groups we prefer working with the discrete group of a lattice . We consider a lattice to be a geometric configuration in $\mathbb{R}^3$ with a symmetry group containing 3 linearly independent translations. Since translations do not have a finite order they do not live in finite groups, so we are looking at a larger infinite group of isometries. *A group G of isometries is discrete if given any point p of $\mathbb{R}^3$ and any bounded set S in $\mathbb{R}^3$ only finitely many points in the G-orbit of p lie in S.*

There are several other criteria in Yale's book however one easy sufficient one is that the group is a symmetry group of a lattice. Often the orbit of a non-random point, such as {0,0,0} will suggest the lattice.

Our best choice of a bounded set will be a solid cube. The Mathematica data type Region with be useful here.

In[43]:=
```
Rcube = Region[Style[Cube[2], Opacity[.5]]]
```

Out[43]=



Here a point {x,y,z} will be in the cube if $-1 \le x \le 1,\ -1 \le y \le 1,\ -1 \le z \le 1$. We can easily test this by

In[54]:=
```
RcubeQ = RegionMember[Rcube, #] &
```

Out[54]=
```
RegionMember[Rcube, #1] &
```

```
In[•]:= RcubeQ[{.9, -.8, .7}]
        RcubeQ[{.2, 1.2, .3}]
Out[•]=
        True
Out[•]=
        False
```

If we use as translations those corresponding to the edges of the cube then ultimately each cube will have the same configuration of G-orbit points so looking at one will be sufficient. Yale also gives some alternate criteria for discreteness, the G-orbit points in a cube should be separated or there should be no clusters.

We can use this with our finite group algorithm. It will run but not terminate if we use isometries such as translations, glide reflections or screw displacements which are not of finite order.

Our first example will be the group of translations of our DBboxes, the generating association will be

```
In[45]:= tasTDB := <|1 → τb[{1, 0, 0}], 2 → τb[{0, 1, 0}], 3 → τb[{0, 0, 1}], 4 → ηx, 5 → ηy, 6 → ηz|>
```

The half turn will give us the identity and inverses. Use **tpe** above, then

```
In[•]:= GTDB[3] = transGroup[tasTDB, tpe, 3];
```

» number of group elements calculated 83

Let our point be the somewhat random point

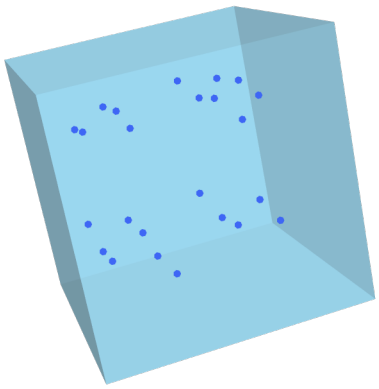
```
In[61]:= p := {-.33, .21, -.47}
```

```
In[46]:= orbitGp[G_, tas_, p_] := Prepend[Table[TasTF[k, tas]@p, {k, G}], p]
```

```
In[•]:= orbp = orbitGp[GTDB[3], tasDB, p];
        Length[orbp]
        Length[Select[orbp, RcubeQ]]
Out[•]=
        84
Out[•]=
        25
```

```
In[•]:= Show[Rcube, Graphics3D[{Blue, PointSize[.02], Point[Select[orbGB3p, RcubeQ]]}]]
```

Out[•]=



We now increase **n** in our finite group function and only give the counts

```
In[53]:= GTDB[4] = transGroup[tasTDB, tpe, 4];
```

» number of group elements calculated 200

```
In[62]:= orb4p = orbitGp[GTDB[4], tasTDB, p];
         Length[orb4p]
         Length[Select[orb4p, RcubeQ]]
```

Out[63]=

201

Out[64]=

31

```
In[66]:= GTDB[5] = transGroup[tasTDB, tpe, 5];
```

» number of group elements calculated 410

```
In[67]:= orb5p = orbitGp[GTDB[5], tasTDB, p];
         Length[orb5p]
         Length[Select[orb5p, RcubeQ]]
```

Out[68]=

411

Out[69]=

33

So it appears the number of points in our RCube is stabilized at near 32 even though the orbit size is increasing rapidly. Since we have three independent translations in our group it follows easily that this cube is the same as any other, so we are fairly confident that the entire group is discrete.

Here is another example of a non-discrete group. The half turns are added to get a group.

```
In[70]:= ξ2 := screwDisplacementTF3D[Pi / 3, {{0, 0, 0}, {0, 2, 3}}]
```

```
In[71]:= tasTDB2 := ⟨|1 → τb[{1, 0, 0}], 2 → τb[{0, 1, 0}],
           3 → τb[{0, 0, 1}], 4 → ξ2, 5 → ηx, 6 → ηy, 7 → ηz|⟩
```

running the above with this example

In[78]:=
```
GTDB2[3] = transGroup[tasTDB2, tpe, 3];
```

» number of group elements calculated 193

So we have a group because the generators all have inverses.

In[83]:=
```
orbB3p = orbitGp[GTDB2[3], tasTDB2, p];
Length[orbB3p]
Length[Select[orbB3p, RcubeQ]]
```

Out[84]=

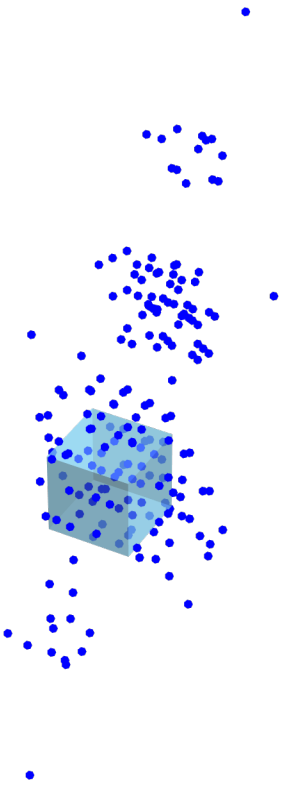
194

Out[85]=

25

In[86]:=
```
Show[Rcube, Graphics3D[{PointSize[.02], Blue, Point[orbB3p]}]]
```

Out[86]=



This is much less organized . Increasing **n** we have

```
GTDB2[4] = transGroup[tasTDB2, tpe, 4];
```

» number of group elements calculated 853

Out[•]=

853

```
orbB4p = orbitGp[GTB2[4], tasTDB2, p];
Length[orbB4p]
Length[Select[orbB4p, RcubeQ]]
```

Out[•]=

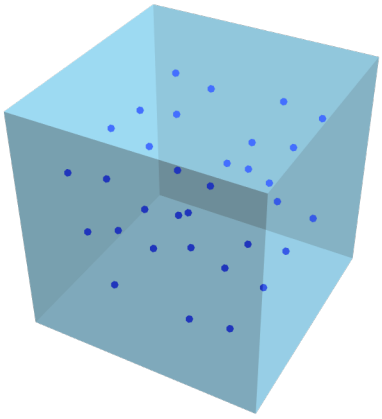
853

Out[•]=

33

```
Show[Rcube, Graphics3D[{Blue, PointSize[.02], Point[Select[orbB4p, RcubeQ]]}]]
```

Out[•]=



(due to time constraints the following 3 cells are not evaluatable .)

```
GTDB2[5] = transGroup[GTBA2, tpe, 5];
```

» number of group elements calculated 3719

Out[•]=

3719

```
orb5p = orbitGp[GTB2[5], tasTDB2, p];
Length[orb5p]
Length[Select[orb5p, RcubeQ]]
```

Out[•]=

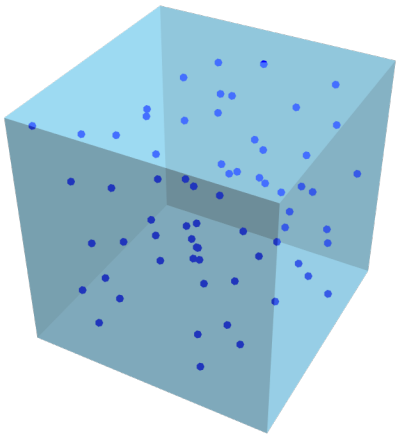
3719

Out[•]=

67

```
Show[Rcube, Graphics3D[{Blue, PointSize[.02], Point[Select[orb5p, RcubeQ]]}]]
```

Out[•]=



To go farther I need a faster computer, but we see this number is nowhere near stabilizing and points are begining to cluster So for our purposes, at least, this is not a discrete group.

I mention again that if a group of isometries can be seen as a group of isometries of a polygon, then it will be discrete .

The main part of this chapter is a discussion of the 32 discrete isometry groups, based on the groups defined in Yale's book *Geometry and Symmetry* pages 107-108. It is not necessary that you have this book as these groups will be defined. They will each have a notebook in the directory crystallographicDiscreteGroups. See "indexCDG.html" on my website.