GS5. Dynamic Even Isometric Transformation Functions for $\mathbb{R}^3$

5.1 Introduction

In the study of motion physicists and mathematicians often show a curve that an object, real or abstract, follows. But for an actual object, besides changing position in space it often changes its orientation as it moves. We see this in such things as astronomy, the moon is rotating on its axis as it circles spinning earth, in travel, an airplane changes direction on the ground and in the air, and in sports, many sports with a ball try to spin the ball to get it to curve, or athletes change orientation as they spin or swim or dive.

Taking this into account a point is not descriptive of the movement of the actual object . If we think dynamically in **Mathematica**, a *Transformation Function* gives both location and orientation an is a better representation. In this chapter I will look at this both analytically and geometrically in Euclidean three space. As I am thinking of actual objects which in practice cannot be reflected I use even isometries which we have seen can be nicely described by Mathematica's TransformationFunction construct. For those readers who are not familiar see my Geometry and Symmetry Chapters 1,2 on my website (PDF) or *[BOOK] Transformation functions and symmetry* posted on my *Wolfram Community* page (.nb).

Analytically they can be represented by a 4×4 matrix. For example

Out[•]=

$$\text{TransformationFunction}\left[\left(\begin{array}{ccc|c} 0.840847 & -0.303623 & -0.448094 & 2. \\ 0.176301 & 0.936339 & -0.303623 & 3. \\ 0.511755 & 0.176301 & 0.840847 & 4. \\ \hline 0. & 0. & 0. & 1. \end{array}\right)\right]$$

For an even isometry the upper left 3×3 part outlined will be an orthogonal 3×3 matrix. This I call the orthogonal part. The column vector in the first 3 rows is the translation part. The last row is always {0,0,0,1} for a Euclidean transformation. Mathematica gives many functions to create such transformation matrices such as

In[•]:=
```
σ = RotationTransform[Pi / 5., {1, -2, 1}]
```

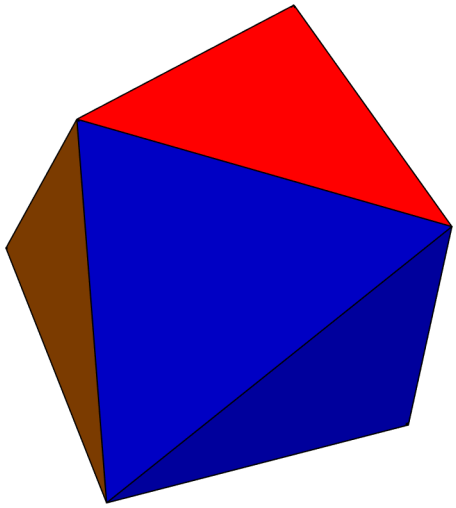
Out[•]=

$$\text{TransformationFunction}\left[\left(\begin{array}{ccc|c} 0.840847 & -0.303623 & -0.448094 & 0. \\ 0.176301 & 0.936339 & -0.303623 & 0. \\ 0.511755 & 0.176301 & 0.840847 & 0. \\ \hline 0. & 0. & 0. & 1. \end{array}\right)\right]$$

In the next section I will introduce a *dynamic form* which will be more useful for moving these TransformationFunctions around. In section 5.3 I will give a geometric abstract model, or as computer games people call it an *avatar*. I call mine a *spindle.*

In[•]:=
```
Show[Spindle[σ], Boxed → False]
```

Out[•]=



Each location (position and orientation) in space will have its own spindle.

Using these two ideas I will show how to represent motion of an object in space .

5.2 Analytic representation.

In this Chapter I will be using the Mathematica functions

In[•]:=
```
TranslationTransform[{1, 2, 4}]
```

Out[•]=

$$\text{TransformationFunction}\left[\left(\begin{array}{ccc|c} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 4 \\ \hline 0 & 0 & 0 & 1 \end{array}\right)\right]$$

In[•]:=
```
RotationTransform[Pi / 5., {0, 0, 1}]
```

Out[•]=

$$\text{TransformationFunction}\left[\left(\begin{array}{ccc|c} 0.809017 & -0.587785 & 0. & 0. \\ 0.587785 & 0.809017 & 0. & 0. \\ 0. & 0. & 1. & 0. \\ \hline 0. & 0. & 0. & 1. \end{array}\right)\right]$$

In[•]:=
```
ω = RotationTransform[{{0, 0, 1.}, {1, -2, 1}}]
```

Out[•]=

$$\text{TransformationFunction}\left[\left(\begin{array}{ccc|c} 0.88165 & 0.236701 & 0.408248 & 0. \\ 0.236701 & 0.526599 & -0.816497 & 0. \\ -0.408248 & 0.816497 & 0.408248 & 0. \\ \hline 0. & 0. & 0. & 1. \end{array}\right)\right]$$

In fact, only the first two of these are needed but the third is useful in constructing isometric transformation functions. We may want inverses, one should use my InverseTF function to get inverses of transformation functions

```
In[•]:= InverseTF[ω]
```

```
Out[•]=
TransformationFunction[( 0.88165   0.236701  -0.408248 | 0. )
                       ( 0.236701  0.526599   0.816497 | 0. )
                       ( 0.408248 -0.816497   0.408248 | 0. )
                       (---------------------------------|---)
                       (   0.        0.         0.     | 1. )]
```

Since ω was an orthogonal transformation it is only necessary to take the transpose of the orthogonal part. Incidentally I have a function to extract the orthogonal part of one of our isometric TransformationFunctions

```
In[71]:= orthPart[α_] := Take[TransformationMatrix[α], 3, 3]
```

```
In[•]:= orthPart[ω]
```

```
Out[•]=
{{0.88165, 0.236701, 0.408248},
 {0.236701, 0.526599, -0.816497}, {-0.408248, 0.816497, 0.408248}}
```

A TransformationFunction will be called orthogonal if the orthPart is orthogonal and the translation part is {0,0,0}. The following bit of code takes a linear transformation to a transformation function so every orthogonal matrix gives rise to an orthogonal TransformationFunction.

```
In[72]:= LT2TF[M_] := TransformationFunction[Append[Join[M, {{0}, {0}, {0}}, 2], {0, 0, 0, 1}]]
```

Note with this any isometric TransformationFunction can be written as the composition of the orthogonal part with the transformation part

For example our first example above

```
In[•]:= α = TransformationFunction[
          ( 0.8408474953124563`  -0.30362332628329014`  -0.4480941478790364` | 2.` )
          ( 0.17630132253325503`  0.9363389981249824`   -0.3036233262832902` | 3.` )
          ( 0.5117551497540539`   0.17630132253325503`   0.8408474953124563` | 4.` )
          (-------------------------------------------------------------------|----)
          (          0.`                    0.`                  0.`         | 1.` )]
```

```
Out[•]=
TransformationFunction[( 0.840847 -0.303623 -0.448094 | 2. )
                       ( 0.176301  0.936339 -0.303623 | 3. )
                       ( 0.511755  0.176301  0.840847 | 4. )
                       (-----------------------------|---)
                       (   0.        0.        0.    | 1. )]
```

```
In[•]:= τ = TranslationTransform[α@{0, 0, 0}]
```

```
Out[•]=
TransformationFunction[( 1. 0. 0. | 2. )
                       ( 0. 1. 0. | 3. )
                       ( 0. 0. 1. | 4. )
                       (----------|---)
                       ( 0. 0. 0. | 1. )]
```

In[•]:=
```
ϑ = orthPart[α]
```

Out[•]=
```
{{0.840847, -0.303623, -0.448094},
 {0.176301, 0.936339, -0.303623}, {0.511755, 0.176301, 0.840847}}
```

In[•]:=
```
τ@*LT2TF[ϑ]
```

Out[•]=
$$\text{TransformationFunction}\left[\left(\begin{array}{ccc|c} 0.840847 & -0.303623 & -0.448094 & 2. \\ 0.176301 & 0.936339 & -0.303623 & 3. \\ 0.511755 & 0.176301 & 0.840847 & 4. \\ \hline 0. & 0. & 0. & 1. \end{array}\right)\right]$$

Again we are working with only even orthogonal matrices, that is orthogonal matrices with determinant 1, I have a routine to show that the matrix can be expressed in the form

`RotationMatrix[`θ`, axis]`

where θ is an angle and axis is a point on the axis of rotation, here it is given by a single non-zero 3-vector defining the axis a a line through the origin.

In[73]:=
```
EvenOrth2Rotation[U1_] := Module[{U, eigs, v1, v2, b, c, M, R, w},
  U = Orthogonalize[U1 + RandomReal[{-1*^-5, 1*^-5}, {3, 3}]];
  If[ Det[U] ≠ 1, Echo["Not even, use OddOrth2RR"]; Abort[]];
  eigs = Eigensystem[U];
  If[eigs[[1, 1]] == 1., v1 = Chop[eigs[[2, 1]],
   If[eigs[[1, 2]] == 1., v1 = Chop[eigs[[2, 2]],
    If[eigs[[1, 3]] == 1., v1 = Chop[eigs[[2, 3]], Return["Fail"]]]]];
  b = Normalize[RandomReal[{-1, 1}, 3]];
  v2 = Normalize[b - (v1.b) v1];
  w = U.v2;
  c = ArcCos[v2.w];
  R = Chop[RotationMatrix[c, v1]];
  If[Norm[R - U] > .01, c = -c];
  R = Chop[RotationMatrix[c, v1]];
  If[Norm[R - U] < .01, {c, v1}, Fail]]
```

Note that this is numerical and gives an approximate answer. The reason is that there are singular examples for which an exact versions of this does not work. There is a small perturbation built in which almost always (probability=1) works so we can have an answer.

Consider the orthogonal matrix above

In[•]:=
```
M = orthPart[ω]
```

Out[•]=
```
{{0.88165, 0.236701, 0.408248},
 {0.236701, 0.526599, -0.816497}, {-0.408248, 0.816497, 0.408248}}
```

In[•]:=
```
EvenOrth2Rotation[M]
```

Out[•]=
```
{1.15027, {0.894427, 0.447214, 0.0000129983}}
```

```
In[•]:= M1 = RotationTransform[1.1502635291612031`,
          {0.8944246040914284`, 0.44721876919763043`, 8.558504669629488`*^-6}]
```

Out[•]=

$$\text{TransformationFunction}\left[\left(\begin{array}{ccc|c} 0.881647 & 0.236695 & 0.408258 & 0. \\ 0.236711 & 0.5266 & -0.816493 & 0. \\ -0.408249 & 0.816497 & 0.408247 & 0. \\ \hline 0. & 0. & 0. & 1. \end{array}\right)\right]$$

Note that M is not exactly the orthPart of M1. But it is close enough for our plotting usage.

Putting this together with our earlier comment that every isometry of 3 space is given by a composition of a translation with a rotation we can write every isometry in the dynamic form {f,g,h} where f is a point in 3-space, g is a real number and h is a non-zero triple defining a axis through the origin. We have a function sending this point to a Transformation Function

```
In[74]:= DITF[{f_, g_, h_}] := TranslationTransform[f]@*RotationTransform[g, h]
```

There is an right inverse

```
In[75]:= TF2DITF[α_] := Module[{p, r, a, M, β},
           p = α@{0, 0, 0};
           β = TranslationTransform[-p]@*α;
           M = orthPart[β];
           If[Norm[M.Transpose[M] - IdentityMatrix[3]] > .001 || Abs[Det[M] - 1] > .001,
            Echo["Not even isometry"];
            Abort[]];
           {r, a} = EvenOrth2Rotation[M];
           Chop[{p, r, a}, 1.*^-5]]
```

For example consider α above

```
In[•]:= α
```

Out[•]=

$$\text{TransformationFunction}\left[\left(\begin{array}{ccc|c} 0.840847 & -0.303623 & -0.448094 & 2. \\ 0.176301 & 0.936339 & -0.303623 & 3. \\ 0.511755 & 0.176301 & 0.840847 & 4. \\ \hline 0. & 0. & 0. & 1. \end{array}\right)\right]$$

```
In[•]:= DITF[TF2DITF[α]]
```

Out[•]=

$$\text{TransformationFunction}\left[\left(\begin{array}{ccc|c} 0.840852 & -0.303622 & -0.448086 & 2. \\ 0.176303 & 0.936339 & -0.303622 & 3. \\ 0.511747 & 0.176303 & 0.840852 & 4. \\ \hline 0. & 0. & 0. & 1. \end{array}\right)\right]$$

This numerically very close . On the other hand

```
In[•]:= d1 = {{2, 1, -3}, 4.27, {1, 1, 2}};

In[•]:= TF2DITF[DITF[d1]]
```

Out[•]=

```
{{2., 1., -3.}, -2.01318, {0.40825, 0.408243, 0.816498}}
```

The point {2, 1, -3} is the same and the axis is just the normalization of the axis

In[•]:=
```
Normalize[{1., 1, 2}]
```

Out[•]=
```
{0.408248, 0.408248, 0.816497}
```

But the rotations are different, note the difference is a multiple of 2 Pi

In[•]:=
```
4.27 - (-2.01319)
```

Out[•]=
```
6.28319
```

In fact we have relations

```
DITF[{f, g, h}] = DITF[{f, -g, -h}]
DITF[{f, g, h}] = DITF[f, g + 2 k Pi, h] for k an integer.
```

The second one is good for stationary DITF only, that is where {f,g,h} are constants. For moving DITF later we note that adding a multiple of 2k Pi makes DITF spin faster about its axis.

DITF gives not only a form with fewer variables but allows us to compare two isometries .

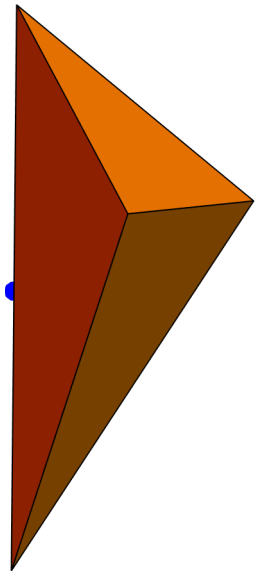
5.3 Geometric representation

For representing motion with moving orientation we use our spindle representation as a generic avatar. Specifically a slice is given by the tetrahedral Polyhedron inscribed in a sphere with the 4 sides

In[76]:=
```
S1 = {{{0.9486832980505138`, 0.`, 0.3162277660168379`},
    {0.`, 0.9486832980505138`, 0.3162277660168379`}, {0.`, 0.`, 1.`}},
   {{0.9486832980505138`, 0.`, 0.3162277660168379`},
    {0.`, 0.9486832980505138`, 0.3162277660168379`}, {0.`, 0.`, -1.`}},
   {{0.9486832980505138`, 0.`, 0.3162277660168379`},
    {0.`, 0.9486832980505138`, 0.3162277660168379`}, {0.`, 0.`, 1.`}},
   {{0.`, 0.`, 1.`}, {0.`, 0.`, -1.`}, {0.9486832980505138`, 0.`, 0.3162277660168379`}},
   {{0.`, 0.9486832980505138`, 0.3162277660168379`}, {0.`, 0.`, 1.`}, {0.`, 0.`, -1.`}}};
```

In[•]:=
```
Graphics3D[
 {{Orange, Polyhedron[S1]}, {Blue, PointSize[.05], Point[{0, 0, 0}]}}, Boxed → False]
```

Out[•]=



The long side has vertices {0, 0, 1} and {0, 0, -1}, this is our axis. The vertices are all on the unit sphere.

In[•]:=
```
σ4 = RotationTransform[Pi / 2, {0, 0, 1}]
```

Out[•]=

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

Now for every isometric TransformationFunction θ we get a full spindle

In[77]:=
```
Spindle[θ_] :=
 Graphics3D[{Opacity [1], {Red, Polyhedron[θ@S1]}, {Green, Polyhedron[θ@*σ4@S1]},
   {Yellow, Polyhedron[θ@*σ4@*σ4@S1]}, {Blue, Polyhedron[θ@*σ4@*σ4@*σ4@S1]}}]
```

Our base spindle is given by the identity isometry

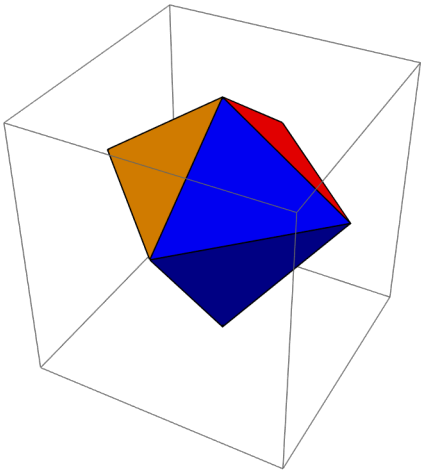
In[•]:=
```
ι = TransformationFunction[IdentityMatrix[4]]
```

Out[•]=

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

In[•]:=
```
Spindle[ι]
```

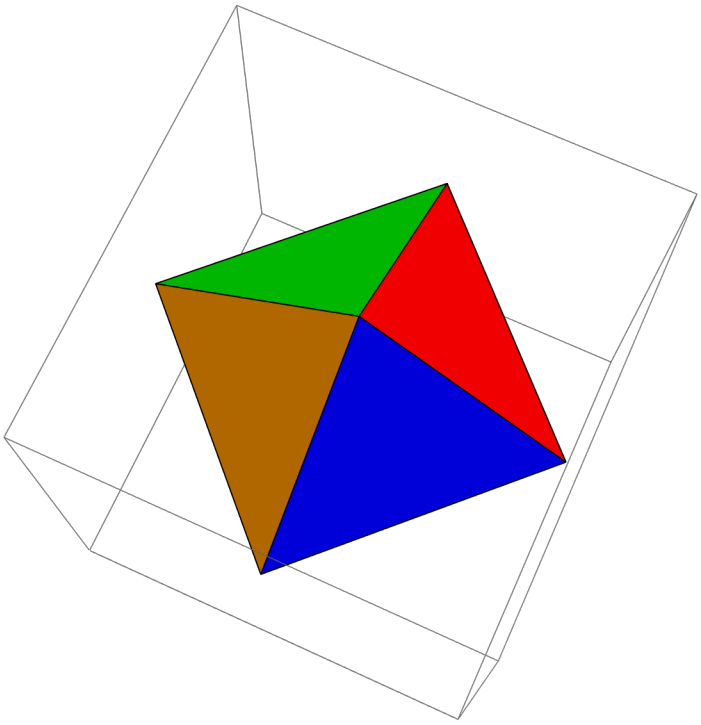
Out[•]=



If we look at this from the top

In[•]:=
```
Spindle[ι]
```

Out[•]=

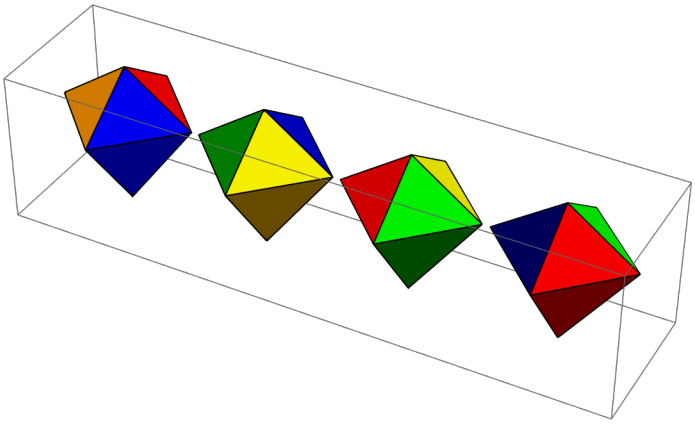


this is the union of 4 slices, going clockwise red, blue, yellow and green. Using a translation to keep them apart

In[•]:=
```
τ12 := TranslationTransform[{2, 0, 0}]
```

In[•]:=
```
Show[Spindle[ι], Spindle[τ12@*σ4],
 Spindle[τ12@*τ12@*σ4@*σ4], Spindle[τ12@*τ12@*τ12@*σ4@*σ4@*σ4]]
```

Out[•]=



We can change the orientation, suppose

In[•]:=
```
η = RotationTransform[Pi, {1, 1, 0}]
```

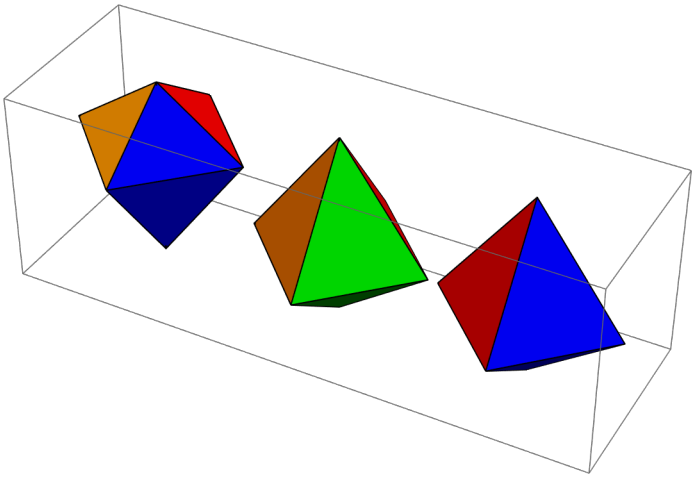
Out[•]=

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

Some other examples

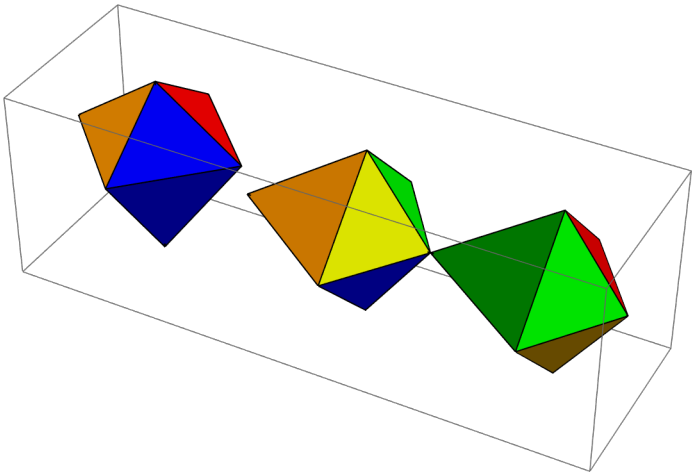
In[•]:=
```
Show[Spindle[ι], Spindle[τ12@*η], Spindle[τ12@*τ12@*η@*σ4@*σ4]]
```

Out[•]=



In[•]:=
```
κ := RotationTransform[{{0, 0, 1}, {1, 0, 0}}]
Show[Spindle[ι], Spindle[τ12@*κ], Spindle[τ12@*τ12@*κ@*σ4]]
```

Out[•]=



In chapter 2 we saw that 4 general position points determined a transformation function . Thus the 4 points on the spindle part determine a spindle in space, so any spindle in space which is isometric to our base is given by a isometric Transformation Function.

5.4 Simple examples

If the path is known parametrically this is fairly simple. The first example works directly from the transform matrix. Here we have a football thrown in a spiral pass. As all our examples this is not to scale, our football spindle is much too large for the field, but that is so you can see it well. We first define a field, .

In[78]:=
```
football = {{Green, EdgeForm[{Thickness[.05], LightYellow}],
     Polygon[{{-5, -12, 0}, {-5, 12, 0}, {5, 12, 0}, {5, -12, 0}}]},
    {LightYellow, Thickness[.01], Line[Table[{{-4.9, 2 i, 0}, {4.9, 2 i, 0}}, {i, -5, 5}]]},
    {Gray, Thickness[.01], Line[{{-1.5, -11.5, 0}, {-1.5, -11.5, 4}}],
     Line[{{1.5, -11.5, 0}, {1.5, -11.5, 4}}], Line[{{-1.5, -11.5, 2}, {1.5, -11.5, 2}}],
     Line[{{-1.5, 11.5, 0}, {-1.5, 11.5, 4}}], Line[{{1.5, 11.5, 0}, {1.5, 11.5, 4}}],
     Line[{{-1.5, 11.5, 2.}, {1.5, 11.5, 2.}}]}};
```

Our dynamic transformation function is

In[79]:=
```
Ω[s_] = TranslationTransform[{0, s, 2.5}]@*
  RotationTransform[{{0, 0, 1}, {0, 1, 0}}]@*RotationTransform[5 s, {0, 0, 1}]
```

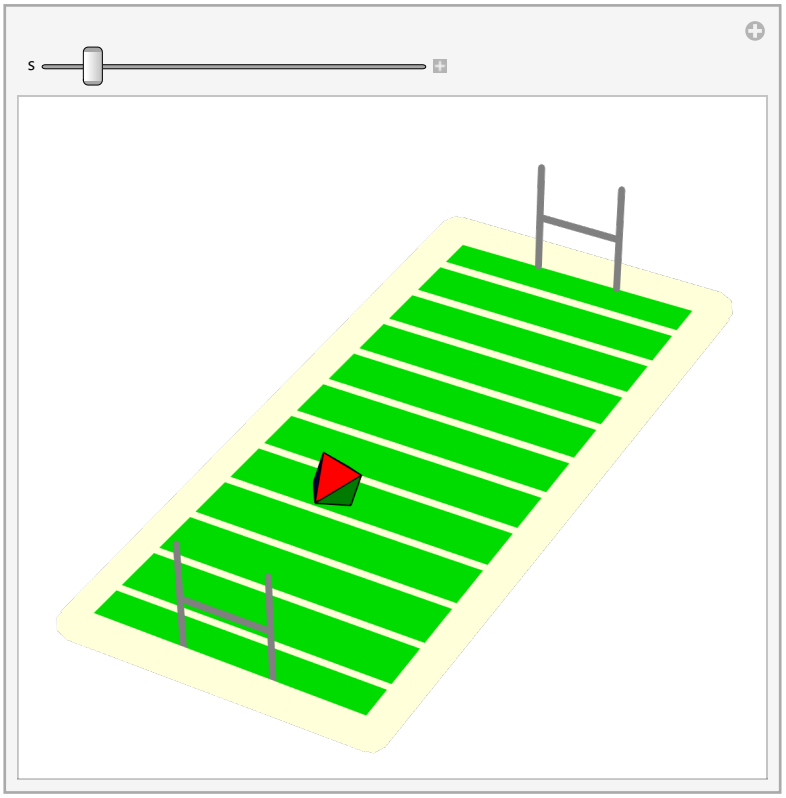
Out[79]=

$$\text{TransformationFunction}\left[\left(\begin{array}{ccc|c} \text{Cos}[5\,s] & -\text{Sin}[5\,s] & 0 & 0 \\ 0 & 0 & 1 & s \\ -1.\,\text{Sin}[5\,s] & -1.\,\text{Cos}[5\,s] & 0 & 2.5 \\ \hline 0 & 0 & 0 & 1 \end{array}\right)\right]$$

Then our pass is given by moving the control

In[•]:=
```
Manipulate[Show[Graphics3D[football], Spindle[Ω[s]], Boxed → False], {s, -7, 10}]
```

Out[•]=

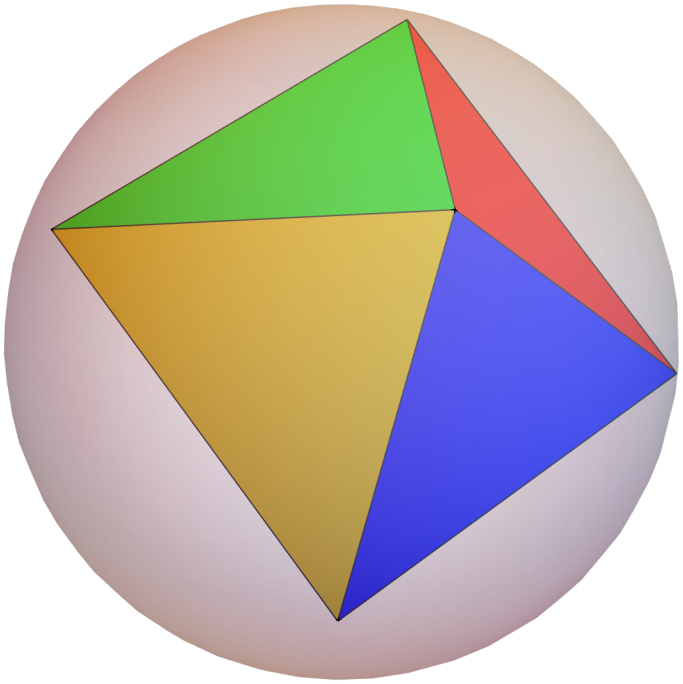


Our next uses our dynamic notation It shows the moon orbiting the earth. For spherical shapes, such as balls or planets we add a sphere of low opacity around our spindle. By default the sphere is of size 1 but it can be increased with an option. The spindle stays the same.

In[80]:=
```
Options[SpindleBall] := {radius → 1, opacity → .4}
SpindleBall[θ_, OptionsPattern[]] := Show[Graphics3D[{Opacity[OptionValue[opacity]],
    Sphere[θ@{0, 0, 0}, OptionValue[radius]]}], Spindle[θ], Boxed → False]
```

In[•]:=
```
SpindleBall[TranslationTransform[{2, 3, 4}]@*RotationTransform[Pi / 4, {1, 1, 1}]]
```

Out[•]=



Again not to scale, the moon position is given by

In[82]:=
```
M[θ_] := DITF[{{10 Cos[θ], 9 Sin[θ], 1}, θ - Pi / 4, {0, 0, 1}}]
```

and earth is

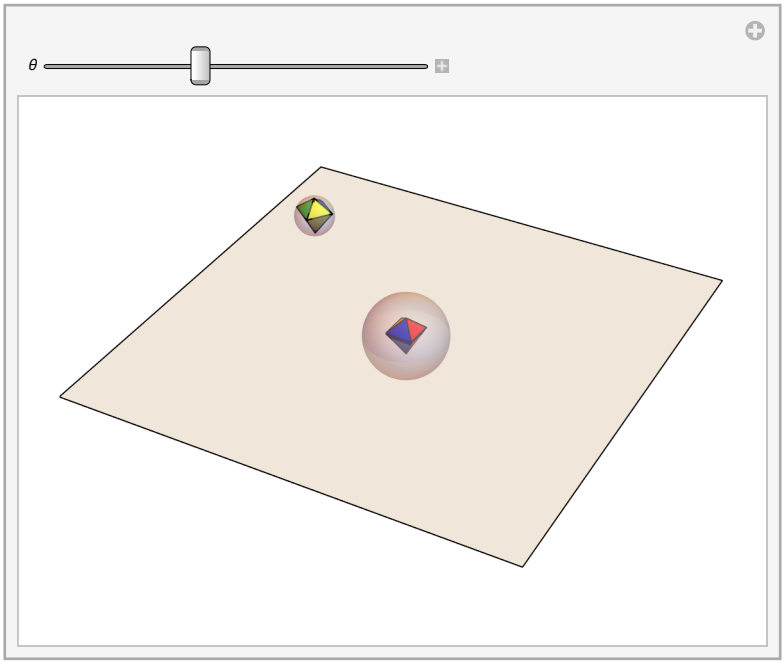
In[83]:=
```
E[θ_] := DITF[{{0, 0, 0}, -28 * θ, {0, 0, 1}}]
```

To pin the earth down,

In[84]:=
```
earthMoon = {{GrayLevel[.8], Opacity[.3],
    Polygon[{{-11, -11, 0}, {-11, 11, 0}, {11, 11, 0}, {11, -11, 0}}]}};
```

In[•]:=
```
Manipulate[Show[Graphics3D[earthMoon], SpindleBall[M[θ]],
  SpindleBall[E[θ], radius → 2], Boxed → False], {θ, 0, 2 Pi}]
```

Out[•]=



If you orient this graphic on its side you can see the orbit of the moon is not in the plane of the equator of earth. Note that the moon always shows its yellow side to the earth and counting you may see the earth spinning on its axis 28 times for one rotation of the moon, this is best seen if you move the control manually.

5.5 More Complicated examples

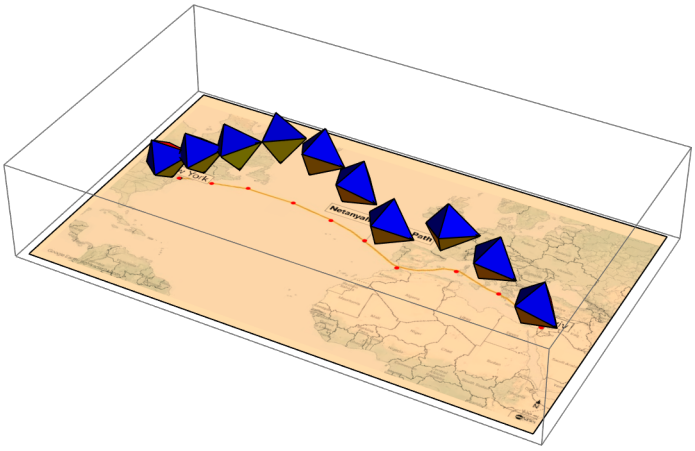
In September 2025 Prime Minister Netanyahu of Israel flew from Tel Aviv to New York for a session of the United Nations. For political reasons he did not want to cross the air space of any European or Arabic nation. So he took a circuitous path which was posted by someone on the internet.

In[97]:=
```
NetFlight :=
```

While this picture shows his 2 dimensional route, it does not show the airplane's altitude or orientation. We can imagine this with a dynamic spindle as an avatar for the airplane. By interpolating the path adding an altitude and orientation. This will not be to scale, in particular the altitude is measured in different units than ground distance for display purposes.

First I interpolated the flight path into 10 parts, and by hand estimated the direction and altitude at each boundary point. I then calculated a dynamic transformation function at each of these points.

In[ ]:=



I made a table of the dynamic DITF value for each of these transformations, the first entry being a parameter value. I used 2 dimensional coordinates from the image.

In[85]:=

```
nData = {{-5, {18.`, 4.8`, 1.`}, 1.5707936296377019`, {-0.999999999971193`, 0, 0}},
   {0, {18, 4.8`, 1}, 1.362142837669556`,
    {-0.3835388080258974`, -0.9214582596014546`, -0.06174672906585432`}},
   {2.37`, {15.9`, 5.8`, 1.5`}, 1.3332301964122943`, {-0.305821062133473`,
     -0.9520890073704005`, 0}}, {4.39`, {14.`, 6.25`, 2}, 1.5317308281102573`,
    {0.01060629203202947`, -0.977474298692654`, 0.21078781265679356`}},
   {6.56`, {12, 5.4`, 2}, 1.4075745860668583`, {-0.2659511176961344`,
     -0.956711241686803`, 0.11820999545839028`}}, {8.64`, {10.2`, 6.3`, 2.5`},
    1.3061014831542193`, {-0.3426134579179421`, -0.9394764597654964`, 0}},
   {10.48`, {8.5`, 6.8`, 3}, 1.4905265579994675`,
    {-0.29779452583917526`, -0.9484177428814601`, 0.10872996536308312`}},
   {12.31`, {6.7`, 7.1`, 3}, 1.9198336373613176`,
    {-0.1986690262228601`, -0.974336077255705`, 0.1058292330957085`}},
   {14.54`, {4.7`, 7.1`, 2}, 1.9806605979000196`,
    {-0.008776987685273005`, -0.9932154274362622`, 0.11595723000216723`}},
   {16, {3.3`, 6.7`, 1.5`}, 1.926797498606207`,
    {0.14394533252309674`, -0.9871253089112266`, 0.06973783586928153`}},
   {17.5`, {2, 6.4`, 1}, 1.931093694715417`,
    {0.17577378252923156`, -0.9838215620278427`, 0.03462241274755564`}}};
```

The set of parameter values is

In[86]:=
```
nt = {-5, 0, 2.3790754506740637`, 4.394639887748701`, 6.567771268958774`,
  8.641415404291546`, 10.482610668243742`, 12.307439427333208`,
  14.543507404832997`, 16.082987836667062`, 17.507768521544563`}
```

Out[86]=
```
{-5, 0, 2.37908, 4.39464, 6.56777, 8.64142, 10.4826, 12.3074, 14.5435, 16.083, 17.5078}
```

I have a helper function that takes such a set of increasing parameters and a set of values at each point which could be 1,2 or 3 , or higher, dimensional and returns a piecewise linear parameterized function.

In[87]:=
```
PLP[tt_, V_] := Module[{m, tab},
  m = Length[tt];
  If[Length[V] ≠ m, Echo["Length problem"]; Abort[]];
  tab = Table[{(V⟦i + 1⟧ * tt⟦i⟧ - V⟦i⟧ * tt⟦i + 1⟧ + (V⟦i⟧ - V⟦i + 1⟧) s) / (tt⟦i⟧ - tt⟦i + 1⟧),
     tt⟦i⟧ ≤ s < tt⟦i + 1⟧}, {i, m - 1}];
  Piecewise[tab, V⟦m⟧]]
```

I then obtain piecewise linear parameterized function for f, g, h , position, rotation ,axis of my DIT-F[{f,g,h}] coordinates for each transformation function. For example

In[88]:=
```
fnet = PLP[nt, Flatten[Take[nData, All, {2}], 1]]
```

Out[88]=
```
{18., 4.8, 1.}                                                          -5 ≤ s < 0
{-0.420331 (-42.8234 + 2.1 s),                                          0 ≤ s < 2.37908
 -0.420331 (-11.4196 - 1. s), -0.420331 (-2.37908 - 0.5 s)}
{-0.496139 (-36.5677 + 1.9 s),                                          2.37908 ≤ s < 4.39464
 -0.496139 (-10.6197 - 0.45 s), -0.496139 (-1.83381 - 0.5 s)}
{-0.460165 (-39.2131 + 2. s), -0.460165 (-17.3175 + 0.85 s), 2.}        4.39464 ≤ s < 6.56777
{-0.482243 (-36.7057 + 1.8 s),                                          6.56777 ≤ s < 8.64142
 -0.482243 (-5.28668 - 0.9 s), -0.482243 (-0.863403 - 0.5 s)}
{-0.543125 (-33.4706 + 1.7 s),                                          8.64142 ≤ s < 10.4826
 -0.543125 (-7.27882 - 0.5 s), -0.543125 (-0.28228 - 0.5 s)}
{-0.547997 (-34.3797 + 1.8 s), -0.547997 (-9.26405 - 0.3 s), 3.}        10.4826 ≤ s < 12.3074
{-0.447214 (-39.5965 + 2. s), 7.1, -0.447214 (-19.0156 + s)}            12.3074 ≤ s < 14.5435
{-0.64957 (-27.5965 + 1.4 s),                                           14.5435 ≤ s < 16.083
 -0.64957 (-16.7477 + 0.4 s), -0.64957 (-10.3507 + 0.5 s)}
{-0.701862 (-25.6097 + 1.3 s),                                          16.083 ≤ s < 17.5078
 -0.701862 (-14.3709 + 0.3 s), -0.701862 (-10.1787 + 0.5 s)}
{2, 6.4, 1}                                                             True
```

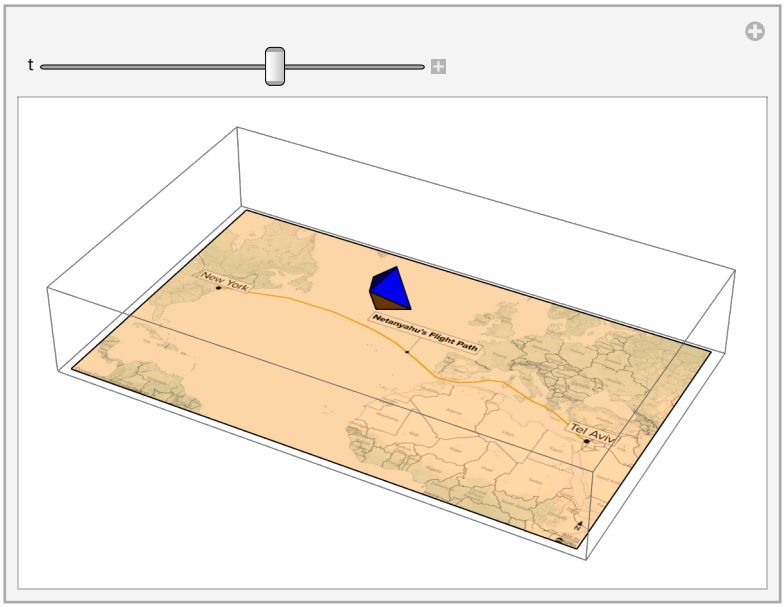
In[89]:=
```
gnet = PLP[nt, Flatten[Take[nData, All, {3}], 1]];
```

In[90]:=
```
hnet = PLP[nt, Flatten[Take[nData, All, {4}], 1]];
```

My flight path is then given by

In[•]:=
```
Manipulate[Show[Graphics3D[{FaceForm[Texture[NetFlight]], GrayLevel[.95],
      Polygon[{{0, 0, 0}, {20, 0, 0}, {20, 12, 0}, {0, 12, 0}}]}],
  Spindle[DITF[{fnet, gnet, hnet} /. s → t]]], {t, -5, 17.7}]
```

Out[•]=



probably best viewed by hand movement of the control. One interesting feature is that between parameter values -5 and 0, the plane is merely turning in place. But this is movement too although not captured by a 2 dimensional map. Note I also keep the blue top of the airplane always pointing up so that Mr. Netanyahu will not spill his coffee.

My last example is a pool swimmer. The swimmer's path is simple, a straight line segment. What makes this interesting is that there are singularities in that the the swimmer must change direction at each end of the pool. To make this as smooth as possible good swimmers use a flip turn. This involves a somersault which leaves the swimmer on its back with feet ideally touching the middle of the cross at the end of the lane. My swimmer is just a spindle but a real swimmer can bend in the knees to push off, saving time and energy. This all takes place in a minimum of time and space. Our problem is to pick the correct versions so this all goes together smoothly. The following table was not easy to construct.

First we define our pool .

In[91]:=
```
pool = {{GrayLevel[.7], Opacity[.5],
    Polygon[{{-2, -2, 0}, {-2, 12 , 0}, {27, 12, 0}, {27, -2, 0}}]}, {RGBColor[0, .4, 1],
    Opacity[.3], Polygon[{{0, 0, .1}, {0, 10, .1}, {25, 10, .1}, {25, 0, .1}}]},
   {Gray, Thickness[.01], Line[{{4, -.2, 0}, {4, -.2, 3}}],
    Line[{{4, 10.4, 0}, {4, 10.4, 3}}], Line[{{21, -.2, 0}, {21, -.2, 3}}],
    Line[{{21, 10.4, 0}, {21, 10.4, 3}}]}, {Gray, Thickness[.005],
    Line[{{4, -.2, 2.5}, {4, 10.4, 2.5}}], Line[{{21, -.2, 2.5}, {21, 10.4, 2.5}}],
    Line[{{0, 10, 0}, {0, 10, -3}}], Line[{{0, 0, 0}, {0, 0, -3}}],
    Line[{{0, 0, -3}, {0, 10, -3}}], Line[{{0, 10, -3}, {25, 10, -3}}],
    Line[{{25, 10, -3}, {25, 0, -3}}], Line[{{25, 10, 0}, {25, 10, -3}}],
    Line[{{0, 0, -3}, {25, 0, -3}}], Line[{{25, 0, 0}, {25, 0, -3}}]},
   {Black, Thickness[.01], Line[{{2, 5, -3}, {23, 5, -3}}],
    Line[{{2, 4.5, -3}, {2, 5.5, -3}}], Line[{{23, 4.5, -3}, {23, 5.5, -3}}],
    Line[{{0, 5, -1}, {0, 5, -2}}], Line[{{0, 4.5, -1.5}, {0, 5.5, -1.5}}],
    Line[{{25, 5, -1}, {25, 5, -2}}], Line[{{25, 4.5, -1.5}, {25, 5.5, -1.5}}]}};
```

Now our table

In[92]:=
```
swimTable = {{1, {1.`, 5.`, -1.5`}, 1.7177643727778618`,
    {-0.3574097334283265`, 0.8628556558297632`, -0.3574050918121926`}},
   {4, {4.`, 5.`, -.5`}, 1.5707871794617763`, {-0.3574023321058912`,
     0.8628568563190498`, 0.35740959487467133`}}, {5, {5.`, 5.`, 0}, 2.6233620316364394`,
    {0.6818100203154313`, 0.2650716811803814`, 0.6818152975943541`}},
   {6, {5.`, 5.`, 0}, 2.6233620316364394`, {0.6818100203154313`, 0.2650716811803814`,
     0.6818152975943541`}}, {19, {19.`, 5.`, 0}, 2.623366846276483`,
    {0.6818096399398168`, 0.26508020075833777`, 0.6818123657217259`}},
   {21, {21.`, 5.`, -0.2`}, 2.684132968828113`,
    {0.7464709456192447`, 0.2902111138873866`, 0.5987976592493934`}},
   {23, {23.`, 5.`, -0.2`}, 3.2653832327719825`,
    {0.9208232228240548`, 0.3788191189320995`, -0.0926318921291628`}},
   {24, {24.`, 5.`, -1.5`}, 3.689616250382108`,
    {0.6785949511024084`, 0.28109097165283137`, -0.678599114347777`}},
   {26, {22.`, 5.`, -1.5`}, 2 Pi - 2.593558125730592`,
    {0.6786001838122628`, 0.28108564378724005`, -0.6785960885436018`}},
   {31, {17.`, 5.`, -0.5`}, 2 Pi - 1.7177676618917466`,
    {-0.3574023321058912`, 0.8628568563190498`, 0.35740959487467133`}},
   {34, {14.`, 5.`, 0}, 4.565417645287839`, {-0.3574023321058912`, 0.8628568563190498`,
     0.35740959487467133`}}, {44, {4.`, 5.`, 0}, 4.565417645287839`,
    {-0.3574023321058912`, 0.8628568563190498`, 0.35740959487467133`}},
   {46, {2.`, 5.`, 0}, 3.0241797478545416`, {-0.32301949009593006`, 0.9460791381370768`,
     -0.024344884472333964`}}, {47, {1.`, 5.`, -1.5`}, 1.7177643727778618`,
    {-0.3574097334283265`, 0.8628556558297632`, -0.3574050918121926`}}};
```

Now we proceed as in the previous example .

In[93]:=
```
tsw = Flatten[Take[swimTable, All, {1}], 1]
```

Out[93]=
$$\{1, 4, 5, 6, 19, 21, 23, 24, 26, 31, 34, 44, 46, 47\}$$

In[94]:=
```
fswim = PLP[tsw, Flatten[Take[swimTable, All, {2}], 1]]
```

Out[94]=

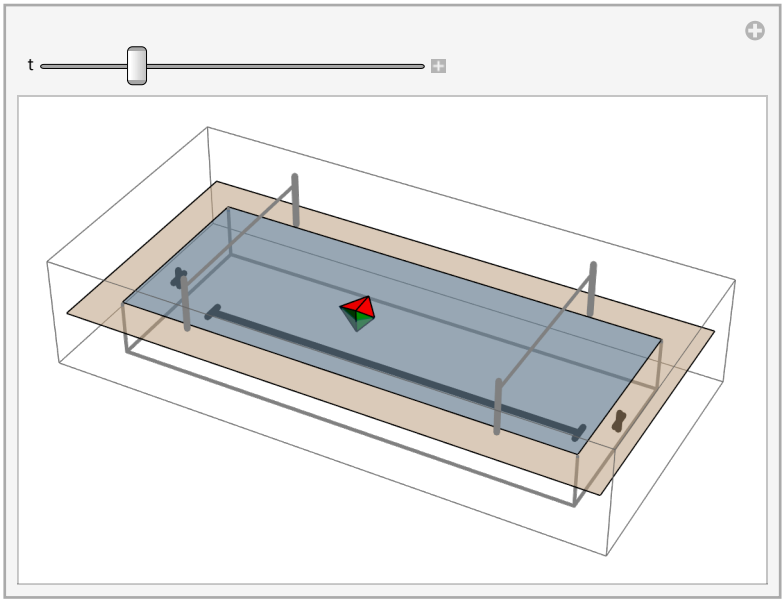
$\left\{\frac{1}{3}\ (0. + 3.\ s),\ 5.,\ \frac{1}{3}\ (-5.5 + 1.\ s)\right\}$	$1 \le s < 4$
$\{0. + 1.\ s,\ 5.,\ -2.5 + 0.5\ s\}$	$4 \le s < 5$
$\{5.,\ 5.,\ 0\}$	$5 \le s < 6$
$\left\{\frac{1}{13}\ (-19. + 14.\ s),\ 5.,\ 0\right\}$	$6 \le s < 19$
$\left\{\frac{1}{2}\ (0. + 2.\ s),\ 5.,\ \frac{1}{2}\ (3.8 - 0.2\ s)\right\}$	$19 \le s < 21$
$\left\{\frac{1}{2}\ (0. + 2.\ s),\ 5.,\ -0.2\right\}$	$21 \le s < 23$
$\{0. + 1.\ s,\ 5.,\ 29.7 - 1.3\ s\}$	$23 \le s < 24$
$\left\{\frac{1}{2}\ (96. - 2.\ s),\ 5.,\ -1.5\right\}$	$24 \le s < 26$
$\left\{\frac{1}{5}\ (240. - 5.\ s),\ 5.,\ \frac{1}{5}\ (-33.5 + 1.\ s)\right\}$	$26 \le s < 31$
$\left\{\frac{1}{3}\ (144. - 3.\ s),\ 5.,\ \frac{1}{3}\ (-17. + 0.5\ s)\right\}$	$31 \le s < 34$
$\left\{\frac{1}{10}\ (480. - 10.\ s),\ 5.,\ 0\right\}$	$34 \le s < 44$
$\left\{\frac{1}{2}\ (96. - 2.\ s),\ 5.,\ 0\right\}$	$44 \le s < 46$
$\{48. - 1.\ s,\ 5.,\ 69. - 1.5\ s\}$	$46 \le s < 47$
$\{1.,\ 5.,\ -1.5\}$	True

In[95]:=
```
gswim = PLP[tsw, Flatten[Take[swimTable, All, {3}], 1]];
```

In[96]:=
```
hswim = PLP[tsw, Flatten[Take[swimTable, All, {4}], 1]];
```

In[•]:=
```
Manipulate[Show[Graphics3D[pool], Spindle[DITF[{fswim, gswim, hswim}] /. s → t]], {t, 1, 47}]
```

Out[•]=



Note the red side is the swimmer's back, yellow is its stomach, the swimmer's right arm is green but left is blue as seen by an observer on the pool deck.

These pictures may be seen in motion on You Tube at https://barryhdayton.space/dynamicGeometry.html