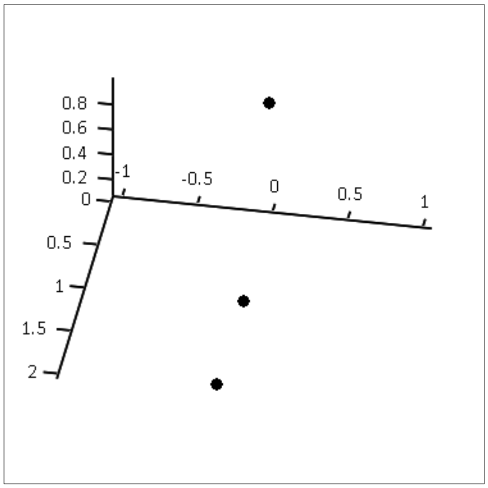
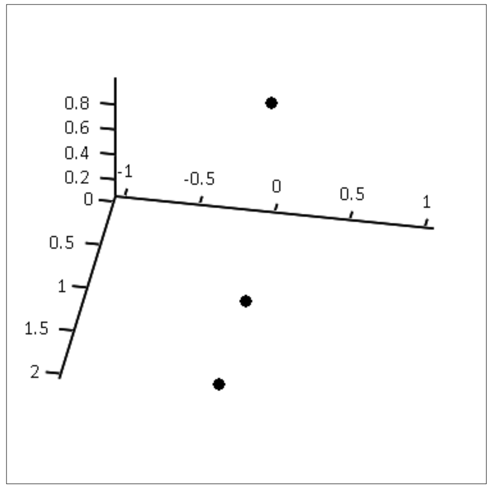
The x, y, and z coordinates of a point: $p := \begin{pmatrix} 0.989 \\ 0.091 \\ 0.119 \end{pmatrix}$

Two attempts at plotting the point:



p



$\left(p_0, p_1, p_2\right)$

What exactly is being plotted up there? Why are there three points? To what do those three points correspond? How can I plot a single point in space?

plottools emulation

The starting point for the worksheet is to determine the data structure that the Mathcad equivalent functions will use.

The Data Structure

Mathcad 3D Plot Data Structure

Mathcad has a number of functions that create data for use within the 3D Plot component. The 2 principle functions are CreateMesh and CreateSpace:

- **CreateMesh(function, [s0, s1, t0, t1], [sgrid, tgrid], [fmap])** Returns a nested array of three matrices representing the x, y, and z-coordinates of a parametric surface defined by the function of two variables in the first argument.
- **CreateSpace(function, [t0, t1], [tgrid], [fmap])** Returns a nested array of three vectors representing the x, y, and z-coordinates of a parametric space curve defined by the function of one variable in the first argument.

The brackets indicate optional arguments; the Mathcad Help gives the default values.

As we shall shortly see, these descriptions are not quite accurate, in that both functions actually return a doubly nested array, with the first level of nesting containing a single array that contains the 'actual' nested array.

For convenience, we will refer to a nested array of three matrices or vectors as a mesh and the higher level nested array as a mesh collection or mesh set.

Let's demonstrate this via a simple example

Define a function

$\text{sincos}(x, y) := \sin(x) \cdot \cos(y)$

Create 2 meshes at different parts of the x-y plane

$aa := \text{CreateMesh}(\text{sincos}, -1, 1, -1, 1, 4, 4)$

$ab := \text{CreateMesh}(\text{sincos}, -2, -1, -2, -1, 4, 4)$

Unnest the first level to get the meshes

$aa0 := aa_0$

$ab0 := ab_0$

Stack the meshes to create composite meshes

$ac := \text{stack}(aa, ab)$

$ac0 := \text{stack}(aa0, ab0)$

Examine the shape (dimensions) of the meshes

$aa = (\{3,1\})$

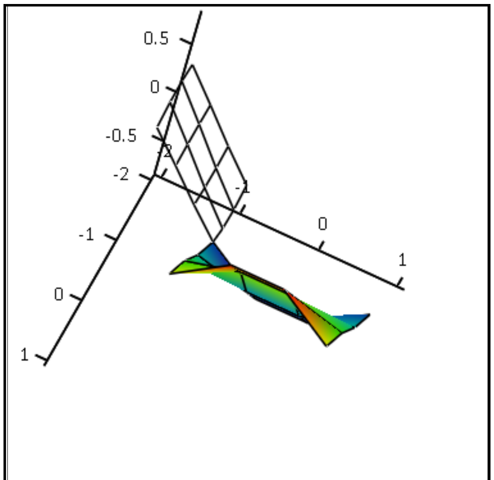
$ac^T = (\{3,1\} \quad \{3,1\})$

$aa0^T = (\{4,4\} \quad \{4,4\} \quad \{4,4\})$

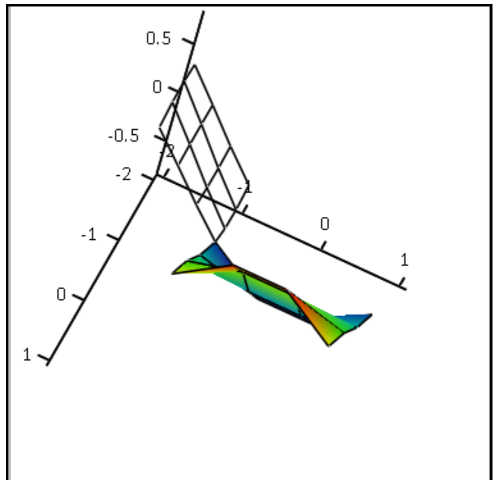
$ac0^T = (\{4,4\} \quad \{4,4\} \quad \{4,4\} \quad \{4,4\} \quad \{4,4\} \quad \{4,4\})$

Plot the meshes

Individual meshes

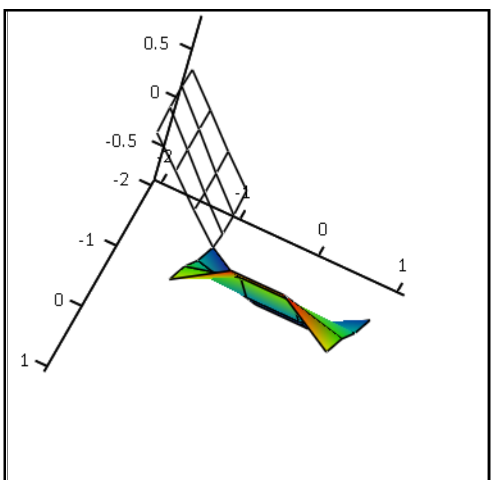


aa , ab

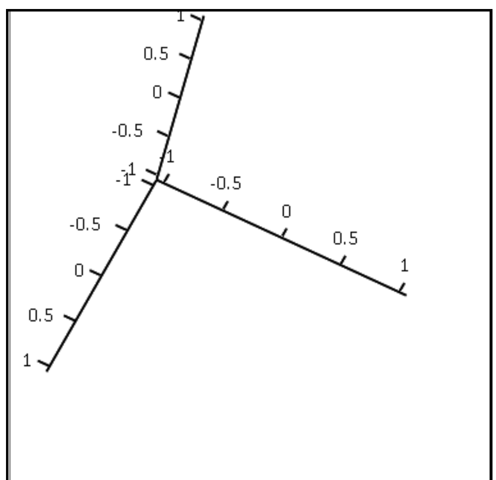


aa0 , ab0

Composite meshes



ac



ac0

As can be seen from the shape of the meshes, does indeed nest the 'nested' array. In the top pair of plots, the nested meshes and plain meshes give the same display when plotted individually. From this alone, it would seem there is little benefit in having the extra layer of nesting. The advantage of this struture can be seen from the behaviour of the composite arrays, where we've simply stacked the arrays together. The left hand plot of the bottom pair gives the same display as the separate collections, but the right hand plot treats them as 6 separate z arrays.

To retain compatibility with the existing Mathcad plot functions and 3D component, we will retain the mesh collection as the data structure for the plottool emulation.

Joining meshes

Mathcad's 3D plot component has several disadvantages, some of which are based on the default settings it applies to each subplot. The 3D plot component creates a separate subplot ('Plot') for each mesh or z-array. However, it applies an unfilled simple line mesh appearance to each plot by default. If the user wants to change the appearance of the Plots, then they have to manually change each plot.

Some of the objects we will create below comprise many meshes, which makes manually setting the plots a time-consuming and error-prone task. One way of simplying matters is to join meshes together to create single mesh rather than simply stacking meshes as above.

To do this, we create 2 functions, stackmesh and augmentmesh, that combine 2 meshes, a,b, into a single mesh in a manner analogous to their array counterparts. Unfortunately, another limitation of Mathcad is that user functions cannot have variable length argument lists, so that, unlike Mathcad's built-in function stack, we cannot simply write stackmesh(a,b,c) to combine 3 meshes, but must write stackmesh(a,stackmesh(b,c)); as a slight aid, we define specific functions, stackmesh3/augmentmesh3 and stackmesh4/augmentmesh4, that allow 3 or 4 meshes to be combined in a single call. However, the use must exercise caution to ensure that it makes sense to join meshes in such a fashion, as one of the examples below will indicate.

$$\mathrm{stackmesh}(a,b) := \left| \begin{array}{ll} \begin{pmatrix} \mathrm{stack}(a_0,b_0) \\ \mathrm{stack}(a_1,b_1) \\ \mathrm{stack}(a_2,b_2) \end{pmatrix} & \text{if } \ \mathrm{IsScalar}\left[(a_0)_{0,0}\right] \\ \overrightarrow{\mathrm{stackmesh}(a,b)} & \text{otherwise} \end{array} \right.$$

$$\mathrm{stackmesh3}(a,b,c) := \mathrm{stackmesh}(a,\mathrm{stackmesh}(b,c))$$

$$\mathrm{stackmesh4}(a,b,c,d) := \mathrm{stackmesh}(a,\mathrm{stackmesh}(b,\mathrm{stackmesh}(c,d)))$$

$$\mathrm{augmentmesh}(a,b) := \left| \begin{array}{ll} \begin{pmatrix} \mathrm{augment}(a_0,b_0) \\ \mathrm{augment}(a_1,b_1) \\ \mathrm{augment}(a_2,b_2) \end{pmatrix} & \text{if } \ \mathrm{IsScalar}\left[(a_0)_{0,0}\right] \\ \overrightarrow{\mathrm{augmentmesh}(a,b)} & \text{otherwise} \end{array} \right.$$

$$\mathrm{augmentmesh3}(a,b,c) := \mathrm{augmentmesh}(a,\mathrm{augmentmesh}(b,c))$$

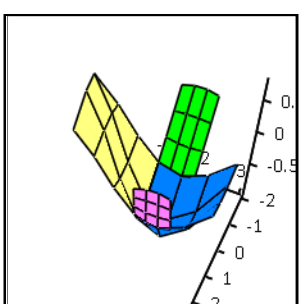
$$\mathrm{augmentmesh4}(a,b,c,d) := \mathrm{augmentmesh}(a,\mathrm{augmentmesh}(b,\mathrm{augmentmesh}(c,d)))$$

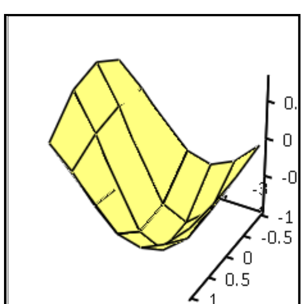
Example:

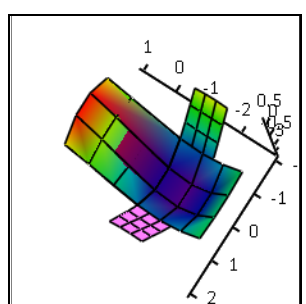
$ac := \mathrm{CreateMesh}(\mathrm{sincos},-1,-3,-1,1,4,4)$ create 2 additional meshes

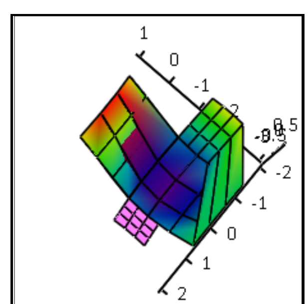
$ad := \mathrm{CreateMesh}(\mathrm{sincos},-2,-1,1,2,4,4)$

$j2 := \mathrm{stackmesh}(aa,ac)$ $j3 := \mathrm{stackmesh3}(aa,ac,ab)$ plot all 4 meshes


aa , ab , ac , ad


j2


j2 , ab , ad


j3 , ad

The first plot, shows all four meshes individually coloured. The yellow and blue meshes (aa and ac) are effectively continuous and may be joined, as shown in the second and third plots, where the joined mesh only uses one Plot tab in the 3D component.

However, the fourth plot shows that it is not reasonable to join every adjoining mesh due to the way the 3D plot component interprets the data. In this instance, it draws a surface between the last edge of j2 and the first edge of ab, which is not what is desired.

zmesh(zmat,xvec,yvec): generates a 3D plot's x and y matrices from a z matrix (zmat), given the x and y axis vectors (xvec and yvec) as inputs and returns the result as a meshset.

$$\mathrm{zmesh}(zmat, xvec, yvec) := \left| \begin{array}{l} M_0 \leftarrow \left| \begin{array}{l} xvec \leftarrow \mathrm{fillint}(\mathrm{rows}(zmat)) \quad \text{if} \quad \mathrm{IsScalar}(xvec) \\ \mathrm{repcol}(xvec, \mathrm{cols}(zmat)) \end{array} \right. \\ M_1 \leftarrow \left| \begin{array}{l} yvec \leftarrow \mathrm{fillint}(\mathrm{cols}(zmat)) \quad \text{if} \quad \mathrm{IsScalar}(yvec) \\ \mathrm{reprow}(yvec, \mathrm{rows}(zmat)) \end{array} \right. \\ M_2 \leftarrow zmat \\ (M) \end{array} \right.$$

zmeshlim(xvec,yvec,zmat): generates a 3D plot's x and y matrices from a z matrix (zmat), given the min/max values for the x and y axes and returns the result as a meshset.

$$\mathrm{zmeshlim}(zmat, xmin, xmax, ymin, ymax) := \left| \begin{array}{l} xvec \leftarrow \left| \begin{array}{l} 0 \quad \text{if} \quad xmin = xmax \\ \mathrm{linspace}(xmin, xmax, \mathrm{rows}(zmat)) \quad \text{otherwise} \end{array} \right. \\ yvec \leftarrow \left| \begin{array}{l} 0 \quad \text{if} \quad ymin = ymax \\ \mathrm{linspace}(ymin, ymax, \mathrm{cols}(zmat)) \quad \text{otherwise} \end{array} \right. \\ \mathrm{zmesh}(xvec, yvec, zmat) \end{array} \right.$$

mesh2xyz(M): converts the meshes within a meshset into an equivalent nested array of (x,y,z) vectors.

$$\mathrm{mesh2xyz}(M) := \left| \begin{array}{l} \text{for} \quad k \in 0 .. \mathrm{last}(M) \\ \quad \left| \begin{array}{l} mesh \leftarrow M_k \\ (x \quad y \quad z) \leftarrow (mesh_0 \quad mesh_1 \quad mesh_2) \\ \text{for} \quad i \in 0 .. \mathrm{rows}(x) - 1 \\ \quad \text{for} \quad j \in 0 .. \mathrm{cols}(x) - 1 \\ \quad \quad v_{i,j} \leftarrow \mathrm{stack}(x_{i,j}, y_{i,j}, z_{i,j}) \\ xyz_k \leftarrow v \end{array} \right. \\ xyz \end{array} \right.$$

$$\mathrm{mesh2vec}(M) := \mathrm{mesh2xyz}(M)$$

xyz2mesh(N): converts a nested array of (x,y,z) vectors into an equivalent meshset.

$$\mathrm{xyz2mesh}(N) := \left| \begin{array}{l} \text{for} \quad k \in 0 .. \mathrm{last}(N) \\ \quad \left| \begin{array}{l} vecarray \leftarrow N_k \\ \text{for} \quad i \in 0 .. \mathrm{rows}(vecarray) - 1 \\ \quad \text{for} \quad j \in 0 .. \mathrm{cols}(vecarray) - 1 \\ \quad \quad (x_{i,j} \quad y_{i,j} \quad z_{i,j}) \leftarrow \left| \begin{array}{l} v \leftarrow vecarray_{i,j} \\ (v_0 \quad v_1 \quad v_2) \end{array} \right. \\ meshset_k \leftarrow (x \quad y \quad z)^T \end{array} \right. \\ meshset \end{array} \right.$$

Limits(M): returns a 3x2 matrix with the first column containing the minimum values of the meshset M's x,y,z values and the second column the maximum values.

$$\mathrm{Limits}(M) := \left| \begin{array}{l} (\mathrm{maxd} \quad \mathrm{mind}) \leftarrow \left(\begin{pmatrix} -\infty & -\infty & -\infty \end{pmatrix}^T \quad \begin{pmatrix} \infty & \infty & \infty \end{pmatrix}^T \right) \\ \text{for } k \in 0 .. \mathrm{last}(M) \\ \quad \left| \begin{array}{l} \mathrm{mesh} \leftarrow M_k \\ \text{if } \mathrm{rows}(\mathrm{mesh}) = 1 \\ \quad \left| \begin{array}{l} \mathrm{maxd}_2 \leftarrow \mathrm{if}\left(\max(\mathrm{mesh}_0) > \mathrm{maxd}_2, \max(\mathrm{mesh}_0), \mathrm{maxd}_2\right) \\ \mathrm{mind}_2 \leftarrow \mathrm{if}\left(\min(\mathrm{mesh}_0) < \mathrm{mind}_2, \min(\mathrm{mesh}_0), \mathrm{mind}_2\right) \end{array} \right. \\ \text{for } d \in 0 .. 2 \qquad \text{otherwise} \\ \quad \left| \begin{array}{l} \mathrm{maxd}_d \leftarrow \mathrm{if}\left(\max(\mathrm{mesh}_d) > \mathrm{maxd}_d, \max(\mathrm{mesh}_d), \mathrm{maxd}_d\right) \\ \mathrm{mind}_d \leftarrow \mathrm{if}\left(\min(\mathrm{mesh}_d) < \mathrm{mind}_d, \min(\mathrm{mesh}_d), \mathrm{mind}_d\right) \end{array} \right. \end{array} \right. \\ \mathrm{augment}(\mathrm{mind}, \mathrm{maxd}) \end{array} \right.$$

Plottool Function Equivalents

$$\mathrm{line2vec}(\mathrm{lin}) := \left| \begin{array}{l} \mathrm{lin} \leftarrow \mathrm{lin}_0 \\ v \leftarrow \mathrm{stack}\left[(\mathrm{lin}_0)_0, (\mathrm{lin}_1)_0, (\mathrm{lin}_2)_0 \right] \\ w \leftarrow \mathrm{stack}\left[(\mathrm{lin}_0)_1, (\mathrm{lin}_1)_1, (\mathrm{lin}_2)_1 \right] \\ (v \quad w)^T \end{array} \right.$$

Rotation

The Maple function rotate operates on Maple's 2D and 3D data structures and accepts several forms of argument; of particular interest to us are the 3D forms.

rotate(M,q,f,r)
rotate(M,q,p1,p2)

where M is a 3D data structure, q,f,r are angles and p1,p2 are 3D points. In the first form, the angles q,f,r represent rotation around the x-axis (roll), y-axis (pitch) and z-axis (yaw) respectively, ie, a combined rotation of M around the origin. In the second form, p1 and p2 define a vector around which M is rotated by q.

We shall implement rotate as three related functions that: iterate through a mesh collection, apply a rotation to a mesh and generate a rotation matrix, respectively.

Function R3

R3 takes 3 arguments, f,q,y, and returns a 3D rotation matrix.

If q is an array, then if y is zero then **R3** assumes q is a mesh representing a line, otherwise it assumes both q and y are vectors and that they represent two points on the line about which it will generate a rotation through an angle f.

Otherwise **R3** assumes they are the axis rotation angles.

$$
\mathbf{R3}(\phi,\theta,\psi) := \left|\begin{array}{l}
\text{if}\quad \text{IsArray}(\theta) \\
\quad\left|\begin{array}{l}
\begin{pmatrix}\theta\\ \psi\end{pmatrix} \leftarrow \text{line2vec}(\theta)\quad \text{if}\quad \psi = 0 \\
v \leftarrow \dfrac{\psi-\theta}{|\psi-\theta|} \\
\begin{pmatrix} a & b & c \end{pmatrix} \leftarrow v^{T} \\
\begin{pmatrix} c\phi & s\phi \end{pmatrix} \leftarrow \begin{pmatrix} \cos(\phi) & \sin(\phi) \end{pmatrix} \\
(1-c\phi)\cdot\begin{pmatrix} a^{2} & a\cdot b & a\cdot c \\ b\cdot a & b^{2} & b\cdot c \\ c\cdot a & c\cdot b & c^{2} \end{pmatrix} + \begin{pmatrix} c\phi & -c\cdot s\phi & b\cdot s\phi \\ c\cdot s\phi & c\phi & -a\cdot s\phi \\ -b\cdot s\phi & a\cdot s\phi & c\phi \end{pmatrix}
\end{array}\right. \\
\text{otherwise} \\
\quad\left|\begin{array}{l}
\begin{pmatrix} c\theta & c\phi & c\psi \end{pmatrix} \leftarrow \begin{pmatrix} \cos(\theta) & \cos(\phi) & \cos(\psi) \end{pmatrix} \\
\begin{pmatrix} s\theta & s\phi & s\psi \end{pmatrix} \leftarrow \begin{pmatrix} \sin(\theta) & \sin(\phi) & \sin(\psi) \end{pmatrix} \\
\begin{pmatrix} c\psi\cdot c\theta & c\psi\cdot s\theta\cdot s\phi - c\phi\cdot s\psi & s\psi\cdot s\phi + c\psi\cdot c\phi\cdot s\theta \\ c\theta\cdot s\psi & c\psi\cdot c\phi + s\psi\cdot s\theta\cdot s\phi & c\phi\cdot s\psi\cdot s\theta - c\psi\cdot s\phi \\ -s\theta & c\theta\cdot s\phi & c\theta\cdot c\phi \end{pmatrix}
\end{array}\right.
\end{array}\right.
$$

Function rotatemesh

rotatemesh takes 4 arguments, mesh,q,f,r, and applies **R3** to each element of mesh.

$$
\text{rotatemesh}(\text{mesh},\phi,\theta,\psi) := \left|\begin{array}{l}
R \leftarrow \mathbf{R3}(\phi,\theta,\psi) \\
\begin{pmatrix} x & y & z \end{pmatrix} \leftarrow \begin{pmatrix} \text{mesh}_0 & \text{mesh}_1 & \text{mesh}_2 \end{pmatrix} \\
\text{nrows} \leftarrow \text{rows}(x) \\
\text{ncols} \leftarrow \text{cols}(x) \\
\text{for}\quad i \in 0..\text{nrows}-1 \\
\quad \text{for}\quad j \in 0..\text{ncols}-1 \\
\qquad \begin{pmatrix} x_{i,j} & y_{i,j} & z_{i,j} \end{pmatrix} \leftarrow \left|\begin{array}{l}
v \leftarrow \begin{pmatrix} x_{i,j} & y_{i,j} & z_{i,j} \end{pmatrix}^{T} \\
\left[R\cdot(v-\phi)+\phi\right]^{T}\quad \text{if}\quad \text{IsArray}(\phi) \\
(R\cdot v)^{T}\quad \text{otherwise}
\end{array}\right. \\
\begin{pmatrix} x & y & z \end{pmatrix}^{T}
\end{array}\right.
$$

Function rotate

rotate is the Maple plottool function equivalent. It takes 4 arguments, mesh,q,f,r, and rotates them as per **R3**.

$$
\text{rotate}(\text{meshset},\phi,\theta,\psi) := \left|\begin{array}{l}
\text{return}\ \ \text{rotatemesh}(\text{meshset},\phi,\theta,\psi)\quad \text{if}\quad \text{cols}(\text{meshset}_0) > 1 \\
\text{for}\quad k \in 0..\text{last}(\text{meshset}) \\
\quad \text{mesh}_k \leftarrow \text{rotatemesh}(\text{meshset}_k,\phi,\theta,\psi) \\
\text{mesh}
\end{array}\right.
$$

$$
\text{rotatev}(\text{meshset},v) := \text{rotate}(\text{meshset},v_0,v_1,v_2)
$$

Reflection

The Maple function reflect also operates on Maple's 2D and 3D data structures and accepts several forms of argument; of particular interest to us are the 3D forms.

reflect(M,p1)

reflect(M,p1,p2)
reflect(M,p1,p2,p3)

where M is a 3D data structure and p1,p2,p3 are 3D points. In the first form, M is reflected about the point p1, in the second form, p1 and p2 define a line around which M is reflected and, in the final form, p1,p2 and p3 define a plane in which M is reflected (more accurately, p1&p2 and p1&p3 define 2 lines which in turn define the plane)..

We shall implement rotate as three related functions that: iterate through a mesh collection, apply a rotation to a mesh and generate a rotation matrix, respectively.

Function reflect3

reflect3 takes 3 arguments,u,v,w, (all 3D points, u being common to the 2 plane-defining lines) and returns a 3D reflection matrix.

$$\mathrm{reflect3}(u,v,w) := \left| \begin{array}{l} \text{return } u \quad \text{if} \quad \mathrm{IsScalar}(v) \\ \text{return } \dfrac{v-u}{|v-u|} \quad \text{if} \quad \mathrm{IsScalar}(w) \\ p \leftarrow v-u \\ q \leftarrow w-u \\ \dfrac{p \times q}{|p \times q|} \end{array} \right.$$

Function reflectmesh

reflectmesh takes 4 arguments, mesh,u,v,w, and applies reflect3 to each element of mesh.

$$\mathrm{reflectmesh}(mesh,u,v,w) := \left| \begin{array}{l} R \leftarrow \mathrm{reflect3}(u,v,w) \\ (x \quad y \quad z) \leftarrow (mesh_0 \quad mesh_1 \quad mesh_2) \\ \text{for } \quad i \in 0 .. \mathrm{rows}(x)-1 \\ \quad \text{for } \quad j \in 0 .. \mathrm{cols}(x)-1 \\ \qquad (x_{i,j} \quad y_{i,j} \quad z_{i,j}) \leftarrow \left| \begin{array}{l} s \leftarrow (x_{i,j} \quad y_{i,j} \quad z_{i,j})^T \\ p \leftarrow R \quad \text{if} \quad \mathrm{IsScalar}(v) \\ \text{otherwise} \\ \quad \left| \begin{array}{l} p \leftarrow (s \cdot R) \cdot R \quad \text{if} \quad \mathrm{IsScalar}(w) \\ p \leftarrow s - (s \cdot R) R \quad \text{otherwise} \end{array} \right. \\ (2p - s)^T \end{array} \right. \\ (x \quad y \quad z)^T \end{array} \right.$$

Function reflect

reflect is the Maple plottool function equivalent. It takes 4 arguments, mesh,q,f,r, and reflects them as per reflect3.

$$\mathrm{reflect}(mesh,u,v,w) := \left| \begin{array}{l} \text{return } \mathrm{reflectmesh}(mesh,u,v,w) \quad \text{if} \quad \mathrm{cols}(mesh_0) > 1 \\ \text{for } \quad k \in 0 .. \mathrm{last}(mesh) \\ \quad mesh_k \leftarrow \mathrm{reflectmesh}(mesh_k,u,v,w) \\ mesh \end{array} \right.$$

Translation

The Maple function translate operates on Maple's 2D and 3D data structures and accepts several forms of argument; of particular interest to us is the 3D form.

translate(M,Dx,Dy,Dz)

where M is a 3D data structure, and Dx,Dy,Dz are distances to move the M in each of the 3 axes.

We shall implement translate as two related functions that: iterate through a mesh collection, and translate a mesh, respectively.

Function translatemesh

translatemesh takes 4 arguments, mesh,x,y,z, and adds x to the first element (array) of mesh, y to the second and z to the third.

$$\mathrm{translatemesh}(mesh, x, y, z) := \left(mesh_0 + x \quad mesh_1 + y \quad mesh_2 + z\right)^T$$

Function translate

translate is the Maple plottool function equivalent. It takes 4 arguments, mesh,x,y,z and reflects them as per translatemesh. translatev is a variant that takes a 3-vector as an argument instead of individual co-ordinates.

$$\mathrm{translate}(mesh, x, y, z) := \left| \begin{array}{l} \text{return } \mathrm{translatemesh}(mesh, x, y, z) \quad \text{if} \quad \mathrm{cols}(mesh_0) > 1 \\ \text{for} \quad k \in 0 .. \mathrm{last}(mesh) \\ \quad mesh_k \leftarrow \mathrm{translatemesh}(mesh_k, x, y, z) \\ mesh \end{array} \right.$$

$$\mathrm{translatev}(mesh, v) := \mathrm{translate}(mesh, v_0, v_1, v_2)$$

Scaling

The Maple function scale operates on Maple's 2D and 3D data structures and accepts several forms of argument; of particular interest to us are the 3D forms.

scale(M,sx,sy,sz)
scale(M,sx,sy,sz,[x,y,z])

where M is a 3D data structure, and sx,sy,sz are scaling factors to apply to M in each of the 3 axes, and [x,y,z] are the co-ordinates that specify a point to re-scale about. Note that the first form scales about the origin.

We shall only implement the first form of scale and do that as two related functions that: iterate through a mesh collection, and scale a mesh, respectively.

Function scalemesh

scalemesh takes 4 arguments, mesh,x,y,z, and multiplies the first element (array) of mesh by x, second by y and the third by z.

$$\mathrm{scalemesh}(mesh, x, y, z) := \overrightarrow{\left(mesh \cdot \mathrm{stack}(x, y, z)\right)}$$

Function scale

scale is the Maple plottool function equivalent. It takes 4 arguments, mesh,x,y,z and reflects them as per scalemesh.

$$\mathrm{scale}(mesh, u, v, w) := \left| \begin{array}{l} \text{return } \mathrm{scalemesh}(mesh, u, v, w) \quad \text{if} \quad \mathrm{cols}(mesh_0) > 1 \\ \text{for} \quad k \in 0 .. \mathrm{last}(mesh) \\ \quad mesh_k \leftarrow \mathrm{scalemesh}(mesh_k, u, v, w) \\ mesh \end{array} \right.$$

$$\mathrm{scalev}(mesh, v) := \mathrm{scale}(mesh, v_0, v_1, v_2)$$

Miscellaneous

As an aside, Maple's norm(x,2) applied to a vector returns the vector's magnitude.

$$\mathrm{norm}(x, n) := |x|$$

Function line

point is the Maple plottool function equivalent. It takes 3 arguments, x,y,z , the 3D co-ordinates of point, and creates a mesh representing that point. pointv is a variant that takes a 3-vector (representation of a point); vec2pt is an alias for pointv. The inverse function of pointv is pt2vec

$$\text{point}(a,b,c) := \begin{array}{|l} (x_0 \;\; y_0 \;\; z_0) \leftarrow (a \;\; b \;\; c) \\ p_0 \leftarrow (x \;\; y \;\; z)^T \\ p \end{array}$$

$$\text{pointv}(v) := \begin{array}{|l} (x_0 \;\; y_0 \;\; z_0) \leftarrow (v_0 \;\; v_1 \;\; v_2) \\ p_0 \leftarrow (x \;\; y \;\; z)^T \\ p \end{array}$$

$$\text{vec2pt}(v) := \text{pointv}(v)$$

$$\text{pt2vec}(pt) := \begin{array}{|l} pt \leftarrow pt_0 \\ \text{stack}(pt_0, pt_1, pt_2) \end{array}$$

line is the Maple plottool function equivalent. It takes 2 arguments, a,b , a pair of 3D vectors, and creates mesh representing a line segment between a and b.

$$\text{line}(a,b) := \begin{array}{|l} ab \leftarrow \text{augment}(a,b)^T \\ ln_0 \leftarrow \left(ab^{\langle 0\rangle} \;\; ab^{\langle 1\rangle} \;\; ab^{\langle 2\rangle}\right)^T \\ ln \end{array}$$

polygon(p,n,s) returns a polygon of order n lying in the xy plane, centred at p and of radius (scale) s

$$\text{polygon}(p,n,s) := \begin{array}{|l} p \leftarrow \text{stack}(p,p,p) \quad \text{if} \quad \text{IsScalar}(p) \\ \theta \leftarrow \dfrac{2\pi}{n} \\ \text{for} \quad k \in 0..n \\ \quad \begin{array}{|l} x_k \leftarrow \cos(k\cdot\theta) \\ y_k \leftarrow \sin(k\cdot\theta) \\ z_k \leftarrow 0 \end{array} \\ poly_0 \leftarrow \overrightarrow{\left(s\cdot(x + p_0 \;\; y + p_1 \;\; z + p_2)^T\right)} \\ poly \end{array}$$

rectange(p1,p2,s) returns a rectangle lying between points p1 and p2, scaled by s

$$\text{rectangle}(p1,p2,s) := \left(\overrightarrow{\left(\left[s\cdot\left[\begin{pmatrix} p1_0 & p1_0 \\ p2_0 & p2_0 \end{pmatrix} \;\; \begin{pmatrix} p1_1 & p2_1 \\ p1_1 & p2_1 \end{pmatrix} \;\; \begin{pmatrix} p1_2 & p2_2 \\ p1_2 & p2_2 \end{pmatrix}\right]\right]^T\right)}\right)$$

square(p,s) returns a square lying in the xy plane, centred at p and of radius (scale) s

$$\text{square}(p,s) := \begin{array}{|l} p \leftarrow \text{stack}(p,p,p) \quad \text{if} \quad \text{IsScalar}(p) \\ \text{rectangle}(p - \text{stack}(1,1,0), p + \text{stack}(1,1,0), s) \end{array}$$

cuboid(p1,p2,s) returns a cuboid lying between points p1 and p2, scaled by s

$$\text{cuboid}(p1, p2, s) := \left|\begin{array}{l} s \leftarrow \text{stack}(s, s, s) \quad \text{if} \quad \text{rows}(s) = 0 \\ pa \leftarrow \text{stack}(0, 0, 0) \\ pb \leftarrow p2 - p1 \\ sqa \leftarrow \left(\left[\begin{pmatrix} pa_0 \\ pb_0 \\ pb_0 \\ pa_0 \\ pa_0 \end{pmatrix} \begin{pmatrix} pa_1 \\ pa_1 \\ pb_1 \\ pb_1 \\ pa_1 \end{pmatrix} \begin{pmatrix} pa_2 \\ pa_2 \\ pa_2 \\ pa_2 \\ pa_2 \end{pmatrix}\right]^T\right) \\ sq1 \leftarrow \text{augmentmesh}(sqa, \text{translatev}(sqa, \text{stack}(0, 0, pb_2))) \\ sq \leftarrow \left(\left[\begin{pmatrix} pa_0 \\ pb_0 \\ pb_0 \\ pa_0 \\ pa_0 \end{pmatrix} \begin{pmatrix} pa_1 \\ pa_1 \\ pa_1 \\ pa_1 \\ pa_1 \end{pmatrix} \begin{pmatrix} pa_2 \\ pa_2 \\ pb_2 \\ pb_2 \\ pa_2 \end{pmatrix}\right]^T\right) \\ sq2 \leftarrow \text{augmentmesh}(sq, \text{translatev}(sq, \text{stack}(0, pb_1, 0))) \\ \text{translatev}(\text{scalev}(\text{augmentmesh}(sq1, sq2), s), p1) \end{array}\right.$$

Because of the way the plot component draws, a cuboid is made up of 2 hollow cuboids, with the second rotated to cover the hole in the first. The order of sides is important to avoid cross-over diagonals.

cube(p,s) returns a cube aligned with the xyz axes, centred at p and of radius (scale) s

$$\text{cube}(p, s) := \text{translatev}(\text{cuboid}(\text{stack}(-1, -1, -1), \text{stack}(1, 1, 1), s), p)$$

BoundingBox(M,s) returns a cuboid lying between the limits of M, scaled by s

$$\text{BoundingBox}(M, s) := \left|\begin{array}{l} lim \leftarrow \text{Limits}(M) \\ \text{cuboid}\left(lim^{\langle 0\rangle}, 1.1 \cdot lim^{\langle 1\rangle}, s\right) \end{array}\right.$$

a cleverer version might shear or rotate the bounding box to lie closer to the true bounds

BoundingCube(M,s) returns a cube lying, completely bounding M and scaled by s

$$\text{BoundingCube}(M, s) := \left|\begin{array}{l} ones \leftarrow \text{stack}(1, 1, 1) \\ lim \leftarrow \text{Limits}(M) \\ (maxd \quad mind) \leftarrow (\max(lim) \quad \min(lim)) \\ (maxd \quad mind) \leftarrow (maxd \cdot ones \quad mind \cdot ones) \\ \text{cuboid}\left(lim^{\langle 0\rangle}, lim^{\langle 1\rangle}, s\right) \end{array}\right.$$

cylinder(p,r,h) returns a cylinder of radius and height h, aligned along the z-axis with a base at the origin.

$$\text{cylinder}(p, r, h) := \left| \begin{array}{l} p \leftarrow \text{stack}(p, p, p) \quad \text{if} \quad \text{IsScalar}(p) \\ N \leftarrow 32 \\ \text{for} \quad k \in 0 .. N \\ \quad \left| \begin{array}{l} x_k \leftarrow r \cdot \cos\left(k \cdot \frac{2 \cdot \pi}{N}\right) \\ y_k \leftarrow r \cdot \sin\left(k \cdot \frac{2 \cdot \pi}{N}\right) \\ z_k \leftarrow 0 \end{array} \right. \\ v_0 \leftarrow (x \quad y \quad z)^T \\ v \leftarrow \text{augmentmesh}(v, \text{translatev}(v, \text{stack}(0, 0, h))) \\ \text{translatev}(v, p) \end{array} \right.$$

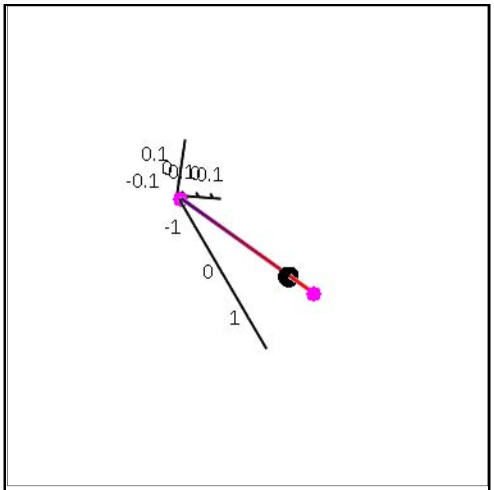
plottools emulation

$\mathbf{pt} := \text{vec2pt}(p)$

plot below shows a line passing through point p

$\text{scl} := 1.6$

$\mathbf{ln} := \text{line}(-\text{scl} \cdot p, \text{scl}\, p)$



pt, **ln**

$$\mathbf{pt} = \begin{bmatrix} \begin{bmatrix} (0.989) \\ (0.091) \\ (0.119) \end{bmatrix} \end{bmatrix} \qquad \mathbf{ln} = \begin{bmatrix} \begin{bmatrix} \begin{pmatrix} -1.582 \\ 1.582 \end{pmatrix} \\ \begin{pmatrix} -0.146 \\ 0.146 \end{pmatrix} \\ \begin{pmatrix} -0.19 \\ 0.19 \end{pmatrix} \end{bmatrix} \end{bmatrix}$$

aa = $\left(\{3,1\}\right)$

j3

$\text{identity}(1) = (1)$

$\text{testmesh} := \text{zmesh}(\text{fillmat}(2, 2), 0, 0)$

$\text{testmesh} = \blacksquare$

$$\left|\begin{array}{l} id \leftarrow \text{identity}(2) \\ \begin{pmatrix} a_0 & a_1 & a_2 \end{pmatrix} \leftarrow \begin{pmatrix} id & id & id \end{pmatrix} \\ (a) \end{array}\right. = (\{3,1\})$$

$\text{mesh2xyz}(\text{testmesh}) = \blacksquare$

$\text{xyz2mesh}(\text{mesh2xyz}(\text{testmesh})) = \blacksquare$

testing

$\mathrm{Limits}(\mathrm{testmesh}) = \blacksquare$

translate(testmesh , 1 , 2 , 3) = ∎

translatev(testmesh , stack(1 , 1 , 1)) = ∎

scale(testmesh , 1 , 2 , 3) = ∎

identity(1) = (1)

$\mathrm{point}(1,2,3) = (\{3,1\})$ $\quad$ $\mathrm{pointv}(\mathrm{stack}(1,2,3)) = (\{3,1\})$

$\mathrm{pt2vec}(\mathrm{point}(1,2,3))^T = (1 \quad 2 \quad 3)$

$(\mathrm{line}(\mathrm{stack}(1,2,3), \mathrm{stack}(3,2,1))_0)_0 = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$

$\mathrm{identity}(1) = (1)$

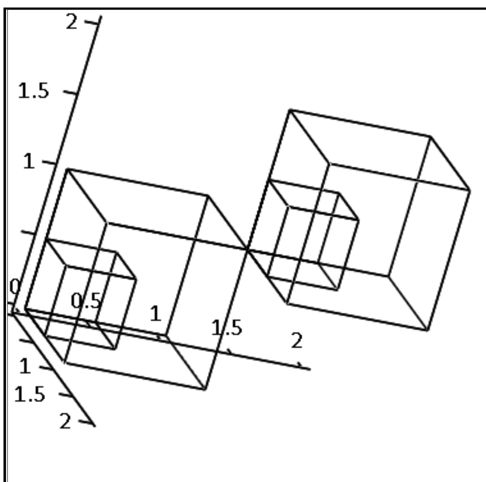
$p1 := \mathrm{stack}(1,1,1)$ $\quad$ $p2 := \mathrm{stack}(2,2,2)$

$s := 1$

$\mathrm{line2vec}(\mathrm{line}(\mathrm{stack}(1,2,3), \mathrm{stack}(3,2,1))) = \begin{pmatrix} \{3,1\} \\ \{3,1\} \end{pmatrix}$

$\mathrm{rectangle}(p1, p2, s) = (\{3,1\})$

$$
\mathrm{cuboid}(p1, p2, s) := \left|
\begin{array}{l}
s \leftarrow \mathrm{stack}(s, s, s) \quad \text{if} \quad \mathrm{rows}(s) = 0 \\
sqa \leftarrow \left(\left[\begin{pmatrix} p1_0 \\ p2_0 \\ p2_0 \\ p1_0 \\ p1_0 \end{pmatrix} \begin{pmatrix} p1_1 \\ p1_1 \\ p2_1 \\ p2_1 \\ p1_1 \end{pmatrix} \begin{pmatrix} p1_2 \\ p1_2 \\ p1_2 \\ p1_2 \\ p1_2 \end{pmatrix} \right]^T \right) \\
sq1 \leftarrow \mathrm{augmentmesh}\left(sqa, \mathrm{translatev}\left(sqa, \mathrm{stack}\left(0, 0, p2_2 - p1_2\right)\right)\right) \\
sq \leftarrow \left(\left[\begin{pmatrix} p1_0 \\ p2_0 \\ p2_0 \\ p1_0 \\ p1_0 \end{pmatrix} \begin{pmatrix} p1_1 \\ p1_1 \\ p1_1 \\ p1_1 \\ p1_1 \end{pmatrix} \begin{pmatrix} p1_2 \\ p1_2 \\ p2_2 \\ p2_2 \\ p1_2 \end{pmatrix} \right]^T \right) \\
sq2 \leftarrow \mathrm{augmentmesh}\left(sq, \mathrm{translatev}\left(sq, \mathrm{stack}\left(0, p2_1 - p1_1, 0\right)\right)\right) \\
\mathrm{scalev}\left(\mathrm{augmentmesh}\left(sq1, sq2\right), s\right)
\end{array}
\right.
$$



$\mathrm{cuboid}(zero \leftarrow \mathrm{stack}(0, 0, 0), one \leftarrow \mathrm{stack}(1, 1, 1), 1), \mathrm{cuboid}(zero, one, 0.5), \mathrm{cuboid}(one, two \leftarrow \mathrm{stack}(2, 2, 2), 1).$

$\text{identity}(1) = (1)$

$cuboid(one, two, 0.5)$