# Putting a Scene in Three Dimensions onto a Two-Dimensional Canvas

*Note: You may notice differences between this Maple worksheet and the equivalent Mathematica notebook. These differences were introduced to preserve the content of these modules and were necessary because of major functional differences between Maple and Mathematica.*

## Introduction

OBJECTIVE: Learn to use linear transformations to map points from three dimensions onto a two-dimensional space.

Since time immemorial, artists have faced a problem that computer graphics artists are facing today, and that is how to picture three-dimensional objects on a two-diemnsional plane. The representation of the objects may vary, depending on the positions of the "eye of the beholder". In the project that follows, we analyze two aspects of this problem. Both are linear mappings of points in three-dimensional space to a plane. Part 1 addresses a problem suggested in the text (Section 10.3, Exercise 61) in which the viewpoint belongs to a single beholder. The remainder of the lab focuses on parallel projections that resemble a situation in which the results of X-rays demonstrate the need for a CAT-Scan. The linear algebra behind such mappings is introduced in Part V.

## Technology Guidelines

NOTE: If you have just finished a worksheet, **restart** *Maple* before executing a new worksheet.
TO OPEN SECTIONS,
Click on the **PLUS** sign at the left hand side of the screen *or* select **Expand All Sections** from the **View** drop down menu.

TO STOP AN EXECUTION
Click on **STOP** button from the toolbar.

ORDER OF EXECUTION
Execute commands in the order given. Do not skip any *Maple* Input lines within a given worksheet

Alternatively, you can execute the entire worksheet by selecting the **Execute Worksheet** command from the **Edit** drop down menu.

SAVING WORKSHEETS.

You can save anytime to any directory you choose, and it is wise to save often.
EXPERIENCING MAJOR PROBLEMS
Save if appropriate, and if appropriate then shut down *Maple* and start it up again.

# Part I: Mapping to the *Y-Z* Plane from a Single Point

First, consider the problem from the text, where all points are mapped to the *y-z* plane. The viewer is at the point ( $x_0$ , 0, 0), and any point (*x*, *y*, *z*) between the viewer and the *y-z* plane appears at the point (0, *t y*, *t z*) on the *y-z* plane, where *t* will be a function of the viewer's position and the *x*-coordinate of the point in three-dimensional space. Verify that $t = \frac{x_0}{x_0 - x}$ .

The following commands demonstrate this mapping by computing image points as specified after randomly generating points in the domain that lie between the observer and the *y-z* plane.

```
> restart:

  Random:=proc(a, b)
    evalf(rand(ceil(a*10^Digits)..floor(b*10^Digits))()/10^Digits);
  end:

> numberofpoints:=1000:
  print(`The number of domain and image points is`, numberofpoints);
  print(``);
  x0:=500:
  t:=(x,x0)->x0/(x0-x):
  newx:=(x,y,z)->0:
  newy:=(x,y,z)->t(x,x0)*y:
  newz:=(x,y,z)->t(x,x0)*z:
  domain:=[seq([Random(0,100), Random(-100,100), Random(-100,100)], i=1..numberofpoir
  image:=seq([newx(domain[i,1],domain[i,2],domain[i,3]),newy(domain[i,1],domain[i,2],dom
  [i,3]),newz(domain[i,1],domain[i,2],domain[i,3])], i=1..nops(domain)):
  print(`The first ten image points are:`);
  partimage:=[[`x-coord`,`y-coord`, `z-coord`],seq(image[i],i=1..10)]:
  matrix(partimage);
```

*The number of domain and image points is,* 1000

*The first ten image points are:*

| *x-coord* | *y-coord* | *z-coord* |
|---|---|---|
| 0 | 36.82621163 | -64.83627172 |
| 0 | 51.45256027 | 20.90575855 |
| 0 | -8.797423039 | -100.1167747 |
| 0 | 71.02496360 | -68.63719172 |
| 0 | -51.01277621 | 29.83121344 |
| 0 | -83.23317910 | 94.56664495 |
| 0 | 68.39612041 | 104.2513658 |
| 0 | 100.6193323 | 61.49784542 |
| 0 | 24.26498384 | 3.493372745 |
| 0 | 36.75900859 | -35.62974349 |

To view the points in the image, you need to first load in a graphics package.

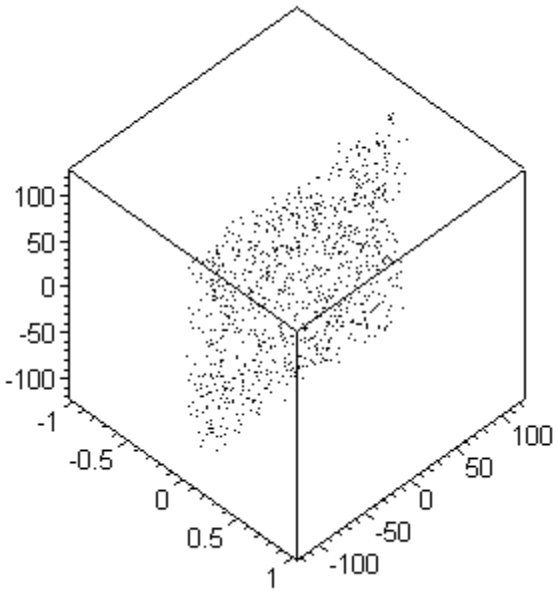
```
> with(plots):
  with(plottools):
```

Warning, the name changecoords has been redefined

Warning, the assigned name arrow now has a global binding

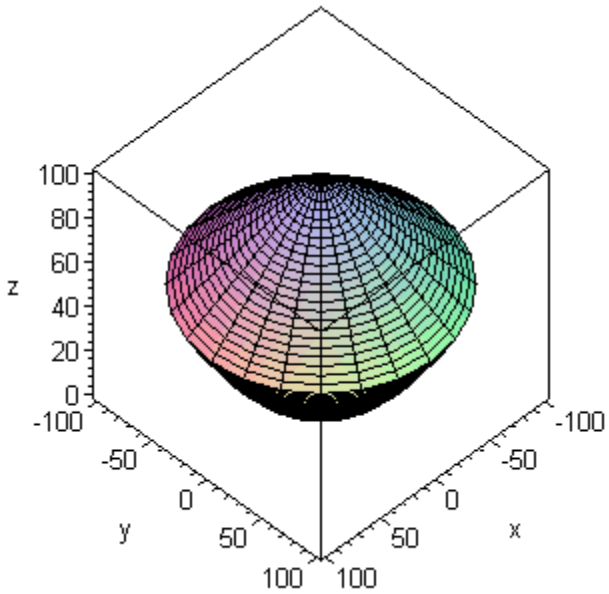
Once the graph is plotted, click and drag on the image to view it from different angles.

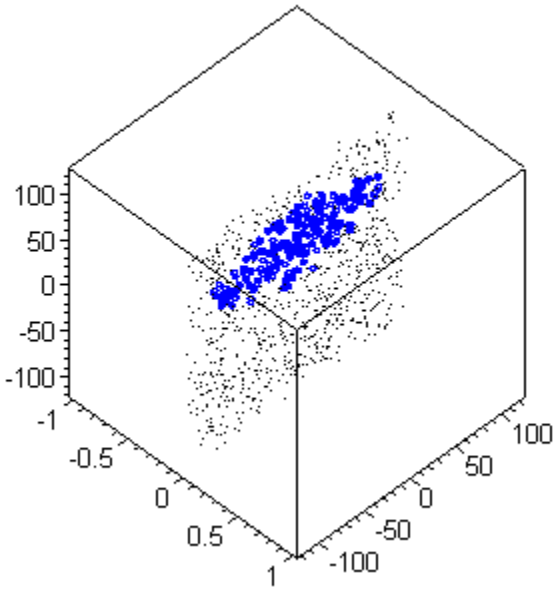
```
> sp:=pointplot3d([image], color=black, axes=boxed, orientation=[-45,45]):
  print(sp);
```




Let's suppose that there is an object in the region bounded approximately by $z = 100 - .5\sqrt{x^2+y^2}$ and $z = .5e\text{-}2\,(x^2+y^2)$ . We will first plot the object in three-dimensional space and then separate out the image points that have been mapped from that region and plot them in blue.

```
> p3:=cylinderplot(200-2*z, theta=0..2*Pi, z=50..100):
  p4:=cylinderplot(sqrt(200*z), theta=0..2*Pi, z=0..50, axes=boxed):
  display({p3,p4});
  listout:=[]:
  listin:=[]:
  for i from 1 to nops(domain) do
   if domain[i,3] < 100-.5*sqrt(domain[i,1]^2+domain[i,2]^2) and domain[i,3] >
      .005*(domain[i,1]^2+domain[i,2]^2) then
      listin:=[op(listin),[image[i,1],image[i,2],image[i,3]]]:
    else listout:=[op(listout),[image[i,1],image[i,2],image[i,3]]]:
   fi:
  od:
  pout:=pointplot3d(listout, color=black,axes=boxed):
  pin:=pointplot3d(listin, color=blue, axes=boxed, symbol=circle):
  display3d({pout,pin}, orientation=[-45,45]);
```




# You Try It: Part I

If you had a parallel projection onto the *y*-*z* plane, any (*x*, *y*, *z*) would map to (0, *y*, *z*). In the above problem, let $x_0$ get larger and larger, and compare your image points to your domain points to see if you are approaching this state. Do this by making the number in **x0** bigger and bigger.

```
> numberofpoints:=100:
  print(`The number of domain and image points is`, numberofpoints);
  print(``);
  x0:=10000:
```

```
t:=(x,x0)->x0/(x0-x):
newx:=(x,y,z)->0:
newy:=(x,y,z)->t(x,x0)*y:
newz:=(x,y,z)->t(x,x0)*z:
domain:=[seq([Random(0,100), Random(-100,100), Random(-100,100)], i=1..numberofpoir
image:=seq([newx(domain[i,1],domain[i,2],domain[i,3]),newy(domain[i,1],domain[i,2],dom
[i,3]),newz(domain[i,1],domain[i,2],domain[i,3])], i=1..nops(domain)):
print(`The first ten domain and image points are`);
temp1:=seq([[seq(domain[i,j],j=1..3)],[seq(image[i,j],j=1..3)]],i=1..10);
```

*The number of domain and image points is,* 100

*The first ten domain and image points are*

$$
\begin{aligned}
temp1 := &[[75.53010936, 58.13111223, 23.97365928], [0, 58.57351865, 24.15611065]],\\
&[[90.18637823, -51.52326014, -97.28296156], [0, -51.99215859, -98.16830597]],\\
&[[11.03742804, -81.86807260, 34.68734348], [0, -81.95853371, 34.72567168]],\\
&[[43.00027274, -18.03551349, 87.90264650], [0, -18.11340160, 88.28226261]],\\
&[[63.32382463, 93.08347924, 1.399919606], [0, 93.67667577, 1.408840925]],\\
&[[46.35251842, 18.23594292, -0.7398487794], [0, 18.32086473, -0.7432941346]],\\
&[[5.448976409, 84.39081970, -79.22877086], [0, 84.43682915, -79.27196599]],\\
&[[51.38064516, 84.59822827, 82.56680812], [0, 85.03514436, 82.99323274]],\\
&[[9.962420544, 81.24782966, 69.01100716], [0, 81.32885292, 69.07982742]],\\
&[[69.62462235, -87.77020435, 98.12970342], [0, -88.38558565, 98.81771805]]
\end{aligned}
$$

In the above matrix, each domain point is followed by its corresponding point in the image. Are your image values for $y$ and $z$ close to your domain values for $y$ and $z$ respectively?

# Part II: Parallel Mapping to Any Plane Using a Linear Transformation - Identifying the Image

We now consider another type of linear mapping onto a plane. Instead of projections from a single point, we consider parallel projections onto a plane. You could think of this as a generalization of the above case, where the beholder moves farther and farrther away ( ? ). Your domain is the set

of all points in three-dimensional space. Begin by finding the image for several points in your domain, and use any three of these distinct noncollinear points in your image to write the equation of the plane in which they lie. Verify that other image points you found satisfy the equation of this plane, and then plot the image to verify visually that the image is a plane.

The transformation described below is defined as follows:
newx = $x + y + 2z$

newy = $2x$ + y - z

newz = $3x + 2y + z$

Other transformations will work, provided that exactly one of the right-hand functions can be written as a linear combination of the other two. In this case, the top right function equals the third one minus the second one. This restriction guarantees that all your image points will lie in a plane.

Without loss of generality, we focus on domain points with integer coordinates between -100 and 100. We use integers to avoid making roundoff errors when we check to see that image points lie on the plane. One thousand image points are computed, and all are plotted, but only the first 10 are listed.

```
> numberofpoints:=1000:
  newx:=(x,y,z)->x+y+2*z:
  newy:=(x,y,z)->2*x+y-z:
  newz:=(x,y,z)->3*x+2*y+z:
  print(`The number of domain and image points is`, numberofpoints);
  print(``);
  a:=rand(-100..100):
  for i from 1 to numberofpoints do A[i]:=[a(),a(),a()] od:
  domain:=convert(A,list):
  image:=seq([newx(domain[i,1],domain[i,2],domain[i,3]),newy(domain[i,1],domain[i,2],dom
  [i,3]),newz(domain[i,1],domain[i,2],domain[i,3])], i=1..nops(domain)):
  print(`The first ten image points are:`);
  print(``);
  partimage:=[[`x-coord`,`y-coord`, `z-coord`],seq(image[i],i=1..10)]:
  matrix(partimage);
```

*The number of domain and image points is,* 1000

*The first ten image points are:*

| x-coord | y-coord | z-coord |
|---|---|---|
| -24 | -206 | -230 |
| 154 | 39 | 193 |
| -146 | -10 | -156 |
| 197 | -33 | 164 |
| 151 | -15 | 136 |
| 212 | 209 | 421 |
| -44 | -279 | -323 |
| 165 | -68 | 97 |
| 165 | -49 | 116 |
| 161 | -15 | 146 |

To determine the equation of the plane in which the first three image points lie, we specify two vectors in the plane and compute their cross product. Knowing that one point in the image is [0, 0, 0] guarantees that the plane passes through the origin.

```
> vector1:=image[1]-image[2]:
  vector2:=image[1]-image[3]:
  norm1:=linalg[crossprod](vector1, vector2):
  norm2:=[norm1[1]/abs(norm1[1]),norm1[2]/abs(norm1[2]),norm1[3]/abs(norm1[3])]:
  print(`The normal to the plane is`, norm2);
  normaltoplane:=norm2[1]*x+ norm2[2]*y+ norm2[3]*z=0:
  print(`The equation of the plane onto which the points are projected is`, normaltoplane);
```

*The normal to the plane is,* [-1, -1, 1]

*The equation of the plane onto which the points are projected is,* $-x - y + z = 0$

Check to see if all points you found in the domain satisfy the equation of the plane. We start our iterator at 0 and then increase it by one eeach time an image point satisfies the equation of the plane.

```
> works:=0:
  for i from 1 to numberofpoints do
   if lhs(eval(normaltoplane, {x=image[i][1], y=image[i][2], z=image[i][3]}) = rhs(normaltopl
  then
   works:=works+1:
  fi:
  od:
  works;
```

1000

It looks as though, as predicted, every point in the image lies on the specified plane.

# You Try It: Part II

The transformation described below is defined as follows:

newx = $2x + 2y + z$

newy = $4x + 4y - 7z$

newz = $3x + 3y + 9z$

Chang the appropriate expression

```
> numberofpoints:=1000:
  newx:=(x,y,z)->2*x+2*y+z:
  newy:=(x,y,z)->4*x+4*y-7*z:
  newz:=(x,y,z)->3*x+3*y+9*z:
  print(`The number of domain and image points is`, numberofpoints);
  print(``);
  a:=rand(-100..100):
  for i from 1 to numberofpoints do A[i]:=[a(),a(),a()] od:
  domain:=convert(A,list):
  image:=seq([newx(domain[i,1],domain[i,2],domain[i,3]),newy(domain[i,1],domain[i,2],dom
  [i,3]),newz(domain[i,1],domain[i,2],domain[i,3])], i=1..nops(domain)):
  print(`The first ten image points are:`);
  print(``);
  partimage:=[[`x-coord`,`y-coord`, `z-coord`],seq(image[i],i=1..10)]:
  matrix(partimage);
```

*The number of domain and image points is,* 1000

*The first ten image points are:*

| x-coord | y-coord | z-coord |
|---|---|---|
| 228 | 492 | 312 |
| -135 | -117 | -330 |
| 2 | -644 | 543 |
| 61 | 437 | -171 |
| 128 | -320 | 672 |
| 192 | 1068 | -282 |
| -303 | -1209 | 48 |
| 128 | -392 | 732 |
| -42 | -966 | 672 |
| 232 | 626 | 213 |

Find the plane in which the image points lie.

```
> vector1:=image[1]-image[2]:
  vector2:=image[1]-image[3]:
  norm1:=linalg[crossprod](vector1, vector2):
  print(`The normal to the plane is`, norm1);
  normaltoplane:=norm1[1]*x+ norm1[2]*y+ norm1[3]*z=0:
  print(`The equation of the plane onto which the points are projected is`, normaltoplane);
```

*The normal to the plane is,* [-869991, 228945, 274734]

*The equation of the plane onto which the points are projected is,* $-869991\,x + 228945\,y + 274734$

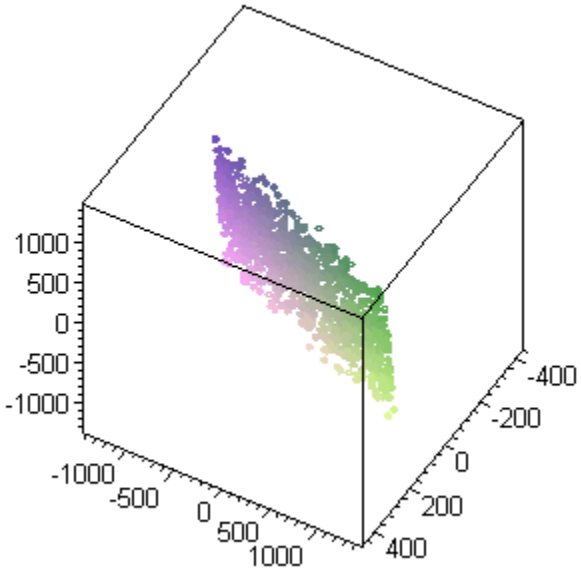
Check to see if all points you found in the domain satisfy the equation of the plane. Start our iterator at 0, and then increase it by one each time an image point satisfies the equation of the plane.

```
> works:=0:
  for i from 1 to numberofpoints do
    if lhs(eval(normaltoplane, {x=image[i][1], y=image[i][2], z=image[i][3]}) = rhs(normaltopl
  then
    works:=works+1:
  fi:
  od:
  works;
```

1000

Does every point in the image lies on the plane specified?

> **pointplot3d([image], axes=boxed, symbol=circle, orientation=[30,45]);**




# Part III: Analyzing the Mapping

We return to the first mapping defined in Part II. Identify the points in the domain that map to the origin $O(0,0,0)$, and verify that they all lie along the same line. Write the equatins for this line.

> **newx:=(x,y,z)->x+y+2*z:**
> **newy:=(x,y,z)->2*x+y-z:**
> **newz:=(x,y,z)->3*x+2*y+z:**
> **unassign('x,y,z');**
> **solve({newx(x,y,z), newy(x,y,z), newz(x,y,z)},{x,y,z});**

$$\{x = 3z, z = z, y = -5z\}$$

Now select another point in the image, say $P\{10,-20,-10\}$, and find the domain points that map to $P$. Note that they all fall along a line and observe how this line is related to the line, of points that map to the origin.

> **solve({newx(x,y,z)=10, newy(x,y,z)=-20, newz(x,y,z)=-10},{x,y,z});**

$$\{y = 40 - 5z, x = -30 + 3z, z = z\}$$

Check this out for another point in the image $Q(-40,70,30)$, and see if you can make a

generalization about the mapping in terms of sets of points in the domain that map to a given point in the image.

```
> solve({newx(x,y,z)=-40, newy(x,y,z)=70, newz(x,y,z)=30},{x,y,z});
```

$$\{x = 110 + 3z, y = -150 - 5z, z = z\}$$

# Part IV: Application to Medical X-Ray and CAT-Scan Technologies

We will now demonstrat how points within a tumor would show on an X-ray slide.

```
> unassign('x,y,z,image');
  numberofpoints:=1000:
  newx:=(x,y,z)->x+y+2*z:
  newy:=(x,y,z)->2*x+y-z:
  newz:=(x,y,z)->3*x+2*y+z:
  print(`The number of domain and image points is`, numberofpoints);
  print(``);
  a:=rand(-100..100):
  for i from 1 to 1000 do A[i]:=[a(),a(),a()] od:
  domain:=convert(A,list):
  image:=seq([newx(domain[i,1],domain[i,2],domain[i,3]),newy(domain[i,1],domain[i,2],dom
  [i,3]),newz(domain[i,1],domain[i,2],domain[i,3])], i=1..nops(domain)):
  print(`The first ten image points are:`);
  print(``);
  partimage:=[[`x-coord`,`y-coord`, `z-coord`],seq(image[i],i=1..10)]:
  matrix(partimage);
```

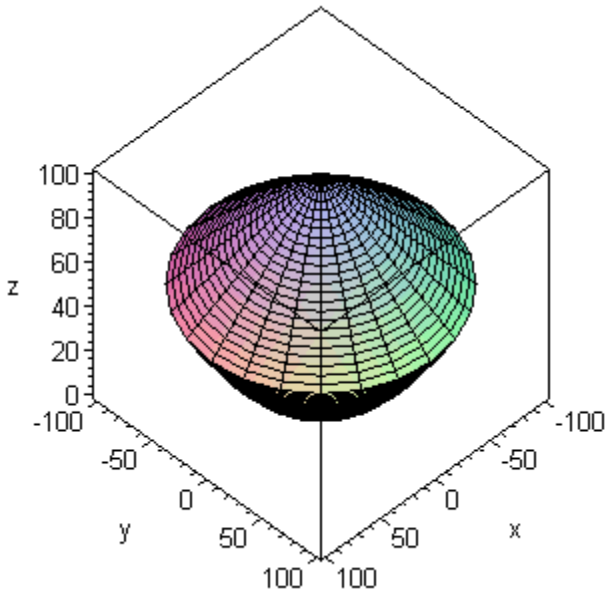
*The number of domain and image points is,* 1000

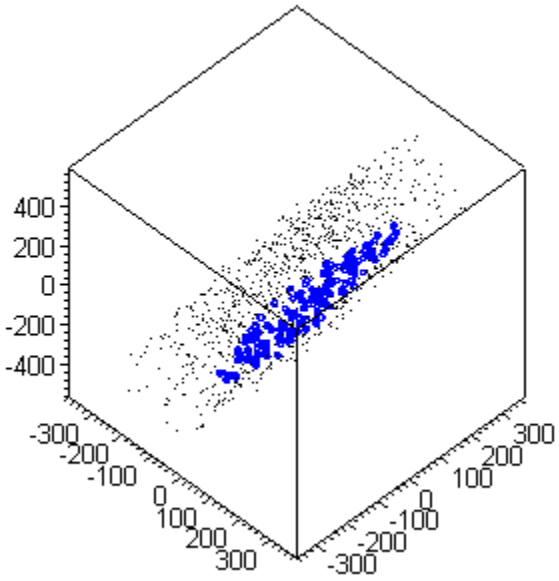
*The first ten image points are:*

| x-coord | y-coord | z-coord |
|---|---|---|
| 223 | 9 | 232 |
| 50 | 119 | 169 |
| -174 | -172 | -346 |
| 178 | -129 | 49 |
| 51 | -229 | -178 |
| -123 | 214 | 91 |
| 114 | 196 | 310 |
| 187 | 75 | 262 |
| 61 | -149 | -88 |
| -154 | -45 | -199 |

Let's suppose that there is a tumor in the region bounded approximately by z = 100 - $.5\sqrt{x^2+y^2}$ and $z = .5e\text{-}2\,(x^2+y^2)$ . We will first plot the object in three-space and then separate out the image points that have been mapped to that region and plot them in green. We must first load in two graphing packages if we have not already done so.

```
> unassign('x,y');
  display({p3,p4});
  listout:=[]:
  listin:=[]:
  for i from 1 to nops(domain) do
   if (domain[i,3] < evalf(100-.5*sqrt(domain[i,1]^2+domain[i,2]^2)) and domain[i,3] >
     evalf(.005*(domain[i,1]^2+domain[i,2]^2))) then
     listin:=[op(listin),[image[i,1],image[i,2],image[i,3]]]:
   else
     listout:=[op(listout),[image[i,1],image[i,2],image[i,3]]]:
   fi:
  od:
  pout:=pointplot3d(listout, color=black,axes=boxed):
  pin:=pointplot3d(listin, color=blue, axes=boxed, symbol=circle):
  display3d({pout,pin}, orientation=[-45,45]);
```




Suppose, for example, that a spot is found at the image point $T1(10,20,30)$. What does that tell you about the location of the possible growth? Based on this image, you cannot precisely identify the placement of the tumor because this mapping is NOT one to one.

> **maptoT1:=solve({newx(x,y,z)=10, newy(x,y,z)=20, newz(x,y,z)=30},{x,y,z});**
> **assign(%);**
> **x1:=x; y1:=y; z1:=z;**

$$maptoT1 := \{y = -\frac{5x}{3} + \frac{50}{3}, z = \frac{x}{3} - \frac{10}{3}, x = x\}$$

$$x1 := x$$

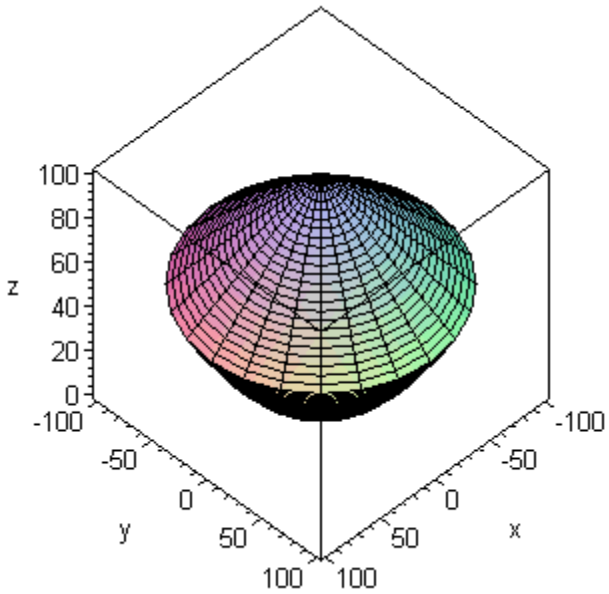
$$yl := -\frac{5x}{3} + \frac{50}{3}$$

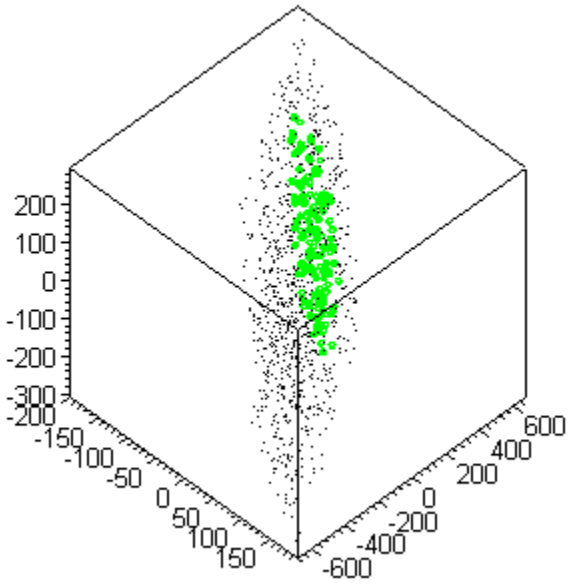
$$zl := \frac{x}{3} - \frac{10}{3}$$

Since this gives you some but not enough information, you must take another X-ray from a different angle. An example is given and you could verify that the points all map to a plane.

```
> unassign('x,y,z,image');
  numberofpoints:=1000:
  newx:=(x,y,z)->-x+y:
  newy:=(x,y,z)->2*x-3*y+2*z:
  newz:=(x,y,z)->-y+2*z:
  print(`The number of domain and image points is`, numberofpoints);
  print(``);
  a:=rand(-100..100):
  for i from 1 to 1000 do A[i]:=[a(),a(),a()] od:
  domain:=convert(A,list):
  image:=seq([newx(domain[i,1],domain[i,2],domain[i,3]),newy(domain[i,1],domain[i,2],dom
  [i,3]),newz(domain[i,1],domain[i,2],domain[i,3])], i=1..nops(domain)):
  display({p3,p4});
  listout:=[]:
  listin:=[]:
  for i from 1 to nops(domain) do
   if (domain[i,3] < evalf(100-.5*sqrt(domain[i,1]^2+domain[i,2]^2)) and domain[i,3] >
      evalf(.005*(domain[i,1]^2+domain[i,2]^2))) then
      listin:=[op(listin),[image[i,1],image[i,2],image[i,3]]]:
    else
      listout:=[op(listout),[image[i,1],image[i,2],image[i,3]]]:
   fi:
  od:
  pout:=pointplot3d(listout, color=black,axes=boxed):
  pin:=pointplot3d(listin, color=green, axes=boxed, symbol=circle):
  display3d({pout,pin}, orientation=[-45,45]);
```

*The number of domain and image points is,* 1000




If you were somehow able to detect that the SAME spot now shows up on the image at *T*2(-90,250,70), what would that tell you about the possible location of the tumor? As before, you can get information on the point in the domain that mapped to this image point.

```
> maptoT2:=solve({newx(x,y,z)=-90, newy(x,y,z)=250, newz(x,y,z)=70},{x,y,z});
  assign(maptoT2);
  x2:=x;y2:=y;z2:=z;
  eqn2:=x2+y2+z2:
```

$$maptoT2 := \{z = z, x = 20 + 2\,z, y = -70 + 2\,z\}$$

$$x2 := 20 + 2z$$

$$y2 := -70 + 2z$$

$$z2 := z$$

If we put this information together with the previous information, we can locate the point on the organ identified. Essentially, we are finding the intersection of two straight lines in three-dimensional space.

NOTE: In this next solve command, you may need to change the variable for which you solve (currently x) to y or z, depending on how the points (x1, y1, z1) and (x2, y2, z2) were solved.

```
> sol:=solve({x1=x2, y1=y2},{x});
```

$$sol :=$$

Let's put this into the specifications for the second line we found.

```
> print(`x=`,eval(x2, sol));
  print(`y=`, rhs(sol[1]));
  print(`z=`,eval(z2,sol));
```

Error, invalid input: eval received NULL, which is not valid for its 2nd argument, eqns

Error, invalid subscript selector

Error, invalid input: eval received NULL, which is not valid for its 2nd argument, eqns

This specifies one of the points in the three-dimensional object where the tumor is present.

You should notice in the analysis how difficult it is to identify the point in the second mapping that is the image of the SAME point as the one we first found. Because of this problem, many more than simply two X-rays need to be taken. To determine the precise location of a tumour the technology that makes this process feasible is called a CT-Scan. Commonly known as the CAT-Scan. The procedure above reflects the fundamental mathematics used to interpret CAT-Scans.

# Part V: Linear Algebra Approach (optional; not in calculus text)

For this section, we load in *Maple's* linear algebra **linalg** package.

```
> restart:
  with(LinearAlgebra):
```

```
> M:=<<1,1,2>|<2,1,-1>|<3,2,1>>;
```

```
eigval:=[evalf(Eigenvalues(M))];
eigvec:=[evalf(Eigenvectors(M))];
```

$$M := \begin{bmatrix} 1 & 2 & 3 \\ 1 & 1 & 2 \\ 2 & -1 & 1 \end{bmatrix}$$

$$eigval := \left[\begin{bmatrix} 0. \\ 3.791287848 \\ -0.791287848 \end{bmatrix}\right]$$

$$eigvec := \left[\begin{bmatrix} 3.791287848 \\ -0.791287848 \\ 0. \end{bmatrix}, \begin{bmatrix} 2.136634176 & -1.136634176 & -1. \\ 1.481980506 & -0.4819805063 & -1. \\ 1. & 1. & 1. \end{bmatrix}\right]$$

Note the presence of the 0 eigenvalue.

```
> unassign('a,b,c'):
  kernel:=<a,b,c>:
  mat:=Multiply(M, kernel):
  solve({mat[1], mat[2], mat[3]},{a,b,c});
  assign(%):
  bk:=b: ck:=c: ak:=a:
```

$$\{b = b, a = b, c = -b\}$$

If $c = 1$, note how this vector compares to the eigenvector associated with the eigenvalue of 0.

```
> c:=1;
  [ak,bk,c];
```

$$c := 1$$

$$[b, b, 1]$$

You can find the normal to the plane formed by the image by taking the cross product of the eigenvectors associated with the nonzero eigenvalues.

```
> cross1:=<2.136634176,1.481980506,1>;
  cross2:=<-1.36634176,-.4819805063,1>;
  norm1:=CrossProduct(cross1,cross2);
```

$$cross1 := \begin{bmatrix} 2.136634176 \\ 1.481980506 \\ 1 \end{bmatrix}$$

$$cross2 := \begin{bmatrix} -1.36634176 \\ -0.4819805063 \\ 1 \end{bmatrix}$$

$$norm1 := \begin{bmatrix} 1.963961012 \\ -3.502975936 \\ 0.995075831 \end{bmatrix}$$

>

As you did above, you can check to see if all the points in the image lie in the plane.

What do you suppose the existence of a 0 eigenvalue has to do with the fact that the image points all lie in a plane, even though the domain consists of points in three dimensions?