Subject: Re: Shear transformation with Poly_2d
Posted by Craig Markwardt on Mon, 18 Jun 2012 19:17:22 GMT
View Forum Message <> Reply to Message

On Monday, June 18, 2012 11:15:48 AM UTC-4, Helder wrote:
> Hi,
> I'm not going to post a question, rather a solution... The reason is that I was fighting with this a few hours and I thought it would be nice to show a solution to anybody who might care about this.
> I was a bit annoyed that of the affine transformations (scaling, rotation, translation and shear) one is missing... The transformations like scaling and rotation can be accessed using ROT that then calling POLY_2D that actually does the work. Translation can be done (easily) with shift (although this is only possible using integer translation, for non integer translations, the procedure described below may also be used...).
> What is missing is of course shear. If you look in text books you find a matrix for shear that looks like:
> ShearVertical = [[1,0,0],[Vert,1,0],[0,0,1]]
> where Vert is the degree of vertical shear.
> I thought it would be nicest to get this sorted by using this matrix, but I just couldn't get this to work without loops or having to rewrite basic math code... not nice and given that IDL is well suited for image processing I thought that there has to be an IDL way for this.
>
> So here we go. POLY_2D give in the help indications on how to use it and some examples (of course not shear).
> If you want to implement a shear transformation, then you would have to do the following.

```
>
> Img = DIST(200)
> WINDOW, XSIZE=405,YSIZE=200
> TVSCL,Img
> VertShear = 20.0
> KX = [[0.0, 0.0],[1.0, 0.0]]
> KY = [[VertShear, 1.0],[-VertShear/100.0, 0.0]]
> TVSCL,POLY_2D(img,KX, KY,cubic=-0.5), 205, 0, /DEVICE
>
```

> This works fine if you want to shear the image in the middle. If you want shear somewhere else, I could only manage that using POLYWARP.
> In this case, you set some coefficients that act as transformation points (4 is the minimum number of points):

```
>
> s = SIZE(Img,/DIMENSIONS)
> Offset = [50,0]          ;In pixels
> XI = [0, 0, s[0]-1, s[0]-1]+Offset[0]
> YI = [0, s[1]-1, 0, s[1]-1]+Offset[1]
> XO = XI
> VertShear = 20.0
> YO = YI + [-VertShear, -VertShear, VertShear, VertShear]
>
```

> Then with POLYWARP you can retrieve the coefficients KX and KY and use them as above.
>

> POLYWARP, XI, YI, XO, YO, 1, KX, KY
> TVSCL,POLY_2D(Img,KX, KY,cubic=-0.5), 205, 0, /DEVICE
>
> The shearing point is now moved 50 pixel to the right (positive).
> There is probably an easy way to obtain these coefficients in a smooth mathematical way... Happy to hear suggestions!

I think the standard way to implement a shear or rotation about a different origin, is to compose a series of transformations: translate, shear, untranslate.

You can't do translations with a simple matrix transformation. A "translation matrix" can be designed by using an (N+1)x(N+1) matrix and using homogeneous coordinates.
http://en.wikipedia.org/wiki/Translation_%28geometry%29
Not that this is more efficient, but sometimes it can simplify the notation.

Craig

---

## Subject: Re: Shear transformation with Poly_2d
Posted by Helder Marchetto on Tue, 19 Jun 2012 12:52:12 GMT
View Forum Message <> Reply to Message

On Monday, June 18, 2012 9:17:22 PM UTC+2, Craig Markwardt wrote:
> On Monday, June 18, 2012 11:15:48 AM UTC-4, Helder wrote:
>> Hi,
>> I'm not going to post a question, rather a solution... The reason is that I was fighting with this a few hours and I thought it would be nice to show a solution to anybody who might care about this.
>> I was a bit annoyed that of the affine transformations (scaling, rotation, translation and shear) one is missing... The transformations like scaling and rotation can be accessed using ROT that then calling POLY_2D that actually does the work. Translation can be done (easily) with shift (although this is only possible using integer translation, for non integer translations, the procedure described below may also be used...).
>> What is missing is of course shear. If you look in text books you find a matrix for shear that looks like:
>> ShearVertical = [[1,0,0],[Vert,1,0],[0,0,1]]
>> where Vert is the degree of vertical shear.
>> I thought it would be nicest to get this sorted by using this matrix, but I just couldn't get this to work without loops or having to rewrite basic math code... not nice and given that IDL is well suited for image processing I thought that there has to be an IDL way for this.
>>
>> So here we go. POLY_2D give in the help indications on how to use it and some examples (of course not shear).
>> If you want to implement a shear transformation, then you would have to do the following.
>>
>> Img = DIST(200)
>> WINDOW, XSIZE=405,YSIZE=200
>> TVSCL,Img
>> VertShear = 20.0

>>  KX = [[0.0, 0.0],[1.0, 0.0]]
>>  KY = [[VertShear, 1.0],[-VertShear/100.0, 0.0]]
>>  TVSCL,POLY_2D(img,KX, KY,cubic=-0.5), 205, 0, /DEVICE
>>
>>  This works fine if you want to shear the image in the middle. If you want shear somewhere else, I could only manage that using POLYWARP.
>>  In this case, you set some coefficients that act as transformation points (4 is the minimum number of points):
>>
>>  s = SIZE(Img,/DIMENSIONS)
>>  Offset = [50,0]          ;In pixels
>>  XI = [0, 0, s[0]-1, s[0]-1]+Offset[0]
>>  YI = [0, s[1]-1, 0, s[1]-1]+Offset[1]
>>  XO = XI
>>  VertShear = 20.0
>>  YO = YI + [-VertShear, -VertShear, VertShear, VertShear]
>>
>>  Then with POLYWARP you can retrieve the coefficients KX and KY and use them as above.
>>
>>  POLYWARP, XI, YI, XO, YO, 1, KX, KY
>>  TVSCL,POLY_2D(Img,KX, KY,cubic=-0.5), 205, 0, /DEVICE
>>
>>  The shearing point is now moved 50 pixel to the right (positive).
>>  There is probably an easy way to obtain these coefficients in a smooth mathematical way... Happy to hear suggestions!
>
>  I think the standard way to implement a shear or rotation about a different origin, is to compose a series of transformations: translate, shear, untranslate.
>
>  You can't do translations with a simple matrix transformation.  A "translation matrix" can be designed by using an (N+1)x(N+1) matrix and using homogeneous coordinates.
>    http://en.wikipedia.org/wiki/Translation_%28geometry%29
>  Not that this is more efficient, but sometimes it can simplify the notation.
>
>  Craig

Yes, you are absolutely right. However, when dealing with affine transformations the translation is also included as an "affine matrix":
Translation = [[1,0,0],[0,1,0],[tx,ty,1]]
where an affine transformation transforms spatial coordinates (v,w) of the source image to (x,y) in the transformed image. The notation goes like:
[x,y,1] = [v,w,1] TransfMatrix
I have taken my geometry classes a long time ago, but I suppose that in this case the pixel coordinates are two and the matrix has three components, therefore fitting to what you said about using a (N+1)x(N+1) transformation matrix for translations (the last column and last row are all zeros and a one on the diagonal for all transformations except for the translation where numbers come in...). Another way of describing this is saying that affine transformations are linear transformations followed by a translation.

I took the above transformation matrix treatment from Digital Image processing of Gonzales/Woods.

I find it a pity that there is no clear/direct way to implement shear transformations in IDL. As I said, it is possible to do rotation, scaling and translation (with some limitations...), but not shear. I thought that they would be all fit in the same chapter. Even nicer would be the possibility to feed in transformation matrices directly. One could then combine the matrices accordingly and perform the transformation only once. If, for instance, I would like to first rotate and then translate, I would calculate the matrix and then perform the operation, saving computation time (1 matrix calculation and 1 transformation).

At the moment I'm busy with something else, but I'll check if effectively the image is also translated when doing the shear rotation as describe above. So far I have the feeling that this is working, but I will cross check by doing separating the two operations and comparing results.

Helder

---

## Subject: Re: Shear transformation with Poly_2d
Posted by lecacheux.alain on Tue, 19 Jun 2012 15:54:22 GMT
View Forum Message <> Reply to Message

On 19 juin, 14:52, Helder <hel...@marchetto.de> wrote:
>  On Monday, June 18, 2012 9:17:22 PM UTC+2, Craig Markwardt wrote:
>>  On Monday, June 18, 2012 11:15:48 AM UTC-4, Helder wrote:
>>>  Hi,
>>>  I'm not going to post a question, rather a solution... The reason is that I was fighting with this a few hours and I thought it would be nice to show a solution to anybody who might care about this.
>>>  I was a bit annoyed that of the affine transformations (scaling, rotation, translation and shear) one is missing... The transformations like scaling and rotation can be accessed using ROT that then calling POLY_2D that actually does the work. Translation can be done (easily) with shift (although this is only possible using integer translation, for non integer translations, the procedure described below may also be used...).
>>>  What is missing is of course shear. If you look in text books you find a matrix for shear that looks like:
>>>  ShearVertical = [[1,0,0],[Vert,1,0],[0,0,1]]
>>>  where Vert is the degree of vertical shear.
>>>  I thought it would be nicest to get this sorted by using this matrix, but I just couldn't get this to work without loops or having to rewrite basic math code... not nice and given that IDL is well suited for image processing I thought that there has to be an IDL way for this.
>
>>>  So here we go. POLY_2D give in the help indications on how to use it and some examples (of course not shear).
>>>  If you want to implement a shear transformation, then you would have to do the following.
>
>>>  Img = DIST(200)
>>>  WINDOW, XSIZE=405,YSIZE=200

```
>>>  TVSCL,Img
>>>  VertShear = 20.0
>>>  KX = [[0.0, 0.0],[1.0, 0.0]]
>>>  KY = [[VertShear, 1.0],[-VertShear/100.0, 0.0]]
>>>  TVSCL,POLY_2D(img,KX, KY,cubic=-0.5), 205, 0, /DEVICE
>
```

>>>  This works fine if you want to shear the image in the middle. If you want shear somewhere else, I could only manage that using POLYWARP.
>>>  In this case, you set some coefficients that act as transformation points (4 is the minimum number of points):
>

```
>>>  s = SIZE(Img,/DIMENSIONS)
>>>  Offset = [50,0]          ;In pixels
>>>  XI = [0, 0, s[0]-1, s[0]-1]+Offset[0]
>>>  YI = [0, s[1]-1, 0, s[1]-1]+Offset[1]
>>>  XO = XI
>>>  VertShear = 20.0
>>>  YO = YI + [-VertShear, -VertShear, VertShear, VertShear]
>
```

>>>  Then with POLYWARP you can retrieve the coefficients KX and KY and use them as above.
>

```
>>>  POLYWARP, XI, YI, XO, YO, 1, KX, KY
>>>  TVSCL,POLY_2D(Img,KX, KY,cubic=-0.5), 205, 0, /DEVICE
>
```

>>>  The shearing point is now moved 50 pixel to the right (positive).
>>>  There is probably an easy way to obtain these coefficients in a smooth mathematical way... Happy to hear suggestions!
>
>>  I think the standard way to implement a shear or rotation about a different origin, is to compose a series of transformations: translate, shear, untranslate.
>
>>  You can't do translations with a simple matrix transformation.  A "translation matrix" can be designed by using an (N+1)x(N+1) matrix and using homogeneous coordinates.
>>  http://en.wikipedia.org/wiki/Translation_%28geometry%29
>>  Not that this is more efficient, but sometimes it can simplify the notation.
>
>>  Craig
>
>  Yes, you are absolutely right. However, when dealing with affine transformations the translation is also included as an "affine matrix":
>  Translation = [[1,0,0],[0,1,0],[tx,ty,1]]
>  where an affine transformation transforms spatial coordinates (v,w) of the source image to (x,y) in the transformed image. The notation goes like:
>  [x,y,1] = [v,w,1] TransfMatrix
>  I have taken my geometry classes a long time ago, but I suppose that in this case the pixel coordinates are two and the matrix has three components, therefore fitting to what you said about using a (N+1)x(N+1) transformation matrix for translations (the last column and last row are all zeros and a one on the diagonal for all transformations except for the translation where numbers

come in...). Another way of describing this is saying that affine transformations are linear transformations followed by a translation.
> I took the above transformation matrix treatment from Digital Image processing of Gonzales/Woods.
>
> I find it a pity that there is no clear/direct way to implement shear transformations in IDL. As I said, it is possible to do rotation, scaling and translation (with some limitations...), but not shear. I thought that they would be all fit in the same chapter. Even nicer would be the possibility to feed in transformation matrices directly. One could then combine the matrices accordingly and perform the transformation only once. If, for instance, I would like to first rotate and then translate, I would calculate the matrix and then perform the operation, saving computation time (1 matrix calculation and 1 transformation).
>
> At the moment I'm busy with something else, but I'll check if effectively the image is also translated when doing the shear rotation as describe above. So far I have the feeling that this is working, but I will cross check by doing separating the two operations and comparing results.
>
> Helder
>
>

Following what Craig said, look at the T3D routine (Direct graphics)
or equivalent functions in NG, and use homogeneous coordinates.
While keywords for rotation, scaling and translation matrices are
prebuilt, you also can define your own transformation trough MATRIX
keyword, and realize whatever transform you want. The transform matrix
of a shear transformation is characterized by the addition of a
multiple of one row or column to another. You can easily build it from
the identity matrix (function IDENTITY in IDL) and replacing one of
the zero elements by a non zero value.
alx.

---

## Subject: Re: Shear transformation with Poly_2d
## Posted by Helder Marchetto on Thu, 21 Jun 2012 08:31:05 GMT
View Forum Message <> Reply to Message

On Tuesday, June 19, 2012 5:54:22 PM UTC+2, alx wrote:
> On 19 juin, 14:52, Helder <hel...@marchetto.de> wrote:
>> On Monday, June 18, 2012 9:17:22 PM UTC+2, Craig Markwardt wrote:
>>> On Monday, June 18, 2012 11:15:48 AM UTC-4, Helder wrote:
>>>> Hi,
>>>> I'm not going to post a question, rather a solution... The reason is that I was fighting with this a few hours and I thought it would be nice to show a solution to anybody who might care about this.
>>>> I was a bit annoyed that of the affine transformations (scaling, rotation, translation and shear) one is missing... The transformations like scaling and rotation can be accessed using ROT that then calling POLY_2D that actually does the work. Translation can be done (easily) with shift

(although this is only possible using integer translation, for non integer translations, the procedure described below may also be used...).
>>>> What is missing is of course shear. If you look in text books you find a matrix for shear that looks like:
>>>> ShearVertical = [[1,0,0],[Vert,1,0],[0,0,1]]
>>>> where Vert is the degree of vertical shear.
>>>> I thought it would be nicest to get this sorted by using this matrix, but I just couldn't get this to work without loops or having to rewrite basic math code... not nice and given that IDL is well suited for image processing I thought that there has to be an IDL way for this.
>>
>>>> So here we go. POLY_2D give in the help indications on how to use it and some examples (of course not shear).
>>>> If you want to implement a shear transformation, then you would have to do the following.
>>
```
>>>> Img = DIST(200)
>>>> WINDOW, XSIZE=405,YSIZE=200
>>>> TVSCL,Img
>>>> VertShear = 20.0
>>>> KX = [[0.0, 0.0],[1.0, 0.0]]
>>>> KY = [[VertShear, 1.0],[-VertShear/100.0, 0.0]]
>>>> TVSCL,POLY_2D(img,KX, KY,cubic=-0.5), 205, 0, /DEVICE
```
>>
>>>> This works fine if you want to shear the image in the middle. If you want shear somewhere else, I could only manage that using POLYWARP.
>>>> In this case, you set some coefficients that act as transformation points (4 is the minimum number of points):
>>
```
>>>> s = SIZE(Img,/DIMENSIONS)
>>>> Offset = [50,0]          ;In pixels
>>>> XI = [0, 0, s[0]-1, s[0]-1]+Offset[0]
>>>> YI = [0, s[1]-1, 0, s[1]-1]+Offset[1]
>>>> XO = XI
>>>> VertShear = 20.0
>>>> YO = YI + [-VertShear, -VertShear, VertShear, VertShear]
```
>>
>>>> Then with POLYWARP you can retrieve the coefficients KX and KY and use them as above.
>>
```
>>>> POLYWARP, XI, YI, XO, YO, 1, KX, KY
>>>> TVSCL,POLY_2D(Img,KX, KY,cubic=-0.5), 205, 0, /DEVICE
```
>>
>>>> The shearing point is now moved 50 pixel to the right (positive).
>>>> There is probably an easy way to obtain these coefficients in a smooth mathematical way... Happy to hear suggestions!
>>
>>> I think the standard way to implement a shear or rotation about a different origin, is to compose a series of transformations: translate, shear, untranslate.
>>

>>> You can't do translations with a simple matrix transformation.  A "translation matrix" can be designed by using an (N+1)x(N+1) matrix and using homogeneous coordinates.
>>> http://en.wikipedia.org/wiki/Translation_%28geometry%29
>>> Not that this is more efficient, but sometimes it can simplify the notation.
>>
>>> Craig
>>
>> Yes, you are absolutely right. However, when dealing with affine transformations the translation is also included as an "affine matrix":
>> Translation = [[1,0,0],[0,1,0],[tx,ty,1]]
>> where an affine transformation transforms spatial coordinates (v,w) of the source image to (x,y) in the transformed image. The notation goes like:
>> [x,y,1] = [v,w,1] TransfMatrix
>> I have taken my geometry classes a long time ago, but I suppose that in this case the pixel coordinates are two and the matrix has three components, therefore fitting to what you said about using a (N+1)x(N+1) transformation matrix for translations (the last column and last row are all zeros and a one on the diagonal for all transformations except for the translation where numbers come in...). Another way of describing this is saying that affine transformations are linear transformations followed by a translation.
>> I took the above transformation matrix treatment from Digital Image processing of Gonzales/Woods.
>>
>> I find it a pity that there is no clear/direct way to implement shear transformations in IDL. As I said, it is possible to do rotation, scaling and translation (with some limitations...), but not shear. I thought that they would be all fit in the same chapter. Even nicer would be the possibility to feed in transformation matrices directly. One could then combine the matrices accordingly and perform the transformation only once. If, for instance, I would like to first rotate and then translate, I would calculate the matrix and then perform the operation, saving computation time (1 matrix calculation and 1 transformation).
>>
>> At the moment I'm busy with something else, but I'll check if effectively the image is also translated when doing the shear rotation as describe above. So far I have the feeling that this is working, but I will cross check by doing separating the two operations and comparing results.
>>
>> Helder
>>
>>
>
> Following what Craig said, look at the T3D routine (Direct graphics)
> or equivalent functions in NG, and use homogeneous coordinates.
> While keywords for rotation, scaling and translation matrices are
> prebuilt, you also can define your own transformation trough MATRIX
> keyword, and realize whatever transform you want. The transform matrix
> of a shear transformation is characterized by the addition of a
> multiple of one row or column to another. You can easily build it from
> the identity matrix (function IDENTITY in IDL) and replacing one of
> the zero elements by a non zero value.
> alx.

Hi Alx,
I already tried once to use the T3D routine, but I found this not ideal for my purposes.
The reason is the following. If you have an image, you define the intensity for each given pixel as I(x,y). The transformation matrix will allow you to find the new coordinates (x',y') that will have the same intensity of the original point I(x,y) = I(x',y'). The problem is that (except for trivial cases) the transformation will not match with an integer value and the new intensity in the new image has to be interpolated using either nearest-neighbor, bilinear or cubic methods.
In other words, I would only find out where the new coordinate points would be by first generating the coordinates of the x and y points:
[following Craig's method for generating coordinates]
x = findgen(N)*0.1 - 5.
y = x
xx = x # (y*0 + 1)
yy = (x*0 + 1) # y)
But then the transformation would "only" give me the new position of these coordinates. Is there an easy way to avoid doing the interpolation on my own?
Maybe there is an easy procedure that I missed that solves this issue...
If so, the modification of the transformation matrix should be pretty easy.

Thanks, Helder

---

Subject: Re: Shear transformation with Poly_2d
Posted by lecacheux.alain on Thu, 21 Jun 2012 16:56:27 GMT
View Forum Message <> Reply to Message

On 21 juin, 10:31, Helder <hel...@marchetto.de> wrote:
> On Tuesday, June 19, 2012 5:54:22 PM UTC+2, alx wrote:
>> On 19 juin, 14:52, Helder <hel...@marchetto.de> wrote:
>>> On Monday, June 18, 2012 9:17:22 PM UTC+2, Craig Markwardt wrote:
>>>> On Monday, June 18, 2012 11:15:48 AM UTC-4, Helder wrote:
>>>> > Hi,
>>>> > I'm not going to post a question, rather a solution... The reason is that I was fighting with this a few hours and I thought it would be nice to show a solution to anybody who might care about this.
>>>> > I was a bit annoyed that of the affine transformations (scaling, rotation, translation and shear) one is missing... The transformations like scaling and rotation can be accessed using ROT that then calling POLY_2D that actually does the work. Translation can be done (easily) with shift (although this is only possible using integer translation, for non integer translations, the procedure described below may also be used...).
>>>> > What is missing is of course shear. If you look in text books you find a matrix for shear that looks like:
>>>> > ShearVertical = [[1,0,0],[Vert,1,0],[0,0,1]]
>>>> > where Vert is the degree of vertical shear.
>>>> > I thought it would be nicest to get this sorted by using this matrix, but I just couldn't get this to work without loops or having to rewrite basic math code... not nice and given that IDL is well suited for image processing I thought that there has to be an IDL way for this.

>
>>>>  > So here we go. POLY_2D give in the help indications on how to use it and some examples (of course not shear).
>>>>  > If you want to implement a shear transformation, then you would have to do the following.
>
>>>>  > Img = DIST(200)
>>>>  > WINDOW, XSIZE=405,YSIZE=200
>>>>  > TVSCL,Img
>>>>  > VertShear = 20.0
>>>>  > KX = [[0.0, 0.0],[1.0, 0.0]]
>>>>  > KY = [[VertShear, 1.0],[-VertShear/100.0, 0.0]]
>>>>  > TVSCL,POLY_2D(img,KX, KY,cubic=-0.5), 205, 0, /DEVICE
>
>>>>  > This works fine if you want to shear the image in the middle. If you want shear somewhere else, I could only manage that using POLYWARP.
>>>>  > In this case, you set some coefficients that act as transformation points (4 is the minimum number of points):
>
>>>>  > s = SIZE(Img,/DIMENSIONS)
>>>>  > Offset = [50,0]           ;In pixels
>>>>  > XI = [0, 0, s[0]-1, s[0]-1]+Offset[0]
>>>>  > YI = [0, s[1]-1, 0, s[1]-1]+Offset[1]
>>>>  > XO = XI
>>>>  > VertShear = 20.0
>>>>  > YO = YI + [-VertShear, -VertShear, VertShear, VertShear]
>
>>>>  > Then with POLYWARP you can retrieve the coefficients KX and KY and use them as above.
>
>>>>  > POLYWARP, XI, YI, XO, YO, 1, KX, KY
>>>>  > TVSCL,POLY_2D(Img,KX, KY,cubic=-0.5), 205, 0, /DEVICE
>
>>>>  > The shearing point is now moved 50 pixel to the right (positive).
>>>>  > There is probably an easy way to obtain these coefficients in a smooth mathematical way... Happy to hear suggestions!
>
>>>>  I think the standard way to implement a shear or rotation about a different origin, is to compose a series of transformations: translate, shear, untranslate.
>
>>>>  You can't do translations with a simple matrix transformation.  A "translation matrix" can be designed by using an (N+1)x(N+1) matrix and using homogeneous coordinates.
>>>>  http://en.wikipedia.org/wiki/Translation_%28geometry%29
>>>>  Not that this is more efficient, but sometimes it can simplify the notation.
>
>>>>  Craig
>
>>>  Yes, you are absolutely right. However, when dealing with affine transformations the translation is also included as an "affine matrix":

>>>  Translation = [[1,0,0],[0,1,0],[tx,ty,1]]
>>>  where an affine transformation transforms spatial coordinates (v,w) of the source image to (x,y) in the transformed image. The notation goes like:
>>>  [x,y,1] = [v,w,1] TransfMatrix
>>>  I have taken my geometry classes a long time ago, but I suppose that in this case the pixel coordinates are two and the matrix has three components, therefore fitting to what you said about using a (N+1)x(N+1) transformation matrix for translations (the last column and last row are all zeros and a one on the diagonal for all transformations except for the translation where numbers come in...). Another way of describing this is saying that affine transformations are linear transformations followed by a translation.
>>>  I took the above transformation matrix treatment from Digital Image processing of Gonzales/Woods.
>
>>>  I find it a pity that there is no clear/direct way to implement shear transformations in IDL. As I said, it is possible to do rotation, scaling and translation (with some limitations...), but not shear. I thought that they would be all fit in the same chapter. Even nicer would be the possibility to feed in transformation matrices directly. One could then combine the matrices accordingly and perform the transformation only once. If, for instance, I would like to first rotate and then translate, I would calculate the matrix and then perform the operation, saving computation time (1 matrix calculation and 1 transformation).
>
>>>  At the moment I'm busy with something else, but I'll check if effectively the image is also translated when doing the shear rotation as describe above. So far I have the feeling that this is working, but I will cross check by doing separating the two operations and comparing results.
>
>>>  Helder
>
>>  Following what Craig said, look at the T3D routine (Direct graphics)
>>  or equivalent functions in NG, and use homogeneous coordinates.
>>  While keywords for rotation, scaling and translation matrices are
>>  prebuilt, you also can define your own transformation trough MATRIX
>>  keyword, and realize whatever transform you want. The transform matrix
>>  of a shear transformation is characterized by the addition of a
>>  multiple of one row or column to another. You can easily build it from
>>  the identity matrix (function IDENTITY in IDL) and replacing one of
>>  the zero elements by a non zero value.
>>  alx.
>
>  Hi Alx,
>  I already tried once to use the T3D routine, but I found this not ideal for my purposes.
>  The reason is the following. If you have an image, you define the intensity for each given pixel as I(x,y). The transformation matrix will allow you to find the new coordinates (x',y') that will have the same intensity of the original point I(x,y) = I(x',y'). The problem is that (except for trivial cases) the transformation will not match with an integer value and the new intensity in the new image has to be interpolated using either nearest-neighbor, bilinear or cubic methods.
>  In other words, I would only find out where the new coordinate points would be by first generating the coordinates of the x and y points:
>  [following Craig's method for generating coordinates]

> x = findgen(N)*0.1 - 5.
> y = x
> xx = x # (y*0 + 1)
> yy = (x*0 + 1) # y)
> But then the transformation would "only" give me the new position of these coordinates. Is there an easy way to avoid doing the interpolation on my own?
> Maybe there is an easy procedure that I missed that solves this issue...
> If so, the modification of the transformation matrix should be pretty easy.
>
> Thanks, Helder
>
>

Maybe I am missing something.
I understand that, after the transformation, you get some irregular gridding. Then "triangulate/trigrid" and/or "griddata" or even "warp_tri" should solve your problem.