Subject: Re: image data extraction
Posted by landers on Fri, 03 Jun 1994 13:39:24 GMT
View Forum Message <> Reply to Message

Finding image data along a line...

This just picks points from the image. It doesn't do any image interpolation or anything like that. Just finds the closest x,y to the line.

Just some code fragments I sucked out of one of my codes:

```
; Line endpoints are (x1,y1), (x2,y2) (these are indices into the image)
; image size is (Mx,My)

; There are two methods here.  The first works better for more
; horizontal cuts, the second for more vertical.  You could put a test
; on the slope of the line, and choose your method accordingly.

.....................
;;;;;;;;;;;;;;;;;;
; Horizontal cuts:
; first figure out how many points you want to grab between the ends.

Nx = Abs( x2 - x1 ) > 1
Ny = Abs( y2 - y1 )

; find the x and y indices into the image along the cut
; get one point for every x pixel in the cut line, and figure the
; corresponding y points

thex = Long( Lindgen( Nx + 1 ) + x1 )
they = Long( Lindgen( Nx + 1 ) * Float(Ny)/Nx + y1 )

image_line = image( they * Mx + thex )

.....................
;;;;;;;;;;;;;;;;;;
; Vertical cuts:

Nx = Abs( x2 - x1 )
Ny = Abs( y2 - y1 ) > 1

; get one point for every y pixel in the cut line

thex = Long( Lindgen( Ny + 1 ) * Float(Nx)/Ny + x1 )
they = Long( Lindgen( Ny + 1 ) + y1 )

image_line = image( they * Mx + thex )
```

;Dave

---

Subject: Re: image data extraction
Posted by sterner on Fri, 03 Jun 1994 17:30:19 GMT
View Forum Message <> Reply to Message

landers@tsunami.dseg.ti.com (David Landers) writes:

> Finding image data along a line...

> This just picks points from the image.  It doesn't  do any image
> interpolation or anything like that.  Just finds the closest x,y to the
> line.
   ... Text dropped  ...

There is now an easy way to do this.  Recent versions of IDL include
a function called INTERPOLATE which does linear, bilinear, or
trilinear interpolation depending on the dimension of the input array.

Let z be an array to extract a line from.
Let (ix1,iy1) and (ix2,iy2) be endpoints of the desired line.
Let n be the number of points desired along the line.

The 2-d indices into array z along the line are:

```
x = ix1 + (ix2-ix1)*findgen(n)/(n-1)
y = iy1 + (iy2-iy1)*findgen(n)/(n-1)
```

The desired values (bilinearly interpolated) along the line are now simply:

```
c = interpolate(z,x,y)
```

If you would like nearest neighbor interpolation do:

```
c2 = interpolate(z,round(x),round(y))
```

where ROUND is another fairly recent addition to IDL.

This also works if x and y trace a curve instead of a straight line.
You could use the mouse to draw a curve on an image and extract the
values along that curve.

Warning: INTERPOLATE has a keyword to do cubic interpolation.
It's easy to use, just add /cubic to the call.

However, don't assume this is better. Check it very carefully before
you rely on it. A number of people have reported problems with /cubic
in other routines (CONGRID, ROT). If the result of the cubic interp.
is carefully examined it will be found to have high frequency wiggles
in the values (with a period of roughly 20 some pixels for the case
I just checked). As of V3.6 beta this problem still exists (for
INTERPOLATE anyway). The default, bilinear, works very well.

Ray Sterner sterner@tesla.jhuapl.edu
Johns Hopkins University North latitude 39.16 degrees.
Applied Physics Laboratory West longitude 76.90 degrees.
Laurel, MD 20723-6099

---

Subject: The cubic keyword in interpolate
Posted by sterner on Mon, 13 Jun 1994 13:34:15 GMT
View Forum Message <> Reply to Message

In an earlier post I gave a warning about using the CUBIC keyword in
INTERPOLATE, CONGRID, ROT:

> Warning: INTERPOLATE has a keyword to do cubic interpolation.
> It's easy to use, just add /cubic to the call.
> However, don't assume this is better. Check it very carefully before
> you rely on it. A number of people have reported problems with /cubic
> in other routines (CONGRID, ROT). If the result of the cubic interp.
> is carefully examined it will be found to have high frequency wiggles
> in the values (with a period of roughly 20 some pixels for the case
> I just checked). As of V3.6 beta this problem still exists (for
> INTERPOLATE anyway). The default, bilinear, works very well.

I have been advised that this option is actually working correctly. I
now believe that to be true. However I suspect a number of people
misunderstood how this option actually works. Here is some code
to demonstrate:

```
x=indgen(4)    ; Some random data.
y=[1.,2,5,10]
x2=findgen(100)/100.*3.  ; Interpolate between points.
yl=interpolate(y,x2)  ; Linear (L).
yn=interpolate(y,round(x2)) ; Nearest Neighbor (NN).
yc=interpolate(y,x2,/cubic) ; Cubic convolution (CC).
plot,x,y   ; Original data (4 points).
oplot,x2,yn   ; NN
oplot,x2,yl,psym=-4  ; L
oplot,x2,yc   ; CC
```

The four original points are connected by 3 straight lines.

Nearest Neighbor interpolation, yn, is the worst case.
Linear interpolation, yl, is better.
Cubic interpolation, yc, might be expected to give a nice smooth curve through the points, but that is a misunderstanding. The cubic interpolation is working correctly but it is not a spline fit.

The following code plots the expected curve.

```
r=nr_spline(x,y)  ; Set up spline.
ycs=nr_splint(x,y,r,x2)  ; Do spline interpolation.
oplot,x2,ycs
```

Unfortunately this doesn't help for 2-d interpolation.

Besides making processed images look better, the cubic convolution option does have other useful applications. Here is some code that shows how high spatial frequencies are restored better using the /cubic keyword:

```
x = findgen(1000)   ; Generate a curve.
y = sin(x/(1+x/200.))
plot,x,y    ; Plot entire curve.
```

Just look at the high frequency part of the data:

```
plot,x,y,xran=[0,50],yran=[-1.2,1.2]
```

Note that it is somewhat ragged due to undersampling. Now interpolate up to a larger number of points using /cubic:

```
x2 = findgen(20000)/20.
y2 = interpolate(y,x2,/cubic)
oplot,x2,y2
```

Note that the interpolated curve appears better and might also be more useful for locating peaks. The peaks are not exactly where they should be but are closer than the uninterpolated peaks. Also note that the interpolated curve overshoots the true values as can be better seen by adding horizontal lines:

```
plots,[0,1000],[1,1]
plots,[0,1000],-[1,1]
```

Ray Sterner  sterner@tesla.jhuapl.edu
Johns Hopkins University  North latitude 39.16 degrees.
Applied Physics Laboratory  West longitude 76.90 degrees.
Laurel, MD 20723-6099

---