Subject: Re: Image warping in IDL
Posted by JD Smith on Fri, 10 Nov 2006 22:51:34 GMT
View Forum Message <> Reply to Message

On Fri, 10 Nov 2006 10:05:25 +0100, Wox wrote:

> ;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
> ; img points to the input image
> imgs=size(*img)
> seval=indgen(imgs[1])
> teval=indgen(imgs[2])
> xmap=seval#replicate(1b,imgs[2])
> ymap=replicate(1b,imgs[1])#teval
>
> ; X-distortion
> xmap+=bsplineint2Dcp(xuvec,xvvec,xp,xq,xn,xm,xh,xk,seval,tev al,xCP) ; ->
> these parameters come from an external source ; Y-distortion
> ymap+=bsplineint2Dcp(yuvec,yvvec,yp,yq,yn,ym,yh,yk,seval,tev al,yCP) ; ->
> these parameters come from an external source
>
> ; At this point we have the corresponding output pixel (non-integer) for
> each input pixel

Here's a possibility:

Take the fractional xmap, ymap pair of vectors, and use HIST_ND to bin into a unit cell grid the size of the output array, saving the reverse indices. Most bins (== output pixels) will have 3 input pixel or fewer mapped into them (depending on how severe the warping is).

Using the "histogram of histogram" method to loop over the small number of values in the original histogram frequency distribution (e.g. 1,2,3), and build up the output array as so:

1. Calculate the fraction of the input pixels xmap[inds],ymap[inds] which fall on each of the 9 neighbors surrounding that output bin. 5 will have zero fraction, and can be disregarded. This will depend on your pixel coordinate system (I use 0.5, 0.5 as center of first pixel, other people use other systems). See below.

2. Accumulate in the output array "f_pix * input[pix]" for each of the 4 pixels that it overlaps, careful of falling off the edge.

3. Accumulate a separate array of f_pix for each of the 4.

4. Divide the output by the accumulated f_pix array, to get the fractional area-weighted average.

All of these steps can be done without a loop. The only loop will be very short (maybe 2-3 iterations at most), over the unique repeat counts in the original 2D histogram.

This is a flux-conserving algorithm similar to drizzle, but vastly simplified by the lack of rotation/skew/etc. between input and output pixels. If a given output pixel has no input pixel "touching it", the f_pix array will be zero there after everything, and you'll have to interpolate from neighbors.

Here's a suggestion for calculating f_pix from two (potentially long) 1xn vectors, xm & ym. This is the part drizzle has to do the hard way by clipping polygons:

```
outpix=floor(xm) + im_width*floor(ym)
dx=xm-floor(xm)-.5 & dy=ym-floor(ym)-.5
sx=1-2*(dx lt 0.) & sy=1-2*(dy lt 0.)
dx=fabs(dx) & dy=fabs(dy)

f_pix=[dx*dy, (1.-dx)*dy, dx*(1.-dy), (1.-dx)*(1.-dy)]
off_x=[sx,    0,          sx,         0            ]
off_y=[sy,    sy,         0,          0            ]

output_pix=rebin(outpix,4,npix) + off_x + im_width * off_y
add_vals=rebin(input[inpix],4,npix)*f_pix
```

You'll also want to zero out the f_pix and add_vals for pixels off the array.

Note that you can't now just say:

```
output[output_pix]+=add_vals
fpix_sum[output_pix]+=f_pix,
```

since there are likely many duplicates among output_pix. See the histogram tutorial for a method using (you guessed it) HISTOGRAM. In fact, another dual histogram would do nicely there.

Yes, that's a lot of HISTOGRAMs:

H1. 2D histogram of fractional mapping coords in output pixel grid.
H2. Histogram of mapped pixel density in H1.
H3. Histogram of output pixels, for each "repeat count" density from H2.
H4. Histogram of repeat counts in the output pixels from H3.

H1 and H2 are called once. H3 and H4 are called once for each number

of repeats in the 2d map histogram (i.e. 2 or 3 times, likely). If you code this up, let us know whether it was faster.

JD
