Subject: Re: Image warping in IDL
Posted by Jeff Hester on Sat, 18 Nov 2006 23:22:56 GMT
View Forum Message <> Reply to Message

Wox wrote:

> On Wed, 08 Nov 2006 10:07:40 -0700, JD Smith <jdsmith@as.arizona.edu>
> wrote:
>
>> I don't see how forward and reverse mapping in this context are any
>> different from each other.
>
>
> The approach is different. And yes, if you just had the tie points in
> input and output image, you could choose between forward and reverse
> mapping.
>
> But as stated after the description of forward and reverse mapping: "I
> only have the surfaces (Xi,Yi)->Xo and (Xi,Yi)->Yo". I have these
> surfaces as 2D splines, I don't have the anchor points.
>
> So I can't just "swap the anchor points", because I don't have them, I
> only have the coefficients of two 2D splines.
>
> The thing is, how to "swap these surfaces", if you know what I mean.
>
> Off course, one could define some arbitrarly tie points, evaluate the
> spline for them and then "swap the anchor points" to do reverse
> mapping. But how to define these tiepoints?
>

Apologies if I'm repeating something that has been said, but when I am faced with this problem, I do the following:

(1) Set up a grid of points x_i, y_i spanning the image that you want to warp, then transform them into eta_i, xce_i in space you are warping into. (This is the transformation that you know how to do.)

(2) Do a least squares fit for some function, (x_i, y_i) = F(eta_i, xce_i) using these sample points.

(3) Do the "reverse" transformation in the standard way, marching through the output (eta, xce) space using F() to map the regularly gridded coordinates back into the original image.

This runs very efficiently in IDL since you can transform large arrays of coordinates at once. (I usually do it with a loop over rows, but there is no reason you can't pass coordinates for the entire array in

one go.)

The key is in choice of the mapping function. Functions of the form x = a + b*eta + c*xce + d*eta^2 + e*xce^2 + f*eta*xce + (whatever order terms you care about) usually work pretty well for my applications (which typically involve optical distortions). You can look at the errors in the transformation to judge whether they are good enough for your purposes.

The other thing that you can do is treat the coordinate transformation as an interpolation problem from an irregularly gridded array to get values for x_i, y_i on a regular grid of points eta_i, xce_i. Then as above use these values to do the reverse transformation.

As an aside, I sometimes have to put images through a long string of transformations. This is messy, because each time you transform an image you introduce resampling errors. So I set up a couple of coordinate arrays, X and Y, in the original image. (X just has the x coordinate for each pixel, while Y has the Y coordinate.) I then put the X and Y arrays through each transformation that I apply to the image I am working with. At the end of the sequence, the transformed X and Y arrays contain the non-integer coordinates in the input array corresponding to each pixel in the output array. This provides the coordinates that I need to go back and redo the entire sequences of resampling operations in a single step.

Hope this helps.

Jeff Hester