Changes to fix itk::PathToImageFilter.

Release 1.0

Zachary Pincus

February 14, 2006

zpincus-at-stanford.edu

Abstract

I have made some changes to make itk::PathToImageFilter to make it a usable base class. There were some problems with the CVS version, including: (1) this "base class" had actual filter functionality, (2) the mechanism for specifying the size, spacing, origin, etc., of the output image was incomplete, contrary to ITK convention, and hard to extend, and (3) the documentation specified that the filter would calculate the size of the output image from the path, but it did not.

I solved these problems by (1) moving the filter functionality to itk::TracePathOnImageFilter, (2) requiring that `filter->GetOutput()->Set...(...)` should be used to set the information for the output image, and (3) providing methods to calculate the correct image region a path inhabits, and a boolean flag to specify whether that calculation should be done (default true).

There are a few points to make about these modifications. First, I believe that they are fully backward compatible with the current PathToImageFilter, and make use of the proper ITK deprecation macros. This should be checked by someone else, however.

Second, the mechanism that I chose for users to modify the output image information and regions may not be perfect. I elected that if a user desired to change image metadata, they should simply use `filter->GetOutput()->SetOrigin/Spacing/etc`. I'm not sure if this is really the correct behavior. Also, for a user to set a custom image region on the output, the user would have to remember to call `filter->CalculatePathRegionOff()` before `filter->GetOutput()->SetRegions(...)`. I am not sure if this is optimal.

Third, I was not sure where to place the code which calculates the correct region for the output image. Placing that code in `GenerateOutputRequestedRegion()` seems wrong, because setting only the RequestedRegion of the output leads to the requested region being outside of the largest possible region, causing an exception. But setting all of the image regions within a GenerateOutputRequestedRegion call seems incorrect. What I elected to do is almost certainly incorrect: I provided an `Update()` implementation that gets the output image, calls `SetRegions()` on that image with the region calculated from the path (if CalculatePathRegion is on), and then calls `Superclass::Update()`. I suspect that this may interact poorly with the pipeline mechanism if only a sub-region really needs to be requested. I'm not sure though, since that's all still a bit opaque to me.

Finally, there is the matter of precisely how the region that a path inhabits is computed. Given that the input is a generic path, there is only a very slow way of computing the region – calling `IncrementInput()` repeatedly on the path. However, if the path is something like a PolyLineParametricPath, you can find out

its region simply by looking at the vertices. So ideally, the `CalculatePathRegion()` function would be partially specialized based on the path type. However, I don't think that ITK supports partial specialization currently. (Is this still the case?) As such, I made `CalculatePathRegion()` a non-member function templated only on the path type, and provided specializations for `PolyLineParametricPath<2>` and `PolyLineParametricPath<3>`. (Aside: is there any proper mechanism for making a partial specialization where the template parameter is itself templated? e.g. specializing `CalculatePathRegion()` for `PolyLineParametricPath<n>`, where n is the template parameter of the specialization?) Anyhow, the above template stuff is a bit ugly. If there are better ideas, I would love to hear them.

But, the good news is that everything seems to work. I made a test case as part of checking this filter out, so I'm including that with this journal submission.