Using a Mask to Decrease Computation Time for SpatialObject to Image Conversions

Release 1.0

Dan Mueller[1]

July 23, 2006

[1]Queensland University of Technology, Brisbane, Australia

Abstract

This article presents an approach for decreasing the computational time for converting a *sparse* `SpatialObject` to an `Image`. The method is applicable for `SpatialObjects` which occupy less volume than the total output `Image` (which occurs often, especially when using `TubeSpatialObjects` to represent vessels). A new filter `MaskedSpatialObjectToImageFilter` is introduced which dramatically decreases the execution time by creating a mask of relevant pixels. The mask is computed by calling the conventional `SpatialObjectToImageFilter` at a lower resolution, which is significantly faster for large images. The mask is then up-sampled to control which areas are further processed.

Keywords: ITK, SpatialObject, Image, convert, fast, time, speed

1 Introduction

The ITK `SpatialObject` library provides an effective and powerful mechanism for representing objects as simple geometric models. For many applications it is useful to convert an `SpatialObject` model to an `Image`. Currently the `SpatialObjectToImageFilter` can be used in such occasions, however the computational time for conversion of large images ($> 256 \times 256 \times 256$ pixels) can be very long.

This article introduces a new filter `MaskedSpatialObjectToImageFilter` which uses a mask to decrease the time spent converting *sparse* `SpatialObjects`. By 'sparse' I mean `SpatialObjects` which occupy less space than the output `Image`. This situation is common, especially for applications using tubes to represent vessels.

In this article I briefly discuss the implementation and usage of the new filter, and demonstrate the speed increases with a number of 'toy' and real datasets.

2 Implementation

2.1 Design

Currently `MaskedSpatialObjectToImageFilter` extends `SpatialObjectToImageFilter` and overrides `GenerateData()`, however it may be possible to re-factor `SpatialObjectToImageFilter::GenerateData()` to allow the sharing of much of this code. Some parameters and helper methods have been added to the new filter.

In terms of additional methods, the `GenerateMask()` method is the most interesting. If the user does not provide a mask (as the second input) this method is responsible for automatic mask generation. It does this by converting the input `SpatialObject` to a low-resolution mask. The computation of this low-resolution mask is significantly less than at full-size. A binary dilation is applied to ensure the low-resolution mask covers the required area, and finally the mask is resampled to full-size.

The `MaskedSpatialObjectToImageFilter` has two additional parameters:

`MaskResampleFactor`: this parameter controls the size of the low-resolution mask. The default value is `4.0`. Using this default value, for the example of a $512 \times 512 \times 512$ output image, the mask will be generated at the smaller size of $128 \times 128 \times 128$ (which takes orders of magnitude less time than the full-size).

`MaskDilationSize`: this parameter controls the size of the morphological structuring element used to expand the mask image. This dilation ensures that the production of the mask at the lower resolution does not miss included regions. The default value is `2`, however this parameter ultimately depends of the `MaskResampleFactor`.

2.2 Usage

Source code for this section can be found in `Testing/itkMaskedSpatialObjectToImageFilterTests.cxx`. Using the new filter is almost identical to `SpatialObjectToImageFilter`. Firstly we must declare the filter type:

```
68      typedef itk::GroupSpatialObject<Dimension> GroupType;
69      typedef itk::MaskedSpatialObjectToImageFilter<GroupType, ImageType>
70          SpatialObjectToImageFilterType;
```

We now read a `GroupType SpatialObject`:

```
75          SpatialReaderType::Pointer readerSO = SpatialReaderType::New();
76          readerSO->SetFileName(InputSpatialObjectFilename);
77          readerSO->Update();
```

create the filter, plug-in the input `SpatialObject`, and setup the various parameters:

```
80          SpatialObjectToImageFilterType::Pointer filterConvert =
81             SpatialObjectToImageFilterType::New();
82          filterConvert->SetInput(readerSO->GetGroup());          //Set Input
83          SpatialObjectToImageFilterType::SizeType size;          //Set Size
84          size[0] = Size0;
85          size[1] = Size1;
86          size[2] = Size2;
87          filterConvert->SetSize(size);
88          SpatialObjectToImageFilterType::SpacingType spacing;    //Set Spacing
89          spacing[0] = Spacing0;
90          spacing[1] = Spacing1;
91          spacing[2] = Spacing2;
92          filterConvert->SetSpacing(spacing);
93          filterConvert->SetUseObjectValue(false);                //Setup Parameters
94          filterConvert->SetInsideValue(itk::NumericTraits<PixelType>::max());
95          filterConvert->SetOutsideValue(itk::NumericTraits<PixelType>::min());
96          filterConvert->SetMaskResampleFactor(MaskResampleFactor);
97          filterConvert->SetMaskDilationSize(MaskDilationSize);
98          filterConvert->Update();
```

3 Results

To demonstrate the achieved speed increases I have put together two 'toy' examples and two real examples. The real examples are `TubeSpatialObject` representations of the coronary arteries extracted from multi-slice spiral computed tomography angiography (MS-CTA) images. These example files are included with the source code for this article in the `Images` folder. Maximum intensity projections (MIP) of the example images are shown in Figure 1, and the dimensions and spacings are listed in `Images/Readme.txt`.

(a) Toy1 (b) Toy2 (c) Real1 (d) Real2

Figure 1: Maximum intensity projection (MIP) representations of the example images.

Each of the four input `SpatialObjects` were converted to an `Image` using both the original and masked methods. These conversions were repeated five times and the mean taken as the result. Unfortunately due to the large times associated with the conversion, the two real datasets were converted to approximate half-size output images (ie. $256 \times 256 \times 256$). The file `Testing/CMakeLists.txt` contains the script used to generate the results. The computer specifications on which these tests were run are as follows: Intel Pentium 4, 2.8GHz dual processors, 1GB RAM, on Windows XP, with ITK-2.8.1, compiled using Microsoft Visual Studio .NET 7.1. Figure 2 depicts the resulting mean computational times using both approaches. It can be clearly seen that the new mask-based method is significantly faster for large, sparse `SpatialObjects` (ie. Toy2, Real1 and Real2). Even for smaller, less sparse objects (ie. Toy1) the new method is comparable.

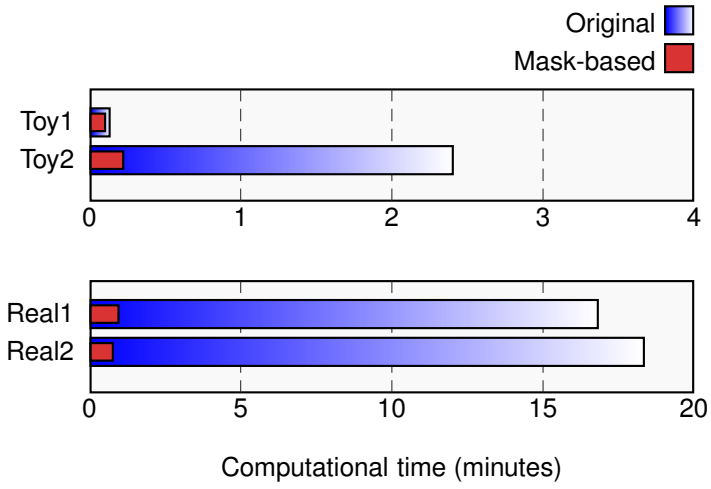

Figure 2: Experimental results of computational time for some 'toy' and real datasets.

4 Conclusions

This paper presented an approach for decreasing the computation time for converting an `SpatialObject` to an `Image`. The speed increases are realised by only computing the `SpatialObject` value at points within a mask. The mask is generated at a lower resolution in significantly less time than full-size. This approach assumes that the overhead for the low-resolution mask generation is less than converting the `SpatialObject` to the full size output image (which is the case for many applications including vascular models). The speed-ups were demonstrated using a number of toy and real examples. While only `TubeSpatialObject` examples were used, this approach is applicable to *any* desired `SpatialObject` representation.