The Iteration Direction Optimized Image Iterator

Release 0.00

Wanlin Zhu[1]

February 27, 2006

[1] National Laboratory of Pattern Recognition
Institute of Automation, Chinese Academy of Sciences

Abstract

This document describes the iteration direction optimized Image Iterators . Comparing with the original corresponding iterators, the new iterators provide automatic iteration direction optimization and give more flexible. They are more efficient when the fastest moving direction has small size. The derived template class ImagePieceWiseLinearIteratorWithIndex is a generic class that can be replace the ImageLinearIterator and ImageSliceIterator. The paper present the implementation and examples of these classes. This paper is accompanied with the source code and experiments.

Contents

1 Introduction

Iterator classes like itk::ImageRegionIterator and itk::ImageRegionIteratorWithIndex are very essential for ITK system. As a result, their performance contribute largely to most filters. We improved the original template class itk::ImageConstIterator and itk::ImageConstIteratorWithIndex to optimize its performance when the fastest moving direction is thin. Moreover the improved iterator is more flexible for iteration direction selection.

2 Implementation

A new template class named itk::DirectionArray inherited from itk::FixedArray is defined. It is a representation of itk::Image iteration directions and has a method to sort image iteration directions based on the size of input region. For example, when the size of the requested region of an itk::Image is [100, 200, 100], the optimized iteration directions are [1,0,2].

Based on the current version itk::ImageConstIterator and itk::ImageConstIteratorWithIndex, we plus new data members to them, an itk::DirectionArray object and two iterators point to the first and last iteration direction respectively. The iteration directions are stored in itk::DirectionArray object,such as [2,1,0,...], where 0 direction is the fastest moving direction. They are optimized automatically or can be specified by user. Only directions among itk::DirectionArray::m_ItBegin and itk::DirectionArray::m_ItEnd will be iterated. The mechanism bring flexibility that you can specify any directions with any order to iterate a region of itk::image. When create an object of itk::ImageConstIterator or itk::ImageConstIteratorWithIndex or their derived classes, the iteration directions will automatically re-sort based on the input region size(One can turn off the automatic optimization by input bool variable false to constructor). The usage of the optimized iterators are exactly the same as itk::ImageConstIterator and itk::ImageConstIteratorWithIndex.

The new classes are as follows:

- itk::DirectionArray
- itk::ImageOptimizedConstIterator
- itk::ImageRegionOptimziedConstIterator
- itk::ImageRegionOptimizedIterator
- itk::ImageOptimizedConstIteratorWithIndex
- itk::ImageRegionOptimizedConstIteratorWithIndex
- itk::ImageRegionOptimizedIteratorWithIndex
- itk::ImagePieceWiseLinearConstIteratorWithIndex
- itk::ImagePieceWiseLinearIteratorWithIndex

3 performance

We performed experiments to test the performance of optimized iterators. all experiments are done with empty loops with given region. when image size is $1 \times 256 \times 256$, there are $256 * 256$ times loop. All experiments are performed on the following iterators:

1. itk::ImageRegionIterator
2. itk::ImageRegionIteratorWithIndex
3. itk::ImageRegionOptimizedIterator
4. itk::ImageRegionOptimizedIteratorWithIndex

No.	$1 \times 256 \times 256$	$256 \times 1 \times 256$	$256 \times 256 \times 1$	$256 \times 256 \times 256$
1	0.02599	0.00090 s	0.00088 s	0.22409 s
2	0.00504	0.00228 s	0.00226 s	0.57043 s
3	0.00098	0.00098 s	0.00094 s	0.24284 s
4	0.00232	0.00233 s	0.00236 s	0.59339 s
5	/	/	/	0.51076 s
6	/	/	/	0.50138 s
7	/	/	/	0.59501 s

Table 1: Execution times.

5. itk::ImageLinearIteratorWithIndex
6. itk::ImageSliceIteratorWithIndex
7. itk::ImagePieceWiseIteratorWithIndex

From table.1, The optimized iterators run faster than original ones when the first iteration direction has small size. The optimized iterators have the same performance for a give size image regardless the size order. It's the desired property. The performance ehancement for iterators with index is not as obvious as iterators without index. When iteration directions have similar size , the optimized iterators are slightly slower than orignal ones. It may caused by the additional dereference needed by changing direction. We only give one experiments results for ImagePieceWiseLinearIteratorWithIndex, It is an iterator that divide the iteration into two parts, in both part, it will optimize the iteration directions. From our experiments, it is not faster than the linear or slice iterator.However it is generic and can bring more flexibility.

4 Usage

The usage of these iterators are the same as their corresponding orignal ones. Some additional methods are provided to supply more flexibility. For eample, declare an iterator

```
ImageRegionOptimizedIteratorWithIndex<ImageType, dimension>       it(image,region);
for(it.GoToBegin(); !it.IsAtEnd(); ++it)
{
// do something.
}
```

If you do not want to declare an iterator with optimization, you can turn off it by adding a bool variable like follows

```
ImageRegionOptimizedIteratorWithIndex<ImageType, dimension>       it(image, region, false);
```

Now the declared iterator behave exactly the same as ImageRegionIteratorWithIndex iterator. for iterators with index, you can specify the iteration directions. such as

```
it.SetIterationDimension(2);
```

Then the iterator will only iterate the first and second directions. (Important: the SetIterationDimension() method is not supported for iterators without index). The ImagePieceWiseLinearIteratorWithIndex can be used to replace ImageLinearIteratorWithIndex and ImageSliceIteratorWithIndex. Fox example:

```
ImagePieceWiseLinearIteratorWithIndex       pit(image,region);
pit.SetDirection(1);
```

Now it behave the same as ImageLinearIteratorWithIndex. if input

```
ImagePieceWiseLinearIteratorWithIndex       pit(image,region);
pit.SetDirections(1,2);
```

The iterator behave the same as ImageSliceIteratorWithIndex.

5 Example Code

```
#include "itkImage.h"
#include "itkImageRegionIterator.h"
#include "itkImageRegionConstIterator.h"
#include "itkImageRegionIteratorWithIndex.h"
#include "itkImageRegionConstIteratorWithIndex.h"
#include "itkImageRegionOptimizedIterator.h"
#include "itkImageRegionOptimizedConstIterator.h"
#include "itkImageRegionOptimizedIteratorWithIndex.h"
#include "itkImageRegionOptimizedConstIteratorWithIndex.h"
#include "itkTimeProbe.h"
#include <iomanip>
////////////////////////////////////////////////////////////////////
int main(int argc, char *argv[])
{
if(argc < 2)
{
std::cerr<<"Input Parameters error!"<<std::endl;
return -1;
}
const unsigned int dimension   = 3;
const unsigned int testnumbers = 50;
typedef  float                                          PixelType;
typedef  itk::Image<PixelType,dimension>               ImageType;
typedef  ImageType::RegionType                          RegionType;
typedef  ImageType::IndexType                           IndexType;
typedef  ImageType::SizeType                            SizeType;
typedef  itk::TimeProbe                                  TimeProbeType;

typedef  itk::ImageRegionOptimizedConstIteratorWithIndex<ImageType>
ImageRegionOptimizedConstIteratorWithIndexType;
typedef  itk::ImageRegionOptimizedIteratorWithIndex<ImageType>
ImageRegionOptimizedIteratorWithIndexType;
typedef  itk::ImageRegionConstIteratorWithIndex<ImageType>
ImageRegionConstIteratorWithIndexType;
typedef  itk::ImageRegionIteratorWithIndex<ImageType>
ImageRegionIteratorWithIndexType;
typedef  itk::ImageRegionOptimizedConstIterator<ImageType>
ImageRegionOptimizedConstIteratorType;
typedef  itk::ImageRegionOptimizedIterator<ImageType>
ImageRegionOptimizedIteratorType;
```

```
typedef  itk::ImageRegionConstIterator<ImageType>
ImageRegionConstIteratorType;
typedef  itk::ImageRegionIterator<ImageType>
ImageRegionIteratorType;


IndexType         ind;
SizeType          size;
for(unsigned int i = 0; i < dimension; i++)
{
ind[i]    = 0;
size[i]   = 300;
}

RegionType region(ind,size);
ImageType ::Pointer = ImageType::New();
image->SetRegions(region);
image->Allocate();

for(unsigned int i = 0; i < dimension; i++)
{
ind[i]  = 10;
size[i] = 256;
}
size[0] = 1;
region.SetIndex(ind);
region.SetSize(size);

TimeProbeType time;

for(int i = 0; i < testnumbers; ++i)
{
ii = 0;
time.Start();

//1. Image region iterator with index
// ImageRegionIteratorWithIndexType                 it(image,region);
// ImageRegionConstIteratorWithIndexType           it(image,region);
// ImageRegionOptimizedIteratorWithIndexType       it(image,region);
// ImageRegionOptimizedConstIteratorWithIndexType it(image,region);
// it.SetIterationDimension(2);

// for(it.GoToReverseBegin(); !it.IsAtReverseEnd(); --it);

//2. Image region iterator
// ImageRegionIteratorType                           it(image,region);
// ImageRegionConstIteratorType                     it(image,region);
// ImageRegionOptimizedIteratorType                it(image,region);
// ImageRegionOptimizedConstIteratorType           it(image,region);

// for(it.GoToEnd(); !it.IsAtBegin(); --it);

for(it.GoToBegin(); !it.IsAtEnd(); ++it);
```

```
time.Stop();
}
     std::cout << std::setprecision(6)
  <<"Image Region Iterator Mean Clock Time = "
  <<time.GetMeanTime()<<std::endl;
return 0;

}
```

6 Conclusion

The optimized image iterators bring more flexibility coming with good performance,They can be treat as the improved version of the current version.

References

[1] L. Ibanez, W. Schroeder, L. Ng, and J. Cates. *The ITK Software Guide*. Kitware, Inc. ISBN 1-930934-10-6, http://www.itk.org/ItkSoftwareGuide.pdf, first edition, 2003.