
A small rework for the Gaussian Derivative Image Function

Release 2.0

Dan Mueller¹

February 28, 2006

¹Queensland University of Technology, Brisbane, Australia

Abstract

The `itk::GaussianDerivativeImageFunction` computes the derivative at the specified physical or pixel location. Unfortunately it has a number of deficiencies, for which I suggest possible solutions.

Keywords: ITK, ImageFunctions, GaussianDerivativeImageFunction

1 The Problem

Gradient or derivative information is important for many image processing tasks. The Insight Toolkit (ITK) has various means to compute the derivative; both of a whole image, and at specific locations. We concern ourselves with `itk::ImageFunctions` which allow users to compute the derivative at specified physical or pixel locations. The `itk::CentralDifferenceImageFunction` uses the central differences method to compute gradient information; however, while useful for some applications, it can suffer from noise artefacts due to small intensity variations. The `itk::GaussianDerivativeImageFunction` overcomes this problem by applying a Gaussian smoothing operator before calculating the gradient information.

Recently a bug ([# 2891](#)) was reported which highlighted the problem that `itk::GaussianDerivativeImageFunction` did not correctly handle points in physical space. This bug was fixed and committed as r1.14 (see [ITK CVS](#)). Unfortunately (as far as I can tell), while this solved the initial problem of evaluating points in physical space, it introduced a new issue of no longer evaluating the derivative at true continuous locations. In the fix (r1.14) all continuous points/indices are now cast to the nearest discrete index. Previously (r1.13) the method `RecomputeContinuousGaussianKernel(double* offset)` was invoked by `Evaluate(...)` with the difference between the continuous and discrete point passed in as the offset. I assume this offset was used to interpolate at continuous points. The method `RecomputeContinuousGaussianKernel(double* offset)` no longer being invoked.

Furthermore, the `itk::GaussianDerivativeImageFunction` contains repetitive code blocks (which are discouraged by extreme programming practices). Finally, it appears this class is only implemented for images with 2 dimensions.

2 The Proposed Solution

I propose a slight rework for this class which addresses the issues of code duplication, and evaluating the derivative at true continuous locations:

1. Remove the duplicated `RecomputeContinuousGaussianKernel()` method and all associated calls by the `SetSigma(..)` and `SetExtent(..)` methods. NOTE: keep the `RecomputeContinuousGaussianKernel(double* offset)` method.
2. Remove the `m_OperatorArray` field, and reference in `PrintSelf(..)`.
3. Create a protected method `EvaluateAtIndexWithOffset(IndexType index, double* offset)`.
4. Change the `EvaluateAtIndex(..)` method to create an offset of all zeros and then call `EvaluateAtIndexWithOffset(..)`.
5. Change the new (r1.14) `Evaluate(..)` method, but compute the offset before calling `EvaluateAtIndexWithOffset(..)`.
6. Change the new (r1.14) `EvaluateAtContinuousIndex(..)` method, but as above compute the offset before calling `EvaluateAtIndexWithOffset(..)`.

These proposed changes, to the best of my knowledge, would allow for the `itk::GaussianDerivativeImageFunction` to compute derivative information at true continuous points (ie. not casting to the nearest `Index`). All evaluation would now be handled by the `EvaluateAtIndexWithOffset(..)` method, avoiding code duplication. These changes would introduce some *limited* overhead for evaluating the function at exact (ie. discrete) indices.

3 Conclusions

The proposed changes address the discussed issues with the `itk::GaussianDerivativeImageFunction`. These changes do not change the API and therefore should adhere to ITK's backwards compatibility requirements. It should also be noted that this class is currently only implemented for images with 2 dimensions. I have not addressed this dimensionality issue in this article (any takers?).

Appendix A Source code

```

1  /*=====
2
3  Program:    Insight Segmentation & Registration Toolkit
4  Module:     $RCSfile: itkGaussianDerivativeImageFunction.h,v $
5  Language:   C++
6  Date:       $Date: 2006/02/17 $
7  Version:    $Revision: 1.15 $
8
9  Copyright (c) Insight Software Consortium. All rights reserved.
10 See ITKCopyright.txt or http://www.itk.org/HTML/Copyright.htm for details.
11
12     This software is distributed WITHOUT ANY WARRANTY; without even
13     the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
14     PURPOSE. See the above copyright notices for more information.
15
16  =====*/
17 #ifndef _itkGaussianDerivativeImageFunction_h
18 #define _itkGaussianDerivativeImageFunction_h
19
20 #include "itkNeighborhoodOperatorImageFunction.h"
21 #include "itkImageFunction.h"
22 #include "itkGaussianDerivativeSpatialFunction.h"
23 #include "itkGaussianSpatialFunction.h"
24
25 namespace itk
26 {
27
28 /**
29  * \class GaussianDerivativeImageFunction
30  * \brief Compute the gaussian derivatives of an the image at a specific
31  *       location in space, i.e. point, index or continuous index.
32  * This class is templated over the input image type.
33  * \sa NeighborhoodOperator
34  * \sa ImageFunction
35  */
36 template <class TInputImage, class TOutput=double>
37 class ITK_EXPORT GaussianDerivativeImageFunction :
38     public ImageFunction< TInputImage,
39         Vector<TOutput, ::itk::GetImageDimension<TInputImage>::ImageDimension>,
40         TOutput >
41 {
42 public:
43
44     /**Standard "Self" typedef */
45     typedef GaussianDerivativeImageFunction Self;
46
47     /** Standard "Superclass" typedef*/
48     typedef ImageFunction<TInputImage,
49         Vector<TOutput, ::itk::GetImageDimension<TInputImage>::ImageDimension>,
50         TOutput > Superclass;
51
52     /** Smart pointer typedef support. */
53     typedef SmartPointer<Self> Pointer;
54     typedef SmartPointer<const Self> ConstPointer;
55
56     /** Method for creation through the object factory.*/

```

```

57 itkNewMacro(Self);
58
59 /** Run-time type information (and related methods). */
60 itkTypeMacro( GaussianDerivativeImageFunction, ImageFunction );
61
62 /** InputImageType typedef support.*/
63 typedef TInputImage InputImageType;
64 typedef typename InputImageType::PixelType InputPixelType;
65 typedef typename InputImageType::IndexType IndexType;
66
67 /** Dimension of the underlying image. */
68 itkStaticConstMacro( ImageDimension2, unsigned int,
69 InputImageType::ImageDimension );
70
71 typedef ContinuousIndex<TOutput, itkGetStaticConstMacro( ImageDimension2 )>
72 ContinuousIndexType;
73
74
75 typedef Neighborhood<InputPixelType, itkGetStaticConstMacro( ImageDimension2 )>
76 NeighborhoodType;
77 typedef Neighborhood<TOutput, itkGetStaticConstMacro( ImageDimension2 )>
78 OperatorNeighborhoodType;
79
80 typedef Vector<TOutput, itkGetStaticConstMacro( ImageDimension2 )> VectorType;
81 typedef typename Superclass::OutputType OutputType;
82 typedef FixedArray<OperatorNeighborhoodType,
83 2*itkGetStaticConstMacro( ImageDimension2 )> OperatorArrayType;
84 typedef NeighborhoodOperatorImageFunction<InputImageType, TOutput>
85 OperatorImageFunctionType;
86 typedef typename OperatorImageFunctionType::Pointer OperatorImageFunctionPointer;
87
88 typedef GaussianDerivativeSpatialFunction<TOutput, 1>
89 GaussianDerivativeFunctionType;
90 typedef typename GaussianDerivativeFunctionType::Pointer
91 GaussianDerivativeFunctionPointer;
92
93 typedef GaussianSpatialFunction<TOutput, 1> GaussianFunctionType;
94 typedef typename GaussianFunctionType::Pointer GaussianFunctionPointer;
95
96 /** Point typedef support. */
97 typedef Point<TOutput, itkGetStaticConstMacro( ImageDimension2 )> PointType;
98
99 /** Evaluate the in the given dimension at specified point */
100 virtual OutputType Evaluate( const PointType& point ) const;
101
102 /** Evaluate the function at specified Index position*/
103 virtual OutputType EvaluateAtIndex( const IndexType & index ) const;
104
105 /** Evaluate the function at specified ContinuousIndex position.*/
106 virtual OutputType EvaluateAtContinuousIndex(
107 const ContinuousIndexType & index ) const;
108
109 /** The variance for the discrete Gaussian kernel. Sets the variance
110 * independently for each dimension, but
111 * see also SetVariance(const double v). The default is 0.0 in each
112 * dimension. If UseImageSpacing is true, the units are the physical units
113 * of your image. If UseImageSpacing is false then the units are pixels.*/
114 void SetSigma( const double* sigma );
115 void SetSigma( const double sigma );

```

```

116  const double* GetSigma() const {return m_Sigma;}
117
118  /** Set the extent of the kernel */
119  void SetExtent( const double* extent);
120  void SetExtent( const double extent);
121  const double* GetExtent() const {return m_Extent;}
122
123  /** Set the input image.
124   * \warning this method caches BufferedRegion information.
125   * If the BufferedRegion has changed, user must call
126   * SetInputImage again to update cached values. */
127  virtual void SetInputImage( const InputImageType * ptr );
128
129  protected:
130      GaussianDerivativeImageFunction();
131      GaussianDerivativeImageFunction( const Self& ){};
132      ~GaussianDerivativeImageFunction(){};
133      void operator=( const Self& ){};
134      void PrintSelf(std::ostream& os, Indent indent) const;
135
136      /** The main worker function for evaluating the function at a given
137       * discrete index and offset (NOTE: the offset will be zeros if
138       * we are evaluating at an exact discrete index). */
139      virtual OutputType EvaluateAtIndexWithOffset( const IndexType & index,
140                                                    const double* offset ) const;
141      void RecomputeContinuousGaussianKernel(const double* offset) const;
142
143  private:
144
145      double                m_Sigma[ImageDimension2];
146
147      /** Array of 1D operators. Contains a derivative kernel and a gaussian
148       * kernel for each dimension */
149      mutable OperatorArrayType    m_ContinuousOperatorArray;
150
151      /** OperatorImageFunction */
152      OperatorImageFunctionPointer m_OperatorImageFunction;
153      double m_Extent[ImageDimension2];
154
155      /** Flag to indicate whether to use image spacing */
156      bool m_UseImageSpacing;
157
158      /** Neighborhood Image Function */
159      GaussianDerivativeFunctionPointer m_GaussianDerivativeFunction;
160      GaussianFunctionPointer          m_GaussianFunction;
161
162  };
163
164  } // namespace itk
165
166  #ifndef ITK_MANUAL_INSTANTIATION
167  #include "itkGaussianDerivativeImageFunction.txx"
168  #endif
169
170  #endif

```

Listing 1: Proposed itkGaussianDerivativeImageFunction.h

```

1  /*=====
2
3  Program:    Insight Segmentation & Registration Toolkit
4  Module:     $RCSfile: itkGaussianDerivativeImageFunction.txx,v $
5  Language:   C++
6  Date:       $Date: 2006/02/17 $
7  Version:    $Revision: 1.15 $
8
9  Copyright (c) Insight Software Consortium. All rights reserved.
10 See ITKCopyright.txt or http://www.itk.org/HTML/Copyright.htm for details.
11
12     This software is distributed WITHOUT ANY WARRANTY; without even
13     the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR
14     PURPOSE. See the above copyright notices for more information.
15
16  =====*/
17 #ifndef __itkGaussianDerivativeImageFunction_txx
18 #define __itkGaussianDerivativeImageFunction_txx
19
20 #include "itkGaussianDerivativeImageFunction.h"
21
22 namespace itk
23 {
24
25  /** Set the Input Image */
26  template <class TInputImage, class TOutput>
27  GaussianDerivativeImageFunction <TInputImage, TOutput>
28  ::GaussianDerivativeImageFunction()
29  {
30      typename GaussianFunctionType::ArrayType mean;
31      mean[0]=0.0;
32      for(unsigned int i=0;i<itkGetStaticConstMacro(ImageDimension2);i++)
33      {
34          m_Sigma[i] = 1.0;
35          m_Extent[i] = 1.0;
36      }
37      m_UseImageSpacing = true;
38      m_GaussianDerivativeFunction = GaussianDerivativeFunctionType::New();
39      m_GaussianFunction = GaussianFunctionType::New();
40      m_OperatorImageFunction = OperatorImageFunctionType::New();
41      m_GaussianFunction->SetMean(mean);
42      m_GaussianFunction->SetNormalized(false); // faster
43      m_GaussianDerivativeFunction->SetNormalized(false); // faster
44  }
45
46  /** Print self method */
47  template <class TInputImage, class TOutput>
48  void
49  GaussianDerivativeImageFunction <TInputImage, TOutput>
50  ::PrintSelf(std::ostream& os, Indent indent) const
51  {
52      this->Superclass::PrintSelf(os,indent);
53      os << indent << "UseImageSpacing: " << m_UseImageSpacing << std::endl;
54
55      os << indent << "Sigma: " << m_Sigma << std::endl;
56      os << indent << "Extent: " << m_Extent << std::endl;
57
58      os << indent << "ContinuousOperatorArray: "

```

```

59     << m_ContinuousOperatorArray << std::endl;
60     os << indent << "OperatorImageFunction: "
61     << m_OperatorImageFunction << std::endl;
62     os << indent << "GaussianDerivativeFunction: "
63     << m_GaussianDerivativeFunction << std::endl;
64     os << indent << "GaussianFunction: "
65     << m_GaussianFunction << std::endl;
66 }
67
68 /** Set the input image */
69 template <class TInputImage, class TOutput>
70 void
71 GaussianDerivativeImageFunction<TInputImage,TOutput>
72 ::SetInputImage( const InputImageType * ptr )
73 {
74     Superclass::SetInputImage(ptr);
75     m_OperatorImageFunction->SetInputImage(ptr);
76 }
77
78 /** Set the variance of the gaussian in each direction */
79 template <class TInputImage, class TOutput>
80 void
81 GaussianDerivativeImageFunction<TInputImage,TOutput>
82 ::SetSigma( const double* sigma)
83 {
84     unsigned int i;
85     for (i=0; i<itkGetStaticConstMacro(ImageDimension2); i++)
86     {
87         if ( sigma[i] != m_Sigma[i] )
88         {
89             break;
90         }
91     }
92     if ( i < itkGetStaticConstMacro(ImageDimension2) )
93     {
94         for (i=0; i<itkGetStaticConstMacro(ImageDimension2); i++)
95         {
96             m_Sigma[i] = sigma[i];
97         }
98     }
99 }
100
101
102 /** Set the variance of the gaussian in each direction */
103 template <class TInputImage, class TOutput>
104 void
105 GaussianDerivativeImageFunction<TInputImage,TOutput>
106 ::SetSigma (const double sigma)
107 {
108     unsigned int i;
109     for (i=0; i<itkGetStaticConstMacro(ImageDimension2); i++)
110     {
111         if ( sigma != m_Sigma[i] )
112         {
113             break;
114         }
115     }
116     if ( i < itkGetStaticConstMacro(ImageDimension2) )
117     {

```



```

177 {
178     OutputType gradient;
179
180     //Recompute the kernel
181     this->RecomputeContinuousGaussianKernel(offset);
182
183     //Compute gradient value
184     for(unsigned int idim=0; idim<itkGetStaticConstMacro(ImageDimension2); idim++)
185     {
186         // Apply each gaussian kernel to a subset of the image
187         InputPixelType pixel = this->GetInputImage()->GetPixel(index);
188         double value = pixel;
189
190         // Apply Gaussian blurring first
191         for(unsigned int jdim=0; jdim<itkGetStaticConstMacro(ImageDimension2); jdim++)
192         {
193             if(idim != jdim)
194             {
195                 unsigned int id = 2*jdim+1; // select only gaussian kernel;
196                 unsigned int center =
197                     (unsigned int) ((m_ContinuousOperatorArray[id].GetSize()[jdim]-1)/2);
198                 TOutput centerval = m_ContinuousOperatorArray[id][center];
199                 m_ContinuousOperatorArray[id][center] = 0;
200                 m_OperatorImageFunction->SetOperator(m_ContinuousOperatorArray[id]);
201                 value = m_OperatorImageFunction->EvaluateAtIndex(index)+centerval*value;
202             }
203         }
204
205         // Apply derivative in the direction
206         signed int center =
207             (unsigned int) ((m_ContinuousOperatorArray[2*idim].GetSize()[idim]-1)/2);
208         TOutput centerval = m_ContinuousOperatorArray[2*idim][center];
209         m_ContinuousOperatorArray[2*idim][center] = 0;
210         m_OperatorImageFunction->SetOperator(m_ContinuousOperatorArray[2*idim]);
211         value = m_OperatorImageFunction->EvaluateAtIndex(index)+centerval*value;
212
213         gradient[idim] = value;
214     }
215
216     return gradient;
217 }
218
219 /** Recompute the gaussian kernel used to evaluate indexes
220 * The variance should be uniform */
221 template <class TInputImage, class TOutput>
222 void
223 GaussianDerivativeImageFunction<TInputImage,TOutput>
224 ::RecomputeContinuousGaussianKernel(
225     const double* offset) const
226 {
227
228     unsigned int direction = 0;
229     for(unsigned int op = 0; op<itkGetStaticConstMacro(ImageDimension2)*2; op++)
230     {
231         // Set the derivative of the gaussian first
232         OperatorNeighborhoodType dogNeighborhood;
233         typename GaussianDerivativeFunctionType::InputType pt;
234         typename OperatorNeighborhoodType::SizeType size;
235         size.Fill(0);

```

```

236     size[direction] = (unsigned long)(m_Sigma[direction]*m_Extent[direction]);
237     dogNeighborhood.SetRadius(size);
238
239     typename GaussianDerivativeFunctionType::ArrayType s;
240     s[0] = m_Sigma[direction];
241     m_GaussianDerivativeFunction->SetSigma(s);
242
243     typename OperatorNeighborhoodType::Iterator it = dogNeighborhood.Begin();
244
245     unsigned int i=0;
246     while(it != dogNeighborhood.End() )
247     {
248         pt[0]= dogNeighborhood.GetOffset(i)[direction]-offset[direction];
249
250         if( (m_UseImageSpacing == true) && (this->GetInputImage()) )
251         {
252             if (this->GetInputImage()->GetSpacing()[direction] == 0.0)
253             {
254                 itkExceptionMacro(<< "Pixel spacing cannot be zero");
255             }
256             else
257             {
258                 pt[0] *= this->GetInputImage()->GetSpacing()[direction];
259             }
260         }
261         (*it)= m_GaussianDerivativeFunction->Evaluate(pt);
262         i++;
263         it++;
264     }
265
266     m_ContinuousOperatorArray[op] = dogNeighborhood;
267
268     // Set the gaussian operator
269     m_GaussianFunction->SetSigma(s);
270     op++;
271     OperatorNeighborhoodType gaussianNeighborhood;
272     gaussianNeighborhood.SetRadius(size);
273
274     it = gaussianNeighborhood.Begin();
275
276     i=0;
277     double sum = 0;
278     while(it != gaussianNeighborhood.End() )
279     {
280         pt[0]= gaussianNeighborhood.GetOffset(i)[direction]-offset[direction];
281
282         if( (m_UseImageSpacing == true) && (this->GetInputImage()) )
283         {
284             if (this->GetInputImage()->GetSpacing()[direction] == 0.0)
285             {
286                 itkExceptionMacro(<< "Pixel spacing cannot be zero");
287             }
288             else
289             {
290                 pt[0] *= this->GetInputImage()->GetSpacing()[direction];
291             }
292         }
293
294         (*it)= m_GaussianFunction->Evaluate(pt);

```

```

295     sum += (*it);
296     i++;
297     it++;
298 }
299
300 // Make the filter DC-Constant
301 it = gaussianNeighborhood.Begin();
302 while(it != gaussianNeighborhood.End() )
303 {
304     (*it) /= sum;
305     it++;
306 }
307
308 m_ContinuousOperatorArray[op] = gaussianNeighborhood;
309 direction++;
310 }
311 }
312
313 /** Evaluate the function at the specifed index */
314 template <class TInputImage, class TOutput>
315 typename GaussianDerivativeImageFunction<TInputImage,TOutput>::OutputType
316 GaussianDerivativeImageFunction<TInputImage,TOutput>
317 ::EvaluateAtIndex(const IndexType& index) const
318 {
319     //Compute offset
320     double offset[itkGetStaticConstMacro(ImageDimension2)];
321     for(unsigned int i=0; i<itkGetStaticConstMacro(ImageDimension2);i++)
322     {
323         offset[i] = 0;
324     }
325
326     //Evaluate
327     return this->EvaluateAtIndexWithOffset( index, offset );
328 }
329
330 /** Evaluate the function at the specifed point */
331 template <class TInputImage, class TOutput>
332 typename GaussianDerivativeImageFunction<TInputImage,TOutput>::OutputType
333 GaussianDerivativeImageFunction<TInputImage,TOutput>
334 ::Evaluate(const PointType& point) const
335 {
336     //Convert Point to ContinuousIndex then Index
337     ContinuousIndexType cindex;
338     IndexType index;
339     this->ConvertPointToContinuousIndex( point, cindex )
340     this->ConvertContinuousIndexToNearestIndex( cindex, index )
341
342     //Compute offset
343     double offset[itkGetStaticConstMacro(ImageDimension2)];
344     for(unsigned int i=0; i<itkGetStaticConstMacro(ImageDimension2);i++)
345     {
346         offset[i] = cindex[i] - index[i];
347     }
348
349     //Evaluate
350     return this->EvaluateAtIndexWithOffset ( index, offset );
351 }
352
353 /** Evaluate the function at specified ContinuousIndex position.*/

```

```
354 template <class TInputImage, class TOutput>
355 typename GaussianDerivativeImageFunction<TInputImage,TOutput>::OutputType
356 GaussianDerivativeImageFunction<TInputImage,TOutput>
357 ::EvaluateAtContinuousIndex(const ContinuousIndexType & cindex) const
358 {
359     //Convert cindex to Index
360     IndexType index;
361     this->ConvertContinuousIndexToNearestIndex( cindex, index );
362
363     //Compute offset
364     double offset[itkGetStaticConstMacro(ImageDimension2)];
365     for(unsigned int i=0; i<itkGetStaticConstMacro(ImageDimension2);i++)
366     {
367         offset[i] = cindex[i] - index[i];
368     }
369
370     //Evaluate
371     return this->EvaluateAtIndexWithOffset ( index, offset );
372 }
373
374 } // end namespace itk
375
376 #endif
```

Listing 2: Proposed itkGaussianDerivativeImageFunction.txx