Spherical Wavelet ITK Filter

Release 0.00

Yi Gao[1], Delphine Nain[1,2], Xavier LeFaucheur[1,3], Allen Tannenbaum[1,2,3]

June 13, 2007

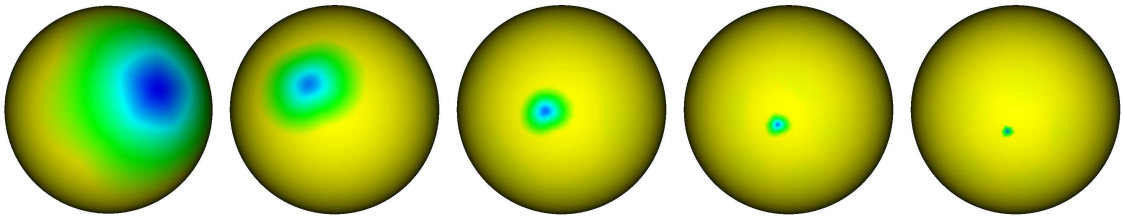
[1]Department of Biomedical Engineering, Georgia Institute of Technology
[2]College of Computing, Georgia Institute of Technology
[3]Department of Electrical and Computer Engineering, Georgia Institute of Technology

Abstract

This paper describes the Insight Toolkit (ITK) Spherical Wavelet object: itkSWaveletSource. This ITK object is an implementation of a paper by Schröder and W. Sweldens, "Spherical Wavelets: Efficiently Representing Functions on the Sphere" [8], with pseudo-code given in their paper entitled "Spherical wavelets: Texture processing" [7]. In these papers, Swelden et. al. show how to do decompose a scalar signal defined on a spherical mesh into spherical wavelet coefficients (analysis step, also called forward transform), and vice-versa (synthesis step, also called inverse transform). We have implemented the spherical wavelet transform in ITK entitled itkSWaveletSource object, which will take the scalar function defined on a spherical mesh as input and apply spherical wavelet analysis and synthesis on it. In this paper, we describe our code and provide the user with enough details to reproduce the results which we present in this paper. This filter has a variety of applications including shape representation and shape analysis of brain surfaces, which was the initial motivation for this work.

This paper is accompanied with the source code, input data, parameters and output data that the authors used for validating the spherical wavelet transform described in this paper.

Contents



(a) Scaling function, scale 0 (b) Wavelet function, scale 1 (c) Wavelet function, scale 2 (d) Wavelet function, scale 3 (e) Wavelet function, scale 4

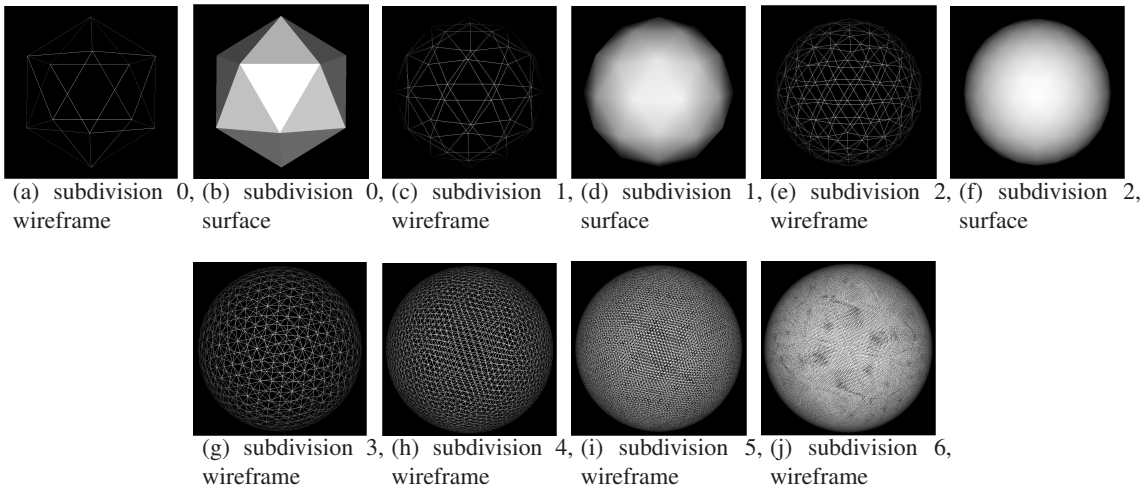
Figure 1: Visualization of selected basis function. The color corresponds to the value of the functions. (a) Visualization of a scaling function at scale 0 on the sphere (b-e) Visualization of a wavelet function for various scales.

Spherical wavelets are used in various fields such as computer graphics for texture analysis applications [7], in computer vision for data compression applications [11], or astronomy for data analysis applications [10]. In citesch-swe:env, Sweldens et. al describe a fast spherical wavelet transform on the discrete sphere. The forward transform takes as input a scalar function defined on the discrete sphere and outputs coefficients that describe the features of the signal in a local fashion in both frequency and space. Conversely, the backward transform reconstructs a signal on the sphere from a set of spherical wavelet coefficients. This transform is computationally attractive as the associated transform is linear in the number of vertices of the sphere. Once in the wavelet domain, filtering operators such as smoothing, enhancement, edge finding, and noise removal can be performed in a straightforward fashion. Additionally, coefficient analysis reveals at what frequency and spatial location the signal content lies. Recently, the fast spherical wavelet transform has been used in medical imaging for shape representation [2], shape analysis [13, 12], segmentation [3, 4] and statistical shape analysis []. In this paper, we present an ITK implementation of the fast spherical wavelet transform from [5]. We present applications and code of interest to the medical imaging community.

1 Algorithm

1.1 Spherical Wavelet Transform

A spherical wavelet basis is composed of functions defined on the sphere that are localized in space and characteristic scales and therefore match a wide range of signal characteristics, from high frequency edges to slowly varying harmonics [1]. The basis is constructed of scaling functions defined at the coarsest scale and wavelet functions defined at subsequent scales. A *scaling function* is a function on the standard unit sphere ($\mathbb{S}$) denoted by $\varphi_{j,k} : \mathbb{S} \rightarrow \mathbb{R}$ where j is the scale of the function and k is a spatial index that indicates where on the surface the function is centered. A usual shape for the scaling function is a hat function defined to be 1 at its center and to decay linearly to 0. Figure 1(a) shows a scaling function for resolution $j = 0$ and a select k. The color on the sphere indicates the value of the function $\varphi_{0,k}$ at every point on the sphere. As the resolution j increases, the support of the scaling function decreases. A wavelet function is denoted by $\psi_{j,m} : \mathbb{S} \rightarrow \mathbb{R}$. At a particular scale j, wavelet functions are combinations of resolution j and $(j+1)$ scaling functions. Figures 1(b)- 1(e) show wavelet functions for different values of j and a select m. Note that the



(a) subdivision 0, wireframe (b) subdivision 0, surface (c) subdivision 1, wireframe (d) subdivision 1, surface (e) subdivision 2, wireframe (f) subdivision 2, surface

(g) subdivision 3, wireframe (h) subdivision 4, wireframe (i) subdivision 5, wireframe (j) subdivision 6, wireframe

Figure 2: Visualization of the Recursive Partitioning of an icosahedron

support of the functions becomes smaller as the resolution increases. Together, the coarsest level scaling function and all wavelet scaling functions construct a basis for the function space L^2:

$$L^2 = \{\varphi_0, k | k \in N_0\} \bigcup \{\psi_j, m | j \geq 0, m \in N_{j+1}\}. \tag{1}$$

A given function $f : \mathbb{S} \rightarrow \mathbb{R}$ can be expressed in the basis as a linear combination of basis functions and coefficients:

$$f(\mathbf{x}) = \sum_k \lambda_{0,k} \varphi_{0,k}(\mathbf{x}) + \sum_{0 \leq j} \sum_m \gamma_{j,m} \psi_{j,m}(\mathbf{x}). \tag{2}$$

Scaling coefficients $\lambda_{0,k}$ represent the low pass content of the signal f, localized where the associated scaling function has support, whereas wavelet coefficients $\gamma_{j,m}$ represent localized band pass content of the signal, where the band pass frequency depends on the resolution of the associated wavelet function and the localization depends on the support of the function.

1.2 Discrete Fast Spherical Wavelet Transform

In this paper, we present the implementation of the discrete biorthogonal spherical wavelets functions defined on a 3D mesh proposed by Schröder and Sweldens [8, 7]. These are second-generation wavelets adapted to manifolds with non-regular grids. The main difference with the classical wavelet is that the filter coefficients of second generation wavelets are not the same throughout, but can change locally to reflect the changing (non translation invariant) nature of the surface and its measure. This means that wavelet functions defined on a mesh are not scaled and shifted versions of the function on a coarser grid, although they are similar in shape, in order to account for the varying shape of mesh triangles.

Multi-resolution Analysis Mesh

The second-generation spherical wavelets that we use are defined on surfaces which are topologically equivalent to the unit sphere ($\mathbb{S}$) and equipped with a multi-resolution mesh. A multi-resolution mesh is created

by recursively subdividing an initial polyhedral mesh so that each triangle is split into 4 "child" triangles at each new subdivision (resolution) level. This is done by adding a new midpoint at each edge, and connecting midpoints together. This process is shown on Figures 2. The starting shape is an icosahedron with 12 vertices and 20 faces, and at the fourth subdivision level, it contains 5120 faces and 2562 vertices. Table xx summarizes the number of vertices and triangles for a given subdivision level.

Any shape (not necessarily a sphere) that is equipped with such a multi-resolution mesh can be used to create a spherical wavelet basis and perform the spherical transform of a signal defined on that mesh. The number of basis functions and coefficients will equal the number of vertices of the mesh of analysis, which in turn depends on the number of subdivisions of the icosahedron used to create the analysis mesh. Table xx summarizes the number of scaling functions and wavelet functions for a given level of subdivision.

Fast Spherical Wavelet Transforms

The algorithm for the fast discrete spherical wavelet transform (FSWT) is given in [7]. In our implementation of the algorithm, we followed the notations of the pseudo-code as closely as possible.

Here, we sketch the transform algorithm in matrix form which gives a more compact and intuitive notation to understand the transform. In our implementation, we use the FSWT given in the pseudo-code of [7] in our implementation.

If there exist N vertices on the mesh, a total of N basis functions are created, composed of N_0 scaling functions (where N_0 is the initial number of vertices before the base mesh is subdivided, with $N_0 = 12$ for the icosahedron) and N_r wavelet functions for $r = 1,...R$ where N_r is the number of new vertices added at subdivision r ($N_1 = 30, N_2 = 120, N_3 = 480, N_4 = 1920$, etc for the subdivision of the icosahedron). In this paper, we will refer to the set of scaling and wavelet basis functions as basis functions as a shorthand.

In matrix form, the set of basis functions can be stacked as columns of a matrix Φ of size $N \times N$ where each column is a basis function evaluated at each of the N vertices. The basis functions are arranged by increasing resolution (subscript j) and within each resolution level, by increasing spatial index (subscript k). Since the basis functions are biorthogonal, $\Phi^T\Phi \neq Id$ (the identity matrix), so the inverse basis Φ^{-1} is used for perfect reconstruction, since $\Phi^{-1}\Phi = Id$.

Any finite energy scalar function evaluated at N vertices, denoted by the vector F of size $N \times 1$, can be transformed into a vector of basis coefficients C of size N using the *Forward Wavelet Transform*:

$$C = \Phi^{-1}F, \tag{3}$$

and recovered using the *Inverse Wavelet Transform*:

$$F = \Phi C. \tag{4}$$

The vector of coefficients C is composed of coefficients associated with each basis function in Φ. It contains scaling coefficients as its first N_0 entries, then wavelet coefficients associated with wavelet functions of resolution 1 for the next N_1 entries, and so forth, all the way to wavelet coefficients of resolution R for the last N_R entries. Therefore in total the vector contains N entries.

2 User's Guide

2.1 Software Requirements

Insight Toolkit(ITK) is needed to compile and run the code. The spherical wavelet coefficients and/or reconstructed scalar function values are in plain text files. The subdivided sphere, if written to file by the function provided, is of ITK meta file type.

Both KWVisu[6] and the dispMetaWithColor can be used to display the spherical mesh meta file.

2.2 Visualizing the Analysis Mesh

The following code is given in the file `itkSWaveletTest1.cxx`

```
SphereMeshSourceType::Pointer  mySphereMeshSource = SphereMeshSourceType::New();
int n = atoi(argv[1]);
if (n>=7 && n<0)
{
  std::cerr<<"Finest resolution can't be less than 0, and best
              to be no larger than 6..."<<std::endl;
  exit(-1);
}
mySphereMeshSource->SetResolution( n );
```

The first line creates the spherical wavelet object. This object will contain the finest multi-resolution analysis mesh (an icosahedron that has been divided n times), the function to be analyzed and the coefficients that result from the transform. Line 2 reads the level of subdivision `n` from the command line. Lines 3-8 check that the level of subdivision is in the right bounds[1]. Line 9 sets the total number of icosahedron subdivisions that will produce the finest resolution mesh.

```
PointType center;
center.Fill( 0.0 );
mySphereMeshSource->SetCenter( center );

VectorType scale;
scale.Fill( 1.0 );
mySphereMeshSource->SetScale( scale );
```

Lines 10-12 center the initial icosahedron at $(0,0,0)$ and lines 13-15 scale the vertices of the initial icosahedron to lie on the unit sphere (radius is 1.0).

```
mySphereMeshSource->Modified();
try
{
  mySphereMeshSource->Update();
}
```

[1] We constrained the subdivision level to be no larger than 6 due to a high level of vertices that will increase considerably the amount of memory and time needed for the transform.

```
catch( itk::ExceptionObject & excp )
{
  std::cerr << "Error during Update() " << std::endl;
  std::cerr << excp << std::endl;
}
```

Lines 16-25 update the object. At that point the initial icosahedron is subdivided `n` times, and the object keeps track of the parent relationships between vertices.

```
std::cerr<<"Testing task 1:"<<std::endl;
std::cerr<<"    Given different resolution in the command line, generate and
             visulize subdivided mesh at different resolutions."<<std::endl;

MeshType::Pointer myMesh = mySphereMeshSource->GetOutput();
displayITKmesh(myMesh);
```

In line 29, `mySphereMeshSource->GetOutput()` returns the finest resolution mesh (an icosahedron subdivided `n` times). It's not called here but by `mySphereMeshSource->WriteFinestSubdivisionMeshToMetaFile(fileName)`, the finest subdivision mesh is saved into .meta file. Line 30 displays the ITK mesh by using the VTK renderer.

Figure 2(b) was created by running:

```
itkSWaveletTest1 0
```

Figure 2(d) was created by running:

```
itkSWaveletTest1 1
```

Figure 2(f) was created by running:

```
itkSWaveletTest1 2
```

Figure 2(g) was created by running:

```
itkSWaveletTest1 3
```

Figure 2(h) was created by running:

```
itkSWaveletTest1 4
```

Figure 2(i) was created by running:

```
itkSWaveletTest1 5
```

Figure 2(j) was created by running:

```
itkSWaveletTest1 6
```

2.3 Basic Usage: Forward and Backward Transform

The scalar function defined on the mesh is represented as a vector of scalars, each element associated with one vertex. As the example, here we use the z coordinate of the vertex as the scalar associated with it. This is done by the following piece of codes:

```
std::vector< PointType > v = mySphereMeshSource->GetVerts();
std::vector< PointType >::const_iterator itv = v.begin();
std::vector< PointType >::const_iterator itvEnd = v.end();

std::vector< double > f( v.size() );
std::vector< double >::iterator itf = f.begin();
for ( ; itv != itvEnd; ++itv, ++itf ) *itf = (*itv)[2];
mySphereMeshSource->SetScalarFunction( f );
```

The last line above is to "push" the vector into the wavelet object. Next by

```
mySphereMeshSource->SWaveletTransform();
```

the wavelet coefficents are generated. In this part we omit the manipulation of the coefficients but immediately inverse transform to complete the procedure, which is done by:

```
mySphereMeshSource->inverseSWaveletTransform();
```

For further process, the reconstructed function can be retrieved by the following code:

```
std::vector< double > reF =  mySphereMeshSource->GetReconstructedScalarFunction();
```

The immediately reconstructed function should be a copy of the original one. Here, to see the fidelity of the transformation pair, we calculated the $\infty-$norm of the difference between original and reconstructed function. The result is around 10^{-16}.

2.4 Advanced Usage: Forward Transform, Coefficient Modification/Filtering, Backward Transform

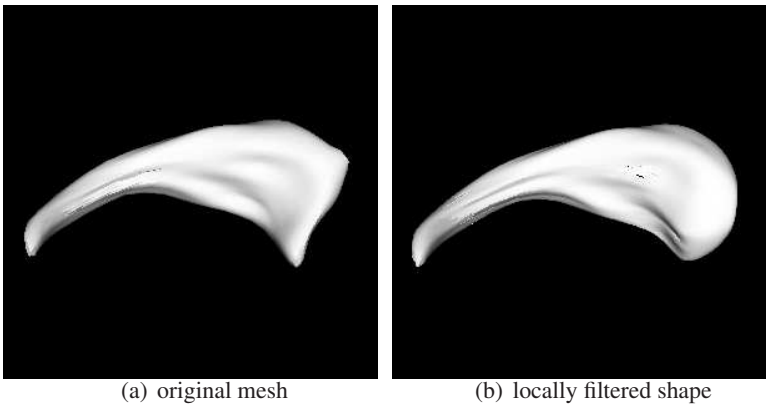
Such technique is analogous to frequency domain signal filtering. For instance, one can filter out those high frequency component in the function, leaving a denoised version of the signal. The advantage of spherical wavelet analysis lies in that the filtering can be done both locally in frequency and spatial domain. This is one of the advantages of wavelet over Fourier based methods.

The coefficients have to be first retrieved then filtered. This can be done by

```
std::vector< std::vector< double > > GetCoefficients( void );
```

It returns all the coefficients in one big structure. The elements of the structure are vectors. The first vector stores all the scaling coefficients while the wavelet coefficients of different resolutions, are in following vectors. Users can modify the returned value and then feed it back for reconstruction, or build a new one of the same structure. Also, several convenient ways are provided to modify the coefficients directly, they are:

```
void SetAllCoarsestScalingCoefficients( std::vector< double > );
void SetAllCoarsestScalingCoefficients( double );
void SetScalingCoefficientAtCoarsestScale( int, double );
void SetAllWaveletCoefficientsOfAllScales( double );
void SetAllWaveletCoefficientsAtOneScale( int scale, double dWaveletCoeff );
void SetWaveletCoefficientAtScale( int scale, int idx, double waveletCoeff );
```



(a) original mesh (b) locally filtered shape

Figure 3: Locally filtering the shape

One can tell the functions and usage by their names and arguments.

After manipulation in the wavelet domain, users can reconstruct the signal according to the procedures mentioned in the previous section.

Visualization of a Basis Function

Till now we already have all we need to visualize the basis functions. Simply set the coefficient we want to one and all the others to zero, followed by inverse transform and we'll have the basis function defined on the mesh.

Two remarks are to be made here. First, the object keeps an internal flag indicating whether the wavelet transform has been done. Only when it's true, further modification of coefficients or reconstruction can be applied. Thus we have to first decompose an (arbitrary)signal to make this flag true and then modify the coefficients and reconstruct.

Secondly, there are two formats for the output scalar function. One is just a list of floating numbers and the other format is preceded by some header information for KWVisu[6], an open source mesh visualization software, to load in and to display. Figure 1(a) is an instant of scaling function at the coarsest level with the highest level to be 5.

By similar approach we can visualize wavelet functions at different resolution level. `itkSWaveletTest4.cxx` gives and example for this.

3 Applications

As an example, we show how the spherical wavelet transform object can be applied to the shape analysis area. The surface of Caudate Nucleus as the original shape.

A shape with spherical topology can be mapped to a sphere thus having the spherical parameterization [9]. After mapping, the x, y and z coordinates of the shape can be treated as three separate scalar functions defined on the sphere and thus can be analyzed by the technique presented here. This can be found in the `filteringShape.cxx`.

In the example, to highlight the local processing property of wavelet transform, we only smooth the scalar functions, the x, y and z coordinates of the caudate surface, around the head region of the shape. The tail part of the shape is kept intact. Thus the head becomes blunt but the sharp tail is preserved. The original shape is shown in Figure 3(a) and its locally filtered version is in Figure 3(b).

4 Conclusions

The spherical wavelet transformation is a powerful tool for signal processing on the sphere. The paper introduced an open source object for carrying out such transformation and analysis. Hopefully it can help in certain areas of signal processing, computer graphics and shape analysis [5].

Currently the code is not fully templated, e.g. the scalar function is defined to be double type since in most cases we need high accuracy for the the scalar function. However, if needed, this can be a direction for further improvement.

References

[1] S. Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, 1998. 1.1

[2] D. Nain, S. Haker, A. Bobick, and A. Tannenbaum. Multiscale 3d shape analysis using spherical wavelets. In *MICCAI*, LNCS 3750, pages 459–467, 2005. (document)

[3] D. Nain, S. Haker, A. Bobick, and A. Tannenbaum. Shape-driven surface segmentation using spherical wavelets. *MICCAI*, pages 66–74, 2006. (document)

[4] D. Nain, S. Haker, A. Bobick, and A. Tannenbaum. Multiscale 3d shape representation and segmentation using spherical wavelets. *IEEE Trans. Med. Imaging, Special Issue on Computational Neuroanatomy*, 2007. Accepted, publication date: April 2007. (document)

[5] D. Nain, M. Styner, M. Niethammer, J. J. Levitt, M. E. Shenton, G. Gerig, A. Bobick, and A. Tannenbaum. Statistical shape analysis of brain structures using spherical wavelets. In *IEEE International Symposium on Biomedical Imaging (ISBI)*, 2007. (document), 4

[6] I. Oguz, G. Gerig, S. Barre, and M. Styner. Kwmeshvisu: A mesh visualization tool for shape analysis. *MICCAI 2006 Open Source Workshop. Insight Journal*, 2006. Insight Journal DSpace link: http://hdl.handle.net/1926/220. 2.1, 2.4

[7] P. Schröder and W. Sweldens. Spherical wavelets: Texture processing. In *Rendering Techniques '95*. Springer Verlag, 1995. (document), 1.2, 1.2

[8] Peter Schröder and Wim Sweldens. Spherical wavelets: Efficiently representing functions on the sphere. *Computer Graphics Proceedings (SIGGRAPH 95)*, pages 161–172, 1995. (document), 1.2

[9] M. Styner, I. Oguz, S. Xu, C. Brechbuehler, D. Pantazis, J. Levitt, M. Shenton, and G. Gerig. Framework for the statistical shape analysis of brain structures using spharm-pdm. *MICCAI 2006 Open Source Workshop. Insight Journal*, 2006. Insight Journal DSpace link: http://hdl.handle.net/1926/215. 3

[10] L Tenorio, A. H Jaffe, S Hanany, and C. H Lineweaver. Applications of wavelets to the analysis of cosmic microwave background maps. *Monthly Notices of the Royal Astronomical Society*, 310(3):823, December 1999. (document)

[11] Ze Wang, Chi-Sing Leung, Yi-Sheng Zhu, and Tien-Tsin Wong. Data compression with spherical wavelets and wavelets for the image-based relighting. *Comput. Vis. Image Underst.*, 96(3):327–344, 2004. (document)

[12] P. Yu, X. Han, F. Segonne, R. L. Buckner, R. Pienaar, P. Golland, P. E. Grant, and B. Fischl. Cortical surface shape analysis based on spherical wavelet transformation. In *IEEE Computer Society Workshop on Mathematical Methods in Biomedical Image Analysis (MMBIA)*, June 2006. (document)

[13] P. Yu, F. Segonne, X. Han, and B. Fischl. Shape analysis of neuroanatomical structures based on spherical wavelets. In *Human Brain Mapping (HBM)*, 2005. (document)