Gridding Graphic Graticules

Nicholas J. Tustison, Brian B. Avants, and James C. Gee

January 22, 2007

Penn Image Computing and Science Laboratory
University of Pennsylvania

Abstract

Certain classes of images find disparate use amongst members of the ITK community for such purposes as visualization, simulation, testing, etc. Currently there exists two derived classes from the `ImageSource` class used for generating specific images for various applications, viz. `RandomImageSource` and `GaussianImageSource`. We propose to add to this set with the class `GridImageSource` which, obviously enough, produces a grid image. Such images are useful for visualizing deformation when used in conjunction with the `WarpImageFilter`, simulating magnetic resonance tagging images, or creating optical illusions with which to amaze your friends.

Keywords: *grid, tagging*

1 Introduction

We propose a grid-producing image class for inclusion within the ITK library. This class can be used to generate n-dimensional grid images.

2 GridImageSource

To produce the grid image, n mutually orthogonal one-dimensional pixel arrays are created where n is the image dimension. Based on the parameters set for each dimension, the values for the elements for the pixel arrays are calculated based on a summation of translated kernel functions (e.g. Gaussian). These arrays are then used to calculate the pixel values for the entire image.

2.1 Options

Once the user sets the parameters for the output image (i.e. size, origin, and spacing), the parameters for the grid can be set with the standard `Get/Set` API and are given as follows:

- `m_GridSpacing` — specifies the *physical* spacing between the grid lines/planes/hyperplanes.
- `m_GridOffset` — specifies the *physical* offset of the grid from the image origin. This value is delimited by `m_GridSpacing`.
- `m_WhichDimensions` — specifies which dimension is gridded (further clarification is given in the examples below).
- `m_Scale` — the multiplicative value for the image intensity values.
- `m_KernelFunction` — determines the grid intensity profile.
- `m_Sigma` — parameter for the variable `m_KernelFunction`.

This class also has multi-thread capabilities.

2.2 Usage

We illustrate usage with the following example which creates the well-known Hermann grid optical illusion [1].

```
5  int itkGridImageSourceTest0( int argc, char *argv[] )
6  {
7    typedef double PixelType;
8    const unsigned int ImageDimension = 2;
9    typedef itk::Image<PixelType, ImageDimension> ImageType;
10
11   // Instantiate the filter
12   typedef itk::GridImageSource<ImageType> GridSourceType;
13   GridSourceType::Pointer gridImage = GridSourceType::New();
14
15   double scale = 255.0;
16   ImageType::SizeType size;
17   ImageType::PointType origin;
18   ImageType::SpacingType spacing;
19   GridSourceType::ArrayType gridSpacing;
20   GridSourceType::ArrayType gridOffset;
21   GridSourceType::ArrayType sigma;
22   GridSourceType::BoolArrayType which;
23
24   // Specify image parameters
25   origin.Fill( 0.0 );
26   size.Fill( 128 );
27   spacing.Fill( 1.0 );
28
29   // Specify grid parameters
30   gridSpacing.Fill( 8.0 );
31   gridOffset.Fill( 0.0 );
32   sigma.Fill( 3 );
33   which.Fill( true );
34
35   // Specify 0th order B-spline function (Box function)
36   typedef itk::BSplineKernelFunction<0> KernelType;
37   KernelType::Pointer kernel = KernelType::New();
38
39   // Set parameters
40   gridImage->SetKernelFunction( kernel );
41   gridImage->SetSpacing( spacing );
```

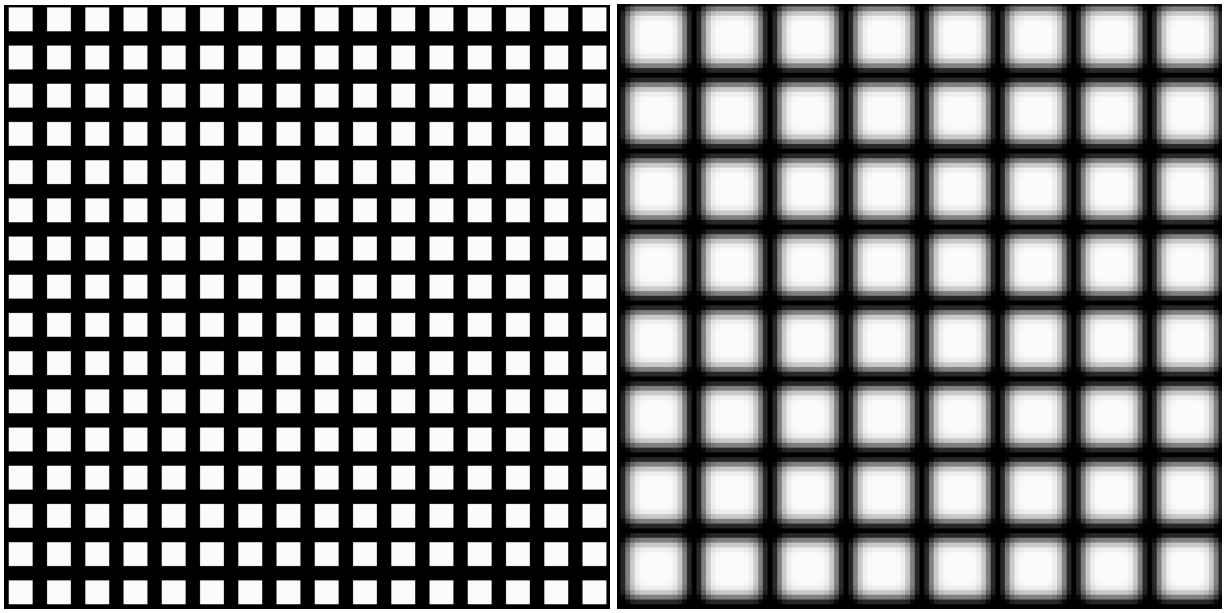

Figure 1: (a) Simple 128×128 Hermann optical illusion grid created using a 0^{th} order B-spline kernel function. (b) 128×128 grid created with a 3^{rd} order B-spline kernel function and twice the grid spacing in both dimensions.

```
42    gridImage->SetOrigin( origin );
43    gridImage->SetSize( size );
44    gridImage->SetGridSpacing( gridSpacing );
45    gridImage->SetGridOffset( gridOffset );
46    gridImage->SetWhichDimensions( which );
47    gridImage->SetSigma( sigma );
48    gridImage->SetScale( scale );
49
50    try
51      {
52      gridImage->Update();
53      }
54    catch (itk::ExceptionObject & err)
55      {
56      std::cout << "ExceptionObject caught !" << std::endl;
57      std::cout << err << std::endl;
58      return EXIT_FAILURE;
59      }
60
61    typedef itk::ImageFileWriter<ImageType> WriterType;
62    WriterType::Pointer writer = WriterType::New();
63    writer->SetFileName( argv[1] );
64    writer->SetInput( gridImage->GetOutput() );
65    writer->Update();
66
67    return EXIT_SUCCESS;
68  }
```

The resulting image is given in Figure 1(a). In order to simulate the box function, we utilized a 0^{th} order B-spline kernel function demonstrated in lines 36-40 above. Changing the kernel function to a higher order B-spline produces the image given in Figure 1(b)

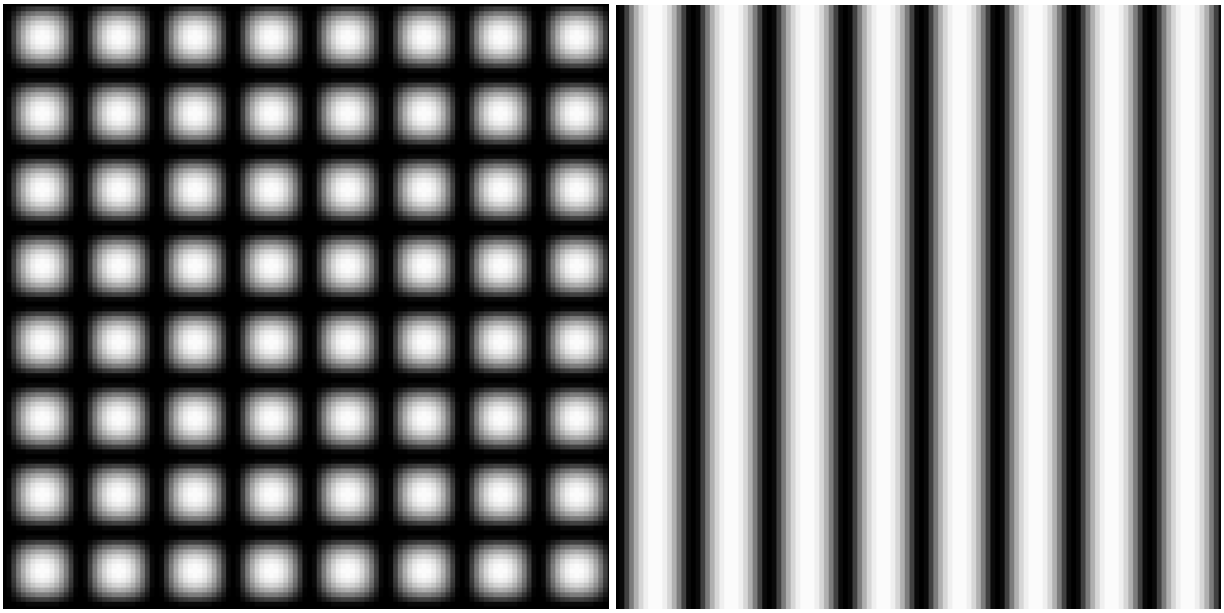

Figure 2: 2-D image slices produced by specifying a grid pattern in only two of the three image dimensions. The default Gaussian kernel function was used.

If the user desires to generate grid hyperplanes in only a subset of the total image dimensions, the variable `m_WhichDimensions` allows such control over the produced pattern illustrated by the following typical usage:

```
163   which.Fill( true );
164   which[ImageDimension-1] = false;
```

where `ImageDimension = 3`. This produces the image slices shown in Figure 2.

Finally, grid parameters can be specified for the different dimensions to produce unorthodox grid patterns such as that found in Figure 3. The specified parameters are as follows

```
220   // Specify grid parameters
221   gridOffset.Fill( 0.0 );
222   gridSpacing[0] = 32.0;
223   gridSpacing[1] = 16.0;
224   sigma[0] = 1.0;
225   sigma[1] = 5.0;
```

References

[1] Wikipedia contributors, “Grid illusion,” *Wikipedia, The Free Encyclopedia*, http://en.wikipedia.org/w/index.php?title=Grid_illusion&oldid=85841165 (accessed January 22, 2007). 2.2

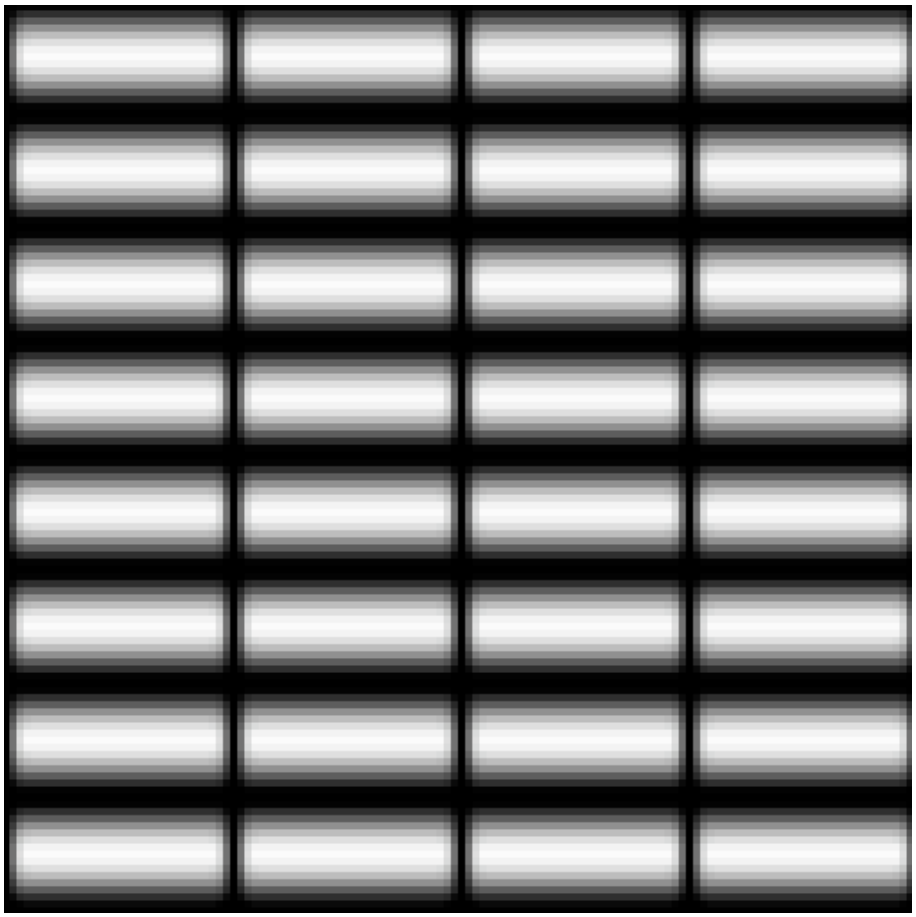

Figure 3: Unorthodox 2-D grid pattern created by varying certain parameters between dimensions. The default Gaussian kernel function was used.