

Supplementary Material to : Motion Clouds: Model-based stimulus synthesis of natural-like random textures for the study of motion perception

Paula Sanz Leon^{*1,2}, Ivo Vanzetta^{†1,2}, Guillaume S. Masson^{‡1,2}, and Laurent U. Perrinet^{§1,2}

¹INCM, CNRS / Aix-Marseille University, France

²Institut de Neurosciences de la Timone, CNRS / Aix-Marseille University, Marseille, France

Computer implementation using Python

In this part, we will briefly describe how Motion Clouds can be implemented while taking into account technical constraints such as discretization and videographic displays. We will also outline the algorithm used to generate our calibrated motion clouds using Python libraries.

Defining Fourier units, discrete units and physical units

In vision research, stimulus parameters depend on experimental conditions such as viewing distance and other properties of the display, such as the refreshing rate. Here, we will define the parameters of interest to implement when computing Motion Clouds based in the parameters showed in Table 1 where give a description of their physical values in one example experimental setup.

Symbol	Magnitude	Value	Unit
D	Viewing distance	570	[mm]
X, Y	Stimulus size	640 x 480	[px]
VA^1	Stimulus width in degrees of visual angle at viewing distance D	38,1	[deg]
f_{rate}	Frame rate	50	[Hz]
T	Stimulus duration	0.6	[sec]

Table 1: Physical units in an optical imaging set-up.

Both N_X and N_Y are determined by the frame (stimulus) size (X and Y), while N_{frame} is determined by the frame rate (f_{rate}) and the stimulus duration (T). These parameters define the stimulus' spatiotemporal periods. In this example we set $N_{frame} = 30$. Additionally, velocities V_x and V_y have arbitrary units with the convention that if $V_x = 1$, it means that average motion is equal to an average displacement of one spatial period over one temporal period and the same applies to V_y . (See Figure 1). In line with this, we had introduced earlier the normalization factor $f_{t_0} = \frac{N_X}{N_{frame}}$. In the spatiotemporal domain implies that there is a translation of a distance VA_X during a period T . We remind that degrees of visual angle are defined by $VA = 2 * \arctan (S/2D)$, where S is stimulus size on the screen (X or Y) and D is the viewing distance.

^{*}Paula.Sanz@univ-amu.fr

[†]Ivo.Vanzetta@univ-amu.fr

[‡]Guillaume.Masson@univ-amu.fr

[§]Corresponding Author, Laurent.Perrinet@univ-amu.fr

Defining stimulus and Fourier cubes

Note first that the visual stimulus \mathbf{I} is a real-valued function, therefore the inverse Fourier transform of our spectrum must be purely real, and its transform must be Hermitian. This means that the frequency component (f_x, f_y, f_t) is the complex conjugate of the component at frequency $(-f_x, -f_y, -f_t)$. Therefore, there is no information in the negative frequency components that is not already available from the positive frequency components. To ensure that, the envelope will always be symmetric with respect to the origin in the Fourier domain, while the phase spectrum will be Hermitian by construction. An alternative consists in taking the real part of the complex inverse Fourier transform of any envelope (symmetric or not). Note that by construction of the Fourier transform, stimuli are generated in the 3D toroidal space and they are invariant up to displacement in multiples of the spatiotemporal period. As a consequence, there is no border or center and moreover any given Motion Cloud may be concatenated in space or time : For instance, playing a Motion Clouds movie in a loop is smooth and there is no abrupt transient. This property is useful to create large stimuli with limited resources by "tiling" a stimulus multiple times. Mathematically, a set of Motion Clouds is generated using normalized input arguments. First, we define the quantization of the Fourier space defined above in cubes of size $N_j, j \in X, Y, frame$, respectively for horizontal, vertical and time axis. In practice we will use the Fast Fourier Transform (FFT). As a consequence, the resulting stimulus cube will be of the same size as the frequency cube and $N_j, j \in X, Y, frame$ should be preferentially defined as an integer power of two. Each frequency axis (in Cartesian coordinates $(f_x, f_y$ and $f_t)$) belongs always to the interval $[-0.5, 0.5]$ although the number of points is different. The frequency resolution is given by $(1/N_X, 1/N_Y, 1/N_{frame})$ and $f_x, f_y, f_t = 0.5$ (in $cyc_{px}, cyc_{py}, cyc_{frame}$) is the Nyquist frequency, i.e., the maximal frequency that can be represented without having undesirable aliasing effects.

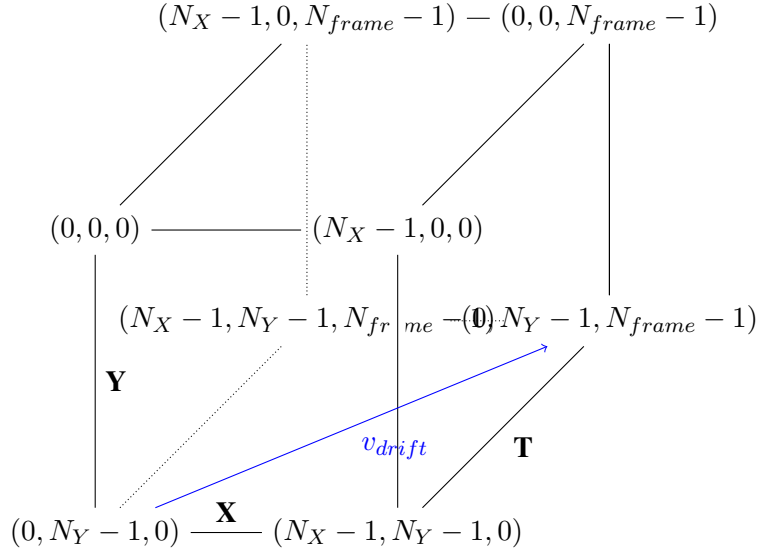


Figure 1

In Figure 2 we show the flow chart of the sequential construction method. We begin by building a three dimensional matrix whose dimensions are given by the input arguments N_X , N_Y and N_{frame} so that $\mathcal{E}(f_x, f_y, f_t) \in \mathbb{R}^{N_X \times N_Y \times N_{frame}}$. The first two define the image size, width and height, respectively. The third dimension is the length of the image-series (number of frames).

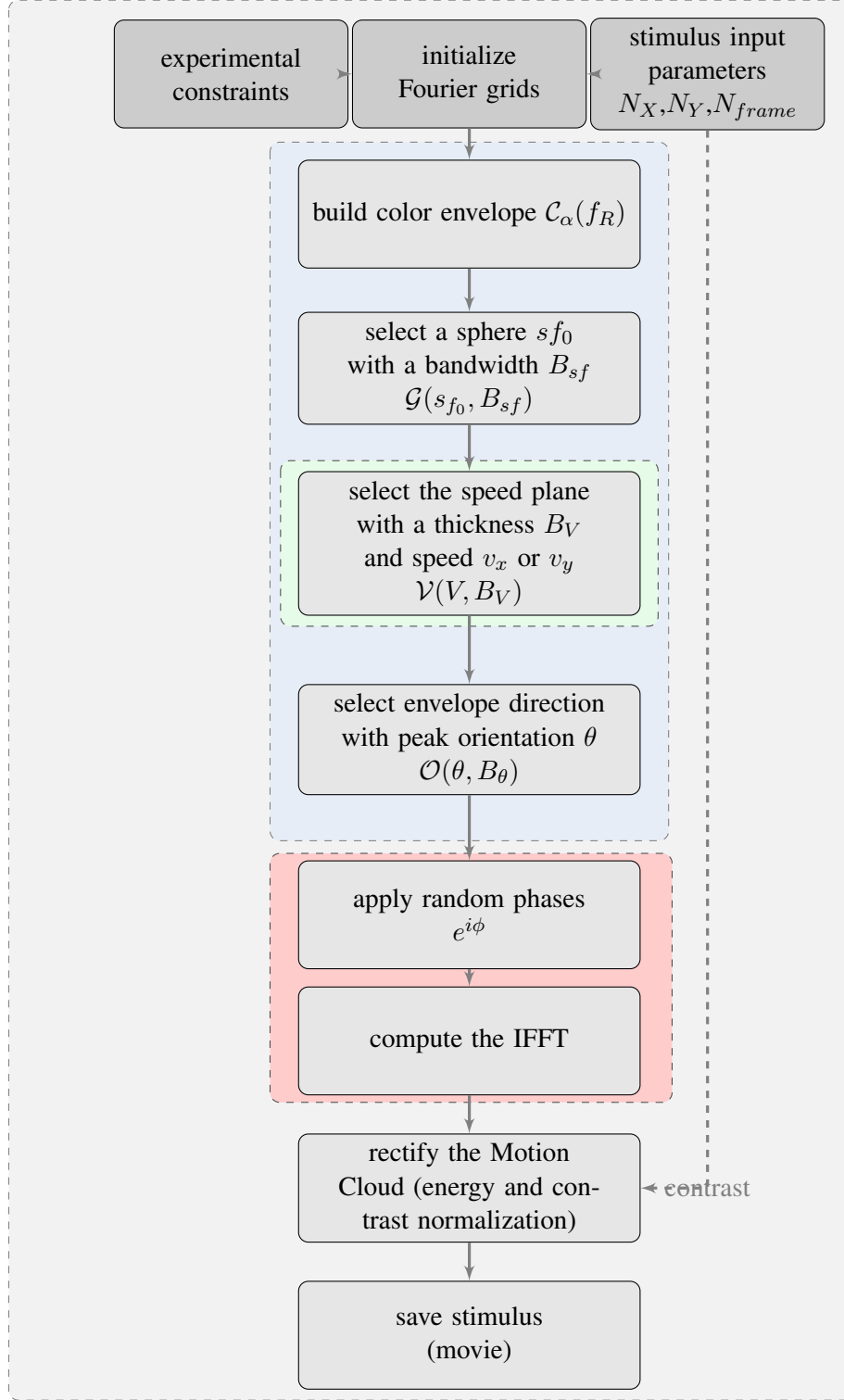


Figure 2

Summary: Flowchart

First, experimental parameters (N_X , N_Y , N_{frame}) are initialized and physical units are normalized (s_{f_0} , V_X , V_Y). Second, the color envelope is generated according to the parameter α . Third, this color envelope (\mathcal{C}_α) is multiplied by the global Fourier envelope constructed by the product of the speed (\mathcal{V}), spatial frequency (\mathcal{G}) and orientation envelopes (\mathcal{O}). The last step in the Fourier domain is to multiply the Fourier modulus by a random phase ($e^{i\phi}$). Thus, after computing the 3-dimensional inverse Fourier transform we obtain a dynamic random phase texture, that is the Motion Cloud movie as a numpy array that can further be processed to be for example stored as a sequence of frames.

Code example

Motion Clouds are built using a collection of scripts that provides a simple way of generating complex stimuli suitable for neuroscience and psychophysics experiments. It is meant to be an open-source package that can be combined with other packages such as PsychoPy or VisionEgg.

All functions are implemented in one main script called *MotionClouds.py* that handles the Fourier cube, the envelope functions as well as the random phase generation and all Fourier related processing. Additionally, all the auxiliary visualization tools to plot the spectra and the movies are included. Specific scripts such as *test_color.py*, *test_speed.py*, *test_radial.py* and *test_orientation.py* explore the role of different parameters for each individual envelope (respectively color, speed, radial frequency, orientation). Our aim is to keep the code as simple as possible in order to be comprehensible and flexible. To sum up, when we build a custom Motion Cloud there are 3 simple steps to follow:

1. set the MC parameters and construct the Fourier envelope, then visualize it as iso-surfaces,

```
1 import MotionClouds as mc
2 import numpy as np
3 fx, fy, ft = mc.get_grids(mc.N_X, mc.N_Y, mc.N_frame) # define Fourier domain
4 envelope = mc.envelope_gabor(fx, fy, ft, V_X=1., V_Y=0., B_V=.1, sf_0=.15, B_sf=.1,
5                               theta=0., B_theta=np.pi/8, alpha=1.) # define an envelope
6 mc.visualize(fx, fy, ft, envelope) # Visualize the Fourier Spectrum
```

2. perform the IFFT and contrast normalization; visualize the stimulus as a 'cube' visualization of the image sequence,

```
1 movie = mc.random_cloud(envelope)
2 movie = mc.rectif(movie)
3 mc.cube(fx, fy, ft, movie, name=name + '_cube') # Visualize the Stimulus
```

3. export the stimulus as a movie (.mpeg format available), as separate frames (.bmp and .png formats available) in a compressed zipped folder, or as a MatlabTM matrix (.mat format).

```
1 mc.anim_save(movie, name, display=False, vext='.mpeg')
```

If some parameters are not given, they are set to default values corresponding to a "standard" Motion Cloud. Moreover, the user can easily explore a range of different Motion Clouds simply by setting an array of values for a determined parameter. Here, for example, we generate 8 MCs with increasing spatial frequency s_{f_0} while keeping the other parameters fixed to default values:

```

1 for sf_0 in [0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6]:
2     name_ = 'figures/' + name + '-sf_0-' + str(sf_0).replace('.', '_')
3     mc.figures_MC(fx, fy, ft, name_, sf_0=sf_0) # function performing plots for a
        given set of parameters

```

Here, we show the source code of *MotionClouds.py*. The test cases are available on request to the corresponding author.

```

1  #!/usr/bin/env python
2  # -*- coding: utf8 -*-
3  """
4
5  Main script for generating Motion Clouds
6
7  (c) Laurent Perrinet – INT/CNRS
8
9  Motion Clouds (keyword) parameters:
10 size      — power of two to define the frame size (N_X, N_Y)
11 size_T    — power of two to define the number of frames (N_frame)
12 N_X       — frame size horizontal dimension [px]
13 N_Y       — frame size vertical dimension [px]
14 N_frame   — number of frames [frames] (a full period in time frames)
15 alpha     — exponent for the color envelope.
16 sf_0      — mean spatial frequency relative to the sampling frequency.
17 ft_0      — spatiotemporal scaling factor.
18 B_sf      — spatial frequency bandwidth
19 V_X       — horizontal speed component
20 V_Y       — vertical speed component
21 B_V       — speed bandwidth
22 theta     — mean orientation of the Gabor kernel
23 B_theta   — orientation bandwidth
24 loggabor  — (boolean) if True it uses a log-Gabor kernel (instead of the traditional
        gabor)
25
26 Display parameters:
27
28 vext       — movie format. Stimulus can be saved as a 3D (x-y-t) multimedia file :
        .mpg movie, .mat array, .zip folder with a frame sequence.
29 ext       — frame image format.
30 T_movie   — movie duration [s].
31 fps       — frame per seconds
32
33 """
34
35 import os
36 DEBUG = False
37 if DEBUG:
38     size = 5
39     size_T = 5
40     figsize = (400, 400) # faster
41 else:
42     size = 7
43     size_T = 7
44     figsize = (800, 800) # nice size, but requires more memory
45
46 import numpy as np
47 N_X = 2**size
48 N_Y = N_X

```

```

49 N_frame = 2**size_T
50 ft_0 = N_X/float(N_frame)
51 alpha = 1.0
52 sf_0 = 0.15
53 B_sf = 0.1
54 V_X = 1.
55 V_Y = 0.
56 B_V = .2
57 theta = 0.
58 B_theta = np.pi/32.
59 loggabor = True
60 vext = '.mpg'
61 ext = '.png'
62 T_movie = 8. # this value defines the duration of a temporal period
63 fps = int(N_frame / T_movie)
64
65 # display parameters
66 try:
67     import progressbar
68     PROGRESS = True
69 except:
70     PROGRESS = False
71
72 # os.environ['ETS_TOOLKIT'] = 'qt4' # Works in Mac
73 # os.environ['ETS_TOOLKIT'] = 'wx' # Works in Debian
74 MAYAVI = 'Import'
75 #MAYAVI = 'Avoid' # uncomment to avoid generating mayavi visualizations (and save
    some memory...)
76 def import_mayavi():
77     global MAYAVI, mlab
78     if (MAYAVI == 'Import'):
79         try:
80             from mayavi import mlab
81             MAYAVI = 'Ok : New and shiny'
82             print('Imported Mayavi')
83         except:
84             try:
85                 from enthought.mayavi import mlab
86                 print('Seems you have an old implementation of MayaVi, but things
                    should work')
87                 MAYAVI = 'Ok but old'
88                 print('Imported Mayavi')
89             except:
90                 print('Could not import Mayavi')
91                 MAYAVI = False
92     elif (MAYAVI == 'Ok : New and shiny') or (MAYAVI == 'Ok but old'):
93         pass # no need to import that again
94     else:
95         print('We have chosen not to import Mayavi')
96 # Trick from http://github.enthought.com/mayavi/mayavi/tips.html : to use offscreen
    rendering, try xvfb :1 -screen 0 1280x1024x24 in one terminal, export DISPLAY
    =:1 before you run your script
97
98 figpath = 'results/'
99 if not(os.path.isdir(figpath)):os.mkdir(figpath)
100
101 def get_grids(N_X, N_Y, N_frame, sparse=True):
102     """
103     Use that function to define a reference outline for envelopes in Fourier
        space.

```

```

104         In general, it is more efficient to define dimensions as powers of 2.
105
106     """
107     if sparse:
108         fx, fy, ft = np.ogrid[(-N_X//2):((N_X-1)//2 + 1), (-N_Y//2):((N_Y-1)//2 +
109             1), (-N_frame//2):((N_frame-1)//2 + 1)] # output is always even.
110     else:
111         fx, fy, ft = np.mgrid[(-N_X//2):((N_X-1)//2 + 1), (-N_Y//2):((N_Y-1)//2 +
112             1), (-N_frame//2):((N_frame-1)//2 + 1)] # output is always even.
113     fx, fy, ft = fx*1./N_X, fy*1./N_Y, ft*1./N_frame
114     return fx, fy, ft
115
116 def frequency_radius(fx, fy, ft, ft_0=ft_0):
117     """
118     Returns the frequency radius. To see the effect of the scaling factor run
119     'test_color.py'
120
121     """
122     N_X, N_Y, N_frame = fx.shape[0], fy.shape[1], ft.shape[2]
123     R2 = fx**2 + fy**2 + (ft/ft_0)**2 # cf . Paul Schrater 00
124     R2[N_X//2 , N_Y//2 , N_frame//2 ] = np.inf
125     return np.sqrt(R2)
126
127 def envelope_color(fx, fy, ft, alpha=alpha, ft_0=ft_0):
128     """
129     Returns the color envelope.
130     Run 'test_color.py' to see the effect of alpha
131     alpha = 0 white
132     alpha = 1 pink
133     alpha = 2 red/brownian
134     (see http://en.wikipedia.org/wiki/1/f\_noise )
135
136     """
137     f_radius = frequency_radius(fx, fy, ft, ft_0=ft_0)**alpha
138     return 1. / f_radius
139
140 def envelope_radial(fx, fy, ft, sf_0=sf_0, B_sf=B_sf, ft_0=ft_0, loggabor=loggabor):
141     """
142     Radial frequency envelope:
143     selects a sphere around a preferred frequency with a shell width B_sf.
144     Run 'test_radial.py' to see the explore the effect of sf_0 and B_sf
145
146     """
147     if sf_0 == 0.: return 1.
148     if loggabor:
149         # see http://en.wikipedia.org/wiki/Log-normal\_distribution
150         fr = frequency_radius(fx, fy, ft, ft_0=1.)
151         env = 1./fr*np.exp(-.5*(np.log(fr/sf_0)**2)/(np.log((sf_0+B_sf)/sf_0)**2))
152         return env
153     else:
154         return np.exp(-.5*(frequency_radius(fx, fy, ft, ft_0=1.) - sf_0)**2/B_sf
155             **2)
156
157 def envelope_speed(fx, fy, ft, V_X=V_X, V_Y=V_Y, B_V=B_V):
158     """
159     Speed envelope:
160     selects the plane corresponding to the speed (V_X, V_Y) with some thickness
161     B_V
162
163     (V_X, V_Y) = (0,1) is downward and (V_X, V_Y) = (1,0) is rightward in the
164     movie.

```

```

158     A speed of V_X=1 corresponds to an average displacement of 1/N_X per frame.
159     To achieve one spatial period in one temporal period, you should scale by
160     V_scale = N_X/float(N_frame)
161     If N_X=N_Y=N_frame and V=1, then it is one spatial period in one temporal
162     period. it can be seen in the MC cube. Define ft_0 = N_X/N_frame
163
164     Run 'test_speed.py' to explore the speed parameters
165
166     """
167     env = np.exp(-.5*((ft+fx*V_X+fy*V_Y))**2/(B_V*frequency_radius(fx, fy, ft, ft_0
168     =1.))**2)
169     return env
170
171 def envelope_orientation(fx, fy, ft, theta=theta, B_theta=B_theta):
172     """
173     Orientation envelope:
174     selects one central orientation theta, B_theta the spread
175     We use a von-Mises distribution on the orientation.
176
177     Run 'test_orientation.py' to see the effect of changing theta and B_theta.
178     """
179     if not(B_theta is np.inf):
180         angle = np.arctan2(fy, fx)
181         envelope_dir = np.exp(np.cos(2*(angle-theta))/B_theta)
182         return envelope_dir
183     else: # for large bandwidth, returns a strictly flat envelope
184         return 1.
185
186 def envelope_gabor(fx, fy, ft, V_X=V_X, V_Y=V_Y,
187                   B_V=B_V, sf_0=sf_0, B_sf=B_sf, loggabor=loggabor,
188                   theta=theta, B_theta=B_theta, alpha=alpha):
189     """
190     Returns the Motion Cloud kernel
191     """
192     envelope = envelope_color(fx, fy, ft, alpha=alpha)
193     envelope *= envelope_orientation(fx, fy, ft, theta=theta, B_theta=B_theta)
194     envelope *= envelope_radial(fx, fy, ft, sf_0=sf_0, B_sf=B_sf, loggabor=loggabor)
195     envelope *= envelope_speed(fx, fy, ft, V_X=V_X, V_Y=V_Y, B_V=B_V)
196     return envelope
197
198 def random_cloud(envelope, seed=None, impulse=False, do_amp=False):
199     """
200     Returns a Motion Cloud movie as a 3D matrix.
201     It first creates a random phase spectrum and then it computes the inverse FFT
202     to obtain
203     the spatiotemporal stimulus.
204
205     - use a specific seed to specify the RNG's seed,
206     - test the impulse response of the kernel by setting impulse to True
207     - test the effect of randomizing amplitudes too by setting do_amp to True
208
209     shape
210     """
211     (N_X, N_Y, N_frame) = envelope.shape
212     amps = 1.
213     if impulse:
214         phase = 0.
215     else:
216         np.random.seed(seed=seed)

```



```

215     phase = 2 * np.pi * np.random.rand(N_X, N_Y, N_frame)
216     if do_amp:
217         amps = np.random.randn(N_X, N_Y, N_frame)
218         # see Galerne, B., Gousseau, Y. & Morel, J.-M. Random phase textures:
            Theory and synthesis. IEEE Transactions in Image Processing (2010).
            URL http://www.biomedsearch.com/nih/Random-Phase-Textures-Theory-
            Synthesis/20550995.html. (basically, they conclude "Even though the
            two processes ADSN and RPN have different Fourier modulus
            distributions (see Section 4), they produce visually similar results
            when applied to natural images as shown by Fig. 11.")
219
220     Fz = amps * envelope * np.exp(1j * phase)
221
222     # centering the spectrum
223     Fz = np.fft.ifftshift(Fz)
224     Fz[0, 0, 0] = 0.
225     z = np.fft.ifftn((Fz)).real
226     return z

```

In *MotionClouds.py* additional functions have been written for displaying purposes such as visualization of the Fourier spectrum and saving the stimulus in different formats.

```

1 ##### Display Tools #####
2
3 def get_size(mat):
4     """
5     Get stimulus dimensions
6
7     """
8     return [np.size(mat, axis=k) for k in range(np.ndim(mat))]
9
10 #NOTE: Python uses the first dimension (rows) as vertical axis and this is the Y in
    the spatiotemporal domain. Be careful with the convention of X and Y.
11
12 def visualize(z, azimuth=290., elevation=45.,
13             thresholds=[0.94, .89, .75, .5, .25, .1], opacities=[.9, .8, .7, .5, .2, .2],
14             name=None, ext=ext, do_axis=True, do_grids=False, draw_projections=True,
15             colorbar=False, f_N=2., f_tN=2., figsize=figsize):
16
17     """ Visualize the Fourier spectrum """
18     import mayavi()
19
20     N_X, N_Y, N_frame = z.shape
21     fx, fy, ft = get_grids(N_X, N_Y, N_frame, sparse=False)
22
23     mlab.figure(1, bgcolor=(1, 1, 1), fgcolor=(0, 0, 0), size=figsize)
24     mlab.clf()
25
26     # Normalize the amplitude.
27     z /= z.max()
28     # Create scalar field
29     src = mlab.pipeline.scalar_field(fx, fy, ft, z)
30     if draw_projections:
31         src_x = mlab.pipeline.scalar_field(fx, fy, ft, np.tile(np.sum(z, axis=0), (
32             N_X, 1, 1)))
33         src_y = mlab.pipeline.scalar_field(fx, fy, ft, np.tile(np.reshape(np.sum(z,
34             axis=1), (N_X, 1, N_frame)), (1, N_Y, 1)))
35         src_z = mlab.pipeline.scalar_field(fx, fy, ft, np.tile(np.reshape(np.sum(z,
36             axis=2), (N_X, N_Y, 1)), (1, 1, N_frame)))

```

```

34
35     # Create projections
36     border = 0.47
37     scpx = mlab.pipeline.scalar_cut_plane(src_x , plane_orientation='x_axes' ,
38         view_controls=False)
39     scpx.implicit_plane.plane.origin = [-border , 1/N.Y, 1/N.frame]
40     scpy = mlab.pipeline.scalar_cut_plane(src_y , plane_orientation='y_axes' ,
41         view_controls=False)
42     scpy.implicit_plane.plane.origin = [1/N.X, border , 1/N.frame]
43     scpz = mlab.pipeline.scalar_cut_plane(src_z , plane_orientation='z_axes' ,
44         view_controls=False)
45     scpz.implicit_plane.plane.origin = [1/N.X, 1/N.Y, -border]
46     scpz.enable_contours = True
47
48     # Generate iso-surfaces at different energy levels
49     for threshold , opacity in zip(thresholds , opacities):
50         mlab.pipeline.iso_surface(src , contours=[z.max()-threshold*z.ptp() , ],
51             opacity=opacity)
52         mlab.outline(extent=[-1./2 , 1./2 , -1./2 , 1./2 , -1./2 , 1./2] ,)
53
54     # Draw a sphere at the origin
55     x = np.array([0])
56     y = np.array([0])
57     z = np.array([0])
58     s = 0.01
59     mlab.points3d(x , y , z , extent=[-s , s , -s , s , -s , s] , scale_factor=0.15)
60
61     if colorbar: mlab.colorbar(title='density' , orientation='horizontal')
62     if do_axis:
63         ax = mlab.axes(xlabel='fx' , ylabel='fy' , zlabel='ft' ,
64             extent=[-1./2 , 1./2 , -1./2 , 1./2 , -1./2 , 1./2] ,
65             )
66         ax.axes.set(font_factor=2.)
67
68     try:
69         mlab.view(azimuth=azimuth , elevation=elevation , distance='auto' , focalpoint
70             ='auto')
71     except:
72         print(" You should upgrade your mayavi version")
73
74     if not(name is None):
75         mlab.savefig(name + ext , magnification=1 , size=figsize)
76     else:
77         mlab.show(stop=True)
78
79     mlab.close(all=True)
80
81 def cube(im , azimuth=-45. , elevation=130. , roll=-180. , name=None ,
82     ext=ext , do_axis=True , show_label=True , colormap='gray' ,
83     vmin=0. , vmax=1. , figsize=figsize):
84
85     """
86     Visualize the stimulus as a cube
87
88     """
89     import mayavi()
90
91     N_X, N_Y, N_frame = im.shape

```

```

90     fx, fy, ft = get_grids(N_X, N_Y, N_frame, sparse=False)
91
92     mlab.figure(1, bgcolor=(1, 1, 1), fgcolor=(0, 0, 0), size=figsize)
93     mlab.clf()
94     src = mlab.pipeline.scalar_field(fx*2., fy*2., ft*2., im)
95
96     mlab.pipeline.image_plane_widget(src, plane_orientation='z_axes',
97                                     slice_index=0, colormap=colormap, vmin=vmin,
98                                     vmax=vmax)
99     mlab.pipeline.image_plane_widget(src, plane_orientation='z_axes',
100                                     slice_index=N_frame, colormap=colormap,
101                                     vmin=vmin, vmax=vmax)
102     mlab.pipeline.image_plane_widget(src, plane_orientation='x_axes', slice_index
103                                     =0,
104                                     colormap=colormap, vmin=vmin, vmax=vmax)
105     mlab.pipeline.image_plane_widget(src, plane_orientation='x_axes', slice_index=
106                                     N_X,
107                                     colormap=colormap, vmin=vmin, vmax=vmax)
108     mlab.pipeline.image_plane_widget(src, plane_orientation='y_axes', slice_index
109                                     =0,
110                                     colormap=colormap, vmin=vmin, vmax=vmax)
111     mlab.pipeline.image_plane_widget(src, plane_orientation='y_axes', slice_index=
112                                     N_Y,
113                                     colormap=colormap, vmin=vmin, vmax=vmax)
114
115     if do_axis:
116         ax = mlab.axes(xlabel='x', ylabel='y', zlabel='t',
117                        extent=[-1., 1., -1., 1., -1., 1.],
118                        ranges=[0., N_X, 0., N_Y, 0., N_frame],
119                        x_axis_visibility=True, y_axis_visibility=True,
120                        z_axis_visibility=True)
121         ax.axes.set(font_factor=2.)
122
123         if not(show_label): ax.axes.set(label_format='')
124
125     try:
126         mlab.view(azimuth=azimuth, elevation=elevation, distance='auto', focalpoint
127                   ='auto')
128         mlab.roll(roll=roll)
129     except:
130         print(" You should upgrade your mayavi version")
131
132     if not(name is None):
133         mlab.savefig(name + ext, magnification=1, size=figsize)
134     else:
135         mlab.show(stop=True)
136
137     mlab.close(all=True)
138
139 def anim_exist(filename, vext='.mpg'):
140     """
141     Check if the movie already exists
142     """
143     return not(os.path.isfile(filename+vext))

```

```

144         centered=False , fps=fps):
145     """
146     Saves a numpy 3D matrix (x-y-t) to a multimedia file.
147
148     The input pixel values are supposed to lie in the [0, 1.] range.
149
150     """
151     import os                                     # For issuing commands to the OS.
152     import tempfile
153     from scipy.misc.pilutil import toimage
154     def make_frames(z):
155         N_X, N_Y, N_frame = z.shape
156         files = []
157         tmpdir = tempfile.mkdtemp()
158
159         if PROGRESS:
160             widgets = ["calculating", " ", progressbar.Percentage(), ' ',
161                        progressbar.Bar(), ' ', progressbar.ETA()]
162             pbar = progressbar.ProgressBar(widgets=widgets, maxval=N_frame).start()
163         print('Saving sequence ' + filename + vext)
164         for frame in range(N_frame):
165             if PROGRESS: pbar.update(frame)
166             fname = os.path.join(tmpdir, 'frame%03d.png' % frame)
167             image = np.rot90(z[:, :, frame])
168             if flip: image = np.flipud(image)
169             toimage(image, high=255, low=0, cmin=0., cmax=1., pal=None,
170                    mode=None, channel_axis=None).save(fname)
171             files.append(fname)
172             if PROGRESS: pbar.update(frame)
173
174         if PROGRESS: pbar.finish()
175         return tmpdir, files
176
177     def remove_frames(tmpdir, files):
178         """
179         Remove frames from the temp folder
180
181         """
182         for fname in files: os.remove(fname)
183         if not(tmpdir == None): os.rmdir(tmpdir)
184
185     if vext == '.mpg':
186         # 1) create temporary frames
187         tmpdir, files = make_frames(z)
188         # 2) convert frames to movie
189 #         cmd = 'ffmpeg -v 0 -y -sameq -loop_output 0 -r ' + str(fps) + ' -i ' +
190 # tmpdir + '/frame%03d.png ' + filename + vext # + ' 2>/dev/null ')
191 #         cmd = 'ffmpeg -v 0 -y -sameq -loop_output 0 -i ' + tmpdir + '/frame%03d.
192 # png ' + filename + vext # + ' 2>/dev/null ')
193 #         print('Doing : ', cmd)
194 #         os.system(cmd) # + ' 2>/dev/null ')
195 #         # To force the frame rate of the output file to 24 fps:
196 #         # ffmpeg -i input.avi -r 24 output.avi
197 #         # 3) clean up
198         remove_frames(tmpdir, files)
199     if vext == '.gif': # http://www.uoregon.edu/~noeckel/MakeMovie.html
200         # 1) create temporary frames
201         tmpdir, files = make_frames(z)
202         # 2) convert frames to movie
203         options = ' -pix_fmt rgb24 -r ' + str(fps) + ' -loop_output 0 '

```

```

202 #         os.system('ffmpeg -i ' + tmpdir + '/frame%03d.png ' + options + filename
+ vext + ' 2>/dev/null')
203         options = ' -set delay 8 -colorspace GRAY -colors 256 -dispose 1 -loop 0 '
204         os.system('convert ' + tmpdir + '/frame*.png ' + options + filename +
+ vext )# + ' 2>/dev/null')
205
206         # 3) clean up
207         remove_frames(tmpdir, files)
208
209     elif vext == '.png':
210         toimage(np.flipud(z[:, :, 0]).T, high=255, low=0, cmin=0., cmax=1., pal=
+ None, mode=None, channel_axis=None).save(filename + vext)
211
212     elif vext == '.zip':
213         tmpdir, files = make_frames(z)
214         import zipfile
215         zf = zipfile.ZipFile(filename + vext, "w")
216         # convert to BMP for optical imaging
217         files_bmp = []
218         for fname in files:
219             fname_bmp = os.path.splitext(fname)[0] + '.bmp'
220             # print fname_bmp
221             os.system('convert ' + fname + ' ppm:- | convert -size 256x256+0 -
+ colors 256 -colorspace Gray - BMP2:' + fname_bmp) # to generate 8-
+ bit bmp (old format)
222             files_bmp.append(fname_bmp)
223             zf.write(fname_bmp)
224         zf.close()
225         remove_frames(tmpdir=None, files=files_bmp)
226         remove_frames(tmpdir, files)
227
228     elif vext == '.mat':
229         from scipy.io import savemat
230         savemat(filename + vext, {'z':z})
231
232     elif vext == '.h5':
233         from tables import openFile, Float32Atom
234         hf = openFile(filename + vext, 'w')
235         o = hf.createCArray(hf.root, 'stimulus', Float32Atom(), z.shape)
236         o = z
237         # print o.shape
238         hf.close()
239
240 def rectif(z, contrast=.9, method='Michelson', verbose=False):
241     """
242     Transforms an image (can be 1,2 or 3D) with normal histogram into
243     a 0.5 centered image of determined contrast
244     method is either 'Michelson' or 'Energy'
245     """
246
247     # Phase randomization takes any image and turns it into Gaussian-distributed
+ noise of the same power (or, equivalently, variance).
248     # See: Peter J. Bex J. Opt. Soc. Am. A/Vol. 19, No. 6/June 2002 Spatial
+ frequency, phase, and the contrast of natural images
249
250     # Final rectification
251     if verbose:
252         print('Before Rectification of the frames')
253         print( 'Mean=', np.mean(z[:]), ', std=', np.std(z[:]), ', Min=', np.min(
+ z[:]), ', Max=', np.max(z[:]), ' Abs(Max)=', np.max(np.abs(z[:])) )

```

```

254
255     z -= np.mean(z[:]) # this should be true *on average* in MotionClouds
256
257     if (method == 'Michelson'):
258         z = (.5* z/np.max(np.abs(z[:]))* contrast + .5)
259     else:
260         z = (.5* z/np.std(z[:]) * contrast + .5)
261
262     if verbose:
263         import pylab
264         pylab.hist(z.ravel())
265
266         print('After Rectification of the frames')
267         print('Mean=', np.mean(z[:]), ', std=', np.std(z[:]), ', Min=', np.min(z
268               [:]), ', Max=', np.max(z[:]))
268         print('percentage pixels clipped=', np.sum(np.abs(z[:]) > 1.)*100/z.size)
269     return z
270
271 def figures_MC(fx, fy, ft, name, V_X=V_X, V_Y=V_Y, do_figs=True, do_movie=True,
272               B_V=B_V, sf_0=sf_0, B_sf=B_sf, loggabor=loggabor,
273               theta=theta, B_theta=B_theta, alpha=alpha, vext=vext,
274               seed=None, impulse=False, verbose=False):
275     """
276     Generates the figures corresponding to the Fourier spectra and the stimulus
277     cubes and
278     movies.
279     The figures names are automatically generated.
280     """
281     if anim_exist(name, vext=vext):
282         z = envelope_gabor(fx, fy, ft, V_X=V_X, V_Y=V_Y,
283                           B_V=B_V, sf_0=sf_0, B_sf=B_sf, loggabor=loggabor,
284                           theta=theta, B_theta=B_theta, alpha=alpha)
285         figures(z, name, vext=vext, do_figs=do_figs, do_movie=do_movie,
286                seed=seed, impulse=impulse, verbose=verbose)
287
288 def figures(z, name, vext=vext, do_figs=True, do_movie=True,
289            seed=None, impulse=False, verbose=False, masking=False):
290     if ((MAYAVI == 'Import') or MAYAVI[:2]=='Ok') and do_figs and anim_exist(name,
291                                     vext=vext): visualize(z, name=name) # Visualize the Fourier
292                                     Spectrum
293
294     if (do_movie and anim_exist(name, vext=vext)) or (MAYAVI and do_figs and
295               anim_exist(name + '_cube', vext=vext)):
296         movie = rectif(random_cloud(z, seed=seed, impulse=impulse), verbose=verbose)
297
298     if (((MAYAVI == 'Import') or MAYAVI[:2]=='Ok') and do_figs and anim_exist(name
299                                     + '_cube', vext=vext)): cube(movie, name=name + '_cube') # Visualize the
300                                     Stimulus cube
301
302     if (do_movie and anim_exist(name, vext=vext)): anim_save(movie, name, display=
303                     False, vext=vext)

```

Both functions **visualize** (line 37) and **cube** (line 100) generate isometric views of a cube. The first one displays isosurfaces enclosing volumes at 6 different energy values with respect to the peak amplitude of the Fourier spectrum. The Cartesian coordinate system is represented by 3 orthogonal grid planes going through the origin. The origin is represented by a black dot where the three 3 orthogonal axes converge. In addition to that, it is also possible to obtain the orthogonal projections onto the corresponding normal planes to the Cartesian axes, illustrated by 10 contour level curves. We enable the projection onto the $f_x - f_t$ and $f_y - f_t$ planes in order to observe the changes in the tilt of the speed plane (reflecting

respectively a change in V_X or V_Y), as well as its thickness. Furthermore, the projection onto the $f_x - f_y$ plane allows us to see the average orientation θ and the spread of the orientation envelope. The outlines delineate the frequency domain extension in Fourier units as described in . The second function draws the isometric view of the movie cube. The first frame of the movie lies on the plane $x - y$, motion direction is seen as diagonal trajectories on the top face ($x - t$ plane) and on the right face ($y - t$ plane), reflecting respectively a change in V_X or V_Y .

Annex

Approximating normal and log-normal distributions

In our implementation we can choose whether to use the log-normal derived function or simply approximate it by a Gaussian envelope. We demonstrate here that:

$$\frac{\ln(f) - \mu}{\sigma} \approx \frac{f - sf_0}{B_{sf}}$$

The log-Gabor envelope is approximately Gaussian in a neighborhood of sf_0 , for $sf - sf_0 \ll B_{sf}$ (for small values of σ , $\ln(1+x)$ is approximately x that is to say the log-normal is approximately Gaussian). Since,

$$\frac{-\log^2\left(\frac{f}{sf_0}\right)}{2 \cdot \log^2\left(\frac{1+B_{sf}}{sf_0}\right)} = -\frac{1}{2} \cdot \left(\frac{\log\left(\frac{f}{sf_0}\right)}{\log\left(\frac{1+B_{sf}}{sf_0}\right)} \right)^2 \quad (1)$$

and

$$\frac{\log\left(\frac{f}{sf_0}\right)}{\log\left(1 + \frac{B_{sf}}{sf_0}\right)} = \frac{\log\left(1 + \frac{f-sf_0}{sf_0}\right)}{\log\left(1 + \frac{B_{sf}}{sf_0}\right)} \quad (2)$$

with $\frac{f}{sf_0} = 1 + \frac{f-sf_0}{sf_0}$.

Then, near sf_0 , i.e. in the neighborhood of sf_0 , and for $f - sf_0 \ll B_{sf}$, this function can be represented by the first order Taylor expansion

$$\frac{\log\left(1 + \frac{f-sf_0}{sf_0}\right)}{\log\left(1 + \frac{B_{sf}}{sf_0}\right)} = \frac{\frac{f-sf_0}{sf_0}}{\frac{B_{sf}}{sf_0}} = \frac{f - sf_0}{B_{sf}} \quad (3)$$

so in the sf_0 neighborhood, the pdf (of f) is:

$$p(f) = \exp \left(\frac{-\log^2 \left(\frac{f}{sf_0} \right)}{2 \cdot \log^2 \left(\frac{1 + B_s f}{sf_0} \right)} \right) \quad (4)$$

$$= \exp \left(-\frac{1}{2} \cdot \left(\frac{\log \left(\frac{f}{sf_0} \right)}{\log \left(\frac{1 + B_s f}{sf_0} \right)} \right)^2 \right) \quad (5)$$

$$= \exp \left(-\frac{1}{2} \left(\frac{f - sf_0}{B_s f} \right)^2 \right) \quad (6)$$

that identifies to the desired normal distribution $\mathcal{N}(f; sf_0, B_s f)$.