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]

[1]INCM, CNRS / Aix-Marseille University, France
[2]Institut 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 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 **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 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/px, cyc/frame) is the Nyquist frequency, i.e., the maximal frequency that can be represented without having undesirable aliasing effects.

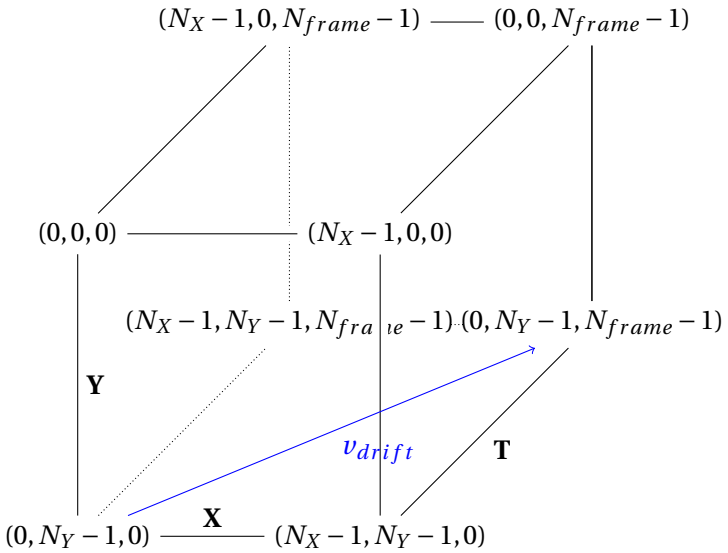

Figure 1

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 $\mathscr{E}(f_x, f_y, f_t) \in \mathbb{R}^{N_X \times N_Y \times N_{frame}}$. The first two define the image size, width and height, respectively. The third dimension is the length of the image-series (number of frames).

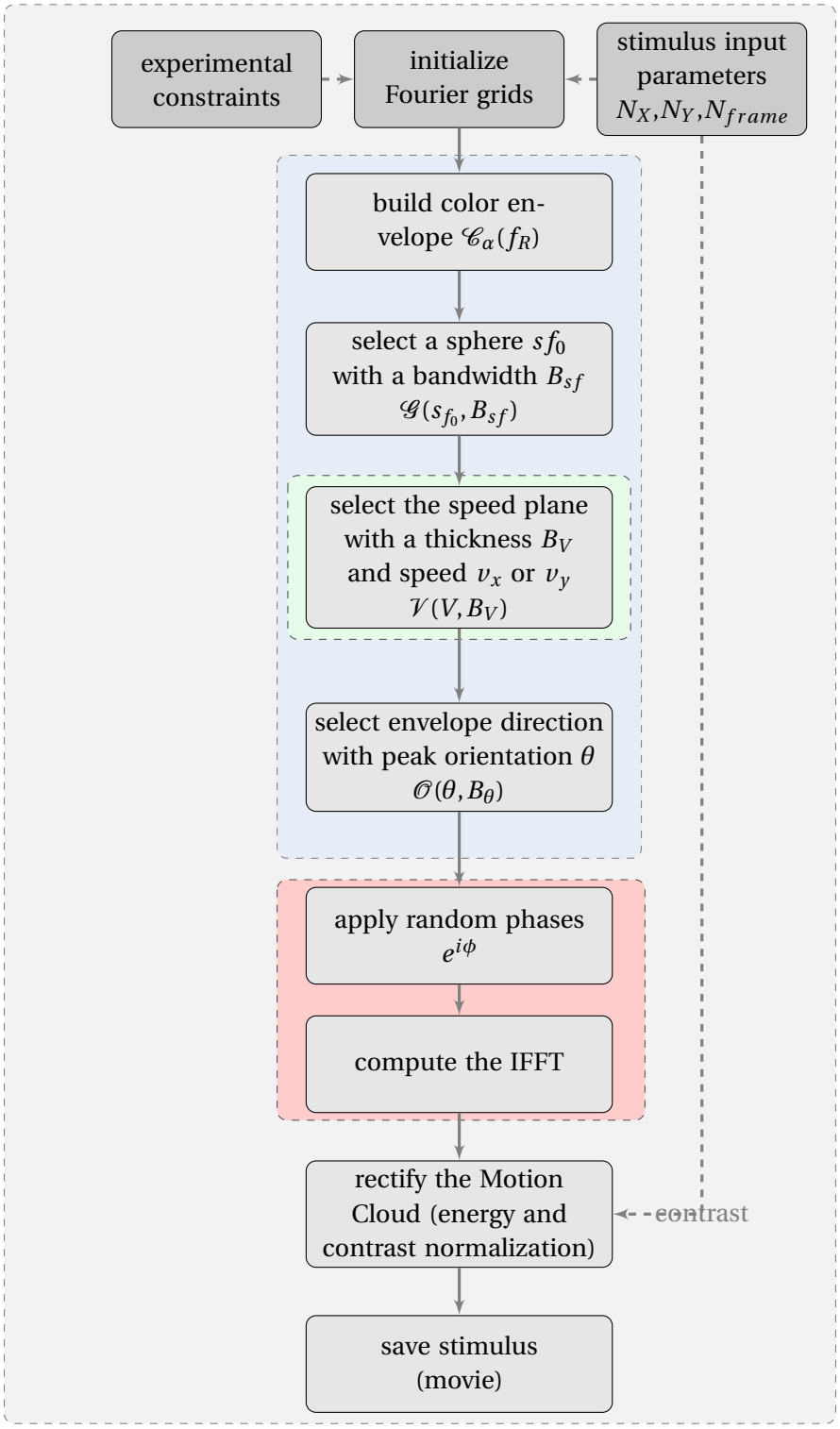
experimental constraints
initialize Fourier grids
stimulus input parameters N_X,N_Y,N_{frame}
build color envelope $\mathscr{C}_\alpha(f_R)$
select a sphere sf_0 with a bandwidth B_{sf} $\mathscr{G}(s_{f_0}, B_{sf})$
select the speed plane with a thickness B_V and speed v_x or v_y $\mathscr{V}(V, B_V)$
select envelope direction with peak orientation θ $\mathscr{O}(\theta, B_\theta)$
apply random phases $e^{i\phi}$
compute the IFFT
rectify the Motion Cloud (energy and contrast normalization)
contrast
save stimulus (movie)

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 ($\mathscr{C}_\alpha$) is multiplied by the global Fourier envelope constructed by the product of the speed ($\mathscr{V}$), spatial frequency ($\mathscr{G}$) and orientation envelopes ($\mathscr{O}$). 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, theta=0.,
      B_theta=np.pi/8, alpha=1.) # define an envelope
5 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 Matlab^{TM} 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:

```
for sf_0 in [0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6]:
  name_ = 'figures/' + name + '-sf_0-' + str(sf_0).replace('.', '_')
  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.

```
#! /usr/bin/env python
# -*- coding: utf8 -*-
"""

Main script for generating Motion Clouds

(c) Laurent Perrinet - INT/CNRS

Motion Clouds (keyword) parameters:
size    -- power of two to define the frame size (N_X, N_Y)
size_T  -- power of two to define the number of frames (N_frame)
N_X     -- frame size horizontal dimension [px]
N_Y     -- frame size vertical dimension [px]
N_frame -- number of frames [frames] (a full period in time frames)
alpha   -- exponent for the color envelope.
sf_0    -- mean spatial frequency relative to the sampling frequency.
ft_0    -- spatiotemporal scaling factor.
B_sf    -- spatial frequency bandwidth
V_X     -- horizontal speed component
V_Y     -- vertical speed component
B_V     -- speed bandwidth
theta   -- mean orientation of the Gabor kernel
B_theta -- orientation bandwidth
loggabor-- (boolean) if True it uses a log-Gabor kernel (instead of the traditional gabor)

Display parameters:

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.
ext        -- frame image format.
T_movie -- movie duration [s].
fps      -- frame per seconds

"""

import os
DEBUG = False
if DEBUG:
    size = 5
    size_T = 5
    figsize = (400, 400)  # faster
else:
    size = 7
    size_T = 7
    figsize = (800, 800) # nice size, but requires more memory

import numpy as np
N_X = 2**size
N_Y = N_X
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 + 1), (-N_frame
                //2):((N_frame-1)//2 + 1)]      # output is always even.
109     else:
110         fx, fy, ft = np.mgrid[(-N_X//2):((N_X-1)//2 + 1), (-N_Y//2):((N_Y-1)//2 + 1), (-N_frame
                //2):((N_frame-1)//2 + 1)]      # output is always even.
111     fx, fy, ft = fx*1./N_X, fy*1./N_Y, ft*1./N_frame
112     return fx, fy, ft
113
114 def frequency_radius(fx, fy, ft, ft_0=ft_0):
115     """
116      Returns the frequency radius. To see the effect of the scaling factor run
117      'test_color.py'
118
119     """
120     N_X, N_Y, N_frame = fx.shape[0], fy.shape[1], ft.shape[2]
121     R2 = fx**2 + fy**2 + (ft/ft_0)**2 # cf . Paul Schrater 00
122     R2[N_X//2 , N_Y//2 , N_frame//2 ] = np.inf
123     return np.sqrt(R2)
124
125 def envelope_color(fx, fy, ft, alpha=alpha, ft_0=ft_0):
126     """
127     Returns the color envelope.
128     Run 'test_color.py' to see the effect of alpha
129     alpha = 0 white
130     alpha = 1 pink
131     alpha = 2 red/brownian
132     (see http://en.wikipedia.org/wiki/1/f_noise )
133     """
134     f_radius = frequency_radius(fx, fy, ft, ft_0=ft_0)**alpha
135     return 1. / f_radius
136
137 def envelope_radial(fx, fy, ft, sf_0=sf_0, B_sf=B_sf, ft_0=ft_0, loggabor=loggabor):
138     """
139     Radial frequency envelope:
140     selects a sphere around a preferred frequency with a shell width B_sf.
141     Run 'test_radial.py' to see the explore the effect of sf_0 and B_sf
142     """
143     if sf_0 == 0.: return 1.
144     if loggabor:
145         # see http://en.wikipedia.org/wiki/Log-normal_distribution
146         fr = frequency_radius(fx, fy, ft, ft_0=1.)
147         env = 1./fr*np.exp(-.5*(np.log(fr/sf_0)**2)/(np.log((sf_0+B_sf)/sf_0)**2))
148         return env
149     else:
150         return np.exp(-.5*(frequency_radius(fx, fy, ft, ft_0=1.) - sf_0)**2/B_sf**2)
151
152 def envelope_speed(fx, fy, ft, V_X=V_X, V_Y=V_Y, B_V=B_V):
153     """
154      Speed envelope:
155      selects the plane corresponding to the speed (V_X, V_Y) with some thickness B_V
156
157     (V_X, V_Y) = (0,1) is downward and  (V_X, V_Y) = (1,0) is rightward in the 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=1.))**2)
168     return env
169
170 def envelope_orientation(fx, fy, ft, theta=theta, B_theta=B_theta):
171     """
172     Orientation envelope:
173     selects one central orientation theta, B_theta the spread
174     We use a von-Mises distribution on the orientation.
175
176     Run 'test_orientation.py' to see the effect of changing theta and B_theta.
177     """
178     if not(B_theta is np.inf):
179         angle = np.arctan2(fy, fx)
180         envelope_dir = np.exp(np.cos(2*(angle-theta))/B_theta)
181         return envelope_dir
182     else: # for large bandwidth, returns a strictly flat envelope
183         return 1.
184
185 def envelope_gabor(fx, fy, ft, V_X=V_X, V_Y=V_Y,
186                    B_V=B_V, sf_0=sf_0, B_sf=B_sf, loggabor=loggabor,
187                    theta=theta, B_theta=B_theta, alpha=alpha):
188     """
189     Returns the Motion Cloud kernel
190
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 to obtain
202     the spatiotemporal stimulus.
203
204     - use a specific seed to specify the RNG's seed,
205     - test the impulse response of the kernel by setting impulse to True
206     - test the effect of randomizing amplitudes too by setting do_amp to True
207 shape
208     """
209     (N_X, N_Y, N_frame) = envelope.shape
210     amps = 1.
211     if impulse:
212         phase = 0.
213     else:
214         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
```

```
    Fz = amps * envelope * np.exp(1j * phase)

    # centering the spectrum
    Fz = np.fft.ifftshift(Fz)
    Fz[0, 0, 0] = 0.
    z = np.fft.ifftn((Fz)).real
    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.

```
########################## Display Tools ########################################

def get_size(mat):
    """
    Get stimulus dimensions

    """
    return [np.size(mat, axis=k) for k in range(np.ndim(mat))]

#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.

def visualize(z, azimuth=290., elevation=45.,
    thresholds=[0.94, .89, .75, .5, .25, .1], opacities=[.9, .8, .7, .5, .2, .2],
    name=None, ext=ext, do_axis=True, do_grids=False, draw_projections=True,
    colorbar=False, f_N=2., f_tN=2., figsize=figsize):

    """ Visualize the  Fourier spectrum """
    import_mayavi()

    N_X, N_Y, N_frame = z.shape
    fx, fy, ft = get_grids(N_X, N_Y, N_frame, sparse=False)

    mlab.figure(1, bgcolor=(1, 1, 1), fgcolor=(0, 0, 0), size=figsize)
    mlab.clf()

    # Normalize the amplitude.
    z /= z.max()
    # Create scalar field
    src = mlab.pipeline.scalar_field(fx, fy, ft, z)
    if draw_projections:
        src_x = mlab.pipeline.scalar_field(fx, fy, ft, np.tile(np.sum(z, axis=0), (N_X, 1, 1)))
        src_y = mlab.pipeline.scalar_field(fx, fy, ft, np.tile(np.reshape(np.sum(z, axis=1), (
            N_X, 1, N_frame)), (1, N_Y, 1)))
        src_z = mlab.pipeline.scalar_field(fx, fy, ft, np.tile(np.reshape(np.sum(z, axis=2), (
            N_X, N_Y, 1)), (1, 1, N_frame)))

        # Create projections
        border = 0.47
        scpx = mlab.pipeline.scalar_cut_plane(src_x, plane_orientation='x_axes', view_controls=
            False)
        scpx.implicit_plane.plane.origin = [-border, 1/N_Y, 1/N_frame]
        scpx.enable_contours = True
        scpy = mlab.pipeline.scalar_cut_plane(src_y, plane_orientation='y_axes', view_controls=
            False)
        scpy.implicit_plane.plane.origin = [1/N_X, border, 1/N_frame]
        scpy.enable_contours = True
        scpz = mlab.pipeline.scalar_cut_plane(src_z, plane_orientation='z_axes', view_controls=
```

```
            False)
        scpz.implicit_plane.plane.origin = [1/N_X, 1/N_Y, -border]
        scpz.enable_contours = True

    # Generate iso-surfaces at differnet energy levels
    for threshold, opacity in zip(thresholds, opacities):
        mlab.pipeline.iso_surface(src, contours=[z.max()-threshold*z.ptp(), ],
                                  opacity=opacity)
        mlab.outline(extent=[-1./2, 1./2, -1./2, 1./2, -1./2, 1./2],)

    # Draw a sphere at the origin
    x = np.array([0])
    y = np.array([0])
    z = np.array([0])
    s = 0.01
    mlab.points3d(x, y, z, extent=[-s, s, -s, s, -s, s], scale_factor=0.15)

    if colorbar: mlab.colorbar(title='density', orientation='horizontal')
    if do_axis:
        ax = mlab.axes(xlabel='fx', ylabel='fy', zlabel='ft',
                       extent=[-1./2, 1./2, -1./2, 1./2, -1./2, 1./2],
                       )
        ax.axes.set(font_factor=2.)

    try:
        mlab.view(azimuth=azimuth, elevation=elevation, distance='auto', focalpoint='auto')
    except:
        print(" You should upgrade your mayavi version")

    if not(name is None):
        mlab.savefig(name + ext, magnification=1, size=figsize)
    else:
       mlab.show(stop=True)

    mlab.close(all=True)

def cube(im, azimuth=-45., elevation=130., roll=-180., name=None,
         ext=ext, do_axis=True, show_label=True, colormap='gray',
         vmin=0., vmax=1., figsize=figsize):

    """
    Visualize the stimulus as a cube

    """
    import_mayavi()

    N_X, N_Y, N_frame = im.shape
    fx, fy, ft = get_grids(N_X, N_Y, N_frame, sparse=False)

    mlab.figure(1, bgcolor=(1, 1, 1), fgcolor=(0, 0, 0), size=figsize)
    mlab.clf()
    src = mlab.pipeline.scalar_field(fx*2., fy*2., ft*2., im)

    mlab.pipeline.image_plane_widget(src, plane_orientation='z_axes',
                                     slice_index=0, colormap=colormap, vmin=vmin, vmax=vmax)
    mlab.pipeline.image_plane_widget(src, plane_orientation='z_axes',
                                     slice_index=N_frame, colormap=colormap,
                                     vmin=vmin, vmax=vmax)
    mlab.pipeline.image_plane_widget(src, plane_orientation='x_axes', slice_index=0,
                                     colormap=colormap, vmin=vmin, vmax=vmax)
```

```
103     mlab.pipeline.image_plane_widget(src, plane_orientation='x_axes', slice_index=N_X,
104                                      colormap=colormap, vmin=vmin, vmax=vmax)
105
106     mlab.pipeline.image_plane_widget(src, plane_orientation='y_axes', slice_index=0,
107                                      colormap=colormap, vmin=vmin, vmax=vmax)
108     mlab.pipeline.image_plane_widget(src, plane_orientation='y_axes', slice_index=N_Y,
109                                      colormap=colormap, vmin=vmin, vmax=vmax)
110
111     if do_axis:
112         ax = mlab.axes(xlabel='x', ylabel='y', zlabel='t',
113                        extent=[-1., 1., -1., 1., -1., 1.],
114                        ranges=[0., N_X, 0., N_Y, 0., N_frame],
115                        x_axis_visibility=True, y_axis_visibility=True,
116                        z_axis_visibility=True)
117         ax.axes.set(font_factor=2.)
118
119         if not(show_label): ax.axes.set(label_format='')
120
121
122     try:
123         mlab.view(azimuth=azimuth, elevation=elevation, distance='auto', focalpoint='auto')
124         mlab.roll(roll=roll)
125     except:
126         print(" You should upgrade your mayavi version")
127
128     if not(name is None):
129         mlab.savefig(name + ext, magnification=1, size=figsize)
130     else:
131         mlab.show(stop=True)
132
133     mlab.close(all=True)
134
135 def anim_exist(filename, vext='.mpg'):
136     """
137     Check if the movie already exists
138
139     """
140     return not(os.path.isfile(filename+vext))
141
142
143 def anim_save(z, filename, display=True, flip=False, vext='.mpg',
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 ' + tmpdir + '/
    frame%03d.png ' + filename + vext # + ' 2>/dev/null')
190         cmd = 'ffmpeg -v 0 -y -sameq -loop_output 0 -i ' + tmpdir + '/frame%03d.png ' +
                filename + vext # + ' 2>/dev/null')
191         # print('Doing : ', cmd)
192         os.system(cmd) # + ' 2>/dev/null')
193         # To force the frame rate of the output file to 24 fps:
194         # ffmpeg -i input.avi -r 24 output.avi
195         # 3) clean up
196         remove_frames(tmpdir, files)
197     if vext == '.gif': # http://www.uoregon.edu/~noeckel/MakeMovie.html
198         # 1) create temporary frames
199         tmpdir, files = make_frames(z)
200         # 2) convert frames to movie
201 #         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[:]), ', 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 cubes and
277     movies.
278     The figures names are automatically generated.
279     """
280     if anim_exist(name, vext=vext):
281         z = envelope_gabor(fx, fy, ft, V_X=V_X, V_Y=V_Y,
282                     B_V=B_V, sf_0=sf_0, B_sf=B_sf, loggabor=loggabor,
283                     theta=theta, B_theta=B_theta, alpha=alpha)
284         figures(z, name, vext=vext, do_figs=do_figs, do_movie=do_movie,
285                     seed=seed, impulse=impulse, verbose=verbose)
286
287 def figures(z, name, vext=vext, do_figs=True, do_movie=True,
288                     seed=None, impulse=False, verbose=False, masking=False):
289     if ((MAYAVI == 'Import') or MAYAVI[:2]=='Ok') and do_figs and anim_exist(name, vext=ext):
            visualize(z, name=name)           # Visualize the Fourier Spectrum
290     if (do_movie and anim_exist(name, vext=vext)) or (MAYAVI and do_figs and anim_exist(name +
            '_cube', vext=ext)):
291         movie = rectif(random_cloud(z, seed=seed, impulse=impulse), verbose=verbose)
292     if (((MAYAVI == 'Import') or MAYAVI[:2]=='Ok') and do_figs and anim_exist(name + '_cube',
            vext=ext)): cube(movie, name=name + '_cube')    # Visualize the Stimulus cube
293     if (do_movie and anim_exist(name, vext=vext)): anim_save(movie, name, display=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 $sf - sf_0 << B_{sf}$ (for small values of σ, $\ln(1+x)$ is approximately x that is to say the log-normal is approximately Gaussian).

Since,

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

and

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

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 $f-sf_0 << B_{sf}$, this function can be represented by the first order Taylor expansion

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

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

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

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

$$= \exp\left(-\frac{1}{2}\left(\frac{f-sf_0}{B_{sf}}\right)^2\right) \tag{6}$$

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