Reconstruction based on a robust and unbiased spherical deconvolution model (RUMBA-SD)

In [ ]:
```
#%pip install bids
```

In [12]:
```
#importing necessary libraries
import numpy as np
from dipy.io.image import load_nifti, save_nifti
from dipy.io.gradients import read_bvals_bvecs
from dipy.core.gradients import gradient_table
from bids.layout import BIDSLayout
import dipy.reconst.dti as dti
from dipy.data import get_fnames
from dipy.io.image import load_nifti_data, load_nifti, save_nifti
from dipy.data import get_fnames, get_sphere
from dipy.core.sphere import HemiSphere
import nibabel as nib
from dipy.reconst.csdeconv import auto_response_ssst
from dipy.reconst.rumba import RumbaSDModel
```

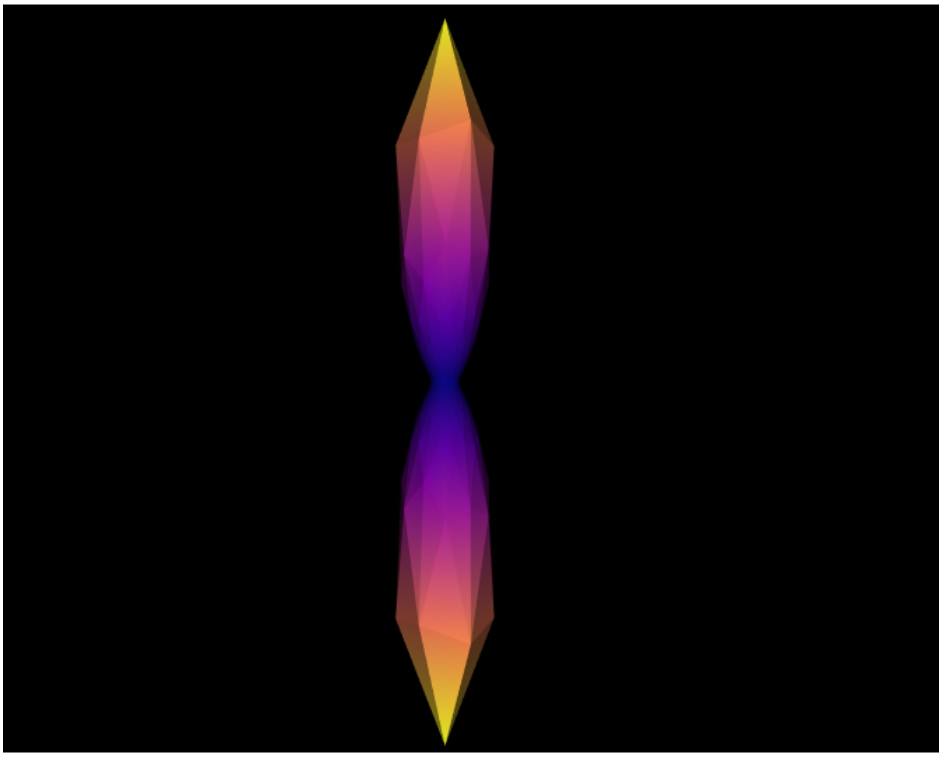
In [22]:
```
gradient_layout = BIDSLayout("./openneuro/ds001907/sub-RC4201/ses-1/", validate=False)
#accessing the openneuro dataset

subj = 'RC4201'

dwi_fname = gradient_layout.get(subject=subj, suffix='dwi', extension='.nii.gz', return_type='file')[0]
bvec_fname = gradient_layout.get( extension='.bvec', return_type='file')[0]
bval_fname = gradient_layout.get( extension='.bval', return_type='file')[0]

dwi_img = nib.load(dwi_fname)
affine = dwi_img.affine

bvals, bvecs = read_bvals_bvecs(bval_fname, bvec_fname)
gtab = gradient_table(bvals, bvecs)

sphere = get_sphere('repulsion724')
```

In [23]:
```
from dipy.segment.mask import median_otsu
import dipy.reconst.dti as dti
from dipy.segment.mask import median_otsu

dwi_data = dwi_img.get_fdata() #diffusion data

maskdata, mask = median_otsu(dwi_data, vol_idx=range(10, 50), median_radius=3,
                             numpass=1, autocrop=True, dilate=2)
print('maskdata.shape (%d, %d, %d, %d)' % maskdata.shape)
print('data.shape (%d, %d, %d, %d)' % dwi_data.shape)
```

```
maskdata.shape (81, 108, 69, 129)
data.shape (128, 128, 72, 129)
```

Fiber response function estimation :

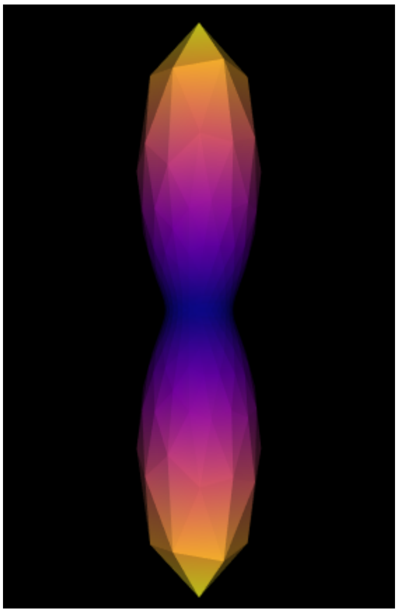
In [10]:
```
#Visualization of the response function
from dipy.sims.voxel import single_tensor_odf
from fury import window, actor

# Enables/disables interactive visualization
interactive = False

scene = window.Scene()

evals = rumba.wm_response
evecs = np.array([[0, 1, 0], [0, 0, 1], [1, 0, 0]]).T

response_odf = single_tensor_odf(sphere.vertices, evals, evecs)
# Transform our data from 1D to 4D
response_odf = response_odf[None, None, None, :]
response_actor = actor.odf_slicer(response_odf, sphere=sphere
                                  )

scene.add(response_actor)
#print('Saving illustration as default_response.png')
#window.record(scene, out_path='default_response.png', size=(200, 200))


window.show(scene)
```



In [11]:
```
scene.rm(response_actor)
```

In [7]:
```
#Another method : Estimation of the fiber response function from the local brain region.
from dipy.sims.voxel import single_tensor_odf
from dipy.reconst.csdeconv import auto_response_ssst
from fury import window, actor

response, _ = auto_response_ssst(gtab, dwi_data, roi_radii=10, fa_thr=0.7)
print(response)
evals = response[0]
evecs = np.array([[0, 1, 0], [0, 0, 1], [1, 0, 0]]).T

response_odf = single_tensor_odf(sphere.vertices, evals, evecs)
# transform our data from 1D to 4D
response_odf = response_odf[None, None, None, :]
response_actor = actor.odf_slicer(response_odf, sphere=sphere,

                                  colormap='plasma')
scene = window.Scene()
scene.add(response_actor)
#print('Saving illustration as estimated_response.png')
#window.record(scene, out_path='estimated_response.png', size=(200, 200))
window.show(scene)
```

```
(array([0.00165202, 0.00039875, 0.00039875]), 69502.95412529839)
```



In [8]:
```
scene.rm(response_actor)
```

Reconstruction of the fODF (Fiber Orientation Distribution Function)

In [24]:
```
response, _ = auto_response_ssst(gtab, dwi_data, roi_radii=10, fa_thr=0.7)
print(response)
```

```
(array([0.00165202, 0.00039875, 0.00039875]), 69502.95412529839)
```

In [25]:
```
#global fit
rumba = RumbaSDModel(gtab, wm_response=response[0], gm_response=None,
                     voxelwise=False, use_tv=False, sphere=sphere)
data_tv = dwi_data[10:50, 40:85, 20:40] #small part of the data

#voxel = dwi_data[29:30,59:60,38:39]
```

In [26]:
```
rumba_fit = rumba.fit(data_tv)

odf = rumba_fit.odf() #odf tensor
```

In [27]:
```
odf.shape
```

Out[27]:
```
(40, 45, 20, 724)
```

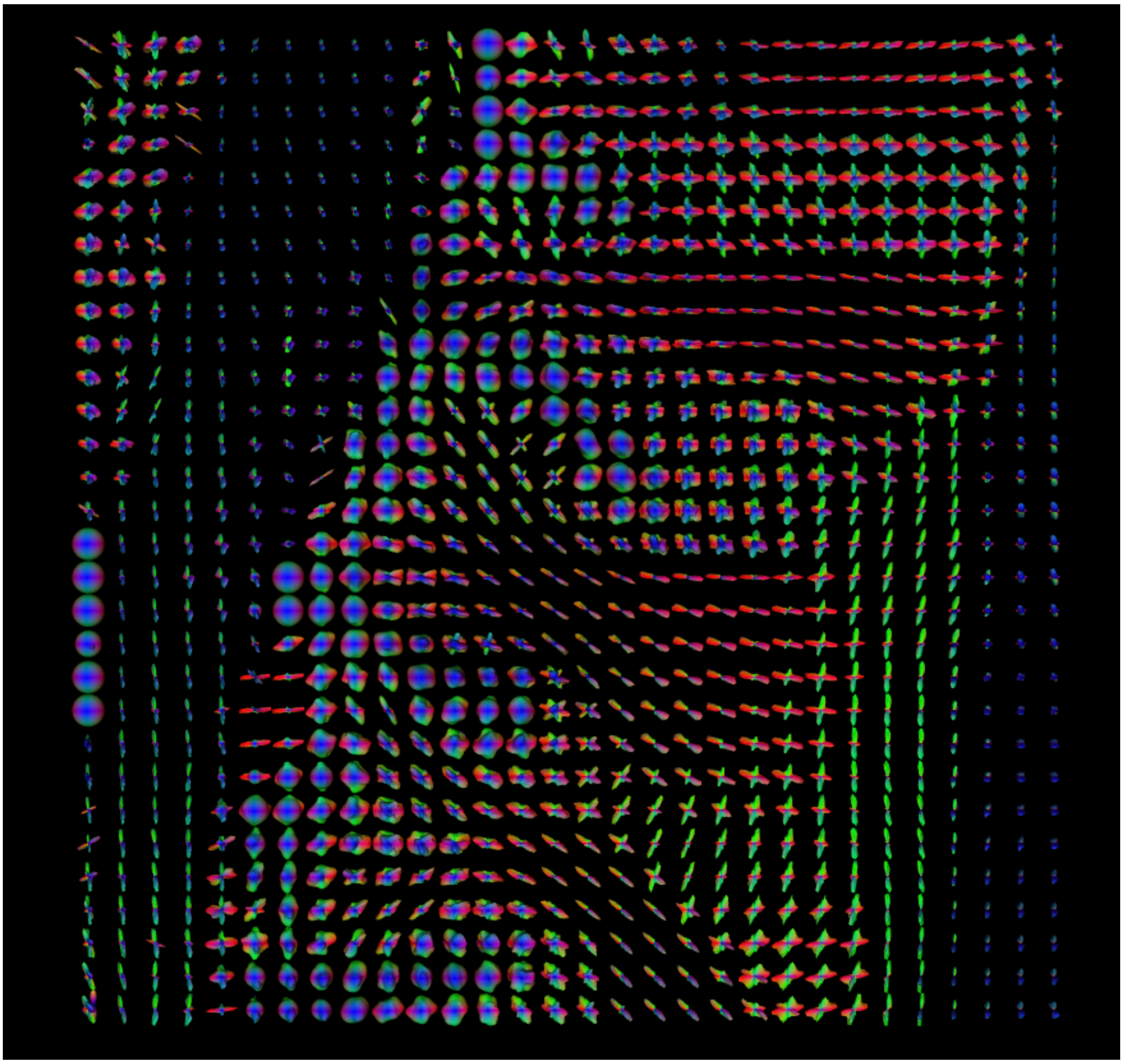
In [16]:
```
#To visualize the fODFs, i combined the fODF with the isotropic components.
combined = rumba_fit.combined_odf_iso
```

In [17]:
```
combined.shape
```

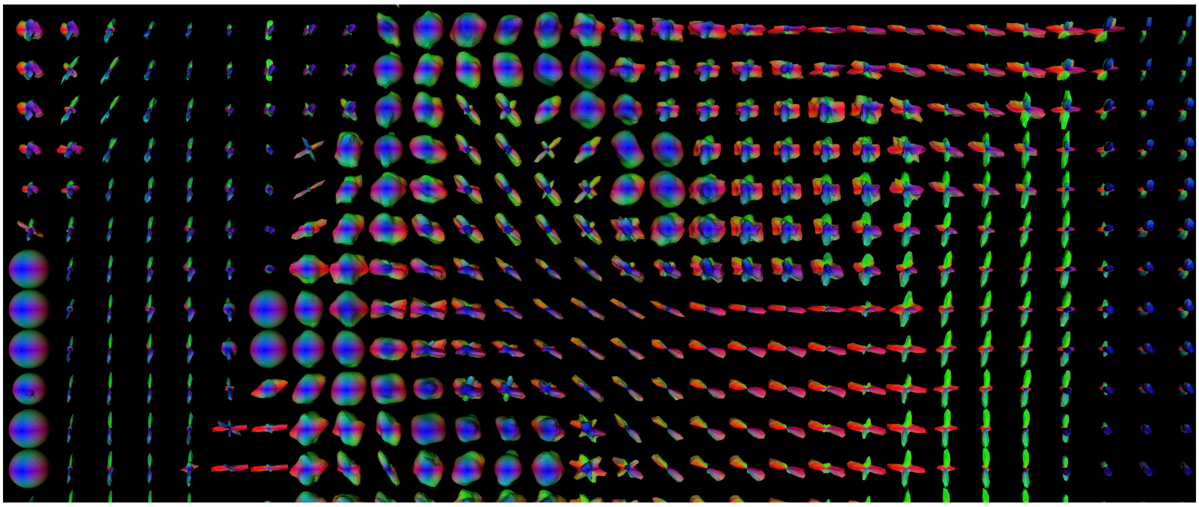
Out[17]:
```
(40, 45, 20, 362)
```

In [30]:
```
from fury import window, actor
combined = rumba_fit.combined_odf_iso
fodf_spheres = actor.odf_slicer(combined[1:4,1:4,:,:], sphere=sphere, norm=False,
                                scale=0.5, colormap=None)
scene = window.Scene()
scene.add(fodf_spheres)
window.show(scene)
```

Visualization of the fODF map: RUMBA-SD



Zoom ++



In [ ]:
```
fodf_spheres.indices
```

In [32]:
```
fodf_spheres.faces
```

Out[32]:
```
array([[  2, 363,   0],
       [  0, 363, 365],
       [366, 364,   1],
       ...,
       [723, 706, 689],
       [723, 719, 685],
       [702, 719, 723]], dtype=uint16)
```

In [33]:
```
fodf_spheres.vertices
```

Out[33]:
```
array([[-0.0914682 ,  0.99569081,  0.01527661],
       [-0.03198756, -0.99848131,  0.04485394],
       [ 0.00657908,  0.99435208,  0.10592761],
       ...,
       [-0.73806889,  0.044762  , -0.67323894],
       [ 0.99918788,  0.03855745, -0.01170055],
       [ 0.71569262, -0.0094363 , -0.69835165]])
```