

```

1  def full_Model2POV(smallmodel,det):
2      # Read a model without hole and convert it into a POV file
3
4      """
5      Comment:
6          Main processing with c-code
7      Input:
8          smallmodel: Small model name (Path to file of small model)
9      Output:
10         aOutputname: Path of POV file of the model
11     Used function:
12         None
13     Used in function:
14         'GeoRayImg.Model2POV'
15     """
16
17     # Get model information
18     dsm = xdibias.Image(smallmodel).readImageData() # Create a numpy array with the
19     image data of the model with XDibias
20
21     # Split path of model file
22     path1, bigmodelname = os.path.split(smallmodel)
23
24     # Comment about the coordinate transformation between model and POV:
25     # x, y and z refers the local defined POV coordinate system:
26     # x: Right
27     # y: Up
28     # z: Depth
29     # Origin: Lower left corner
30     # Pixel center: [0, h, 0]
31     # Intensity I with indices [row, column] refers the coordinate system of the SAR
32     image (and hence the model):
33     # Origin: Upper left corner
34     # Pixel center: [0, 0, h]
35     # Transformation of the two coordinate systems:
36     # x = column
37     # y = dsm[row, column]
38     # z = dsm.shape[0] - 1 - row;
39     # row = dsm.shape[0] - 1 - z
40     # column = x
41     # I[row, column] = y
42
43     # Number of rows and columns of the model
44     rows = dsm.shape[0] # Number of rows of the model
45     cols = dsm.shape[1] # Number of columns of the model
46
47     # -----
48     if len(det.path_to_adapted_pov_file) != 0:
49         fin = open(det.path_to_adapted_pov_file, "r") # Open POV file
50         POVstr_full_scene = fin.read() # Read POV file of full scene
51         fin.close() # Close POV file
52
53     else:
54         POVstr_full_scene = ' '
55
56     # -----
57
58     # Output name of POV file
59     aOutputname = smallmodel + '/%s_full.pov' % bigmodelname
60
61     # Give the definition of the needed matrix
62     modelmarktext = " // %s" % smallmodel
63
64     # Find the pixel that should be saved (at least two neighbour == 1)
65     pixelnr_m = np.zeros((cols, rows)) # Initialisation
66     pixelnr_m = pixelnr_m.astype(np.int) # Convert datatype
67
68     # Main processing with c-code
69     code = '''
70     # line 249 "dem2pov.py" // real line number + 1

```

```

71
72     int x, z;
73     int Pixelordernumber = 0; // the found pixel number
74     double y;
75
76     FILE * file = fopen(aOutputname.c_str() ,"w");
77     if ( ! file) {
78         fprintf(stderr,"Output file could not be opened.\n");
79         exit(0);
80     }
81
82     // 1.5. write the head part of this file
83     fprintf(file,"mesh2 {\n vertex_vectors {%s\n
84 %d\n",modelmarktext.c_str(),rows*cols); // # how many pixel saved
85
86     // 2. loop runs on the coordinate system of the pov. left to right(east), unten
87     // to oben(north)
88     for ( z=0; z<rows; z++) // from lower to upper, (north)
89     {
90         for ( x=0; x<cols; x++ ) // from left to right, (east)
91         {
92             pixelnr_m(x,z) = Pixelordernumber; // generate the pixel order number
93             Pixelordernumber = Pixelordernumber +1;
94
95             y = dsm(rows -1 -z, x);
96
97             /*fprintf(file,"    <%d, %.3f, %d>,\n",x,y,z );*/
98             fprintf(file,"    <%d, %d, %d>,\n",x,int(y),z );//<68, 5627, 115>,
99             //y: .3f or d? or interger ???!!! should change the scale
100             ?????????????????????????????????????????????????????????
101         }
102     }
103
104     //3. loop the triangle
105     fprintf(file," }\n face_indices {\n %d\n", (cols-1)*(rows-1)*2);
106
107     for ( z=1; z<rows; z++) //from lower to upper, (north)
108     {
109         for ( x=0; x<cols-1; x++ ) //from left to right, (east)
110         {
111             // original not consider the height difference of the 4 points
112             /*fprintf(file,"    <%d, %d, %d>,\n",
113             pixelnr_m(x,z),pixelnr_m(x+1,z-1),pixelnr_m(x,z-1));//<69, 1, 0>,*//
114             /*fprintf(file,"    <%d, %d, %d>,\n",
115             pixelnr_m(x,z),pixelnr_m(x+1,z),pixelnr_m(x+1,z-1));//<69, 70, 1>,*//
116
117             //if (fabs(Height(x,z)-Height(x+1,z-1)) <=
118             fabs(Height(x+1,z)-Height(x,z-1)))
119             if ( fabs(dsm(rows -1 -z, x)-dsm(rows -z, x+1)) <= fabs( dsm(rows -1
120 -z, x+1) - dsm(rows -z, x) ) )
121             {
122                 fprintf(file,"    <%d, %d,
123 %d>,\n",pixelnr_m(x,z),pixelnr_m(x+1,z-1),pixelnr_m(x,z-1)); //<69,
124 70, 0>,write the pixel numbers
125                 fprintf(file,"    <%d, %d,
126 %d>,\n",pixelnr_m(x,z),pixelnr_m(x+1,z),pixelnr_m(x+1,z-1));
127             }
128             else
129             {
130                 fprintf(file,"    <%d, %d, %d>,\n",
131                 pixelnr_m(x,z),pixelnr_m(x+1,z),pixelnr_m(x,z-1));
132                 fprintf(file,"    <%d, %d, %d>,\n",
133                 pixelnr_m(x+1,z),pixelnr_m(x+1,z-1),pixelnr_m(x,z-1));
134             }
135         }
136     }
137
138     //4, write end text of the file
139     //fprintf(file," }\n texture {\n pigment { color rgb<1, 1, 1> }\n
140 finish { ambient rgb<0, 0, 0> diffuse 0.5 }\n }\n}\n");
141 //fprintf(file," }\n \n}\n //endmodel \n \n");
142 fprintf(file," }\n \n}\n \n \n \n");

```

```

130
131 // Add geometric information for full scene model (visibility check for depth
132 level > 2)
133 fprintf(file, "%s", POVstr_full_scene.c_str());
134
135 fclose(file);
136 '''
137 err = weave.inline(code,['dsm', 'cols', 'rows', 'aOutputname', 'modelmarktext',
138 'pixelnr_m', 'POVstr_full_scene'], type_converters=converters.blitz)
139
140 print "Model '%s' is converted into '%s' without hole." % (smallmodel, aOutputname
141 )
142
143 # Output
144 return aOutputname
145
146 def read_Metadata(metadatafile, bigmodel, smallmodel, det):
147     # Read parameters from meta data file of image
148     """
149     Input:
150         metadatafile: Orbit meta data file
151         bigmodel: Big model name (Path to file of big model)
152         smallmodel: Small model name (Path to file of small model)
153         det: Definition of input parameters (Look at class 'InputPara' in
154             'Script.py')
155     Output:
156         sensor: Name of sensor
157         filename:
158             SAR case: Filename of image
159             Optical case: Record time of image
160         heading: Heading angle in [°]
161             SAR case: Heading angle of SAR sensor (clockwise) in [°] = Angle
162             from UTM north direction Y to azimuth image axis of SAR image
163             (clockwise) in [°] = Angle from UTM north direction Y to flight
164             direction of SAR sensor (clockwise) in [°]
165             Optical case: Heading angle of optical sensor in [°]
166         AngOfView:
167             SAR case: Angle from UTM north direction Y to line of sight of
168             SAR sensor (clockwise) in [°] = Heading angle of SAR sensor + 90°
169             (clockwise) in [°]
170             Optical case: Angle from UTM north direction Y to line of sight
171             of optical sensor (clockwise) in [°]
172         incidence:
173             SAR case: Interpolated incidence angle on ground of the image
174             according to model center (center of model box) coordinates in [°]
175             Optical case: Incidence angle on ground in [°]
176         meanHeight: Mean height (reference height) of image in [m]
177         PS_columns: Spatial resolution along columns (in UTM east direction X) in
178         [m] --> Is different for different sensors
179         PS_rows: Spatial resolution along rows (in UTM north direction Y) in [m]
180         --> Is different for different sensors
181     Used function:
182         'GeoRayImg.find_1Word'
183         'GeoRayImg.find_2Word'
184         'GeoRayImg.InterpIncidence'
185         'GeoRayImg.RPC2Ang'
186     Used in function:
187         'GeoRayImg.gen_HeaderEndPOV'
188         'GeoRayImg.gen_RaySAR_Para'
189         'GeoRayImg.GeocodeSimuImg1'
190         'GeoRayImg.GeocodeSimuImg2'
191         'GeoRayImg.Model2GeoI'
192         'GeoRayImg.select_ImgOption'
193         'GeoRayImg.create_foldername'
194         'Script.py'
195     """
196
197     # SAR image
198     if det.ImgOption == 1:
199         sensor = find_1Word(metadatafile, 'mission') # Name of SAR sensor

```

```

189     filename = find_1Word(metadatafile, 'filename')[
190         :-4] # Filename of SAR image
191     # Heading angle of SAR sensor in [°] = Angle from UTM north direction Y to
    flight direction of SAR sensor (clockwise) in [°] = Angle from UTM north
    direction Y to azimuth image axis of SAR image (clockwise) in [°]
192     heading = float(find_1Word(metadatafile, 'headingAngle'))
193     # Angle from UTM north direction Y to line of sight of SAR sensor (clockwise)
    in [°] = Heading angle of SAR sensor + 90° (clockwise) in [°]
194     AngOfView = heading + 90
195     # Interpolated incidence angle on ground of the SAR image according to model
    center (center of model box) coordinates in [°]
196     incidence = InterpIncidence(metadatafile, bigmodel, det)
197     # Mean height (reference height) of SAR image in [m]
198     meanHeight = float(find_1Word(metadatafile, 'meanHeight'))
199     # Spatial resolution along columns (in UTM east direction X) of SAR image in
    [m]
200     PS_columns = float(find_2Word(metadatafile, 'pixelSpacing', 'easting'))
201     # Spatial resolution along rows (in UTM north direction Y) of SAR image in [m]
202     PS_rows = float(find_2Word(metadatafile, 'pixelSpacing', 'northing'))
203
204     # Optical image
205     else: # det.ImgOption == 2:
206         # Create an image object of the optical image meta data with XDibias
207         img = xdibias.Image(metadatafile)
208
209         sensor = img.Sensor # Name of optical Sensor
210         filename = img.RecordTime # Record time of optical image
211         heading = img.Azimuth # Heading angle of optical sensor in [°]
212         # Angle from UTM north direction Y to line of sight of optical sensor
    (clockwise) in [°] and incidence angle on ground of optical sensor in [°]
213
214         # !!!!!!!!!!!!!!!
215
216         # AngOfView = 0 # temporary for Data Fusion Contest 2019 (nadir view ->
    incidence angle = 0°)
217         # incidence = 0 # temporary for Data Fusion Contest 2019 (nadir view ->
    incidence angle = 0°)
218
219         # !!!!!!!!!!!!!!!
220
221         AngOfView, incidence = RPC2Ang(smallmodel, metadatafile) # Activate again
    after Data Fusion Contest 2019
222
223         # Mean height (reference height) of optical image in [m]
224         meanHeight = det.meanHeight
225         # Spatial resolution along columns (in UTM east direction X) of optical image
    in [m]
226         PS_columns = img.XCellRes
227         # Spatial resolution along rows (in UTM north direction Y) of optical image
    in [m]
228         PS_rows = img.YCellRes
229
230     # Output
231     return sensor, filename, heading, AngOfView, incidence, meanHeight, PS_columns,
    PS_rows
232
233
234 def gen_HeaderEndPOV(bigmodel, smallmodel, metadatafile, det):
235     # Generate additional code parts (header and end part) for the POV file
236     # 1. Generate header and end part for the POV file
237     # 2. Open the POV file which is already in existence
238     # 3. Add generated header and end part to POV file
239     """
240     Input:
241         bigmodel: Big model name (Path to file of big model)
242         smallmodel: Small model name (Path to file of small model)
243         metadatafile: Orbit meta data file
244         det: Definition of input parameters (Look at class 'InputPara' in
    'Script.py')
245     Output: Additional code parts for the POV file and Information needed for POV-Ray
246         headtext: Header of POV file (Parameters for Ray Tracing)
247         endtext: End part of POV file (Model information)

```

```

248         columns_simu: Number of columns of simulated image
249         rows_simu: Number of rows of simulated image
250     Used function:
251         'GeoRayImg.read_ModelPara'
252         'GeoRayImg.read_Metadata'
253         'GeoRayImg.UpRight'
254         'GeoRayImg.pts'
255         'GeoRayImg.cal_SunPosition'
256     Used in function:
257         'GeoRayImg.POVfile_addPara'
258     """
259
260     global sensor, filename, heading, AngOfView, incidence, meanHeight, PS_columns,
    PS_rows
261
262     # Dummy distance in horizontal plane for calculating sensor position
263     D = 500 # unit [m]
264
265     # Split path of model file
266     path1, bigmodelname = os.path.split(bigmodel)
267
268     # Keep a list of lines that we want to save in POV file
269     headtext = [] # Initialisation: Line to save in header of POV file
270     endtext = [] # Initialisation: Line to save in end part of POV file
271
272     # Read parameters of model
273     model, H_max, H_min, H_mid, L, B, H = read_ModelPara(bigmodel, det)
274
275     # Get information from orbit meta data (Read parameters from meta data file of
    image)
276     sensor, filename, heading, AngOfView, incidence, meanHeight, PS_columns, PS_rows =
    read_Metadata(
277         metadatafile, bigmodel, smallmodel, det)
278
279     # Calculate parameters for Ray Tracing
280     # Working with orthographic camera: Parallel projection
281
282     # GIS data (CityGML)
283     if det.ModelOption == 1:
284         # Look at (Position where the sensor is looking at in POV coordinate system):
        Position of model center (center of model box)
285         x_la = L / 2.0 # in [m]
286         y_la = H_mid # in [m]
287         z_la = B / 2.0 # in [m]
288
289         # Sensor location (Position of signal source in POV coordinate system)
290         # --> Sensor location and Look at have the same POV right coordinate x and
        only differ in POV up coordinate y and POV depth coordinate z
291         x_lo = x_la # in [m]
292         y_lo = y_la + D / math.tan(incidence / 180 * math.pi) # in [m]
293         z_lo = z_la + D # in [m]
294
295     # DSM/DTM/nDSM
296     else: # det.ModelOption == 2:
297         # Look at (Position where the sensor is looking at in POV coordinate system):
        Position of model center (center of model box)
298         x_la = (L - 1 * model.XCellRes) / 2.0 # in [m]
299         y_la = H_mid # in [m]
300         z_la = (B - 1 * model.YCellRes) / 2.0 # in [m]
301
302         # Sensor location (Position of signal source in POV coordinate system)
303         # --> Sensor location and Look at have the same POV right coordinate x and
        only differ in POV up coordinate y and POV depth coordinate z
304
305         if incidence == 0: # special case: ortho image --> incidence angle = 0°
            (new for Data Fusion Contest 2019)
306             x_lo = x_la # in [m]
307             y_lo = y_la + D # in [m]
308             z_lo = z_la
309         else:
310             x_lo = x_la # in [m]
311             y_lo = y_la + D / math.tan(incidence / 180 * math.pi) # in [m]

```

```

312         # in [m]; Sensor is positioned in positive POV depth direction z (Sensor
        looks downwards, top down, in negative POV depth direction z)
313         z_lo = z_la + D
314         # z_lo = z_la - D # in [m]; Sensor is positioned in negative POV depth
        direction z (Sensor looks upwards, from the bottom up, in positive POV
        depth direction z)

315
316     # print "\nLook at (Position where the sensor is looking at in POV coordinate
    system): (x (right),y (up),z (depth)) = (%.2f,%.2f,%.2f) m"%(x_la,y_la,z_la)
317     # print "Sensor location (Position of signal source in POV coordinate system): (x
    (right), y (up), z (depth)) = (%.2f,%.2f,%.2f) m"%(x_lo,y_lo,z_lo)
318     # print "Horizontal distance between sensor location and look at: %.2f m"%D
319
320     # SAR image
321     if det.ImgOption == 1:
322         # Rotation of model to get the right perspective of the defined sensor
        location in the POV coordinate system to the model
323         # Model should be rotated around POV up direction y (in origin of POV
        coordinate system) by 'RotModel' degrees
324         # If RotModel > 0°: Clockwise rotation
325         # If RotModel < 0°: Anticlockwise rotation
326         # Angle of view of SAR sensor can be calculated from 90° and heading angle,
        because SAR sensor is looking perpendicular to the flight direction
327         # Rotation if sensor is positioned in positive POV depth direction z (Sensor
        looks southwards)
328         RotModel = 90 - heading + 360
329         # RotModel = 360 - heading - 90 # Rotation if sensor is positioned in
        negative POV depth direction z (Sensor looks northwards)
330
331         # print "Rotation of model around POV up direction y (in origin of POV
        coordinate system) by %.2f°."%RotModel
332
333         # The azimuth image axis is perpendicular to line of sight (to the left side)
334         # --> The azimuth image axis equates to the flight direction, because SAR
        sensor is looking perpendicular to the flight direction (to the right side)
335         # --> Angle from UTM north direction Y to azimuth image axis of SAR image
        (clockwise) in [°] = Heading angle of SAR sensor (clockwise) in [°] = Angle
        from UTM north direction Y to flight direction of SAR sensor (clockwise) in
        [°]
336         AngImgAxis = heading
337
338         # print "Angle from UTM north direction Y to azimuth image axis of SAR image
        (clockwise): %.2f°"%AngImgAxis
339
340     # Optical image
341     else: # det.ImgOption == 2:
342         # Rotation of model to get the right perspective of the defined sensor
        location in the POV coordinate system to the model
343         # Model should be rotated around POV up direction y (in origin of POV
        coordinate system) by 'RotModel' degrees
344         # If RotModel > 0°: Clockwise rotation
345         # If RotModel < 0°: Anticlockwise rotation
346         # Angle of view of optical sensor must be calculated in 'RPC2Ang'
347         # Rotation if sensor is positioned in positive POV depth direction z (Sensor
        looks southwards)
348         RotModel = - AngOfView
349         # RotModel = 180 - AngOfView # Rotation if sensor is positioned in negative
        POV depth direction z (Sensor looks northwards)
350
351         # print "Rotation of model around POV up direction y (in origin of POV
        coordinate system) by %.2f°."%RotModel
352
353         # The easting image axis is perpendicular to line of sight (to the left side)
354         # --> The easting image axis do not equate to the flight direction, because
        optical sensor is looking in arbitrary direction (independent of the flight
        direction and hence the heading angle)
355         # --> Angle from UTM north direction Y to easting image axis of optical image
        (clockwise) in [°] = Angle from UTM north direction Y to line of sight of
        optical sensor + 90° (clockwise) in [°]
356         AngImgAxis = AngOfView + 90
357
358         # print "Angle from UTM north direction Y to easting image axis of optical

```

```

        image (clockwise): %.2f°"%AngImgAxis
359
360 # Find the size of the projection image = cover area of sensor
361 # (Columns * Rows; in POV software: Rendering with Az X Rg)
362
363 # SAR case:
364 #   up: Height of image plane of simulated SAR image in [m] --> Along range image
axis
365 #   right: Width of image plane of simulated SAR image in [m] --> Along azimuth
image axis
366 # Optical case:
367 #   up: Height of image plane of simulated optical image in [m] --> Along
northing image axis
368 #   right: Width of image plane of simulated optical image in [m] --> Along
easting image axis
369
370 if incidence != 0: # if no ortho image as special case (new for data fusion
contest 2019)
371     up, right = UpRight(AngImgAxis, incidence, L, B, H)
372 else:
373     up = L
374     right = B
375
376 # SAR image
377 if det.ImgOption == 1:
378     # Size of the simulated SAR image: Number of rows of simulated SAR image
379     rows_simu = round(up / PS_rows)
380 # Optical image
381 else: # det.ImgOption == 2:
382     # Size of the simulated optical image: Number of rows of simulated optical
image
383     rows_simu = round(
384         up / (PS_rows * math.cos(incidence / 180.0 * math.pi)))
385
386 # Size of the simulated SAR image and optical image: Number of columns of
simulated SAR image and optical image
387 columns_simu = round(right / PS_columns)
388
389 print "Size of the simulated image:"
390 # print "Height of image plane of simulated image: %.2f m"%up
391 # print "Width of image plane of simulated image: %.2f m"%right
392 # print "Number of rows of the simulated image: %.2f"%rows_simu
393 # print "Number of columns of the simulated image: %.2f"%columns_simu
394
395 # Generate header of POV file and add it to POV file
396
397 pt = '#include "colors.inc"\n#include "finish.inc"\n\n// File produced by
AccuTrans 3D\n\n#version 3.5;\n'
398 pts(pt, headtext) # Save print output in log file
399
400 # SAR image
401 if det.ImgOption == 1:
402     # Contribution file should be created
403     pt = "global_settings {max_trace_level %s SAR_Output_Data 1 SAR_Intersection
0}\n" % det.max_bounce
404     pts(pt, headtext) # Save print output in log file
405 # Optical image
406 else: # det.ImgOption == 2:
407     # Contribution file should not be created
408     pt = "global_settings {max_trace_level %s SAR_Output_Data 0 SAR_Intersection
0}\n" % det.max_bounce
409     pts(pt, headtext) # Save print output in log file
410
411 # Sensor information
412 pt = "#declare Cam = camera {"
413 pts(pt, headtext) # Save print output in log file
414 # Type of sensor
415 pt = "orthographic"
416 pts(pt, headtext) # Save print output in log file
417 # Position where the sensor is looking at
418 pt = "look_at <" + str(x_la) + ", " + str(y_la) + ", " + str(z_la) + ">"
419 pts(pt, headtext) # Save print output in log file

```



```

420 # Sensor location (Position of signal source)
421 pt = "location <" + str(x_lo) + "," + str(y_lo) + "," + str(z_lo) + ">"
422 pts(pt, headtext) # Save print output in log file
423 pt = "right " + str(right) + ' *x'
424 pts(pt, headtext) # Save print output in log file
425 pt = "up " + str(up) + ' *y'
426 pts(pt, headtext) # Save print output in log file
427 pt = "}\n\ncamera{Cam}\n"
428 pts(pt, headtext) # Save print output in log file
429
430 # Light source
431 pt = "light_source {"
432 pts(pt, headtext) # Save print output in log file
433 pt = "0*x"
434 pts(pt, headtext) # Save print output in log file
435 # Color of light source: white
436 pt = "color rgb <1,1,1>"
437 pts(pt, headtext) # Save print output in log file
438 # Type of light source
439 pt = "parallel"
440 pts(pt, headtext) # Save print output in log file
441
442 # SAR image
443 # Light source is at the same position as the sensor (signal source) --> Sun
position = sensor position
444 if det.ImgOption == 1:
445     # Translate light source to position of sensor (signal source)
446     pt = "translate <" + str(x_lo) + "," + \
447         str(y_lo) + "," + str(z_lo) + ">"
448     pts(pt, headtext) # Save print output in log file
449 # Optical image
450 else: # det.ImgOption == 2:
451     # Light source is at the same position as the sensor (signal source)
452     if det.light == 1: # (sun position = sensor position)
453         print "Light source is at the same position as the sensor (signal
source)."
454         # Translate light source to position of sensor (signal source)
455         pt = "translate <" + str(x_lo) + "," + \
456             str(y_lo) + "," + str(z_lo) + ">"
457         pts(pt, headtext) # Save print output in log file
458     # Light source is at that position at which it was during image acquisition
459     else: # det.light == 2: (sun position = real sun position)
460         print "Light source corresponds to sun position."
461         # Calculate position of sun during image acquisition
462         x_sun, y_sun, z_sun = cal_SunPosition(
463             RotModel, metadatafile, det, x_la, y_la, z_la)
464         # Translate light source to that position at which it was during image
acquisition
465         pt = "translate <" + str(x_sun) + "," + \
466             str(y_sun) + "," + str(z_sun) + ">"
467         pts(pt, headtext) # Save print output in log file
468
469 # Light source point at the position of model center (center of model box)
470 pt = "point_at <" + str(x_la) + "," + str(y_la) + "," + str(z_la) + ">"
471 pts(pt, headtext) # Save print output in log file
472 pt = "}\n"
473 pts(pt, headtext) # Save print output in log file
474
475 # Add dummy command for geometry part of the model
476 if bigmodelname == "box_full_scene.txt":
477     pt = "#declare Building = union{\n"
478     pts(pt, headtext) # Save print output in log file
479
480 # DSM/DTM/nDSM
481 if det.ModelOption == 2 and det.object_flag == 1:
482     pt = "#declare %s = " % bigmodelname
483     headtext.append(pt)
484
485 # Define surface reflection parameters of model
486
487 # SAR image
488 if det.ImgOption == 1:

```



```

489     if det.refl_comb == 1:
490         refl_params = 'reflection {0.5} ambient 0 diffuse 0.3 specular 0.5
           roughness 0.0033'
491     elif det.refl_comb == 2:
492         refl_params = 'reflection {0.5} ambient 0 diffuse 0.7 specular 0.5
           roughness 0.0033'
493     elif det.refl_comb == 3:
494         refl_params = 'reflection {0.5} ambient 0 diffuse 1.0 specular 0.5
           roughness 0.0033'
495     # No option of reflection parameters is choosen
496     else: # det.refl_comb != 1 and det.refl_comb != 2 and det.refl_comb != 3:
497         print "\nWarning: The parameter 'refl_comb = %d' in function
           'gen_HeaderEndPOV' is not part of [1,2,3]!" % det.refl_comb
498 # Optical image
499 else: # det.ImgOption == 2:
500     if det.refl_comb == 1:
501         refl_params = 'reflection {0} ambient 0 diffuse 0.3 specular 0'
502     elif det.refl_comb == 2:
503         refl_params = 'reflection {0} ambient 0 diffuse 0.7 specular 0'
504     elif det.refl_comb == 3:
505         refl_params = 'reflection {0} ambient 0 diffuse 1.0 specular 0'
506     # No option of reflection parameters is choosen
507     else: # det.refl_comb != 1 and det.refl_comb != 2 and det.refl_comb != 3:
508         print "\nWarning: The parameter 'refl_comb = %d' in function
           'gen_HeaderEndPOV' is not part of [1,2,3]!" % det.refl_comb
509
510 print "The surface reflection parameters of the model are '%s'." % refl_params
511
512 # Generate end part of POV file and add it to POV file
513
514 # GIS data (CityGML)
515 if det.ModelOption == 1:
516     # Add brace for model
517     if bigmodelname == "box_full_scene.txt":
518         pt = "\n}"
519         pts(pt, endtext) # Save print output in log file
520
521     # Model information
522     pt = "\n\nobject {Building"
523     pts(pt, endtext) # Save print output in log file
524     pt = "rotate <0," + str(RotModel) + ",0>"
525     pts(pt, endtext) # Save print output in log file
526     pt = "translate <" + str(x_la) + "," + \
527         str(y_la) + "," + str(z_la) + ">"
528     pts(pt, endtext) # Save print output in log file
529
530 # DSM/DTM/nDSM
531 else: # det.ModelOption == 2:
532     # Model information
533     # pt = "\n\nobject {%s"%bigmodelname; pts(pt,endtext) # Save print output
           in log file
534     pt = "object {%s" % bigmodelname
535     pts(pt, endtext) # Save print output in log file
536     # Scale model coordinates --> Model coordinates in POV file have unit [m]
           (The unit of the spatial resolution of the model)
537     pt = "scale <" + str(model.XCellRes) + "," + \
538         str(model.ZCellRes) + "," + str(model.YCellRes) + ">"
539     pts(pt, endtext) # Save print output in log file
540     # Translate model to origin of POV coordinate system
541     pt = "translate <" + str(-x_la) + "," + \
542         str(-y_la) + "," + str(-z_la) + ">"
543     pts(pt, endtext) # Save print output in log file
544     # Rotate model in its center around POV up direction y
545     pt = "rotate <0," + str(RotModel) + ",0>"
546     pts(pt, endtext) # Save print output in log file
547     # Translate model to original location in POV coordinate system
548     pt = "translate <" + str(x_la) + "," + \
549         str(y_la) + "," + str(z_la) + ">"
550     pts(pt, endtext) # Save print output in log file
551     # Define texture of model with color pattern (pigment) and surface reflection
           features (finish)
552     pt = "texture {"

```

```

553     pts(pt, endtext) # Save print output in log file
554     pt = "           pigment {color rgb <1.0,1.0,1.0>}"
555     pts(pt, endtext) # Save print output in log file
556     pt = "           finish {%s}" % refl_params
557     pts(pt, endtext) # Save print output in log file
558     pt = "           }"
559     pts(pt, endtext) # Save print output in log file
560     pt = "}"
561     pts(pt, endtext) # Save print output in log file
562
563 # Output
564 return headtext, endtext, columns_simu, rows_simu
565
566
567 def POVfile_addPara(aPovfile, smallmodel, bigmodel, metadatafile, det):
568     # Take POV file and add header and end part of the file as required
569
570     """
571     Input:
572         aPovfile: Name of POV file
573         smallmodel: Small model name (Path to file of small model)
574         bigmodel: Big model name (Path to file of big model)
575         metadatafile: Orbit meta data file
576         det: Definition of input parameters (Look at class 'InputPara' in 'Script.py')
577     Output:
578         POV file with updated header and ending, after translation
579         columns_simu: Number of columns of simulated image
580         rows_simu: Number of rows of simulated image
581     Used function:
582         'GeoRayImg.gen_HeaderEndPOV'
583         'GeoRayImg.TransBigSmallModel'
584     Used in function:
585         'GeoRayImg.Model2GeoI'
586     """
587
588     # Split path of model file
589     path1, smallmodelname = os.path.split(smallmodel)
590
591     # Generate additional code parts (header and end part) for the POV file
592     headtext, endtext, columns_simu, rows_simu = gen_HeaderEndPOV(bigmodel, smallmodel,
593         , metadatafile, det)
594
595     # Translation if 'bigmodel' != 'smallmodel'
596     # --> If so: The simulated image will have the size of the model extent
597
598     # GIS data (CityGML)
599     if det.ModelOption == 1:
600         # Add translation between big model and small model to POV file
601         xtranslate, ytranslate, ztranslate = TransBigSmallModel(bigmodel, smallmodel,
602             det)
603
604         # Add translation between big model and small model to POV file
605         pt = "translate <" + str(xtranslate) + "," + str(ytranslate) + "," + str(
606             ztranslate) + "> // Difference between big model '%s' and small model '%s'\n"
607         % (bigmodel, smallmodel)
608         str_endtext = "".join(endtext)
609         index_translate = str_endtext.find('translate <')
610         str_endtext_new = str_endtext[:index_translate] + pt + str_endtext[
611             index_translate:]
612
613         fin = open(aPovfile, "r") # Open POV file
614         POVstr = fin.read() # Read POV file
615         fin.close() # Close POV file
616
617         # Update filename for output
618         dot_index = smallmodelname.index('.')
619         modelname_pure = smallmodelname[0:dot_index]
620         aPovfile2 = path1 + '/' + modelname_pure + ".pov"
621
622         fout = open(aPovfile2, "w") # Open POV file
623         fout.write("".join(headtext) + POVstr + "\n\n" + str_endtext_new) # Write in
624         POV file

```

```

619         fout.close() # Close POV file
620
621     # DSM/DTM/nDSM
622     else: # det.ModelOption == 2:
623         # Add translation between big model and small model to POV file
624         xtranslate, ytranslate, ztranslate = TransBigSmallModel(bigmodel, smallmodel,
        det)
625
626         # Add translation between big model and small model to POV file
627         pt = "translate <" + str(xtranslate) + "," + str(ytranslate) + "," + str(
        ztranslate) + "> // Difference between big model '%s' and small model '%s'\n"
        % (bigmodel, smallmodel)
628         str_endtext = "".join(endtext)
629         index_translate = str_endtext.find('translate <')
630         str_endtext_new = str_endtext[:index_translate] + pt + str_endtext[
        index_translate:]
631
632         fin = open(aPovfile, "r") # Open POV file
633         POVstr = fin.read() # Read POV file
634
635         fin.close() # Close POV file
636
637         if det.object_flag == 1 and det.plane_flag == 0:
638             a1 = POVstr.find('mesh2')
639             # a2 = POVstr.find('//endmodel')
640
641         if det.object_flag == 0 and det.plane_flag == 1:
642             a1 = POVstr.find('plane')
643             # a2 = POVstr.find('//endplane')
644
645         if det.object_flag == 1 and det.plane_flag == 1:
646             a1 = POVstr.find('mesh2')
647             # a2 = POVstr.find('//endplane')
648
649         if det.depth == 4:
650             a1 = POVstr.find('sphere')
651             # a2 = POVstr.find('//endmodel')
652
653         # Substitute translate and rotate routine for full scene model
654         # (required only for visibility check in processing depth level 2)
655         if len(det.path_to_adapted_pov_file) != 0 and det.object_flag == 1:
656
657             # Find key positions for inserting new code in the POV-Ray file
658             index_translate_full_model = POVstr.find('translate')
659             index_texture_full_model = POVstr.find('texture')
660
661             # Extract strings of full scene model
662             POVstr_part_1 = POVstr[0:index_translate_full_model]
663             POVstr_part_2 = POVstr[index_texture_full_model:]
664
665             bigmodel_temp = path1 + "/preDSM" # name of full scene model (needed for
        visibility check)
666
667             # Calculate necessary shifts along POV-Ray axes
668             xtranslate_temp, ytranslate_temp, ztranslate_temp = TransBigSmallModel(
        bigmodel_temp, smallmodel, det)
669
670             # Generate string for unifying the small and big model in the same
        POV-Ray coordinate system
671
672             # Difference between big model and small model (context: visibility check)
673             shift_small_vs_big = "translate <" + str(xtranslate_temp*(-1)) + "," + str(
        ytranslate_temp*(-1)) + "," + str(ztranslate_temp*(-1)) + ">"
674
675             # Difference between big model and small model (e.g. shift between small
        and extended model)
676             shift_extended_model = "translate <" + str(xtranslate) + "," + str(
        ytranslate) + "," + str(ztranslate) + ">"
677
678             # Build string for geometric object manipulation
679             POVstr = POVstr_part_1 + shift_small_vs_big + "\n" + shift_extended_model
        + "\n" + endtext[4] + endtext[5] + endtext[6] + endtext[7] + endtext[8] +

```

```

        endtext[7] + POVstr_part_2
680
681     fout = open(aPovfile, "w") # Open POV filev
682
683     if det.object_flag == 1:
684         fout.write("".join(headtext) + POVstr[a1:] + str_endtext_new) # Write in
        POV file
685     else:
686         fout.write("".join(headtext) + POVstr) # Write header and plane
        information to file
687
688     fout.close() # Close POV file
689
690 # Output
691 return columns_simu, rows_simu
692
693
694 def POVRay_POV2Img(aPovfile, columns_simu, rows_simu, sensor, det):
695     # SAR case: Run POV-Ray to render optical image based on POV file and to generate
        signal contribution file based on POV file
696     # Optical case: Run POV-Ray to render optical image based on POV file
697     """
698     Input:
699         aPovfile: Name of POV file
700         columns_simu: Number of columns of simulated image
701         rows_simu: Number of rows of simulated image
702         sensor: Name of sensor
703         det: Definition of input parameters (Look at class 'InputPara' in
        'Script.py')
704
705     Output:
706         SAR case: Rendered optical image and contribution file which contains
        signal contributions with coordinates and signal strengths and is
        necessary for Rendering of SAR image in RaySAR
707         Optical case: Rendered optical image
708
709     Used function:
710         None
711
712     Used in function:
713         'GeoRayImg.Model2GeoI'
714     """
715
716     # Simulate the optical image
717
718     # Split path of POV file
719     path1, povfile = os.path.split(aPovfile)
720
721     # Using POV-Ray for doing Ray Tracing
722     os.system(r'cd %s && povray %s +W%d +H%d Output_File_Name=%s -D' %
723              (path1, aPovfile, columns_simu, rows_simu, path1 + '/'))
724     # os.system(r'povray %s +W%d +H%d Output_File_Name=%s
725     +D'%(aPovfile,columns_simu,rows_simu,path1 + '/')') # POV-Ray should render
        optical image and display it
726
727     # SAR image
728     if det.ImgOption == 1:
729         # Change contribution filename
730         contribution_filename = aPovfile[:-4] + '_Contributions.txt'
731         # Change name of contribution
732         check_call('cd %s && mv Contributions.txt %s' %
733                   (path1, contribution_filename), shell=True)
734
735         # print "\nRendering of optical image and simulation of contribution file by
        Ray Tracing using POV-Ray:"
736         # print "The rendered optical image is '%s'"%(aPovfile[:-4] + '.png')
737         # print "The contribution file is '%s'"%contribution_filename
738
739         # Output
740         return contribution_filename
741
742     # Optical image
743     else: # det.ImgOption == 2:
744         # print "\nSimulation of %s image by Ray Tracing using POV-Ray:"%sensor
745         # print "Rendered optical image (%s): %s"%(sensor,aPovfile[:-4] + '.png')

```

```

742
743     # Output
744     return 1
745
746
747 def gen_RaySAR_Para(bigmodel, smallmodel, metadatafile, RaySARparafire, det):
748     # Generate SAR image parameters for the simulation in RaySAR (MATLAB part)
749     """
750     Input:
751         bigmodel: Big model name (Path to file of big model)
752         smallmodel: Small model name (Path to file of small model)
753         metadatafile: Orbit meta data file of SAR image
754         RaySARparafire: Generated output file for SAR image parameters for RaySAR
755         det: Definition of input parameters (Look at class 'InputPara' in
              'Script.py')
756
757     Output:
758         Parameter file with SAR image parameters for RaySAR
759     Used function:
760         'GeoRayImg.read_ModelPara'
761         'GeoRayImg.read_Metadata'
762         'GeoRayImg.UpRight'
763         'GeoRayImg.pts'
764     Used in function:
765         'GeoRayImg.Model2GeoI'
766     """
767
768     global sensor, filename, heading, AngOfView, incidence, meanHeight, PS_columns,
769     PS_rows
770
771     # Dummy distance in horizontal plane for calculating sensor position
772     D = 500 # unit [m]
773
774     # Split path of model file
775     path1, bigmodelname = os.path.split(bigmodel)
776
777     # Keep a list of lines that we want to save in POV file
778     lts = [] # Initialisation: Line to save in POV file
779
780     # Read parameters of model
781     model, H_max, H_min, H_mid, L, B, H = read_ModelPara(bigmodel, det)
782
783     # Get information from orbit meta data (Read parameters from meta data file of
784     image)
785     #sensor, filename, heading, AngOfView, incidence, meanHeight, PS_columns, PS_rows =
786     read_Metadata(metadatafile, bigmodel, smallmodel, det)
787
788     # Find the size of the projection image = cover area of sensor
789     # up: Perspective height of simulated optical image
790     # right: Azimuth of simulated optical image = Azimuth of simulated SAR image
791     up, right = UpRight(heading + 180, 90 - incidence, L, B, H)
792
793     # Size of simulated images (Columns * Rows; in POV software: Rendering with Az X
794     Rg)
795     # Number of columns of simulated optical image and of simulated SAR image
796     columns_simu = round(right / PS_columns)
797     # Number of rows of simulated optical image
798     rows_simu = round(up / (PS_rows * math.sin(incidence / 180 * math.pi)))
799
800     # Calculate the min and max values in range direction for the following SAR
801     simulation in MATLAB (RaySAR)
802     # Slant distance between model center (center of model box) and SAR sensor
803     Sslant = D / math.sin(incidence / 180 * math.pi)
804
805     # Generate SAR image Parameters required for RaySAR simulation in MATLAB
806
807     pt = "PixelSpacing_columns = " + str(PS_columns)
808     pts(pt, lts) # Save print output in log file
809     pt = "PixelSpacing_rows = " + str(PS_rows)
810     pts(pt, lts) # Save print output in log file
811     pt = "Azimuth_Min = " + str(-columns_simu * PS_columns / 2.0)
812     pts(pt, lts) # Save print output in log file
813     pt = "Azimuth_Max = " + str(columns_simu * PS_columns / 2.0)

```

```

808 pts(pt, lts) # Save print output in log file
809 pt = "Range_Min = %.2f" % (
810     (Sslant - up / 2.0) / math.sin(incidence / 180 * math.pi))
811 pts(pt, lts) # Save print output in log file
812 pt = "Range_Max = %.2f" % (
813     (Sslant - up / 2.0) / math.sin(incidence / 180 * math.pi) + rows_simu *
        PS_rows)
814 pts(pt, lts) # Save print output in log file
815 pt = "Bounce_level = " + str(det.max_bounce)
816 pts(pt, lts) # Save print output in log file
817 pt = "dB_range_Min = -25"
818 pts(pt, lts) # Save print output in log file
819 pt = "dB_range_Max = Max"
820 pts(pt, lts) # Save print output in log file
821 pt = "RangeCoord_topdown = 1"
822 pts(pt, lts) # Save print output in log file
823 pt = "Binomial_filt = 1"
824 pts(pt, lts) # Save print output in log file
825 pt = "GroundRangeGeometry = 1"
826 pts(pt, lts) # Save print output in log file
827 pt = "Incidence = " + str(incidence)
828 pts(pt, lts) # Save print output in log file
829
830 # Additional parameters to identify the parameter file
831 pt = "modelfile = %s" % (os.path.join(path1, bigmodelname))
832 pts(pt, lts) # Save print output in log file
833 pt = "%s_image = %s" % (sensor, filename)
834 pts(pt, lts) # Save print output in log file
835 pt = "heading_Az = " + str(heading)
836 pts(pt, lts) # Save print output in log file
837
838 # print "\nInput for calculation of %s simulation parameters for RaySAR:"%sensor
839 # print "%s image: '%s'"%(sensor,filename)
840 # print "Model: '%s'"%(path1 + '/' + bigmodelname)
841 print "\nOutput of %s simulation parameters for RaySAR is stored in '%s'\n" % (
    sensor, RaySARparafilename)
842
843 fileobj = open(RaySARparafilename, "w") # Open parameter file
844 fileobj.write("".join(lts)) # Write in parameter file
845 fileobj.close() # Close parameter file
846
847 # Output
848 return 1
849
850
851 def RaySARSimu(contribution_filename, Output_path, RaySARparafilename):
852     # Run RaySAR of linux version and do SAR simulation
853     """
854     Comment:
855         Contribution file not allowed to be bigger than 50 MB
856     Input:
857         contribution_filename: Contribution file which contains signal
            contributions with coordinates and signal strengths (Output of
            'POVRay_POV2Img')
858         Output_path: Path where the output folder should be saved
859         RaySARparafilename: Parameter file with SAR image parameters for RaySAR
            (Output of 'gen_RaySAR_Para')
860     Output:
861         Directory 'Maps' in 'Output_path'
862         To see: 'Maps/Frames/Ref_Maps/... .tif'
863     Used function:
864         None
865     Used in function:
866         'GeoRayImg.Model2GeoI'
867     """
868
869     # Start RaySAR with MATLAB
870     #os.system(r' ../RaySARLinux/matlab_batcher_raysar.sh raysar_linux_function "' +
        '"%s"'%contribution_filename + '" "' + '"%s"'%Output_path + '" "' +
        '"%s"'%RaySARparafilename + '"')
871
872 import Gen_Refl_Map

```

```

873
874 # Initiate SAR simulation
875 Gen_Refl_Map.raysar_linux_function(
876     contribution_filename, Output_path, RaySARparafire)
877
878 print "\nSAR simulation done."
879 print "\nThe SAR simulation results are stored in '%s/Maps/'" % Output_path
880
881 # Change name of simulated SAR images in folder 'Maps'
882 listing = os.listdir(os.path.join(
883     Output_path, 'Maps', 'Frames', 'Ref_Maps'))
884 for infile in listing:
885     texta, textb = os.path.splitext(infile)
886     bounce = texta
887     if bounce == 'All Reflections_Fr':
888         simpsimun = 's0'
889         bounce = 'All\ Reflections_Fr'
890     elif bounce == 'Single Bounce_Fr':
891         simpsimun = 's1'
892         bounce = 'Single\ Bounce_Fr'
893     elif bounce == 'Double Bounce_Fr':
894         simpsimun = 's2'
895         bounce = 'Double\ Bounce_Fr'
896     elif bounce == 'Triple Bounce_Fr':
897         simpsimun = 's3'
898         bounce = 'Triple\ Bounce_Fr'
899     else: # There is no simulated SAR image available
900         print "No suitable simulated SAR image available for geocoding."
901         simpsimun = 'NOFILE'
902     if simpsimun != 'NOFILE': # Change name
903         # Change name of folder with simulated SAR images in folder 'Maps'
904         check_call('mv %s/Maps/Frames/Ref_Maps/%s.tif' % (Output_path, bounce) +
905             ' %s/Maps/Frames/Ref_Maps/%s.tif' % (Output_path, simpsimun),
906             shell=True)
907         print "Image '%s%s' represents '%s'" % (simpsimun, textb, texta[:-3])

```