CYPRIAN LEWANDOWSKI

EXPORT VOXEL DATA

MATLAB PACKAGE MANUAL

Contents

Introduction

The Export Voxel Data package was written as an interface between voxel data stored as a matrix in Matlab[1] and triangulated mesh format used in 3D file formats such as STL[2] and/or software such as Blender[3] or PovRay[4].

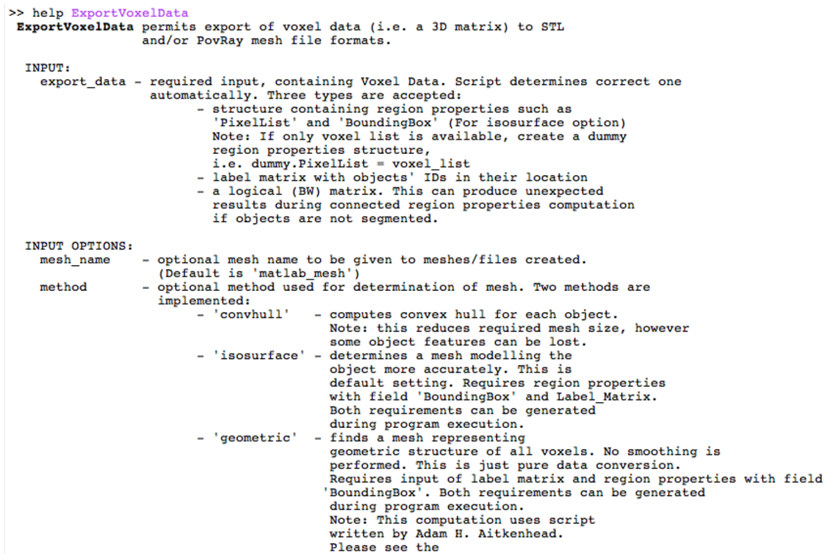
[1] Information about the most recent version is available at: http://www.mathworks.com/products/matlab/.

[2] An excellent description is provided by Wikipedia: http://en.wikipedia.org/wiki/STL_(file_format).

[3] Blender is a free and open-source 3D computer graphics software: http://www.blender.org.

[4] PovRay is a free ray tracing software: http://www.povray.org.

Main script dealing with user interaction is `ExportVoxelData.m`. It contains an associated help document explaining structure of input and output. Figure 1 shows included help file, which can be obtained by entering `help ExportVoxelData` command in Matlab.



```
>> help ExportVoxelData
 ExportVoxelData permits export of voxel data (i.e. a 3D matrix) to STL
                 and/or PovRay mesh file formats.

  INPUT:
    export_data - required input, containing Voxel Data. Script determines correct one
                  automatically. Three types are accepted:
                        - structure containing region properties such as
                          'PixelList' and 'BoundingBox' (For isosurface option)
                          Note: If only voxel list is available, create a dummy
                          region properties structure,
                          i.e. dummy.PixelList = voxel_list
                        - label matrix with objects' IDs in their location
                        - a logical (BW) matrix. This can produce unexpected
                          results during connected region properties computation
                          if objects are not segmented.

  INPUT OPTIONS:
    mesh_name     - optional mesh name to be given to meshes/files created.
                    (Default is 'matlab_mesh')
    method        - optional method used for determination of mesh. Two methods are
                    implemented:
                        - 'convhull'   - computes convex hull for each object.
                                         Note: this reduces required mesh size, however
                                         some object features can be lost.
                        - 'isosurface' - determines a mesh modelling the
                                         object more accurately. This is
                                         default setting. Requires region properties
                                         with field 'BoundingBox' and Label_Matrix.
                                         Both requirements can be generated
                                         during program execution.
                        - 'geometric'  - finds a mesh representing
                                         geometric structure of all voxels. No smoothing is
                                         performed. This is just pure data conversion.
                                         Requires input of label matrix and region properties with field
                                        'BoundingBox'. Both requirements can be generated
                                         during program execution.
                                         Note: This computation uses script
                                         written by Adam H. Aitkenhead.
                                         Please see the
```

Figure 1: Help document is included in the package.

How does the package work?

Process of exporting voxel data can be divided into several stages as illustrated in figure 2. This separation in distinct stages provides the program with modular structure and hence individual elements can be extended or used within body of other scripts.

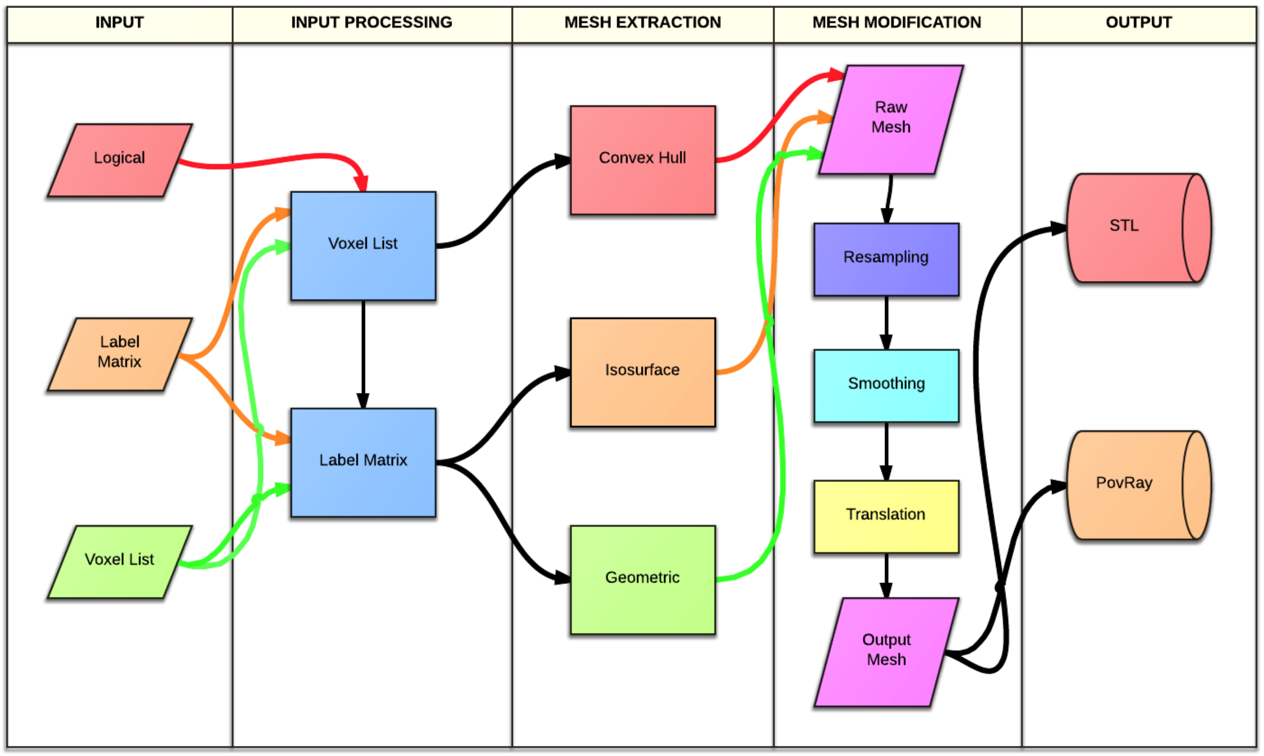


Figure 2: Workflow of the Export Voxel Data package. *Note: Arrows between "Input Processing" and "Mesh Extraction" stages indicate what format is used in each of these methods.*

Data to be exported is inputed to `ExportVoxelData.m` with optional arguments specifying, for example, output format, name, mesh extraction method etc. Following data input, the package performs necessary conversions of the data, so that desired mesh generation method can be used. After export data has been prepared mesh is extracted and enters the MESH MODIFICATION part of the package. At this stage mesh size can be reduced (resampling), smoothed out or translated to another coordinate system. Such prepared mesh is then exported to formats of choice and saved to HDD.

EXPORTED mesh files are not the only information that the package can output. There are additional scripts `ExportObjectsBlender.m` and `ExportObjectsPovRay.m`, which can generate colouring information for exported objects using a scalar array as a key. An example would be ordering of particles by some physical property, such as number of contacts or packing fraction. The POVRAY script can also be used to create a simple scene, which can be rendered out of a box. The BLENDER script outputs a configuration file, which can be used together with a custom written BLENDER add-on `io_mesh_stl_batch`[5] to import exported objects to BLENDER with

[5] All BLENDER add-ons to be mentioned in the manual are located in `Blender Addons` folder within the main package directory.

specified colour and transparency.

Different options and parameters will be discussed in the manual, hence it is worth pointing out the general structure and nomenclature used. Order of options does not matter, but they need to use following command pattern:

```
ExportVoxelData(input_data, 'option_name', option_value, ...
    'option_name_2', option_value_2);
```

For the remainder of the manual, name of an option will be referred to as `option_name` and its value as `option_value`.

Manual layout

In this document most of package features will be presented with a Matlab example and an accompanying output. Code used for generation of examples will be included within body of this document, but there is also an associated `Manual_Examples.m` script, which can be opened with Matlab and used alongside the manual. Most of images used to illustrate output are made with Pleasant3D[6] application, which, among other things, allows for quick viewing of STL files.

[6] Pleasant3D software is a small STL editing application: `http://www.pleasantsoftware.com/developer/pleasant3d/`.

The ExportVoxelData package was written as a by-product of my main Matlab project, which was reconstruction of spaghetti packing, however for all of manual examples, input data can be generated in Matlab with following code (also present in the `Manual_Examples` script):

```
%% Create data
% This cell creates data that is used throughout the ...
    Manual_Data script

% Make coordinate system
[X,Y,Z] = meshgrid(-249:250);

% Nonintersecting objects
% Create empty matrix
BW_NI = false(500,500,500); % Black and White image (logical)

% Add Sphere
BW_NI((X-50).^2+(Y-50).^2+(Z+30).^2 ≤ 400) = true;

% Add Ellipsoid
BW_NI((X/20).^2+(Y/50).^2+(Z/100).^2 ≤ 1) = true;

% Intersecting Objects
% Create empty matrix
BW_I = false(500,500,500);
LM_I = zeros(500,500,500, 'uint8'); % Label matrix
```

```

% Add Sphere
BW_I((X-50).^2+(Y-50).^2+(Z+30).^2 <= 10000) = true;
LM_I((X-50).^2+(Y-50).^2+(Z+30).^2 <= 10000) = 1;

% Add Ellipsoid
BW_I((X/20).^2+(Y/50).^2+(Z/100).^2 <= 1) = true;
LM_I((X/20).^2+(Y/50).^2+(Z/100).^2 <= 1) = 2;

% Determine voxel list through finding region properties
RP_I = regionprops(LM_I, 'PixelList');

clear X Y Z
```

Input

CURRENTLY three input formats are supported: logical, label matrix and voxel list. If necessary the package can be extended to support additional formats, but from experience with my data, voxel list and label matrix are sufficient (and preferred) formats for performing most of data analysis.

Logical

A logical matrix is a matrix where each entry is either `true` or `false`.[7] This visually corresponds to a black and white image. Using the data generated for the purpose of the manual (as described in the Manual Layout section), let's first use the image of two nonintersecting objects, that is of a sphere and an ellipsoid. Output shown in figure 3 can be generated with code:

[7] I prefer to think of it as a scalar field with a limited domain, which can take only two values.

```
1  %% Input — Logical — Example 1
2
3  ExportVoxelData(BW_NI);
```

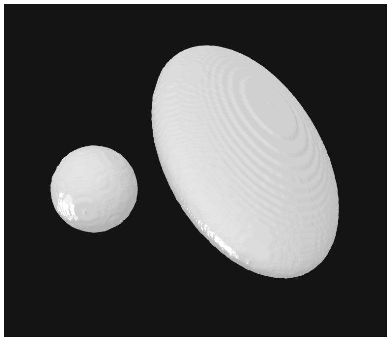


Figure 3: A logical image of nonintersecting sphere and ellipsoid.

Line 3 is the simplest execution of the EXPORTVOXELDATA script.

It automatically detects type, performs extraction of the mesh using `isosurface` method and does one iteration of surface smoothing. By default it outputs to directory of script execution two files `matlab_mesh.stl` and `matlab_mesh.inc`. The first one contains mesh in STL binary format and the second one contains `mesh2`[8] format used for mesh description in POVRAY. Both formats are discussed in chapters "Exporting voxel data to BLENDER " and "Exporting voxel data to POVRAY " respectively.

[8] More information available at: `http://www.povray.org/documentation/view/3.6.1/68/`.

That's it, no more scripting or coding is required. Exported voxel data is ready to be used.

It is worth mentioning that if the logical image contains two objects, which are not separated, i.e. two objects are intersecting or just touching, then the EXPORTVOXELDATA package will detect them as a single object.

Consider executing following code:

```
%% Input — Logical — Example 2

ExportVoxelData(BW_I);
```

As it can be seen in figure 4, this produces no changes that can be observed, but when one would like to export colour information about those two objects, it would be impossible to do so as two objects cannot be distinguished in the logical input format.

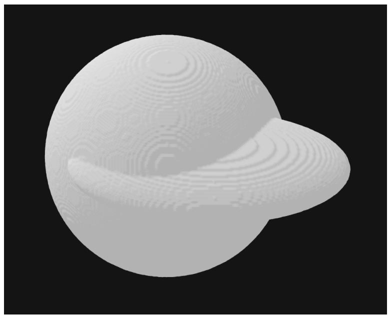


Figure 4: STL file of two intersecting objects.

Label matrix

A label matrix is a matrix of identical dimensions to the logical image, but regions corresponding to different objects are labelled with consecutive integers rather than just simply being labelled as `true`. For well separated objects such as the one in Example 1 this produces no visual difference. For intersecting objects as generated by code below, the output is again visually identical, but during the exporting process, two objects are being exported rather than just one. This has further application when one considers option `object_ids`.

```
%% Input — Label Matrix — Example 3

ExportVoxelData(LM_I);
```

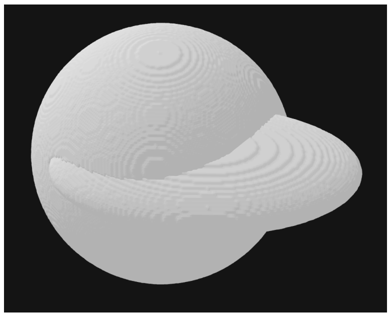


Figure 5: Exported image of two intersecting objects from Label Matrix. *Note: No visual difference is present as compared to* L*OGICAL input data.*

Voxel list

More precisely it is a structure similar to the output of MATLAB's `regionprops`[9] function. Each entry of the structure contains a field, which is an array of voxels that constitute the object. An example of such structure was generated in the Manual layout section under variable name `RP`. Again, using default setting as in the code below, no visual difference can be observed, which is illustrated in figure 6.

[9] Information about the function: `http://www.mathworks.com/help/images/ref/regionprops.html`.

```
%% Input — Voxel List — Example 4

ExportVoxelData(RP_I);
```

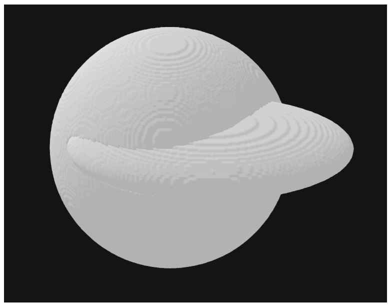


Figure 6: Output from voxel list used as input data.

SO FAR it might seem that there is no significant advantage of using one input format over another, however as more options are being used from the package or if just one format is available, then support for different types of input becomes handy.

Mesh Extraction

Convex hull *method*

This is the most basic method for mesh extraction implemented in the package. It is performed using Matlab's function `convhull`[10], thus it is very computationally efficient. For a given input data it finds exterior voxels of each region, connects them and uses such created surface for mesh extraction.

[10] Information about the function: `http://www.mathworks.com/help/matlab/ref/convhull.html`.

Consider the two nonintersecting objects in logical voxel data format - the `BW_NI` matrix. Command used to export them using Convex hull method is:

```
%% Mesh Extraction — Convexhull — Example 5

ExportVoxelData(BW_NI, 'method', 'convhull');
```

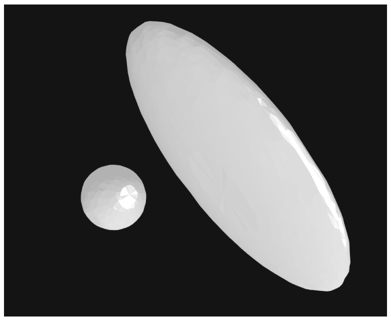


Figure 7: Convex hull exported objects with no intrinsic surface concavity visually appear identical to the isosurface method.

Visually the two objects (shown in figure 7) look very similar to Example 1, however surface of objects appears more triangulated.

It is also worth pointing out differences in file size. For the default method - isosurface, STL binary file is (in this case) 5.8 MB as compared to the convex hull file that is 223 kB.

On the other hand when an object being exported possesses some surface concavity, then convex hull method produces undesirable results, as illustrated in figure 8 obtained with Matlab's code:

```
%% Mesh Extraction — Convexhull — Example 6

ExportVoxelData(BW_I, 'method', 'convhull');
```

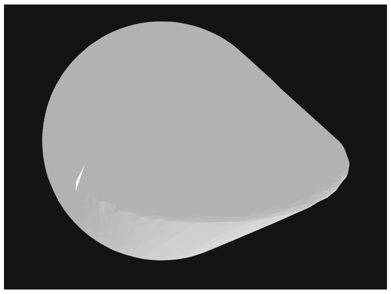


Figure 8: Intersecting objects produce a shape with surface concavity. *Note: In this case if the two objects were inputed as label matrix or voxel list such effect would not have been obtained.*

In such cases one has to fall back to using other mesh extraction methods, which although produce larger output files, can accommodate such surface concavity as shown in figure 9.

Isosurface *method*

This is the default mesh extraction method. The function that performs determination of object's surface is again a Matlab's function `isosurface`[11].

If necessary one can be explicit in the method he uses by executing the ExportVoxelData script in following form:

```
%% Mesh Extraction — Isosurface — Example 7

ExportVoxelData(BW_I, 'method', 'isosurface');
```

[11] Information about the function: http://www.mathworks.com/help/matlab/ref/isosurface.html.

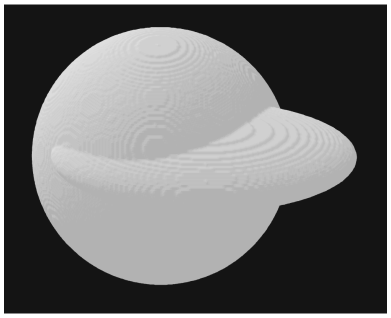


Figure 9: Object with surface concavity is correctly exported using the isosurface mesh extraction method.

Geometric *method*

The implemented geometric method is based on a code developed by Aitkenhead A. H.[12] and adapted to work with the ExportVoxelData package.

In principle this method never has to be used, as essentially it creates the same output to the isosurface method both visually and file size, while being much slower. This is because the voxelisation stage is implemented within interpreted Matlab code rather than as a fast, precompiled Matlab's function that `isosurface` is. Geometric method was included in the package out of academic interest and can be used as shown in the code extract below:

[12] More information about the package: http://www.mathworks.com/matlabcentral/fileexchange/27733-converting-a-3d-logical-array-into-an-stl-surface-mesh.

```
%% Mesh Extraction — Geometric — Example 8

ExportVoxelData(BW_I, 'method', 'geometric');
```

Figure 10 illustrates this example's output. If one zooms to view the surface with more detail and compares it with figure 9, it can be seen that output of geometric method contains a much finer mesh, but the overall shape is not much different.

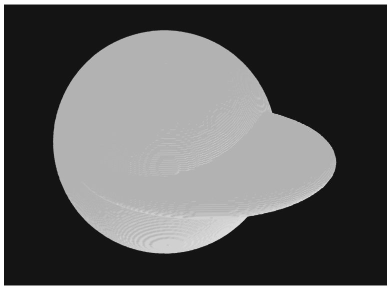


Figure 10: STL output of geometric mesh extraction method.

Mesh Modification

EXTRACTED MESH can be further modified to improve quality of the output. The two possible mesh modifications, which produce visual changes are: resampling and smoothing. Other parameters provided by the package will be discussed in next chapter.

Resampling

As more objects (larger voxel data) is being exported, logically size of output file increases. It is desirable to be able to reduce file size, so that further image processing, such as importing to other programs, rendering images can be done more efficiently, while still producing similar quality of visual output.

This is accomplished by resampling the exported mesh, which means that number of vertices in the mesh is reduced. A script from ISO2MESH[13] package was adapted and included within the EXPORTVOXELDATA package.

[13] More information about the package: `http://iso2mesh.sourceforge.net/`.

Resampling can be varied as illustrated by following code:

```
%% Mesh Modification — Resampling — Example 9

ExportVoxelData(BW_I, 'resample', 0.2);
```

The `option_value` is a number between 0 to 1, which defines fraction of vertices remaining in the resampled mesh. To large extent process of finding appropriate resampling value is based on an educated guess. One has to export data with various values of resampling parameter and decide at which stage visual information is being lost at a cost of reducing output file size.

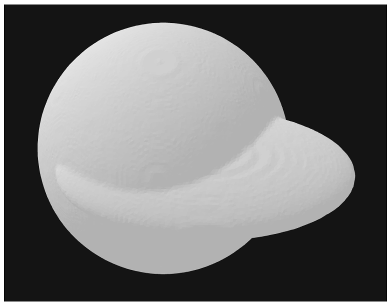


Figure 11: Resampled image with 0.2 original number of vertices remaining.

In this example, exported mesh (Figure 11) contains only 0.2 of original number of vertices, file size is reduced also approximately 5 times and it is a matter of opinion if image visually looks similar to the original mesh - Figure 4.

By default no mesh resampling is done, i.e. `option_value` of `resample` is set as 1.

Smoothing

Voxel data (by definition) is stored in discrete elements. This results in a voxelised mesh having a discrete - "lego" like - texture. To obtain a smooth looking surface, a smoothing package by Kroon. D.[14] was adapted. Main part of the smoothing is implemented in `C`, which is compiled on first package execution on a given workstation.

[14] More information about the package: `http://www.mathworks.com/matlabcentral/fileexchange/26710-smooth-triangulated-mesh`.

Consider first an example code:

```
%% Mesh Modification — Smoothing — Example 10

ExportVoxelData(BW_I, 'smoothing', false);
```

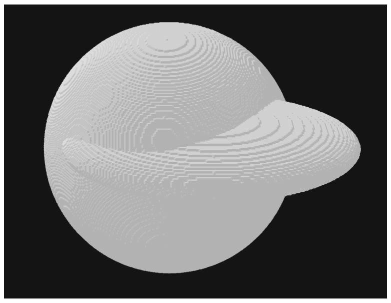


Figure 12: No mesh smoothing.

This produces output that can be seen in figure 12. `option_value` in this case is set as `false`, which implies no smoothing. A scalar value of 0 could be inputed to produce similar result.

The smoothing script has itself five parameters that specify how the smoothing is performed. They can be inputed like:

```
%% Mesh Modification — Smoothing — Example 11

ExportVoxelData(BW_I, 'smoothing', struct('mode',1, ...
    'itt',10, 'lambda',1, 'sigma',1));
```

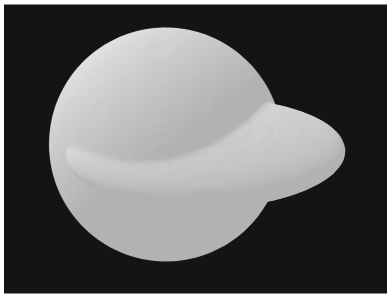


Figure 13: Mesh smoothing with 10 iterations. Virtually no texture due to discrete voxel data nature remains.

This example code produces an exported object (shown in figure 13) for which smoothing procedure was executed 10 times.

All options are described in detail in `N_SmoothMesh.m` file, but an extract of the file is shown below (similar output can be obtained via `help N_SmoothMesh` command):

```
%   mode : value 0 or 1 (default)
%          If zero uses inverse distance between vertices ...
    as weights. The
%          mesh becomes smoother but also edge distances ...
    more uniform
%          If one uses the normalized curvature operator ...
    as weights. The
%          mesh is mainly smoothed in the normal ...
    direction, thus the
%          original ratio in length between edges is ...
    preserved.
%   itt : Number of smoothing iterations (default 1)
%   lambda : Amount of smoothing [0....1] (default 1)
%   sigma : (If mode is 0), Influence of neighbour point is
%              Weight = 1 / ( inverse distance + sigma)    ...
    (default sigma=1);
```

By default following structure is used as configuration of the smoothing script:

`struct('mode',1,'itt',1,'lambda',1,'sigma',1)`

Options

The two options described in previous chapter performed modification of the extracted mesh that produced visual changes. Features described in this chapter affect output location, mesh name, objects exported or coordinate system used by the mesh. They were introduced so that more efficient usage of the ExportVoxelData can be achieved.

`pov` and `stl` - output specific file format

In some cases it might be desirable to export voxel data only in one output file format. For example to export only STL file one has to set `pov` flag as `false` or 0.

```
%% Output — STL only — Example 12

ExportVoxelData(BW_I, 'pov', false);
```

In a similar fashion to export only PovRay file one has to set `stl` flag as `false` or 0.

By default both file formats are exported.

`mesh_name` - change mesh name

In order to change name of exported mesh and, what comes along, also names of exported files, one has to set `mesh_name` flag with a string specifying new mesh name. This is shown in listing below:

```
%% Options — mesh_name — Example 13

ExportVoxelData(BW_I, 'mesh_name', 'new_mesh_name');
```

By default "`matlab_mesh`" is used as `option_value` of `mesh_name`.

`output_dir` - *change output folder*

To change export directory to a given folder, it is necessary to set `option_value` of `output_dir` option. For example, in the following code, output is saved in a folder "`my_output_folder`":

```
%% Options — output_dir — Example 14

ExportVoxelData(BW_I, 'output_dir', 'my_output_folder');
```

`option_value` can be either a relative or absolute path to a folder. In case if the folder does not exist it will be created.

By default all output is saved in directory of code execution, i.e. `option_value` is an empty string.

`label_matrix` - *input label matrix*

If during data analysis label matrix was created alongside of correctly structured voxel list, both can be entered to the package to reduce amount of calculations performed. Code below shows structure of the command:

```
%% Options — label_matrix — Example 15

ExportVoxelData(RP_I, 'label_matrix', LM_I);
```

Visually no different outcome to the one shown in "Input" chapter is obtained, but amount of computations required is reduced.

By default, if option `label_matrix` is empty and label matrix is required for a chosen mesh extraction method (that is both ISOSURFACE and GEOMETRIC methods require label matrix), label matrix will be created from entered input data, unless the input data already is a label matrix.

`object_ids` - *export only chosen objects*

To export only some of objects contained in the voxel data, their ids - integer numbers representing either regions objects occupy in label matrix or entries in region properties (voxel list) structure can be entered.

Consider code below:

```
%% Options — object_ids — Example 16

ExportVoxelData(LM_I, 'object_ids', 1);
```

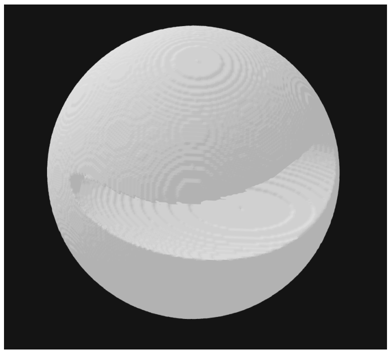


Figure 14: Only specified object is exported. *Note: Shape of the object is due to creation of the label matrix.*

Only the sphere corresponding to the first region is exported as shown in figure 14.

By default all objects are exported from the entered input data.

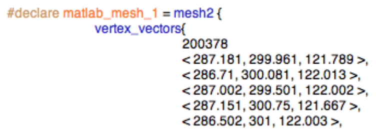
`shift_origin` - translate object's coordinate system

MATLAB's coordinate system starts at $(1,1,1)$ as it is linked to the matrix format. This results in the centre of the exported voxel data being at geometric centre of the matrix, i.e. if matrix has dimensions (A,B,C) the geometric centre is at $(\frac{A}{2},\frac{B}{2},\frac{C}{2})$. Most of 3D visualisation software, for example BLENDER or POVRAY, prefer to have centre of the image at $(0,0,0)$, which makes rotations or other mesh operations easier to implement.

To accomplish this, an option `shift_origin` was added to the EXPORTVOXELDATA package. Consider first output with no origin shift, i.e. essentially just as Example 1, but to make it explicit:

```
%% Options - shift_origin - Example 17

ExportVoxelData(BW_I, 'shift_origin', 0);
```



```
#declare matlab_mesh_1 = mesh2 {
              vertex_vectors{
                            200378
                            < 287.181, 299.961, 121.789 >,
                            < 286.71, 300.081, 122.013 >,
                            < 287.002, 299.501, 122.002 >,
                            < 287.151, 300.75, 121.667 >,
                            < 286.502, 301, 122.003 >,
```

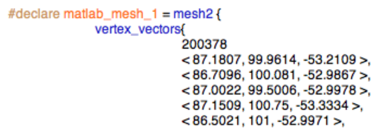
Figure 15: Coordinates of vertices with no origin shift.

Shifting origin produces no difference in terms of object shape. Let's look at coordinates of vertices of the mesh as given in POVRAY `matlab_mesh.inc` file. Part of the file is shown in figure 15.

Now if origin shift is performed:

```
%% Options - shift_origin - Example 18

ExportVoxelData(BW_I, 'shift_origin', 1);
```



```
#declare matlab_mesh_1 = mesh2 {
              vertex_vectors{
                            200378
                            < 87.1807, 99.9614, -53.2109 >,
                            < 86.7096, 100.081, -52.9867 >,
                            < 87.0022, 99.5006, -52.9978 >,
                            < 87.1509, 100.75, -53.3334 >,
                            < 86.5021, 101, -52.9971 >,
```

Figure 16: Coordinates of vertices with estimated value of origin shift.

In this case, as can be seen in figure 16, some of coordinates become negative, which corresponds to the fact that now geometric center is approximately coincident with $(0,0,0)$ in Cartesian coordinate system.

By default no origin shift is performed, that is `shift_origin` is set as 0. It is worth pointing out that this flag accepts only 0 or 1 as input, due to internal code structure imposed by another option of the package - `img_dim`.

`img_dim` - provide dimensions of exported object

Estimate of the geometric centre, which is necessary to perform origin shift, is done from list of voxels occupied by all particles being exported. This estimate can differ in value from the true input matrix dimensions, hence image dimensions can be entered manually as given in following listing:

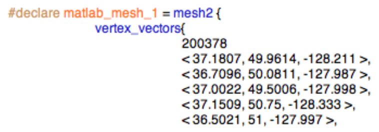
```
%% Options — img_dim — Example 19

ExportVoxelData(BW_I, 'img_dim', size(BW_I))
```

Resulting coordinates of the same vertices as in previous section are presented in figure 17. As it can be seen the difference between these values and those from figure 16 is significant. This can be the case if exported voxel data is mostly empty.

Also inputing just `option_value` for `img_dim` is treated as informing the package that origin shift is to be performed, i.e. it is not necessary to set `shift_origin` to 1.

Image dimensions are not only used for determination of true origin shift. They can also be used to speed up the process of label matrix creation. If one wanted to speed up process of label matrix creation, but not to shift origin of the image, setting `shift_origin` to 0 accomplishes that. This is also the reason why `option_value` of `shift_origin` cannot be a logical type as information whether origin shift to be performed is a default setting or user input has to be kept.



```
#declare matlab_mesh_1 = mesh2 {
                vertex_vectors{
                                200378
                                < 37.1807, 49.9614, -128.211 >,
                                < 36.7096, 50.0811, -127.987 >,
                                < 37.0022, 49.5006, -127.998 >,
                                < 37.1509, 50.75, -128.333 >,
                                < 36.5021, 51, -127.997 >,
```

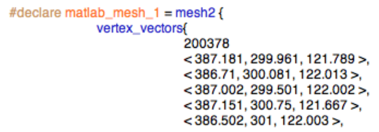
Figure 17: Coordinates of vertices with true value of origin shift.

`img_shift` - translate object's mesh by a given vector

In some cases it might be desirable to shift origin of the mesh by some specified displacement vector. Consider following command:

```
%% Options — img_shift — Example 20

ExportVoxelData(BW_I, 'img_shift', [100, 0, 0]);
```



```
#declare matlab_mesh_1 = mesh2 {
                vertex_vectors{
                                200378
                                < 387.181, 299.961, 121.789 >,
                                < 386.71, 300.081, 122.013 >,
                                < 387.002, 299.501, 122.002 >,
                                < 387.151, 300.75, 121.667 >,
                                < 386.502, 301, 122.003 >,
```

Figure 18: Shift of origin by a given displacement vector.

Figure 18 shows that the x coordinate was shifted by 100 units in positive direction, while y and z coordinates remain unchanged. This corresponds to the entered $(100, 0, 0)$ value of displacement vector as the `option_value`.

By default the displacement vector is set as $(0, 0, 0)$, which corresponds to no additional origin shift.

Exporting voxel data to PovRay

Structure of PovRay `.inc` *files*

Outputted files, to maintain generality, contain just description of mesh structure of the voxel data. Every exported object is labelled in the `.inc` file using following pattern: `mesh_name'_'object_id`. For example if `mesh_name` is `my_mesh` and there are two objects exported, then the first object will have name of its mesh given as `my_mesh_1` and the second object will have name `my_mesh_2`. In addition all exported objects can be referred to as a whole using `mesh_name_all`, i.e. `my_mesh_all` in this case.

Creating PovRay *scene*

ExportVoxelData package contains support for a simple PovRay scene creation, which can be rendered out of the box and shows exported objects together with some colour, texture and transparency setting.

Consider exporting first the `BW_NI` data and colouring all objects in yellow, completely opaque and with no texture. This is accomplished by code in Example 21 in `Manual_Examples.m` file shown below:

```
%% Exporting Voxel Data to PovRay — No colouring — ...
    Example 21

% Export only PovRay files
ExportVoxelData(BW_NI, 'stl', false);

% Define parmeters of the export
scene_name = 'matlab_scene'; % Name of created file
mesh_name  = 'matlab_mesh'; % Name of the .inc file
dim        = size(BW_NI); % Dimensions of the image

% Create a cell array of strings with names of other .inc ...
    files, such as texture
% files etc, to be included as well apart of the mesh ...
    description .inc file
```

```
13 include_files = {'colors'};
14
15 % Create mesh_list of objects to export
16 mesh_list.name = [mesh_name '_all']; % Union of all ...
       exported objects is always accesible with
17                                   % mesh_name_all
18 mesh_list.texture  = ''; % No texture
19 mesh_list.rgb      = [1, 1, 0]; % Set as yellow
20 mesh_list.transmit = 0;
21
22 % Add file with object meshes to the list
23 include_files = [ include_files mesh_name ];
24
25 % Create PovRay scene configuration
26 pov_configuration.scene_name      = scene_name;
27 pov_configuration.include_files   = include_files;
28 pov_configuration.camera_location = [ round(dim(2)/2), ...
       round(dim(1)/2), -(dim(3)+200) ]; % Camera location
29 pov_configuration.camera_look_at  = [ round(dim(2)/2), ...
       round(dim(1)/2), round(dim(3)/2) ]; % Camera look at
30
31 clear mesh_name scene_name include_files
32
33 % Add PovRay ligth sources
34
35 % Preallocate
36 pov_configuration.light_source = zeros(4,3); % Four light ...
       sources
37
38 % Add light sources
39 pov_configuration.light_source(1,:) = [ dim(2)+20, ...
       dim(1)+20, dim(3)+20 ];
40 pov_configuration.light_source(2,:) = [ 1, dim(1)+20, ...
       dim(3)+20 ];
41 pov_configuration.light_source(3,:) = [ dim(2)+20, 1, ...
       dim(3)+20 ];
42 pov_configuration.light_source(4,:) = [ dim(2)+20, ...
       dim(1)+20, 1 ];
43
44 clear dim
45
46 % Create PovRay Scene
47 N_PovCreateScene(pov_configuration, mesh_list);
48
49 clear pov_configuration mesh_list
```

Line 4 indicates exporting only PovRay data, similarly to exporting just STL file in Example 12.

Lines 6 to 9 specify parameters for the export. `scene_name` just sets name for exported `.pov` file, `mesh_name` informs the script under what name was the voxel data exported as it corresponds to the `.inc` file to be included. Note that this does not support exporting voxel data to other directory as provided by `output_dir` option (unless one sets PovRay to search that directory for `.inc` files). `dim` is just a handy short-cut for vector defining dimensions of exported data.

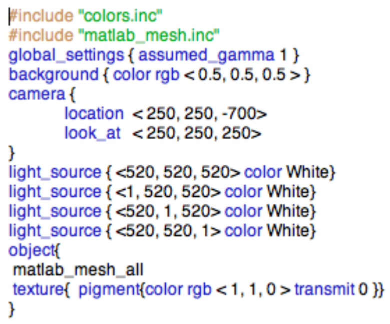
Line 13 can be used to specify what other files are to be included. In this case it includes description of colours, by default provided with every PovRay provided distribution.

Line 15 creates the `mesh_list` structure. First field is `name`, which specifies the name of the mesh to be rendered. In this case it will render all exported objects as provided by the reference `matlab_mesh_all`.

Line 18 sets texture field of the exported mesh. In this case no texture is used, hence it is left as empty.

Line 19 sets `RGB` colour[15] of the object. In this case colour of exported mesh is set as yellow.

[15] More information about specifying colours, transmittance and texture in PovRay available at: `http://www.povray.org/documentation/view/3.7.0/230/`.



```
#include "colors.inc"
#include "matlab_mesh.inc"
global_settings { assumed_gamma 1 }
background { color rgb < 0.5, 0.5, 0.5 > }
camera {
        location  < 250, 250, -700>
        look_at   < 250, 250, 250>
}
light_source { <520, 520, 520> color White}
light_source { <1, 520, 520> color White}
light_source { <520, 1, 520> color White}
light_source { <520, 520, 1> color White}
object{
 matlab_mesh_all
 texture{  pigment{color rgb < 1, 1, 0 > transmit 0 }}
}
```

Figure 19: Exported PovRay file. *Note: All exported objects are referred to with one declaration.*

Line 20 defines `transmit` field of the mesh, which informs PovRay about transparency of the object.

Line 23 appends name of the mesh file to the list of include files. This could be changed if mesh is present in alternative directory as specified by `output_dir` option of the ExportVoxelData script. Please note that all names of libraries to include do not have extension. It is assumed by the script that default extension `.inc` is used.

Lines 25 to 27 create a structure with configuration of PovRay scene just for easier argument passing. This includes the scene name and what files to include.

Lines 28 and 29 append to the structure specification of PovRay camera[16] for this scene, which attempts to center on exported objects in the rendered PovRay scene.

[16] More information about PovRay camera available at: `http://www.povray.org/documentation/view/3.7.0/245/`.

Lines 35 to 42 add information about lightning to the configuration structure. In this case four light sources are added, which are aimed to be located at edges of exported voxel data. Light sources added to PovRay scene are point-like[17] and they emit white light.

[17] More information about PovRay light sources available at: `http://www.povray.org/documentation/view/3.6.1/308/`.

Finally line 47 creates the PovRay scene file (`.pov`) specified by scene configuration structure `pov_configuration` with objects to include described in `mesh_list`.

Exported scene is located in directory of the code execution. Exported `.pov` file is shown in figure 19 and rendered scene is in figure 20.

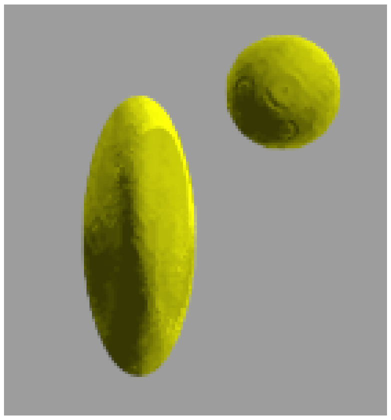


Figure 20: Rendered scene

Similarly to all other files included in the ExportVoxelData package, `N_PovCreateScene` function contains an associated help file, which can be accessed using `help N_PovCreateScene` command in Matlab.

Colouring objects using a scalar array as a key

In most cases it will be desirable to colour exported objects by some property. It can be some physical quantity such as packing fraction, number of contacts or any other scalar array, which can be put into an ordered sequence. The package contains support for that and it is illustrated with Example 22 in the `Manual_Examples`.

Code from Example 21 performing PovRay scene creation is largely unchanged with exception of lines 15 to 20, which are exchanged to:

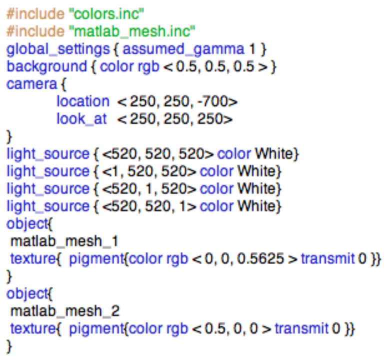
```
1  % Property used for coloring
2  color_criteria = [1, 2]; % Just two different colours
3
4  % Create mesh_list of objects to export with coloring as ...
       a function of a given parameter
5
6  % Parameters of the mesh_list
7  transmit          = 0; % Scalar affecting transparency (0 ...
       opaque, 1 transparent)
8  texture_name      = ''; % No texture is being used
9  number_of_bins    = 2; % Number of bins used for grouping ...
       of colour_criteria, in this case it is superficial
10 color_compression = 0; % compression of color histogram ...
       (0 no, 1 maximal(binary))
11
12 % Obtain a structure with color and transparency ...
       properties of exporte objects
13 mesh_list = N_PovCreateMeshList(mesh_name, ...
       color_criteria, color_compression, number_of_bins, ...
       transmit, texture_name);
```

Line 2 specifies the array with scalar values used for ordering. In this example property used for colouring is just the object identifier.

Lines 6 to 10 define properties of colouring process and exported mesh as indicated by comments included in the code. In this example `number_of_bins` value of 2 is superficial as only two objects exist. `color_compression` permits exclusion of outliers in the key used for ordering.

Line 13 is execution of a function, which performs creation of the `mesh_list` in the same format as in Example 21. Created scene is shown in figure 21 and rendered image is in figure 22.

Script uses `jet`[18] colormap included by default in Matlab, to generate RGB values. Similarly to `N_PovCreateScene`, `N_PovCreateMeshList` also includes documentation, which in depth describes the `mesh_list` structure. This help file is accessible with `help N_PovCreateMeshList` command in Matlab.



```
#include "colors.inc"
#include "matlab_mesh.inc"
global_settings { assumed_gamma 1 }
background { color rgb < 0.5, 0.5, 0.5 > }
camera {
        location  < 250, 250, -700>
        look_at   < 250, 250, 250>
}
light_source { <520, 520, 520> color White}
light_source { <1, 520, 520> color White}
light_source { <520, 1, 520> color White}
light_source { <520, 520, 1> color White}
object{
 matlab_mesh_1
 texture{ pigment{color rgb < 0, 0, 0.5625 > transmit 0 }}
}
object{
 matlab_mesh_2
 texture{ pigment{color rgb < 0.5, 0, 0 > transmit 0 }}
}
```

Figure 21: Exported PovRay file with colouring of objects. *Note: Objects are being referred to using their individual ids.*

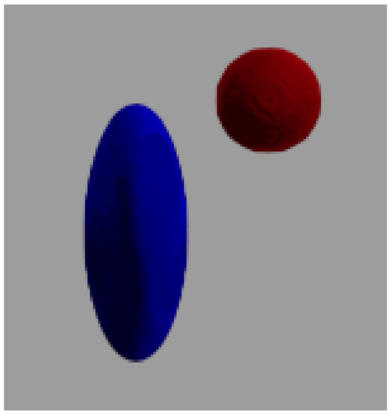


Figure 22: Exported objects are coloured with consecutive integers used as a key. Colormap decreases in colour warmth with increasing key, i.e. red corresponds to lower values.

[18] More information about Matlab's colormaps available at: http://www.mathworks.com/help/matlab/ref/colormap.html.

Exporting voxel data to Blender

Format of STL *files*

A detailed description of STL file format is accessible online[19] and exported STL files follow it. Script responsible for exporting of the STL files, `N_StlAddMesh`, supports both binary and ascii file structure. In order to reduce size of exported STL files, binary format is used in the package, but if necessary it can be changed in `N_3DAddVoxelData.m` file.

[19] More information is available at: `http://www.ennex.com/~fabbers/StL.asp`.

It is important to point out a key difference between exported mesh in PovRay and STL format. While PovRay file structure permits export of distinct objects to a single `.inc` file, which can then be referred to individually, STL format allows only to refer to all exported objects as a whole, i.e. it treats them as a single object. An example, where referring to individual objects in Blender is possible will be given in "Exporting voxel data as individual coloured objects" section, but essentially it requires creation of a separate STL file for each exported object.

Exporting voxel data as a single object

In essence this is already performed by ExportVoxelData package, without any additional coding, as the exported STL file can be readily imported to any 3D visualisation software. For the purpose of this manual Blender will be used due to its open-source nature, ease of implementing add-ons and large online community.

Consider Example 23 in the `Manual_Examples.m` file. A simple STL export is given by following code:

```
%% Exporting Voxel Data to Blender — No colouring — ...
    Example 23

ExportVoxelData(BW_NI, 'pov', false, 'img_dim', size(BW_NI));
```

Note that `img_dim` option is used to make geometric centre of the exported image coincide with global coordinate system origin $(0, 0, 0)$ in Blender. It is not strictly necessary as imported object can be shifted in Blender.

Resulting STL file can be imported to Blender using provided File → Import → Stl (.stl) function. Imported object is usually too big by default and has to be rescaled[20]. This is because exported STL file describes the mesh in Matlab units, whereas the default Blender unit is smaller. Alternatively camera can be zoomed out. Such plain scene can be rendered and is shown in figure 23.

Following the import further modelling, shading etc. can be performed.

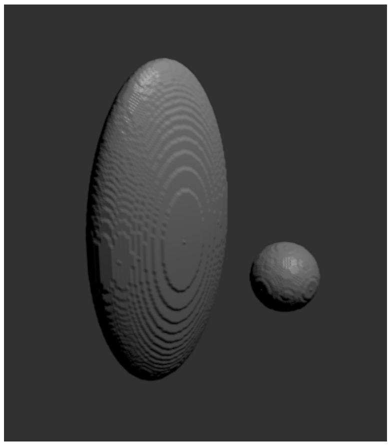


Figure 23: Rendered scene just after importing the STL file and removing the default "Cube" object included in every new Blender scene.

[20] Most easily achieved by hitting "S" key and typing 0.02 just after import is finished

Exporting voxel data as individual coloured objects

Method presented in previous section although simple and straightforward to be done, might not be the most useful way of exporting voxel data in most cases. This is because exported objects cannot be separated in Blender and also it is desirable to output, alongside of object shape, information about its colour or transparency that can be used to represent some physical property.

To accommodate these conditions further features were added to the ExportVoxelData package and an add-on to Blender was written. It permits batch import of STL files and sets their diffuse colour and transparency[21] as specified in a configuration file.

[21] More information about Blender material properties available at: `http://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro/Materials_and_Textures`.

An Example 24 in `Manual_Examples.m` illustrates those features. Its code listing is included below:

```
%% Exporting Voxel Data to Blender - Separate files with ...
    coloring - Example 24

% Export voxel data as separate .stl files only
for i = 1 : 2 % Loop over both objects
    ExportVoxelData(LM_I,...
                    'mesh_name',strcat('Blender_', ...
                        num2str(i)),...
                    'pov',false,...
                    'object_ids', i,...
                    'img_dim',size(LM_I));
end

clear i

% Define parmeters of the export
file_name = 'matlab_export';
mesh_name = 'Blender'; % Note lack of _
dim       = size(LM_I);

```

```
19 % Property used for coloring
20 color_criteria = [1, 2];
21
22 % Create mesh_list of objects to export with coloring as ...
       a function of a given parameter
23 % Parameters of the mesh_list
24 alpha              = 0.4; % Transparency affecting factor
25 number_of_bins     = 2; % Number of bins used for grouping ...
       of colour_criteria
26 color_compression = 0; % compression of color histogram ...
       (0 no, 1 maximal(binary))
27 mesh_list = N_StlCreateMeshList(mesh_name, ...
       color_criteria, color_compression, number_of_bins, ...
       alpha);
28
29 clear color_criteria alpha number_of_bins ...
       color_compression mesh_name
30
31 % Scaling Settings (Empirical Value)
32 blender_max_coord = 3; % Assuming equal aspect ratio
33 scaling_factor = repmat(blender_max_coord / (dim(1)/2), ...
       1, 3); % Assume equal scaling
34
35 clear blender_max_coord dim
36
37 % Create the blender_configuration structure
38 blender_configuration.file_name      = file_name;
39 blender_configuration.scaling_factor = scaling_factor;
40
41 clear file_name scaling_factor
42
43 % Export the properties file
44 N_StlExportToBlender(blender_configuration, mesh_list);
45
46 clear blender_configuration mesh_list
```

Let's discuss the code in pieces. Lines 4 to 10 correspond to exporting every object from the voxel data file into a separate STL file. As far as I am aware, this is the easiest solution to the mentioned limitation of STL file format. It is worth pointing out that if larger number of STL files is being exported, to avoid cluttering of the package execution directory, `output_dir` option could be used. Also please note how file name is being generated sequentially. In principle this could be implemented within the `ExportVoxelData.m` function, but it would require significantly more code lines as compared to the simple `for` loop.

Lines 14 to 30 are very similar to the PovRay Example 22. The only different is in nomenclature. `transmit` is exchanged with `alpha`, but they both are responsible for the same property, i.e. object's transparency in the rendered image.

Lines 31 to 33 determine in a simple way (nearly empirical) scaling of exported objects, so that they will fit Blender's default render region.

Lines 38 and 39 create a `blender_configuration` structure analogous in its role to the `pov_configuration` structure from Example 21 and Example 22.

Finally line 44 creates the `.txt` BLENDER configuration file, which can be used with the provided add-on to import exported objects into BLENDER.

Process of using the batch import STL add-on is identical to using the default import STL function in BLENDER. Following one-time installation of the add-on[22], just use the option menu entry located in BLENDER at FILE → IMPORT → BATCH IMPORT STL (.TXT) and select, in the dialog window that opens, the configuration file. It is necessary to ensure that the configuration file is located in the same directory as exported STL files. After clicking the `"Batch import STL"` button, importing process will start and, depending on number and size of exported STL files, it might take a while until the BLENDER 3D view will appear with imported STL files listed in the `Outliner window`. Following successful import, the scene can be rendered to produce results shown in figure 24.

[22] More information on installing BLENDER add-ons available at: `http://wiki.blender.org/index.php/Doc:2.6/Manual/Extensions/Python/Add-Ons`.

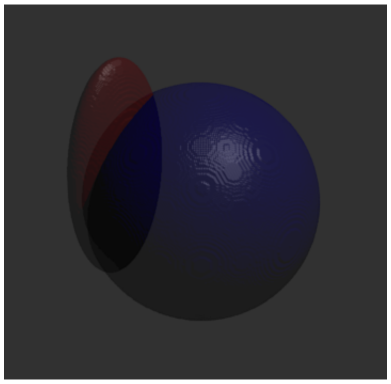


Figure 24: In the rendered scene objects are coloured and partially transparent as specified in the configuration file.

Structure of the configuration file is described in depth in the `N_StlExportToBlender` function and, as usual, is accessible using `help N_StlExportToBlender` command.

Rendering stereoscopic images

VOXEL DATA, STL files and BLENDER scenes all contain 3D information, i.e. objects are represented as three-dimensional structures. When an image or animation is rendered all that information is being lost and two-dimensional structures are being outputted. In principle some of this information can be preserved if stereoscopic output is being used.

BLENDER, being an open-source program with extensive online community, allows to make the process of rendering stereoscopic images relatively straightforward. A stereoscopic camera add-on written by Schneider S.[23] permits an easy creation of such images. For convenience it is included in the `Blender Addons` folder together with the batch import STL add-on.

[23] More information about the package:`http://www.noeol.de/s3d/`.

Following section tries to give a short example of the add-on usage. For more detailed examples please refer to the add-on author's website.

Let's try to output stereoscopic images using the BLENDER scene created in Example 24. If the stereoscopic camera add-on was correctly installed, it should have added its interface to the `Camera` panel in the `Properties` window of camera objects. By following steps showed in author's Youtube video[24] and applying them to the cam-

[24] Video at a time of writing this manual is available at: `http://www.youtube.com/watch?v=usWXat4pt1M`.

era in our scene, it is straightforward to obtain a stereoscopic effect. Rendered stereoscopic images are not included in the manual file as the 3D effect would not be visible on a standard display. Instead a screenshot of Blender workspace used to render them is shown in figure 25.

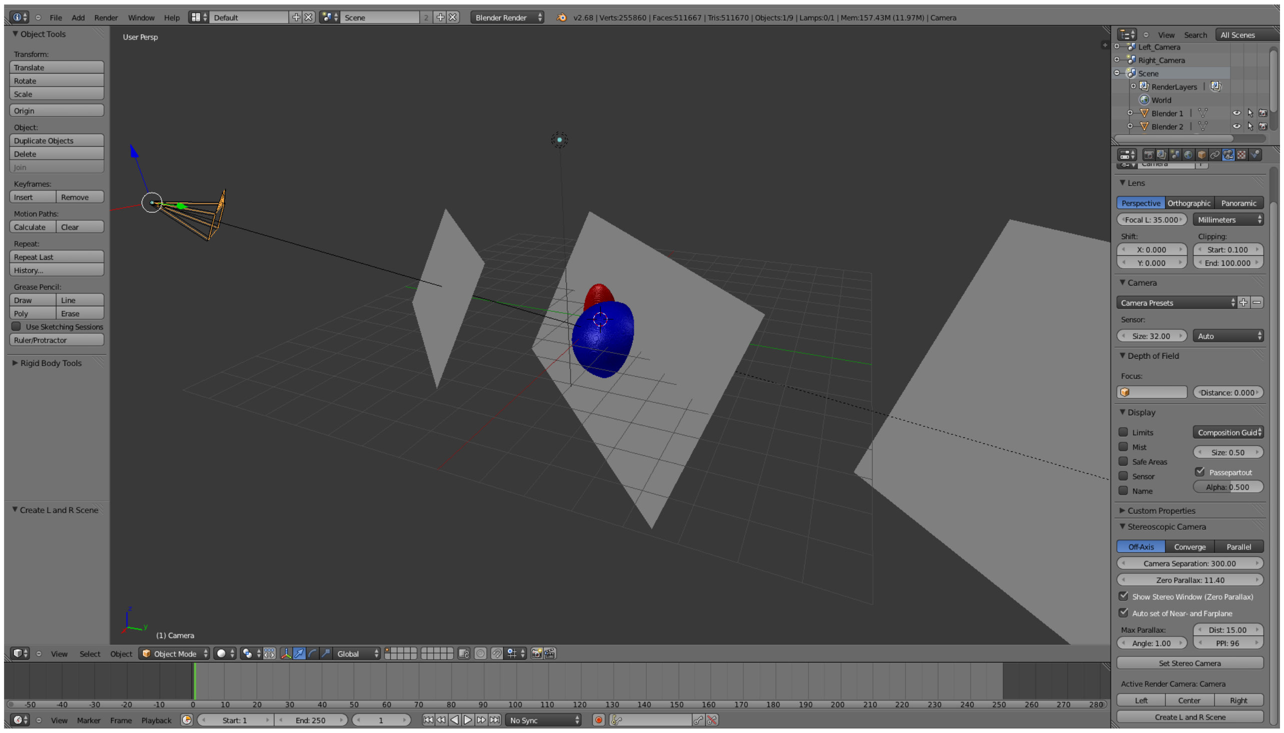


Figure 25: Blender workspace with objects imported using the batch import STL add-on and stereoscopic camera add-on active. *Note: Grey planes are added by the stereoscopic camera package and help to adjust stereoscopic cameras for the best 3D effect.*

The add-on allows to vary several parameters of its setup (for example location of plane of zero-parallax, camera separation or even number of pixels per inch on the final display) and those parameters should be changed in order to produce the best possible visual effect. Furthermore the stereoscopic add-on allows to export images in several formats such as: side by side, above under or red cyan analglyph.

Further examples

Those two objects used in this manual might not be typical voxel data one wants to export mesh of. For this reason there are also two additional example scripts, `Example_ExportVoronoiCellsPovRay.m` and `Example_ExportVoronoiCellsBlender.m`. In order to be able to run those two scripts, a `.mat` file with spaghetti data has to be downloaded[25]. This is because both examples use real voxel data of a spaghetti packing and it would be inconvenient to include a large data file (310 MB) by default with the ExportVoxelData package.

[25] For the time being, file is accessible at my Google Drive: `https://docs.google.com/file/d/0BybuboAGRbuvRzFPa3FzTEdiYzQ/edit?usp=sharing`.

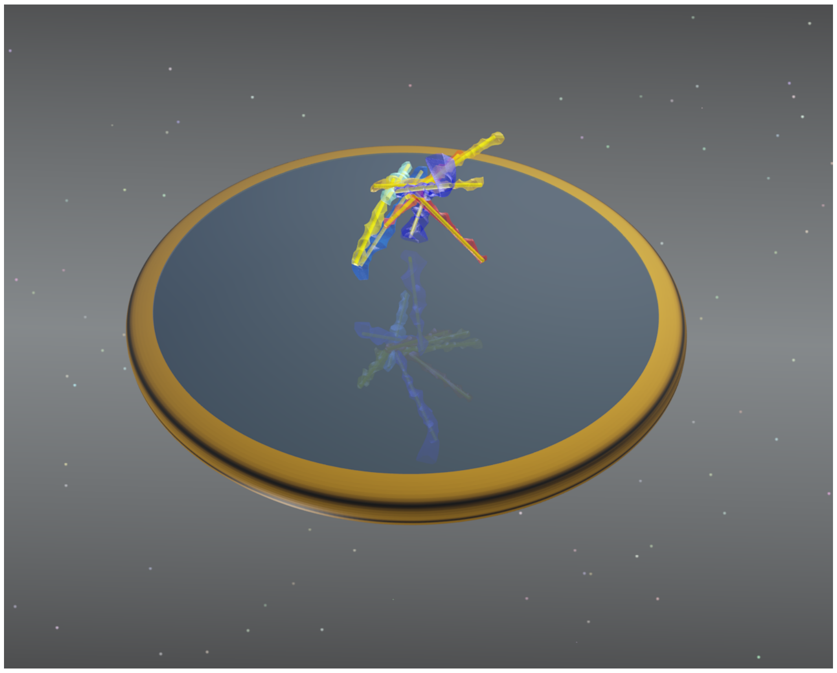


Figure 26: A 2D frame from a sample 3D animation available at: `https://docs.google.com/file/d/0BybuboAGRbuvcEgtUzI2bmVIRUk/edit?usp=sharing`.

Examples were written as extensions of Example 22 and Example 24. They are aimed to illustrate features of the ExportVoxelData package. In both examples it is shown how to combine `mesh_list` for objects that are being colour coded (voronoi cells in this case) with objects of the same colour (spaghetti particles).

Furthermore there also is a Blender file available online[26]. It was created from output of `Example_ExportVoronoiCellsBlender.m` and it renders a working stereoscopic animation using the described add-on by Schneider S. in "Rendering stereoscopic images" section. It is optimised for a 3DTV setup, hence it might require some adjustments to reproduce similar effect on other equipment. A simple 2D render of its content is shown in figure 26.

[26] Again for the time being available at my Google Drive: `https://docs.google.com/file/d/0BybuboAGRbuvVVBHV1FhZ2kyYzQ/edit?usp=sharing`.

Conclusion

Thank you for reading this manual until the very end. I hope it was useful and that I managed to explain some of the features that the ExportVoxelData package provides.

This package was my first computing project of this size, hence I wanted to apologise in advance for any errors that might be present in the code or the manual itself. I would appreciate any suggestions, comments or possible further package improvements. Finally if anything is unclear or further clarification is required, please e-mail me[27] and I will do my best to provide an answer.

[27] My e-mail address: `cyprian.lewandowski@gmail.com` .