Implementation of pore microstructure model generator and pore space analysis tools

Roman Putanowicz

* Cracow University of Technology, Warszawska 24, 31-155 Kraków, Poland

Abstract

Mesoscale modelling of porous materials may require building explicit geometric models for porous microstructure. Such microstructure models are composed of connected voids (pores) embedded in an otherwise impermeable solid. Having such a model at hand it is possible to directly calculate several characteristics of the pore microstructure, for instance pore radii, pore lengths, pore wall surface area as well as spatial distribution statistics. Additionally, having a solver capable of simulating fluid flow through such model, it is possible to gain insight into the effects of pore microstructure characteristics on flow properties, both macroscopic, such as permeability, or microscopic, such as tortuosity. In their recent paper Hyman and Winter [1] discuss an algorithm for stochastic generation of explicit pore structures by thresholding Gaussian random fields. Unfortunately their paper is missing an discussion on effective implementation of the presented algorithm. In our paper we fill this void by providing detailed discussion on implementation side of the algorithm discussed by Hyman and Winter. Besides presenting a microstructure generator which we provide as Open Source program, this paper shows how generated model can exported to data formats suitable for analysis and visualisation with various tools, for instance data formats based on VTK library. The programs described in this paper can form a base for handy tools and reduce the programming effort of building simulation environment for materials microstructure analysis.

Keywords: microstructure generation; voxel models; pore space; programming

* Corresponding author. Tel.: +48-12-628-2567; fax: +48-12-628-2034.
E-mail address: R.Putanowicz@L5.pk.edu.pl
1. Introduction

Mesoscale modelling of porous materials may require building explicit geometric models for porous microstructure. Such microstructure models are composed of connected voids (pores) embedded in an otherwise impermeable solid. Having such a model at hand it is possible to directly calculate several characteristics of the pore microstructure, for instance pore radii, pore lengths, pore wall surface area as well as spatial distribution statistics. From the point of view of modelling microstructure geometry a commonly used approach is to use voxel based models also referred as multidimensional images. Figure 1 illustrates the concept of three dimensional voxel and shows microstructure consisting of several grains modeled as groups of voxels tagged with colours.

![Fig. 1. (a) voxel concept; (b) voxel based microstructure model.](image)

Besides microstructure visualisation, having a solver capable of simulating fluid flow through such model, it is possible to gain insight into the effects of pore microstructure characteristics on flow properties, both macroscopic, such as permeability, or microscopic, such as tortuosity. In their recent paper Hyman and Winter [1] discuss an algorithm for stochastic generation of explicit pore structures by thresholding Gaussian random fields. Unfortunately their paper is lacking an discussion on effective implementation of the presented algorithm. Despite many software tools, building own microstructure generator may be of vital importance. It allows not only to gain deeper insight into algorithms presented in literature but also gives a chance to introduce custom solutions not available in the ready tools.

In this paper we describe implementation of Hyman and Winter algorithm for explicit pore structure generation, with the intention to make it easier to build custom tools by other researcher. Full source code distributed on Open Source license is available at https://femdk.l5.pk.edu.pl/FORM/wiki/MicroGen.

2. Stochastic generation of explicit pore structures

As indicated in the previous section analysis of many phenomena on meso and microlevel may involve generation of geometric model for microstructure. Voxel based models are of particular interest due to their direct link with experimental method of gathering data for volumetric images. There are several ways to generate explicit microstructure models, for instance methods based on overlapping spheres, Voronoi tessellations, random packing of simple solids, or methods mimicking physical processes. Among many possible methods, the stochastic ones are characterised by relative simplicity and inherent parallel flavour. In this paper we use method based on thresholding of generated Gaussian random field as described in Hyman and Winter [1]. The three main steps of the algorithm are as follows:

1. Populate $S \subseteq \mathbb{R}^n$ with realization $\xi(x)$ of a random field whose elements are independent, identically distributed random variables sampled from the continuous uniform distribution on the interval $[a, b]$. 
2. Generate a correlated random topography, $T = k \ast u$ by convolving $u$ with a deterministic kernel, $k$. It is convenient if $\int_{\mathbb{R}^n} k(x) dx = 1$.

3. Produce a pore space realization, $\mathbb{P}$, by thresholding given $T(x)$ with a level threshold $\gamma \in [a, b] : \mathbb{P} = \{ x \in \mathbb{S} | T(x) < \gamma \}$. The resulting pore space is equivalent to the indicator function

$$\chi(x) = \begin{cases} 1, & \text{if } T(x) < \gamma \\ 0, & \text{if } T(x) \geq \gamma \end{cases}$$

Being simple this method is also very flexible because one can design kernel functions to handle specific cases like anisotropy or multiple pore structures. Figure 2 shows an example of two very different pore structures obtained with Gaussian and uniform kernel functions.

![Image](image_url)

**Fig. 2.** Examples of various types of microstructures generated with different kernels (a) Gaussian kernel; (b) pores for Gaussian kernel; (c) uniform kernel; (d) pores for uniform kernel.

### 3. Convolution and Fourier transform

The key element in the algorithm presented in the previous section is the ability to calculate convolution of kernel function and random fields in three dimensional space. As it turns out, the natural basis for effective implementation is the relation between convolution and Fourier transform, and the possibility of efficient calculation of Fourier transform by Discrete Fourier Transform algorithm. For completeness we briefly outline these mathematical concepts, with more details to be found in Hirshchman.and Widder [2], SSE [3].

The convolution of two functions $g(t)$ and $f(t)$ is the function:

$$h(t) = \int_{-\infty}^{\infty} g(t-x) f(x) dx$$  \hspace{1cm} (1)

The following notation is commonly used:
\[(g * f)(t) = \int_{-\infty}^{\infty} g(t - x)f(x)dx\] (2)

This is not hard to show that convolution is commutative, associative, distributive, associative with scalar multiplication. Somehow most important in the context of this paper is the convolution theorem, which states that

\[F(g * f)(s) = Fg(s) \cdot Ff(s)\] (3)

where \(F\) stands for Fourier transform. Versions of this theorem also hold for Laplace transform. Fourier transform in turn can be defined as:

\[\hat{f}(s) = Ff(s) = \int_{-\infty}^{\infty} e^{-2\pi i s t} f(t)dt\] (4)

As elaborated in Hyman and Winter [1], the convolution theorem allows for an efficient evaluation of the convolution kernel \(k(x)\) with realisation of the random field \(u(x)\). The convolution is achieved by transforming a realization of the random field into Fourier space using the discrete Fourier transform, \(F[\cdot]\), point-wise multiplication by the discrete Fourier transform of the kernel, and then applying the inverse transform to the product, i.e.

\[T(x) = F^{-1}[F[k(x)] \circ F[u(x)]]\] (5)

where \(\circ\) denotes point-wise multiplication. The discussion presented in Hyman and Winter [1] solves the mathematical side of the problem of explicit generation of pore structure. The rest of this paper is focused on the implementation aspects, which are important if we aim at robust and effective tools, especially for medium to large microstructure models.

4. Software libraries for volumetric images analysis

Looking at the description above we could guess that the backbone of an implementation of the outlined method is a data structure to represent voxel based geometric model. While it is possible to use multidimensional arrays presented in many languages to implement such structure we must remember that processing efficiency or memory restriction may dictate the use of more sophisticated implementation, for instance tree based. Also, in order not to reinvent the wheel, it is wise to use several general purpose libraries for image manipulation, as one needs to handle the following issues: basic data structure, enumeration of voxelized data, serialization of voxel model and input output facilities, numerical algorithms and visualization. Detailed discussion of available software solutions goes beyond the scope of this paper so we just briefly mention the main component used in our implementation:

- VTK library (Shroeder et al. [4], Shroeder et al. [5], The visualization toolkit [6] and ParaView (Ahrens[7]) visualisation environment. VTK is very rich computer graphics and visualization library that itself could be used as the main component, but at this stage we use it mainly for I/O facilities and data exchange between other tools, for instance to conveniently visualize generated microstructure with ParaView, as illustrated by figures in this paper.
- VoxelCAD (Voxcad [8]) modeler illustrated in Figure 3 provides facilities to inspect and modify models voxel by voxel. This turns to be useful in preparing special data sets for verification of our generator or for adjusting generated microstructures.
- Vigra (Köthe [9], Köthe [10], Köthe [11]) constitutes the heart of our implementation. Is an image processing and analysis library. "Vigra is especially strong for multi-dimensional images, because many algorithms (e.g.
filters, feature computation, superpixels) are implemented for arbitrary high dimensions.” (see Köthe [11]). The main features are:
  o implemented in C++ with Python and Matlab bindings,
  o MIT style license,
  o well documented,
  o several modules: images and multidimensional arrays, STL-like image processing algorithms, filters, segmentation, image analysis, machine learning, mathematical tools, inter-language support.

Fig. 3. VoxCAD interface window.

5. Pore structure generator

Micorgen, our pore structure generator, is implemented in C++ as command line tool. For flexibility and future extension all data describing generated microstructure are read from a file in JSON format. Sample data file is shown in Listing 1.

Listing 1. Microgen data file in JSON format

```
{
  "gamma" : 0.550,
  "micro-size" : [ 50, 50, 50 ],
  "kernel" : {
    "type" : "isotropic-gaussian",
    "lambda" : 0.01
  }
}
```

In the above file gamma stands for threshold level $\gamma$ from Eq. 1, micro-size holds number of voxels in each direction and lambda is the parameter of length scale appearing in Gaussian kernel described below.

The main element of Micorgen is the class MicroGenerator. The purpose of this class is to hold all data related to the generation process. The key data items are:

- three 3D images implemented as three-dimensional arrays of the size given by micro-size parameter from Listing 1. The arrays are:
  o kernelImage – an array holding values of the kernel function $k(x)$,
- randTopoImage – an array holding values of the random field $u(x)$,
- outImage – an array holding the resulting pore space indicator function $\chi(x)$ given by Eq. 1.

- parameters of Cartesian grid that is mapped into 3D image. The parameters are:
  - myDeltas[3] – grid cell size in directions x, y and z, respectively,
  - myOrigin[3] – the anchor of the grid in space – the coordinates of the grid center, usually (0,0,0),
  - myDimensions[3] – the dimensions of the grid in each direction.
- $a, b$ – the limit of the interval from which the values of the random field are sampled, $0 \leq a, b \leq 1$.

The main methods of this class include:

- Generate – the main driver implementing the algorithm described in Section 2,
- GenerateRandomTopoImage(void) – method to generate random field $u(x)$ by calling one of random number generators available in Vigra library.
- GenerateKernelImage(void) – method to generate kernel image, depending on user choice the generation is dispatched to specialised methods like CalculateIsotropicGaussianKernel(const double lambda)
- WriteVTK – method to save generated 3D image in VTK file.

Below we give some code fragments illustrating the kernel part of the generator. The use of Vigra library is the key point as it enables very short, clear implementation. The method Generate shown in Listing 2 corresponds to the algorithm outline presented in Section 2.

**Listing 2. Method MicroGenerator::Generate – top level algorithm driver**

```cpp
void
MicroGenerator::Generate( void ){
    GenerateRandomTopoImage ( ) ;
    GenerateKernel Image ( ) ;
    double gamma = femdk::getDouble ( myInputConfig, "gamma", 0.5 );
    vigra::convolveFFT ( randTopoImage, kernel Image, outImage );
    vigra::transformMultiArray ( out Image , out Image ,
                                vigra::Threshold <double ,double >( 0.0, gamma, 1.0, 0.0 ));
}
```

For generation of random field we use one of the several random number generators provided by Vigra library. Listing 3 shows method that fills the randTopoImage data structure holding random values at voxel centers.

**Listing 3. Method MicroGenerator::GenerateRandomTopoImage for random field**

```cpp
void
MicroGenerator::GenerateRandomTopoImage( void ){
    // Code for generating random field...
}
```
void
MicroGenerator::GenerateRandomTopoImage ( void ){
  vigra::RandomTT800 randomKernel;
  vigra::UniformRandomFunctor<vigra::RandomTT800>
    getrand( 0.0, 1.0 , randomKernel );
  typedef vigra::MultiArray<3, double>::iterator Iter;
  for ( Iter i = randTopoImage.begin();
    i != randTopoImage.end(); ++i ) {
    *i = getrand();
  }
}

The n-dimensional Gaussian kernel is given by:

$$k(x) = \frac{1}{(2\pi)^{n/2}} \exp(-x\Lambda x/2)$$

(6)

where

$$\Lambda = \begin{bmatrix}
\lambda_x & 0 & 0 \\
0 & \lambda_y & 0 \\
0 & 0 & \lambda_z
\end{bmatrix}$$

(7)

is the matrix of the microstructure length scales. The values $\lambda_i$ describe the spread of the kernel in coordinate directions. For isotropic case the input the length scales are set to the same value $\lambda_x = \lambda_y = \lambda_z$.

Method CalculateGaussianKernel in Listing 4 implements Gaussian kernel, already with possibility to generate anisotropic pore structures.

Listing 4. Method MicroGenerator::CalculateGaussianKernel

void
MicroGenerator::CalculateGaussianKernel ( const Matrix3 &Lambda ) {
  const int n = 3 ;
  const double f1 = std::pow(2 * M_PI, n / 2.0 ) ;
  const double det = vigra::linalg::determinant ( Lambda ) ;
  const double factor = 1.0 / ( f1 * sqrt ( det ) ) ;
  const Matrix3 LambdaInv = vigra::linalg::inverse ( Lambda ) ;
  Vector3 x;
  for ( int i = 0; i < myShape[0]; i++){
    x[0] = i - myShape[0]/2.0;
    for ( int j = 0; j < myShape[1]; j++){
      x[1] = j - myShape[1]/2.0;
      for ( int k = 0; k < myShape[2]; k++){
        x[2] = k - myShape[2]/2.0;
        double xe = -VecMatVec3 ( x, LambdaInv, x )/2.0;
        kernelImage( i, j, k ) = factor * exp ( xe ) ;
      }
    }
  }
}

The use of Vigra library allows for very concise implementation that can be extended in several directions. Vigra offers several utilities for processing multidimensional images. A simple example is the function to analyse
voxels connectivity pattern. Thanks to this function it is easy to detect "islands", that is groups of material voxels not connected to the rest microstructure material. Figure 5 illustrates such analysis. Removal of floating material particles in the generated microstructure is done in the postprocessing step to make the model more realistic.

Fig. 5. Detection of ’islands’ that are not connected bulk material region (a) identification of connected components; (b) only islands.

6. Conclusions

In the paper we have discussed some implementation aspects of the microstructure generator based on the algorithm from Hyman and Winter [1]. We have shown that the selection of adequate software components, for this case the Vigra library, results in efficient and flexible implementation. Our program Microgen is available as Open Source project and can be a starting point for development of customised tools that require generation and handling of explicit realisation of material microstructure.

Acknowledgements

This work was financially supported by Polish Ministry of Science and Higher Education within the contract no. L5/44/2015/DS.

References

[1] Hyman J D, Winter C L. Stochastic generation of explicit pore structures by thresholding gaussian random fields, *J Comput Phys* 2014; 277:16–31.
[2] Hirshchman I I, Widder D V. The convolution transform. Princeton mathematical series, Princeton University Press. 1955
[3] SSE, Stanford Engineering Everywhere. URL: http://see.stanford.edu/see/courses.aspx, accessed: 15.10.2014.
[4] Schroeder W J, Avila L S, Hoffman W. Visualizing with vtk: A tutorial, *IEEE Comput. Graph. Appl* 2000; 20:20–27.
[5] Schroeder W, Martin K, Lorensen B. The Visualization Toolkit–An Object-Oriented Approach To 3D Graphics, fourth ed., Kitware, Inc.; 2006
[6] The visualization toolkit. URL: http://www.vtk.org/, accessed: 05.04.2015.
[7] Ahrens J, Geveci B, Law C. Paraview: An end user tool for large data visualization. 2005.
[8] Voxcad. URL: http://www.govcad.com, accessed: 05.04.2015.
[9] Köthe U. Reusable Software in Computer Vision. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA; 1999 pp. 103–132. URL: http://portal.acm.org/citation.cfm?id=553575.
[10] Köthe U. STL-Style Generic Programming with Images, in: *C++ Report Magazine*; 2000; 12:1:24–30. URL: http://kogs-www.informatik.uni-hamburg.de/~koethe/papers/abstracts/GenericProg2DC++Report.html.
[11] Köthe U. Vigra homepage, URL: http://ukoethe.github.io/vigra/, accessed: 15.10.2014.