Procedural Voronoi Foams for Additive Manufacturing — Supplemental

Jonàs Martínez
INRIA

Jérémie Dumas
Université de Lorraine, INRIA

Sylvain Lefebvre
INRIA, Université de Lorraine

In Section 1 we present the formula of the elasticity tensor for isotropic materials. In Section 2 we present a short background on numerical homogenization. Finally, in Section 3 we give the iterative algorithm to generate seeds, and describe the Python code attached in the supplemental material.

We use the following notation:

Notation	Description
E	Young's modulus
v	Poisson's ratio
ϵ	Strain tensor
σ	Stress tensor
C	Elasticity tensor
u	Displacement field
f	Force field
K	Stiffness matrix

1 Isotropic Material Tensor

$$\hat{E} = \frac{E}{(1-2v)(1+v)}, \qquad G = \frac{E}{2(1+v)}$$

$$C^I(E,v) = \begin{pmatrix} \hat{E}(1-v) & \hat{E}v & \hat{E}v & 0 & 0 & 0 \\ \hat{E}v & \hat{E}(1-v) & \hat{E}v & 0 & 0 & 0 \\ \hat{E}v & \hat{E}v & \hat{E}(1-v) & 0 & 0 & 0 \\ 0 & 0 & 0 & G & 0 & 0 \\ 0 & 0 & 0 & 0 & G & 0 \\ 0 & 0 & 0 & 0 & 0 & G \end{pmatrix}$$

2 Background on Homogenization

Homogenization is at the core of most existing work regarding microstructures [Allaire 2012]. We rely on homogenization to predict the large scale behavior of our structures. We therefore give some more precise background regarding this technique.

Homogenization efficiently determines the elasticity tensor of a periodic composite material defined from a unit tile V, having a volume $|V|$. For small deformations of an elastic material, the amount of stress σ is linearly proportional to the strain ϵ, as given by the elasticity tensor C:

$$\sigma = C\epsilon$$

The homogenized elasticity tensor C^H can be derived as [Sanchez-Palencia 1980]:

$$C^H_{rspq} = \frac{1}{|V|}\int_V C_{ijkl}\left(\epsilon^{0(ij)}_{pq} - \epsilon^{ij}_{pq}\right)\left(\epsilon^{0(kl)}_{rs} - \epsilon^{kl}_{rs}\right) dV$$

$\epsilon^{0(ij)}_{pq}$ are prescribed strain fields. $\epsilon^{(ij)}_{pq}$ is obtained as $\epsilon_{pq}(u^{ij})$, where the displacement field u^{ij} is the result of solving the elasticity equations with prescribed strain.

For most problems, homogenization is performed numerically by discretization, solving the elasticity equation with the finite element method (FEM). Our method is an extension to 3D of the 2D implementation of [Andreassen and Andreasen 2014]. The elasticity equation is discretized on a regular grid with N hexahedral elements. Consider six forces f_a with prescribed unit strains (the six different strain coordinate directions in 3D). u^0_e are the six displacement fields corresponding to unit strains, and u_e are the displacement fields resulting from enforcing the corresponding unit strains. The six displacement fields are obtained by solving for linear elasticity $Ku_a = f_a$, with imposed periodic boundary conditions. When the displacements have been obtained, the homogenized elasticity tensor can be found as [Andreassen and Andreasen 2014]:

$$C^H_{ij} = \frac{1}{|V|}\sum_e^N \int_{V_e} \left(u^{0(i)}_e - u^{(i)}_e\right)^T K_e \left(u^{0(j)}_e - u^{(j)}_e\right) dV_e$$

This tensor characterizes the linear elastic behavior of the periodic material.

3 Seed Generation Process

Algorithm 1 shows a stackless version, iterative method for generating seeds in a coarse grid cell.

We also provide two sample Python codes implementing the adaptive sampling method described in our paper, specialized for the 2D case. The files can be found in the `code.zip` archive accompanying this supplemental material, and are organized as follows:

- The first file, `generate_seeds_recursive.py` is a 67 lines python version of our algorithm SubdivideCell, using built-in calls for generating random numbers.
- The second file, `generate_seeds_iterative.py` is a 141 lines stackless version of the same code, with explicit random number generation (using the same generator as the default random number generator of `libstdc++`). This version is amenable to a GPU implementation.

The code works with both `python2` and `python3`, and includes a plot of the result. It depends only on `numpy` and `matplotlib`.

References

Allaire, G. 2012. *Shape optimization by the homogenization method*, vol. 146. Springer Science & Business Media.

Andreassen, E., and Andreasen, C. S. 2014. How to determine composite material properties using numerical homogenization. *Computational Materials Science 83*, 488 – 495.

Sanchez-Palencia, E. 1980. Homogenization in elasticity and electromagnetism. In *Non-Homogeneous Media and Vibration Theory*, vol. 127 of *Lecture Notes in Physics*. Springer Berlin Heidelberg, 84–128.

Algorithm 1: SubdivideCellIterative

Input: Density field ρ, starting coarse cell center cs.
Output: A set of seed, with a density driven by ρ

$ijk \leftarrow cs$;
$d \leftarrow 0$;
$N \leftarrow \emptyset$;
while *true* **do**
  $l \leftarrow coarseCellLength \times 2^{-d}$;
  $c \leftarrow l \times ijk + l/2$; `// center of the current cell`
  $t \leftarrow l^3 \times \rho(c)$; `// target number of seeds in cell`
  if $t \leqslant 2^3$ **then**
    $I = RandomPermutation(subcells)$;
    $n_{min} = \lfloor t \rfloor$; `// minimum number of samples to draw`
    for $i \in [\![0, n_{min}[\![$ **do**
      $N \leftarrow N \cup \{RandomSampleInSubcell(I[i])\}$;
    $p \leftarrow random(0, 1)$;
    if $p \leqslant (t - n_{min})$ **then**
      $N \leftarrow N \cup \{RandomSampleInSubcell(I[n_{min}])\}$;
    `// move up the cells`
    while $ijk \mod 2 = (1, 1, 1) \wedge d > 0$ **do**
      $d \leftarrow d - 1$;
      $ijk \leftarrow \lfloor ijk/2 \rfloor$;
    if $d > 0$ **then**
      `// move to next cell with the same parent`
      $ijk \leftarrow nextOnLevel(ijk)$;
    else
      break;
  else
    `// go to the first child cell`
    $d \leftarrow d + 1$;
    $ijk \leftarrow ijk * 2$;
return N