Contents lists available at ScienceDirect

Powder Technology

journal homepage: www.elsevier.com/locate/powtec

Effective packing of 3-dimensional voxel-based arbitrarily shaped particles

Thomas Byholm, Martti Toivakka, Jan Westerholm *

Abo Akademi University, Laboratory of Paper Coating and Converting and Center for Functional Materials, Porthansgatan 3, FI-20500 Abo, Finland

ARTICLE INFO

Article history:
Received 8 August 2008
Received in revised form 16 January 2009
Accepted 12 July 2009
Available online 26 July 2009

Keywords:
Particle packing
Digital
Voxel
Optimisation
Memory

ABSTRACT

In many research areas including medicine and paper coating, packing of particles together with numerical simulation is used for understanding important material functionalities such as optical and mass transfer properties. Computational packing of particles allows for analysing those problems not possible or difficult to approach experimentally, e.g., the influence of various shapes and size distributions of particles. In this paper a voxel-based algorithm by Jia et al. [X. Jia, R.A. Williams, A packing algorithm for particles of arbitrary shapes, Powder Technology 2001, vol. 120, pp. 175–186.] enabling the packing of arbitrarily shaped particles, is memory- and speed-optimised to allow for simulating significantly larger problems than before. Algorithmic optimisation is carried out using particle shell area reduction decreasing the amount of time spent on collision detection, fast rotation routines including lookup tables, and a bit packing algorithm to utilise memory effectively. Presently several hundreds of thousands of complex arbitrarily shaped particles can be simulated on a desktop machine in a simulation box consisting of more than 10^9 voxels.

1. Introduction

In many different research areas including pharmaceutical science and paper coating there is a clear need to better understand how particles are packed when forming, e.g., a coating. A more detailed knowledge of the packing process would be beneficial in order to understand and optimize the microstructure of different kinds of porous media.

Traditionally analytical methods have been used, packing mathematically defined objects and applying various chemical and physical interactions in order to assess the relative movements and interactions of the objects. In this paper we will focus on computational methods for improving the speed and memory efficiency of a voxel based particle packing approach first presented in [1]. In voxel based representations, solid objects are discretized using elementary volume units, voxels, typically cubes, the 3-dimensional objects analogous to discretized 2-dimensional objects, pixels. The simulations in [1] are based on moving these voxel objects in different directions in space. For clarity, in Fig. 1 we have visualized two pixel objects in 2D-space. Any subsequent translation or rotation of these two objects will retain the approximate shape and size of each object, always filling or not filling a pixel and at most one particle in any given pixel.

Well done code optimisations can have a dramatic effect on the effectiveness of almost any algorithm [2], and hence we are interested in both representing the voxel objects as effectively in computer memory as possible in order to minimize memory usage, and also simulating the movement and interaction of voxel objects as quickly as possible. We will also look at how to optimise low level operations, deeply nested in the algorithm of [1], as these operations are good candidates for optimisation based on their usage frequency. Although many of these operations are fast, their frequency of use will add up to a considerable total time.

Furthermore, since it is often desired to use high resolutions when discretizing objects into voxels, there is a need for considering different kinds of real time data compression for the packing matrix, that is, the 3-dimensional space where the voxels are situated. Since voxel based packings operate on a large 3-dimensional array for representing the packing matrix, memory requirements tend to escalate very quickly. High resolutions are needed for accurately representing a wide particle size distribution as well as extreme aspect ratios. A straightforward way of representing the packing matrix is to use an array containing id-numbers for the respective particles as in Fig. 1. While this enhances speed by not having to erase a particle before moving it, it consumes large amounts of memory. To represent a 4000^3 array with 32-bit elements we would need 256 GB of memory. Furthermore, the spatial locality when accessing voxels in the packing matrix will be very poor and the deepest nested operations of the algorithm consist of intensive reading and writing of data. To overcome this problem, real time bit-packing is applied representing particle positions with 1 and void space with 0. Although a little bit more work need to be done like storing particle metadata separately and always removing them before trying a new position, the increased spatial locality and 32-fold reduced need for memory outweigh the extra steps. These techniques allow for packing of complex sets with arbitrarily shaped particles on normal 64-bit desktop machines. As the amount of computer memory increases steadily future desktops will be able to simulate even higher resolutions and larger

* Corresponding author.
E-mail address: jan.westerholm@abo.fi (J. Westerholm).

0032-5910/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.powtec.2009.07.013

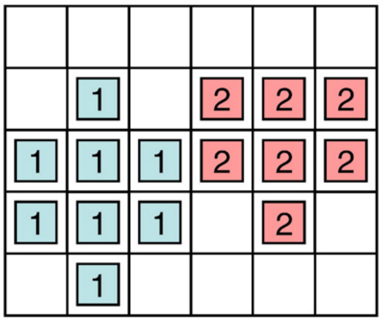

Fig. 1. 2-dimensional illustration of a packing matrix containing two distinct particles. Each particle is described by a set of pixels identified by the id of the particle.

packing matrices. Basic statistic output was added for the packed material, including porosity and rotational distributions of packed particles and RMS surface roughness including a graphical user interface for easy particle definition and batch processing.

We start by introducing the basics of the packing algorithm considered in this paper in Section 2. In Section 3, we consider in detail the different optimisations done to improve the time and memory efficiency of the algorithm. The main results are presented in Section 4 and the conclusions in Section 5.

2. Packing algorithm

The general structure of a particle simulation process can be divided into three logically separated tasks: geometric definition, particle movement rules and collision detection. In the geometric definition we define how particles are represented geometrically and which data structures are used to represent the particles. These auxiliary structures will have a significant impact on memory consumption and the efficiency of particle movements and collision detection. Particle movement rules should capture the essential physics of the packing process while keeping the logic of the rules reasonably simple. Thirdly, the collision detection is a central feature of the program inducing restrictions of particle movements. We will be particularly interested in developing fast collision detection methods by defining the particle geometry in such a way as to support this.

2.1. Particle and voxel data representation

In our particle packing simulations the volume within which particle movements are simulated, the packing matrix, is divided into 3-D unit cubes, voxels. A voxel is the 3D analogue to 2D pixels representing the finest granularity of volume available. Hence e.g. all particles consist of a collection of voxels, and any voxel within the simulation box is either empty or occupied by precisely one particle.

The main idea behind voxel based particle packing is to avoid the time consuming collision detection associated with analytical methods [3], where particles are represented as mathematical objects, a sphere for example being represented by its centre point and radius. Collision detection, or intersection test, is used to determine if a particle due to its movement will be overlapping any other particle, thus enabling us to decide if the particle movement will be allowed or not. In a simple implementation an intersection test has to be done for all voxels of every object in the simulation box. If an intersection with any other object is found, the two are said to collide and the move cannot be accepted. The intersection tests for analytically defined particles are very time consuming and depend largely on the type of object being checked for collision. Although this kind of approach can be optimised in many different ways including neighbour lists [4], octree representations [5] and bounding boxes, it is still very complex for large numbers of particles. The basic complexity is $0(n)$ meaning that n (number of particles) intersection tests have to be performed for every single particle move. Furthermore the complexity increases if we want to pack arbitrarily shaped objects using analytical methods since these are often represented using large numbers of polygons, meaning that the number of intersection tests needed is further increased, depending on the number of polygons used to describe every particle.

In Ref. [1] objects are not represented analytically, instead they are built up from voxels [5]. Every object can be approximated by a set of voxels arranged on a regular grid so that it represents the analytical shape as closely as possible. The desired accuracy can be varied by increasing the resolution, that is decreasing the voxel volume. All the particles simulated are inserted in a large packing matrix. As we are working with voxel data, all coordinates are represented using integers. The packing matrix is a simple 3-dimensional data array containing a particle id or a special void id at every element. When a particle is moved, it is removed from its previous voxel position and written into its new position in the array. The collision detection can then be done by checking all the new positions that the voxels of the particle in question will overlap. If any of these voxels already contains an id belonging to another particle, the move is denied and the particle is reset to its original position. This way we eliminate the analytical intersection tests and we do not need to scan through all particles as the collision detection is based on the packing matrix only.

To enable some of the optimisations that we will deal with in Section 3, all particles are also represented as metadata. For every particle we know the position of its centre point as defined by the user, its current rotation and the original voxel map, so that we can reconstruct it without the simulation box.

2.2. Particle movement

To create a particle packing, particles need to be introduced in one way or another into a simulation box. A simple approach would be to try random (x,y,z) positions until a position is found where the particle does not overlap with any other particle. This procedure is repeated until the desired packing density is reached. The problem with this method is that it is not possible to achieve high packing densities, as the particles are static after they have been added to the packing. This means that large gaps will be formed between particles which are difficult to fill, resulting in low packing densities.

To deal with this problem, various packing algorithms that move the particles according to predefined rules can be implemented, usually involving collision detection to ensure that particles are not moving into the domain of another particle. Since the goal of particle packing is often to make a densely packed structure, we have to find a set of particle movement rules that optimises the particle movement to achieve this goal. We are using the same fast statistical approach as in [1] to achieve this effect. The underlying principle is to introduce one or more particles every nth time step. They can be introduced using many different methods, resulting in slightly different packing structures, but the most straightforward way is to introduce new particles at random (x,y) positions at the top of the simulation box. For every time step, all particles are moved 1 voxel position along any axis (x,y,z) and rotated freely within a preset maximum value, which is set individually for all three axes relative to their initial rotation. The packing property of the algorithm is achieved by assuring that the particle movement is generally downwards according to the following rule. New particle positions are found by assigning a 50% probability to move in the negative directions and 50% probability for moving in the positive directions on the x-, y- or z-axis, allowing moves along every axis simultaneously. The maximum movement is 1 voxel in each direction. To simulate the downwards movement a rebounding probability between 0% and 100% is used, describing how high the probability to accept an upwards movement is. The upwards component of the move is accepted if and only if the rebounding