Warsaw University of Technology

FACULTY OF
POWER AND AERONAUTICAL ENGINEERING

Institute of Heat Engineering

Master's diploma thesis

in the field of study Power Engineering
and specialisation Nuclear Power Engineering

Development of the new method of the pebble bed generation for HTRs

Piotr Klukowski
student record book number 252662

thesis supervisor
Dr inż. Grzegorz Maciej Niewiński

thesis co-advisor
Mgr inż. Piotr Darnowski

Warsaw, 2018

ABSTRACT

High Temperature Pebble Bed Nuclear Reactors (HTR-PB) are very specific because their cores consist of small fuel pebbles, which are fuel and a moderator as one. These pebbles are nearly spherical and are closed in a cylindrical-conical container called a reactor.

One of the problems of a given type of nuclear reactor is the distribution of pebbles inside the reactor. The problem is caused by the fact that although the reactor is self-controlling, we cannot precisely determine the k_{eff} value during a design stage.

Therefore, the primary goal of this work was to solve the problem of random distribution of pebbles in a closed space like a container of a nuclear reactor. The thesis presents one possible method for generating pebble bed in HTR. The model is using vector algebra and mechanics, with a presence of gravity. A pseudo-random method was proposed which simulates physical process of pebble bed formation.

The method was implemented into a computer program developed in MATLAB environment. The source code of the program is described in the thesis. The current version is not as fast to be used for significant power HTRs. However, the new version is redesigned and will be faster about one hundred times, which is enough to create HTR-10 pebble bed in few days.

At the end of the work are shown speed tests and results of generated pebble beds. As an example, was generated HTR-10 pebble bed with twice bigger ball radius, which took about four days.

Keywords: nuclear reactor, HTR, pebble bed, mechanical collision, computer simulation

STRESZCZENIE

Reaktory wysokotemperaturowe ze złożem usypanym (HTR-PB) są bardzo specyficzne, ponieważ ich rdzeń składa się z małych kamyków paliwowych (ang. Fuel pebbles), które są jednocześnie paliwem i moderatorem. Kamyki te są kulistego kształtu, w obudowie cylindryczno-stożkowej.

Głównym problemem reaktora jądrowego danego typu jest rozłożenie kamyków w reaktorze. Dany problem jest spowodowany tym, iż mimo że reaktor jest samo kontrolujący się, podczas projektowania nie możemy precyzyjnie wyznaczyć wartości k_{eff}.

W związku z powyższym głównym celem tej pracy było rozwiązanie za pomocą algebry wektorowej oraz prostej mechaniki zagadnienia dotyczącego losowego rozmieszczenia kul w zamkniętej przestrzeni, z uwzględnieniem grawitacji. Do rozwiązania danego problemu zastosowano metodę pseudolosową, która z odpowiednią dokładnością symuluje rzeczywisty rozkład.

Dodatkowo został napisany skrypt w programie MATLAB, który zawiera implementacje wcześniej wspomnianej metody. Opis kodu został przedstawiony w pracy. Obecna wersja nie jest wystarczająco szybk do uycia dla reaktorów HTR dużej mocy, aczkolwiek obecnie jest tworzona nowa wersja, około stukrotnie szybsza, co wystarczy do generacji złoża reaktora HTR-10 w kilka dni.

Na końcu zostały przeprowadzone testy oraz wygenerowane przykłady złóż usypanych. Jednym z wygenerowanych złóż jest złoże HTR-10 o dwa razy większym promieniu kul, które było generowane przez cztery dni.

Słowa Kluczowe: reaktor jądrowy, HTR, złoże usypane, kolizja mechaniczna, symulacja komputerowa

Statement of the author (authors) of the thesis

Being aware of my legal responsibility, I certify that this diploma:

- has been written by me alone and does not contain any content obtained in a manner inconsistent with the applicable rules,
- had not been previously subject to the procedures for obtaining professional title or degree at the university

Furthermore I declare that this version of the diploma thesis is identical with the electronic version attached.

…… ……..……

date author's (s') signature

Statement

I agree / ~~do not agree~~[*1] to make my diploma thesis available to people, who may be interested in it. Access to the thesis will be possible in the premises of faculty library. The thesis availability acceptance does not imply the acceptance of making the copy of it in whole or in parts.

Regardless the lack of agreement the thesis can be viewed by:

- the authorities of Warsaw University of Technology
- Members of The Polish Accreditation Committee
- State officers and other persons entitled, under the relevant laws in force in the Polish Republic, to free access to materials protected by international copyright laws.

Lack of consent does not preclude the control of the thesis by anti-plagiarism system.

…… ……..……

date author's (s') signature

*1- delete when applicable

Table of contents

1 Introduction

High Temperature Pebble Bed Nuclear Reactors (HTR-PB) are very specific because their cores contain small balls, which are fuel and a moderator as one, they are also called fuel pebbles. These balls are a nearly spherical shape and are closed in a cylindrical-cone container called the reactor. Figure 1 presents a typical scheme of described type of the nuclear reactor. Figure 2 shows a schematic of a single fuel pebble and its interior.

A typical engineering challenge of HTR-PB is finding the distribution of pebbles inside the reactor. The problem is caused by the fact that although the reactor is self-controlling, we cannot precisely determine the k_{eff} value during a design stage.

The primary purpose of this work is to solve the problem of random distribution of spheres (balls) in a closed space, using vector algebra and mechanics, with a presence of gravity. A pseudo-random method was proposed which simulates physical process of pebble bed formation.

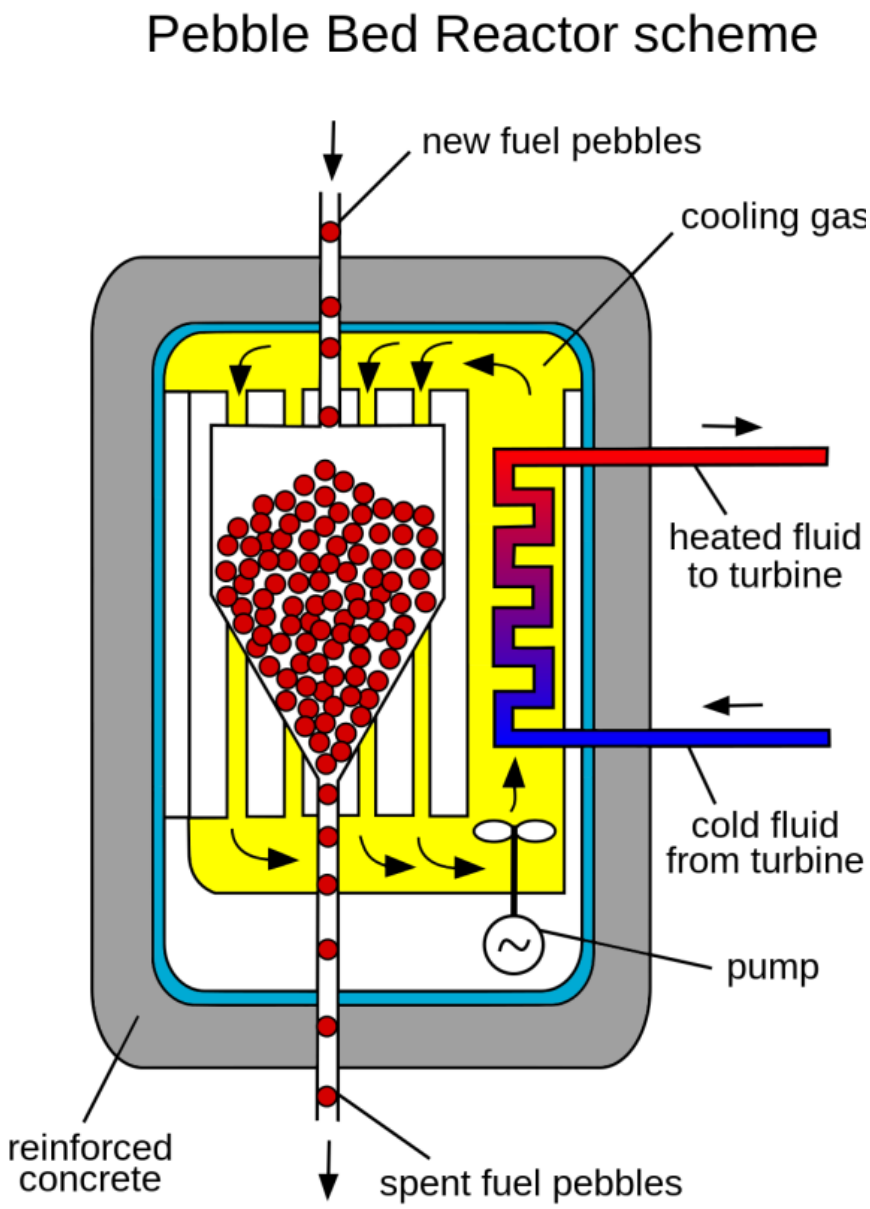

Figure 1. Pebble Bed Reactor (source: [8])

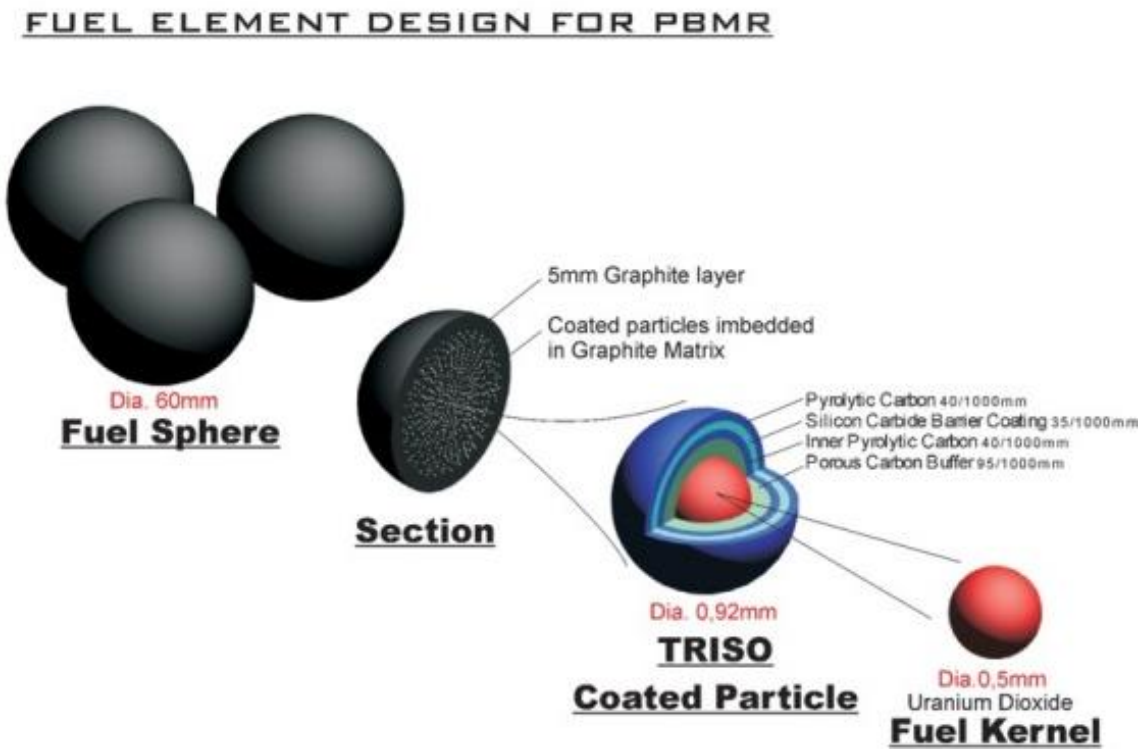

Figure 2 Fuel element in HTR (source: [4])

The methodology of the solution has implemented physical laws. It means that the algorithm simulates physical pebble bed generation process. A computer cannot perfectly simulate reality. However, the model must be discretized. The model's discretization rule is: "Balls are bouncing all the time, and never touch anything." It means that balls are moving all the time, and they are colliding with each other. Between balls, all the time are small gaps.

Thesis model has many advantages like fully randomized positions of balls in the reactor, possibility of setting an intensity of randomization, solution which mimics physical reality. However, the algorithm has one huge disadvantage – convergence time, which depends on the number of balls in a reactor. The dependency of balls between iterations causes another problem with implementation. Therefore, it could be hard to implement this algorithm to real HTR PB reactor. In other hands in this time, people create faster and faster computers.

The developed algorithm is not trivial. It allows generating beds for pebbles with different masses and radiuses. However, all examples shown in this document are made in a case of the same ball radius and mass.

In the next chapter (Model) is presented a mathematical model as an introduction to how calculations are done. The third chapter (Computer program) present how the whole program works. The fourth one (Ball's creation) shows how proper setting balls creation part increased the speed of the algorithm. In the next part is information about randomization of values (5 Randomization). Moreover, the last ones are respectively: Tests and results (5), and Possible improvements (5).

2 Model

The main rule in this model is "Balls are bouncing all the time, and never touch anything." It means that if in next iteration balls overrun each-other or any ball is beyond the boundary, the algorithm calculates collision. In Figure 3 is shown the conception of the model. Red balls are the colliding one. Bounce algorithm diagram is presented in Figure 4; it explains the whole conception.

The first step is initializing all properties like reactor geometry, time step value, balls properties. It is time 0. The next step is the creation of the first layer of balls. Then algorithm starts new time step and accelerates balls. Then algorithm finding any collision that occurred and calculates new balls positions. Collisions could be between balls or with the geometry. Then the program is repeating collision finding and calculating until any collision is not found. It converges thanks to energy dissipation in each collision. If any collisions occurred, the function returns the data of new balls (in the matrix). Next step is to check if the kinetic energy of whole balls is smaller than kinetic energy caused by gravity acceleration. If it is not, the algorithm makes another time step. If it is, the algorithm checks the highest ball's position. If the position is higher than cylinder height minus twice ball's radius, then algorithm stops and returns created pebble bed. Otherwise, the algorithm creates a new layer of balls.

This chapter concerns method model description, which generally is balls bounce model and geometry collision model. Bounce between balls is described in 2.1, and boundary collision is described in 2.2. Code representation of the model is shown in *Appendix C – bounce check, Appendix D – ball's bounce* and *Appendix E – geometry collision*. The main script (Appendix C) diagram is shown in Figure 4, in the green field.

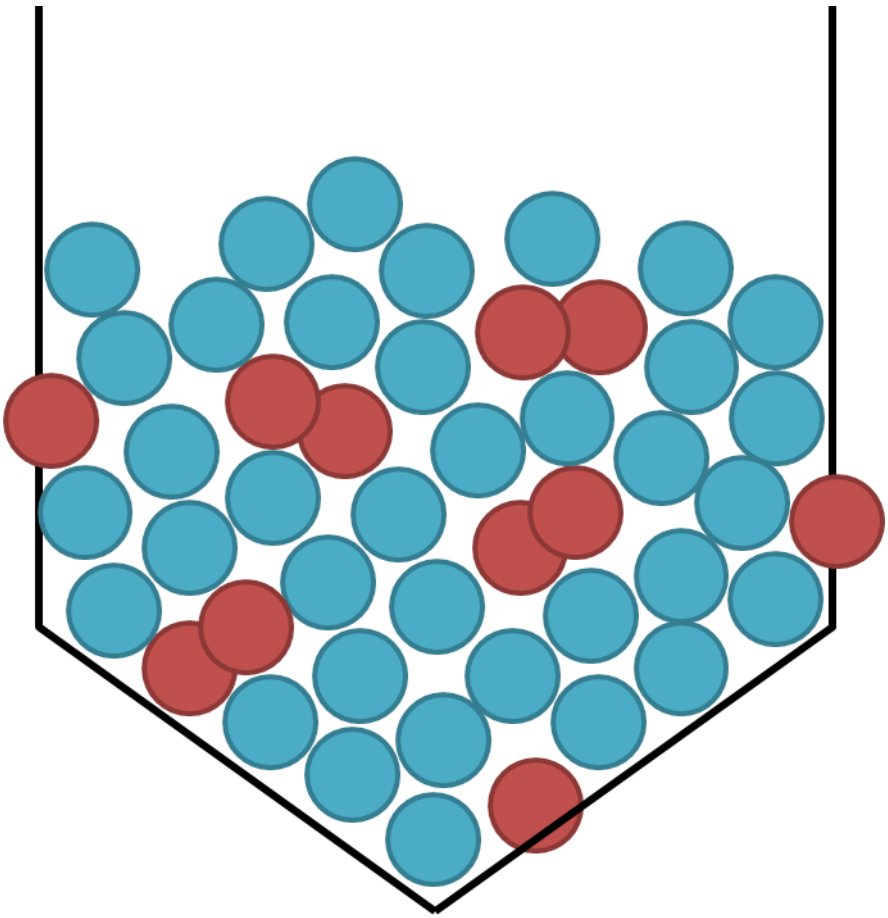

Figure 3 The conception of the model

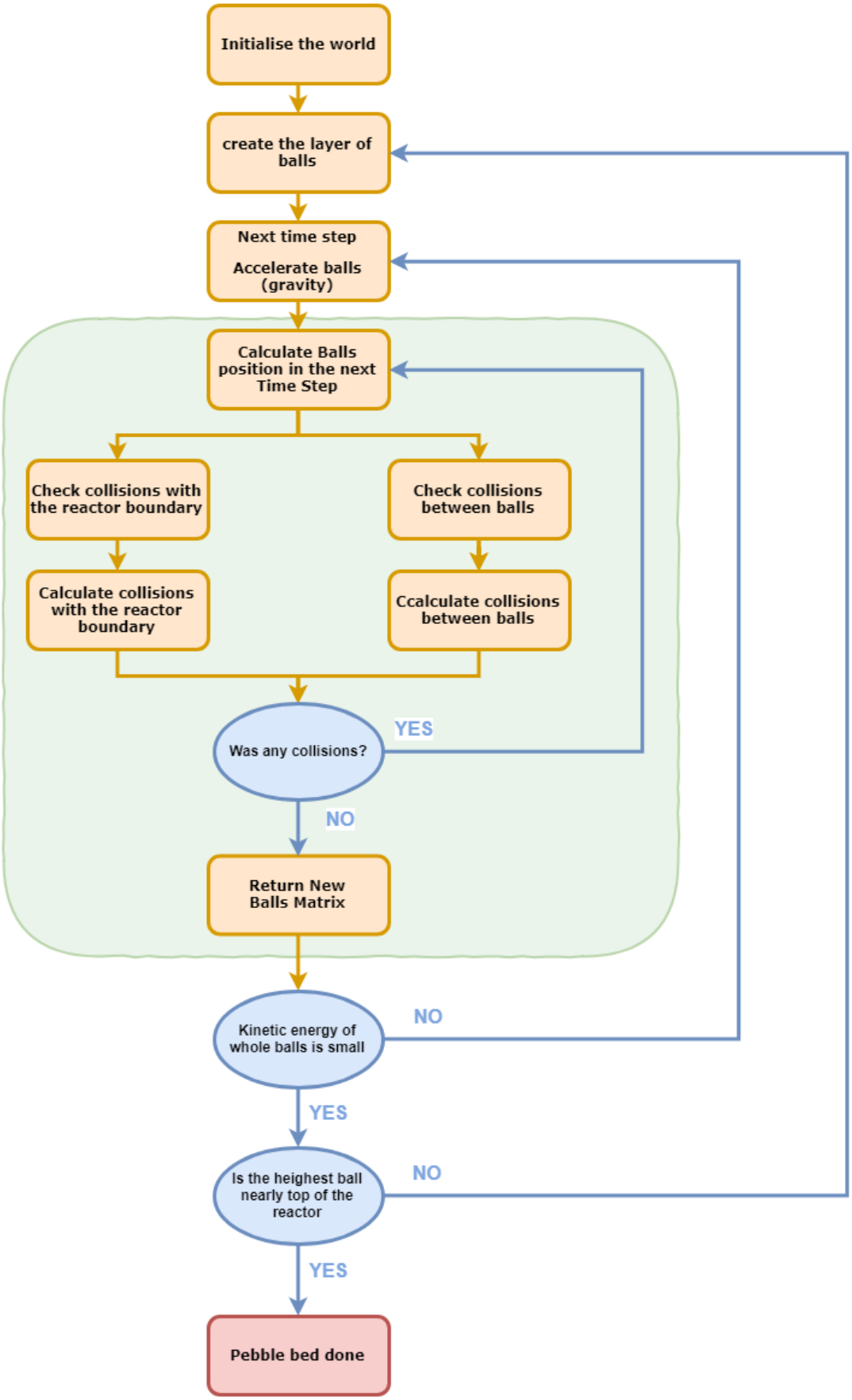

Figure 4 Bounce Algorithm diagram

2.1 Bounce between balls

2.1.1 Bounce description

The first would be explained balls bouncing model. It can be guessed that this model is different towards reality. This difference is the fact that balls never touch each other and the reactor boundary. It means that if in a next iteration balls overrun each-other, the algorithm calculates collision from the previous state. In Figure 5 we can see a real ball's bounce in 3D. The thesis model avoids step 2 – connection (touch), it calculates a state (3) from a state (1). It is essential due to the discretization of the movement.

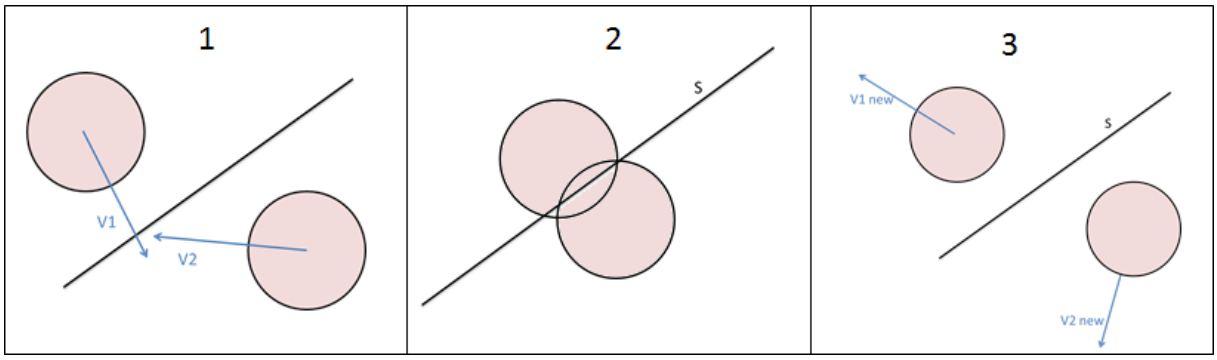

Figure 5. Ball's bounce in real (3D)

Due to never touching balls model there is another problem. The question is how small a time step value should be to avoid balls crossing over. It can happen if the time step is too significant. If time step is too small, calculations are very slow. However, the accuracy is higher. It is crucial information that time step does not influence on algorithm convergence, of course in reasonable values. Energy dissipation guarantee algorithm's convergence.

In Figure 6 is presented the scale of a numerical collision. The scale is shown just before calculations are done. The condition is that the time step has reasonable value.

In conclusion, the output error is decreasing with the time step. It happens, because a distance between two balls before collision decreasing, when time step goes down.

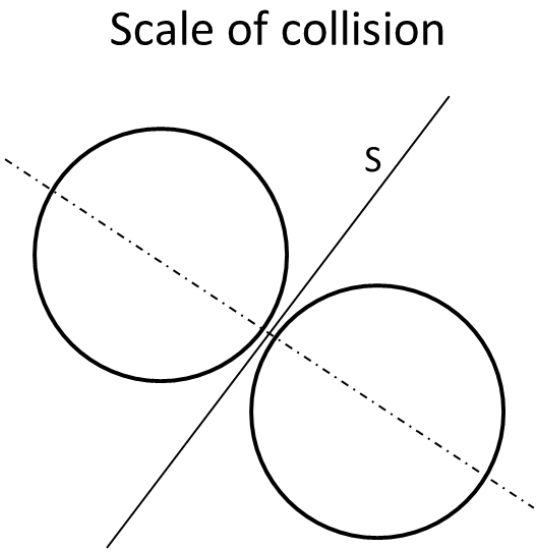

Figure 6. The scale of collision in Algorithm

On the other hand, it must be remembered that the task is pebble bed creation, not high accuracy mechanical bounce calculations. Due to this reason, time step should be as high as possible for fast calculations and should avoid crossing over. More information about setting time step is in 3.3.2.

Code diagram is straightforward, and it is shown in Figure 7. The scheme presents balls collision computational procedure. All vectors concern momentum, because it is only calculated variable.

In *Appendix D – ball's bounce* is the code with a function returning the position of two balls and their properties. The function returns only two colliding balls. The mathematical model and used equations are presented in the next paragraph - 2.1.2.

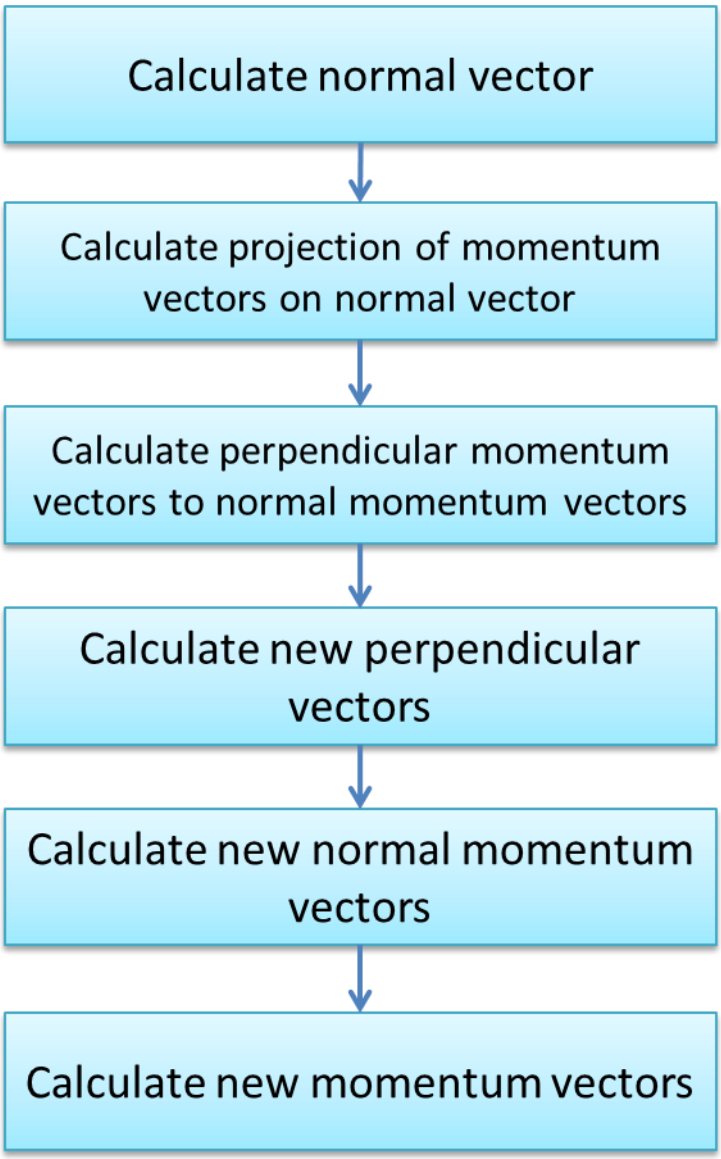

Figure 7 Bounce between balls diagram

2.1.2 Bounce mathematical model

In this paragraph is presented a mathematical description of bouncing balls. The model uses energy conservation law and momentum conservation law. However, before using them balls, momentum vectors must be transformed into a different form. Perpendicular and normal vector, to collision surface normal vector, represents the form. Before mathematical description starts, the reader should be familiar with the description of variables, and their meaning.

Acronym “V” means Velocity, “p” means momentum. Bounded vectors – new ones, have additional mark ‘. The numbers (1) and (2) relate to the first and the second ball. Known variables are positions and velocities.

$$BallCenter2 = [\,x_2 \quad y_2 \quad z_2\,] \qquad BallCenter1 = [\,x_1 \quad y_1 \quad z_1\,]$$

$$\overrightarrow{V_2} = [\,V_{2x} \quad V_{2y} \quad V_{2z}\,] \qquad \overrightarrow{V_1} = [\,V_{1x} \quad V_{1y} \quad V_{1z}\,]$$

$$\overrightarrow{p_2} = m_2 \cdot \overrightarrow{V_2} \qquad \overrightarrow{p_1} = m_1 \cdot \overrightarrow{V_1}$$

$$\overrightarrow{V_2}' = [\,V_{2x}' \quad V_{2y}' \quad V_{2z}'\,] \qquad \overrightarrow{V_1}' = [\,V_{1x}' \quad V_{1y}' \quad V_{1z}'\,]$$

$$\overrightarrow{p_2}' = [\,p_{2x}' \quad p_{2y}' \quad p_{2z}'\,] \qquad \overrightarrow{p_1}' = [\,p_{1x}' \quad p_{1y}' \quad p_{1z}'\,]$$

If the position of two balls is given, then normal collision vector could be calculated, which refers to the normal vector of bounce surface.

$$Normal\ Vector\ to\ surface\ S = \vec{n} = [x_2 - x_1 \quad y_2 - y_1 \quad z_2 - z_1] \tag{1}$$

All following transformations are conducted for the first ball. However, the way for the second ball is the same.

So, the normal and perpendicular momentum vectors can be computed. These vectors are the different representation of vectors in the global frame of reference. Only the form is changing. The next equations show how to transform the vector into a different form.

$$\begin{cases} \overrightarrow{p_{1n}} = \dfrac{\overrightarrow{p_1} \circ \vec{n}}{\|\vec{n}\|} \cdot \vec{n} & (2) \\ \overrightarrow{p_{1n}} + \overrightarrow{p_{1p}} = \overrightarrow{p_1} & (3) \end{cases}$$

Where:

$\overrightarrow{p_{1n}}$ – normal vector

$\overrightarrow{p_{1p}}$ – perpendicular vector to normal vector

From the equation (3) can be got the unknow perpendicular vector, which is presented in the equation (4).

$$\overrightarrow{p_{1p}} = \overrightarrow{p_1} - \overrightarrow{p_{1n}} \tag{4}$$

This model was concerned that during collision a direction of a normal vector changes. A module of the perpendicular vector is nearly the same, little smaller due to energy dissipation. Its natural assumption that balls exchange energy only through normal vector’s direction.

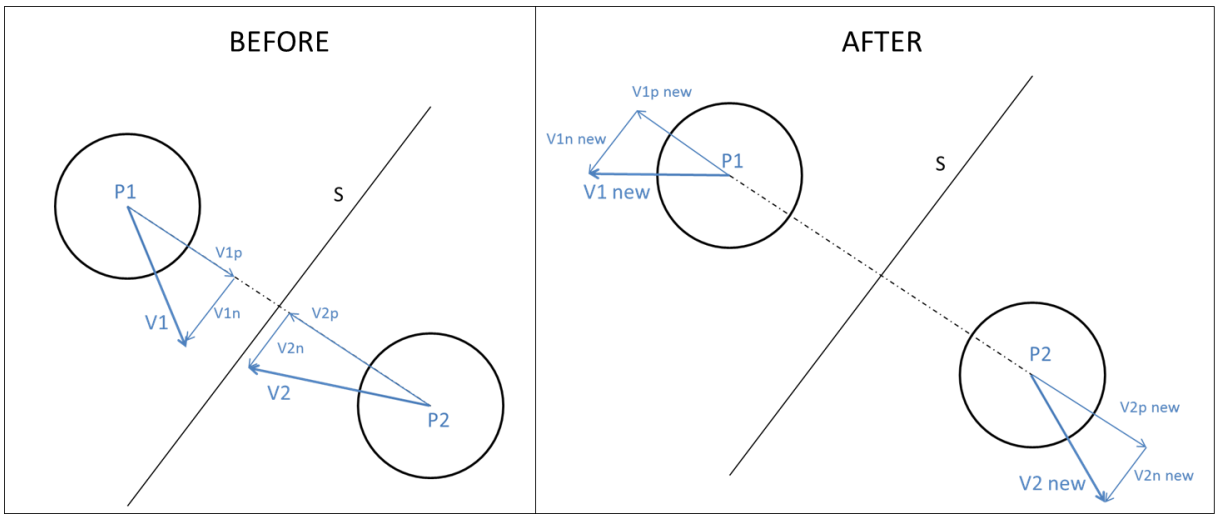

Figure 8. Collision Description

In Figure 8 are shown balls, which collide each other in thesis model. Balls exchange their momentum and energy. Below are presented equations: (5) and (6) describing this phenomenon: respectively momentum and energy conservation laws.

$$\begin{cases} \| \overrightarrow{p_{1n}} \| + \| \overrightarrow{p_{2n}} \| = \left\| \overrightarrow{p_{1n}'} \right\| + \left\| \overrightarrow{p_{2n}'} \right\| & (5) \\ \frac{\| \overrightarrow{p_{1n}} \|^2}{2 \cdot m_1} + \frac{\| \overrightarrow{p_{2n}} \|^2}{2 \cdot m_2} = \frac{\left\| \overrightarrow{p_{1n}'} \right\|^2}{2 \cdot m_1} + \frac{\left\| \overrightarrow{p_{2n}'} \right\|^2}{2 \cdot m_2} & (6) \end{cases}$$

Group of equations: (5) and (6), has two solutions. The first one is straightforward and is shown below as equations (7) and (8).

$$\begin{cases} \left\| \overrightarrow{p_{1n}'} \right\| = \| \overrightarrow{p_{1n}} \| & (7) \\ \left\| \overrightarrow{p_{2n}'} \right\| = \| \overrightarrow{p_{2n}} \| & (8) \end{cases}$$

The second one solution is more complicated and is shown below as equations (9) and (10).

$$\begin{cases} \left\| \overrightarrow{p_{1n}'} \right\| = \| \overrightarrow{p_{1n}} \| \cdot \frac{m_1 - m_2}{m_1 + m_2} + \| \overrightarrow{p_{2n}} \| \cdot \frac{2 \cdot m_1}{m_1 + m_2} & (9) \\ \left\| \overrightarrow{p_{2n}'} \right\| = \| \overrightarrow{p_{1n}} \| \cdot \frac{2 \cdot m_2}{m_1 + m_2} + \| \overrightarrow{p_{2n}} \| \cdot \frac{m_1 - m_2}{m_1 + m_2} & (10) \end{cases}$$

The second solution seems that (9 and 10 equation) would be better if the target was a real solution of ideal elastic balls. Let's consider a situation where m_1 is equal m_2. Then, according to equations (9) and (10), the module of vector p_{1n}' is equal to the module of the vector p_{2n}. Also, a module of the vector p_{2n}' is equal to p_{1n}. It happens due to not implemented elasticity of balls. In this case, could happen situation that if plenty of balls create little pebble bed (all velocities are nearly 0) and another ball hits this pebble bed. This another ball would have momentum (velocity) nearly 0, and all energy would be transferred to this group of balls. This way causes convergence speed problem and the problem with the consistent distribution of balls in the reactor. The reason is that the kinetic energy of this another ball is divided into pebble bed's kinetic energy. The case is shown in Figure 10.

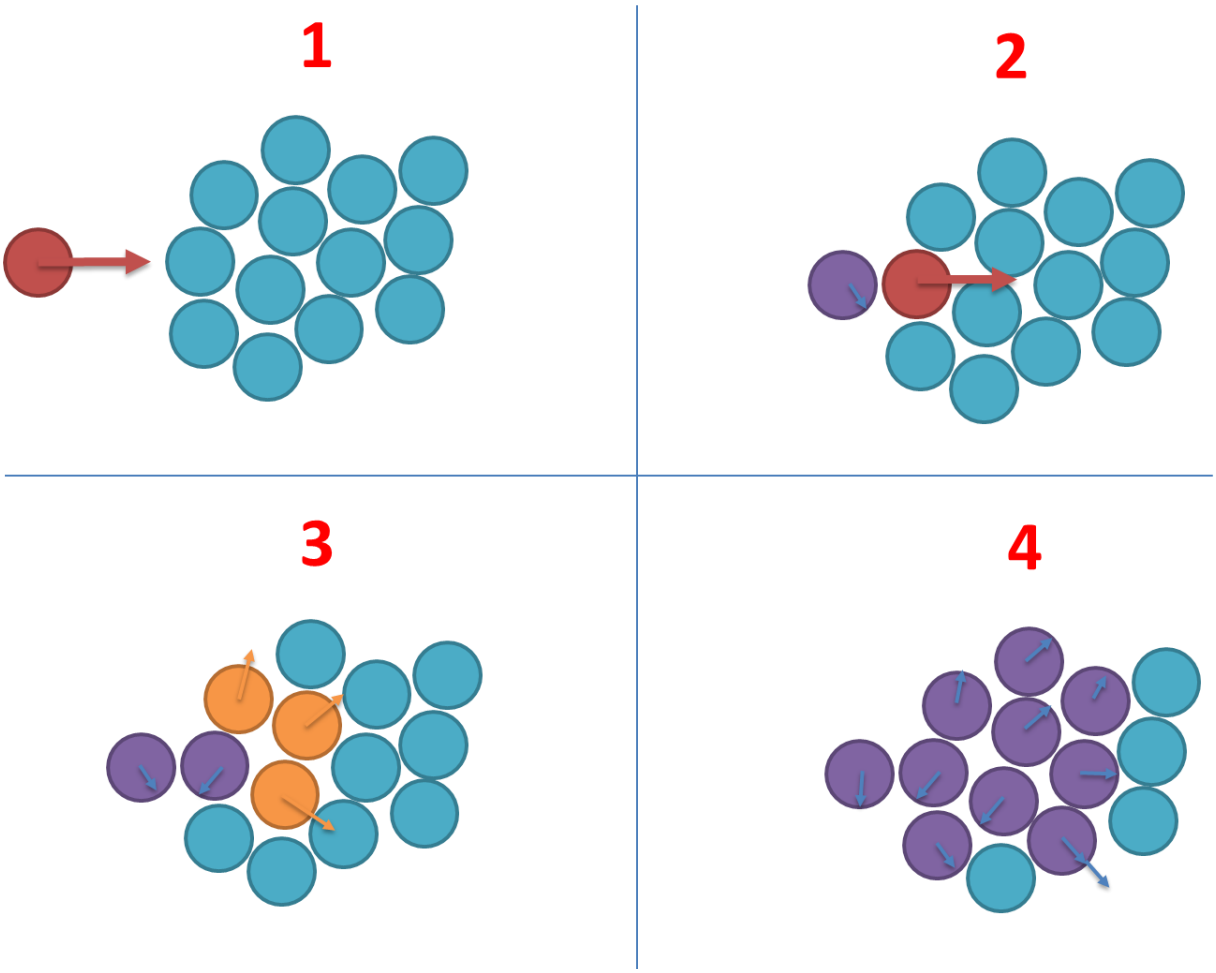

Figure 10 The case of the second solution

Oppositely, the first solution (equations: 7 and 8) makes pebble bed more stable. It means if any ball hits a created bed, this ball will reflect from the bed. The bed is not destroyable. The conception is shown in Figure 10.

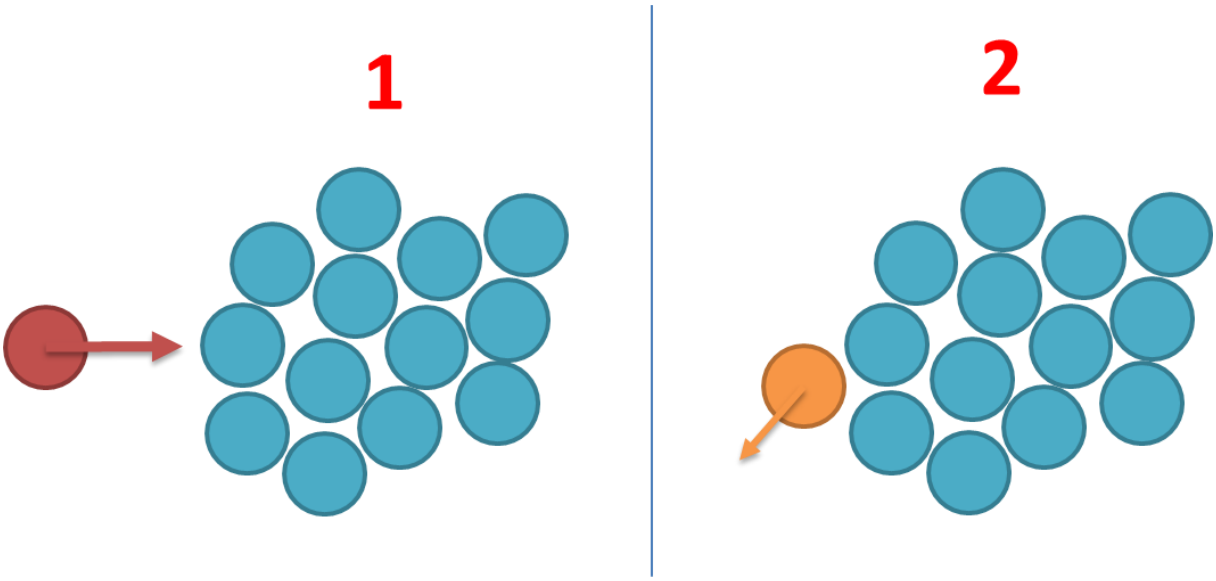

Figure 9 The case of the first solution

Cause of above reasons, it was decided to use simpler solution, the first one. Due to this choice created pebble bed would be not destroyable. It means if any ball strikes group of balls, pebble bed would be still the same. Thanks for this solution, convergence time decreases. Moreover, the solution gives the opportunity to improve the speed of the algorithm, which is described in section 7.1.

Due to the need for convergence, the bounce algorithm has implemented energy dissipation ratio. Otherwise, balls would be bouncing all the time. Moreover, all new vectors are randomized for a little – more information in 5.1.

If the module of a new normal momentum vector is computed, then could be calculated ball's new momentum vector. As it can be seen in Figure 8 new normal vector has the same direction as previous one, but with the reverse return. So old vector must be multiplied by -1, divided by its module and multiplied by a calculated module of new vector and energy dissipation ratio to get a new normal vector. New perpendicular vector has the same direction and returns as an old one. However, a value must be multiplied by energy dissipation vector. At the end to get new momentum vector it has to be summed normal and perpendicular vectors. All equations are presented below, equations from 11 to 16. Feature EDF means Energy Dissipation Ratio.

$$\begin{cases} \overrightarrow{p'_{1n}} = -\overrightarrow{p_{1n}} \cdot \dfrac{\left\| \overrightarrow{p_{1n}{}'} \right\|}{\| \overrightarrow{p_{1n}} \|} \cdot EDF & (11) \\ \overrightarrow{p'_{2n}} = -\overrightarrow{p_{2n}} \cdot \dfrac{\left\| \overrightarrow{p_{2n}{}'} \right\|}{\| \overrightarrow{p_{2n}} \|} \cdot EDF & (12) \\ \overrightarrow{p_{1p}{}'} = \overrightarrow{p_{1p}} \cdot EDF & (13) \\ \overrightarrow{p_{2p}{}'} = \overrightarrow{p_{2p}} \cdot EDF & (14) \\ \overrightarrow{\boldsymbol{p_1}{}'} = \overrightarrow{\boldsymbol{p_{1n}}{}'} + \overrightarrow{\boldsymbol{p_{1p}}{}'} & (15) \\ \overrightarrow{\boldsymbol{p_2}{}'} = \overrightarrow{\boldsymbol{p_{2n}}{}'} + \overrightarrow{\boldsymbol{p_{2p}}{}'} & (16) \end{cases}$$

2.2 Ball collision with the boundary

2.2.1 Collision description

Ball's collision with boundary is very similar to the balls bouncing from a mathematical point of view. The main difference is a way how to find normal vector of a collision. From mathematical and conception point of view, next steps are the same. It means that the algorithm for finding a normal vector works like it is shown in Figure 7. Diagram demonstrated in Figure 11 shows a way how to find a normal vector.

In principle, it is not easy to find a normal vector, due to combined shape of the reactor. Moreover, the main problem was that boundaries of a reactor are not boundaries for ball's centers. By black line in Figure 12 is shown the shape of the reactor. Additionally, on the same figure, by the red dotted line is shown the boundary for ball's centers. It must be done because balls matrix contains only ball's centers positions. Moreover, new boundary (red dotted line) depends on ball's radius, so each ball theoretically has a different boundary.

The *Appendix E – geometry collision* presents MATLAB code with the function which calculates ball's collision with the boundary. The function returns only one ball and its properties.

Figure 11 presents ball-wall interaction computation diagram.

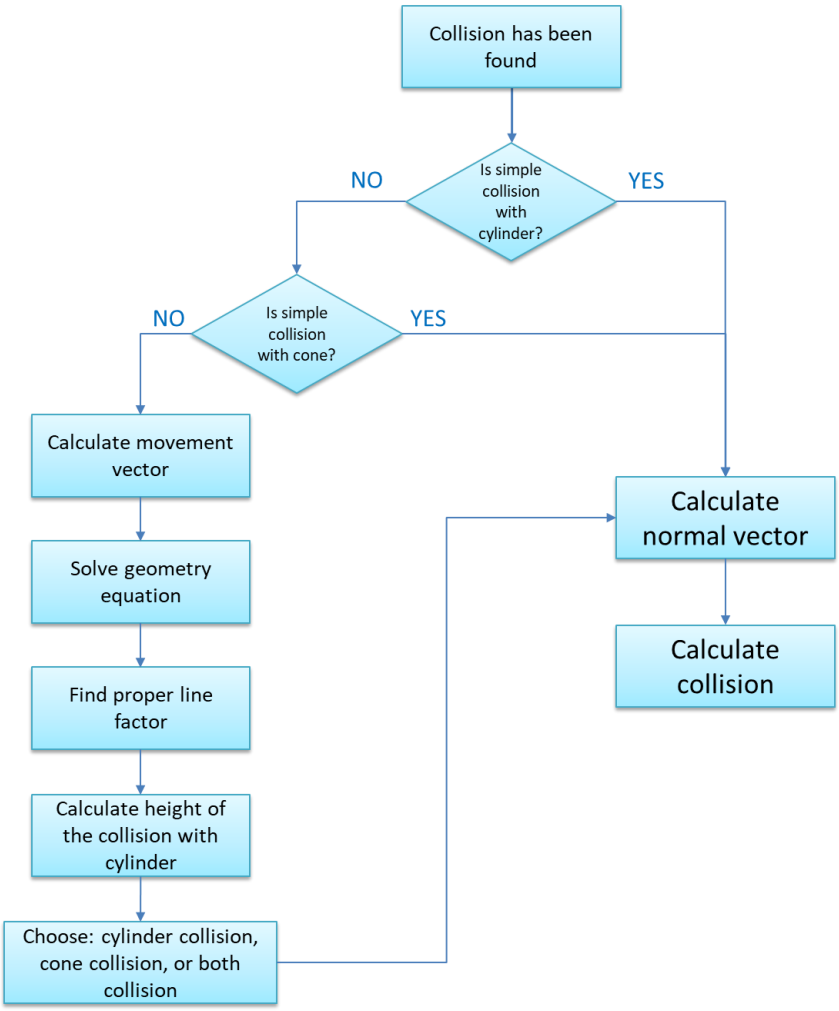

Figure 11 Ball collision with boundary diagram

2.2.2 Collision mathematical model

Reactor core has a shape of a cylinder and a cone at the bottom. Equation (17) presents cone equation, which was applied to the model. The cylinder equation is very similar; the difference is that the right side of the equation is equal to square cylinder radius.

$$x^2 + y^2 = a^2 z^2 \tag{17}$$

Where:

$$a = \frac{cylinder\ diameter}{2 * cone\ height}$$

If point (0,0,0) is chosen as the middle point of the bottom base of the cylinder, cone equation is the following:

$$x^2 + y^2 = a^2 \cdot (z + L)^2 \tag{18}$$

Firstly, geometry equations must be modified because ball's center's boundary is not the same boundary as the reactor boundary. By red dotted line in Figure 12 is presented the ball's centers boundary. Equation (19) presents cylinder modified equation.

$$x^2 + y^2 = (cylinder\ radius - ball\ radius)^2 \tag{19}$$

Equation (20) presents modified cone equation

$$x^2 + y^2 = a^2 \cdot (z + L - delta\ z)^2 \tag{20}$$

The feature *delta z* is presented in Figure 12.

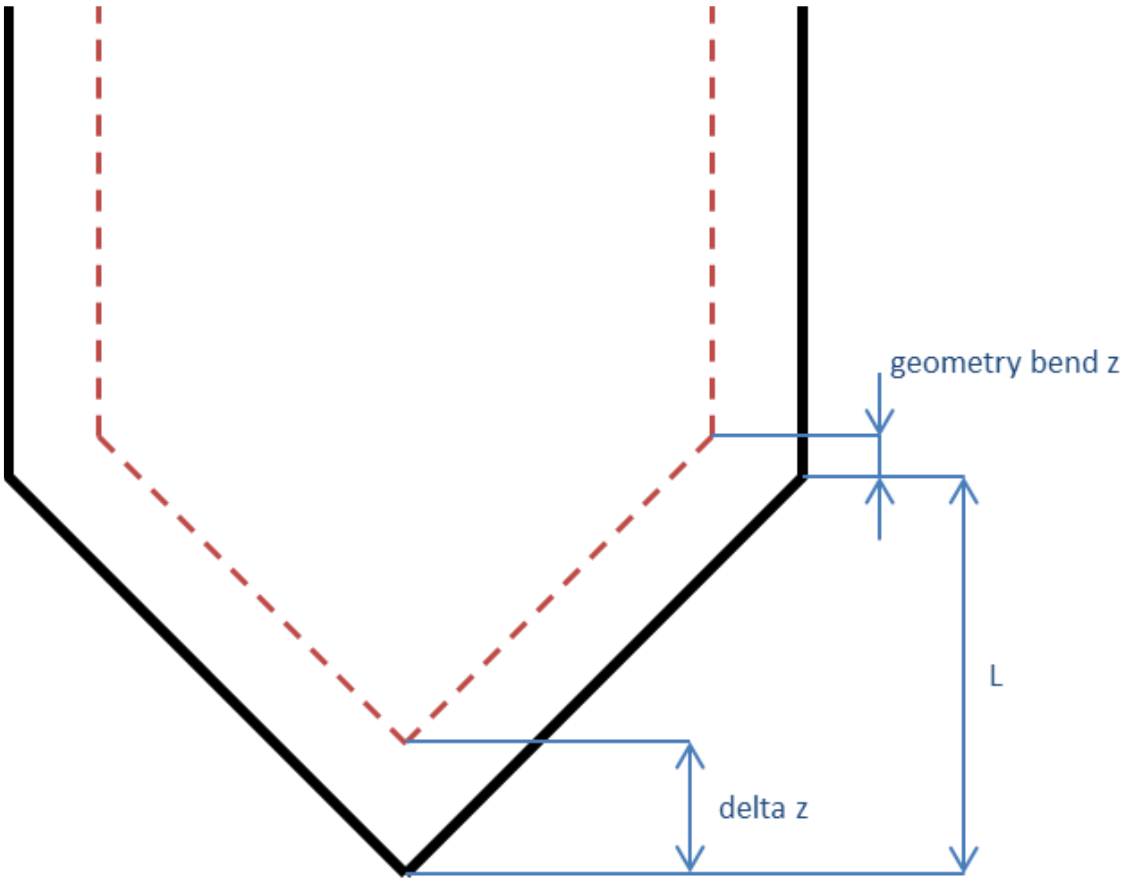

Figure 12 Reactor Boundary Conception

As the first step in the collision calculations is computing two properties shown in Figure 12: *delta z* and *geometry bend z*. The formula for them is shown in equations (21) and (22). It has been done because these values determine the ball's centers boundary.

$$delta\ z = \frac{(4 \cdot (cone\ height)^2 + (cylinder\ diameter)^2)^{\frac{1}{2}}}{reactor\ diameter} * Ball\ Radius \tag{21}$$

$$geometry\ bend\ z = \frac{reactor\ diameter - 2 \cdot Ball\ Radius}{2 \cdot a} - cone\ height + delta\ z \tag{22}$$

Equations give an essential conclusion that features *delta z* and *geometry bend z* are depended on ball's radius. It means if each ball has different size, for each ball new geometry features must be recalculated.

When boundary properties are computed, it can be done the second step. The second step is choosing a collision type: cylinder, cone, or both. Figure 13 presents configurations before and after a collision. The situation before was chosen to be divided into two possibilities: a ball is inside the cylinder (1), and a ball is inside the cone (2). Whereas both situations are divided into three after possible configurations: besides the cylinder (I), below the cone (III), and in the area diagonally from the reactor (II). Orange line marks division area in Figure 13.

Due to that fact, geometry collision is divided into following parts:

- 1 → I - cylinder collision
- 1 → II - special case
- 1 → III - cone collision
- 2 → I - special case
- 2 → II - cone collision
- 2 → III - cone collision

There are three options: cone collision, cylinder collision, and the special case. Cone and cylinder collisions have different normal vectors. If any case is a special one, the algorithm is checking in another way what is a type of collision. In this case could happen situation when is both cone and cylinder collision, which is predicted.

If a collision type is known from the first step (cylinder or cone), there are simple solutions. The problem occurs when it must be calculated there is cone collision, cylinder collision or both. The solution is shown in 2.2.3.

However typical cylinder collision is very simple, the normal vector is given by the equation (23).

$$cylinder\ normal\ vector:\ \vec{n} = [x \quad y \quad 0] \tag{23}$$

Where: x is the ball's position on the x-axis, y is the ball's position on the y-axis and z is the ball's position on the z-axis.

On the other hand, normal cone vector is little more complicated and is presented by the equation (24).

$$cone\ normal\ vector: \vec{n} = [x \quad y \quad grad\ z] \tag{24}$$

Where:

$$grad\ z = -a \cdot (x^2 + y^2)^{1/2} \tag{25}$$

Where: x is the ball's position on the x-axis, y is the ball's position on the y-axis and z is the ball's position on the z-axis. Feature a is the same, like in the equation (17).

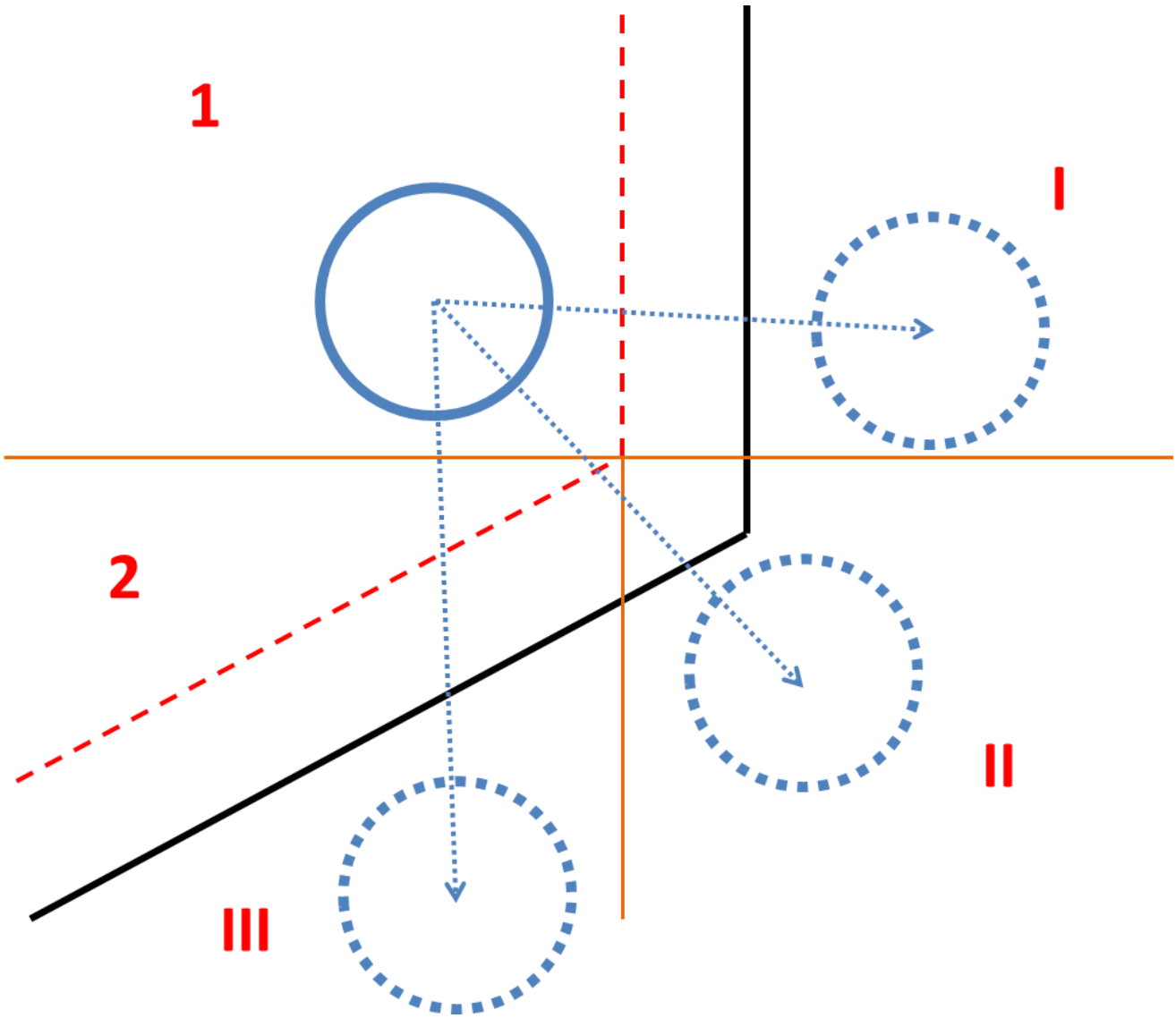

Figure 13 Types of Boundary Collisions

At first glance, it could be strange that in the equation (24) *grad z* does not depend on variable z. It was taken from cone equation (17), where variable *z* can be calculated as a function (x, y). Next, the cone equation was differentiated to get z dimension part of a normal vector.

Equation with grad z (x, y) must be calculated due to the typical situation in the algorithm when a ball which has a collision is not directly placed on the red dotted line (Figure 12). In this situation, all normal cone vectors have the same direction on each intersection plane. It means that direction of a normal vector is only depended on x and y variables. One more important thing must be added. When a normal vector is calculated, then the only direction is needed. Module and return of a vector are unimportant. Thanks to those reasons the algorithm does an advanced trick. It collaterally projects the ball's center position through z-axis on the moved cone surface (red dotted line in Figure 13). Mathematically this trick is done by extracting z from cone equation (17) and differentiating it. Then the algorithm gets z dimensional part of a gradient of the projected point.

The next special case is when x = 0 and y = 0. There is a collision with the cone. It is a problem because according to equations (24) and (25), the normal vector is equal [0 0 0]. It is impossible, thus unacceptable in the algorithm. Because of that if any ball falls straight through z-axis, and collision is detected then a normal vector is the z-dimensional unit vector. The case unit vector is presented in equation (26).

$$cone\ normal\ vector\colon\ \vec{n} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \tag{26}$$

If normal collision vector is founded, then rest of calculations are the same as it was done in the balls bounce's model (2.1.2). The algorithm is using equations: (7), (8), and from (11) to (16).

2.2.3 Special collision case

Special collision calculation means how to determine collision type in cases: 1 → II and 2 → I. The idea is line creation which connects ball's middle points before a collision and after a collision. If this line crossing cylinder boundary above *geometry bend z* in the z-dimension, there is cylinder collision. If this happens below, it is cone collision. In a case when line cross through point geometry bend z, there is both cone and cylinder collision.

Both collision's normal vector is a superposition of cone and cylinder normal vectors. In other words, the direction of a sum of normal cone and cylinder vectors is a solution.

The equations for calculations at the cross point are above. As the first is shown cylinder and line equations – equations from (27) to (31).

$$x^2 + y^2 = R^2 \tag{27}$$

where R is the radius of the cylinder minus ball's radius

$$\begin{cases} x = x_A + n \cdot t & (28) \\ y = y_A + m \cdot t & (29) \\ z = z_A + p \cdot t & (30) \\ [n \quad m \quad p] = [x_B - x_A \quad y_B - y_A \quad z_B - z_A] - direction\ vector & (31) \end{cases}$$

If x and y from line equation are put into cylinder equation, the equation has form presented in equation (32).

$$(x_A + n \cdot t)^2 + (y_A + m \cdot t)^2 - R^2 = 0 \tag{32}$$

Afterward few transformations, the equation has form showed in equation (33).

$$t^2 \cdot (n^2 + m^2) + t \cdot (2 \cdot x_A \cdot n + 2 \cdot y_A \cdot m) + ({x_A}^2 + {y_A}^2 - R^2) = 0 \tag{33}$$

The more comfortable form is presented in equations from (34) to (37).

$$\begin{cases} a \cdot t^2 + b \cdot t + c = 0 & (34) \\ a = (n^2 + m^2) & (35) \\ b = (2 \cdot x_A \cdot n + 2 \cdot y_A \cdot m) & (36) \\ c = ({x_A}^2 + {y_A}^2 - R^2) & (37) \end{cases}$$

Equation (34) is a quadratic equation, which has two solutions presented in equations (38) and (39).

$$\begin{cases} t_1 = \dfrac{-b + \sqrt{\Delta}}{2a} & (38) \\ t_2 = \dfrac{-b - \sqrt{\Delta}}{2a} & (39) \end{cases}$$

The variable Δ is presented in equation (40).

$$\Delta = R^2 \cdot (n^2 + m^2) - (n \cdot y_A - m \cdot x_A)^2 \quad (40)$$

One of these values is a point at which the algorithm is looking. The second is a cross point of the line and cylinder's other side. This point is shown in Figure 14. Points A and B

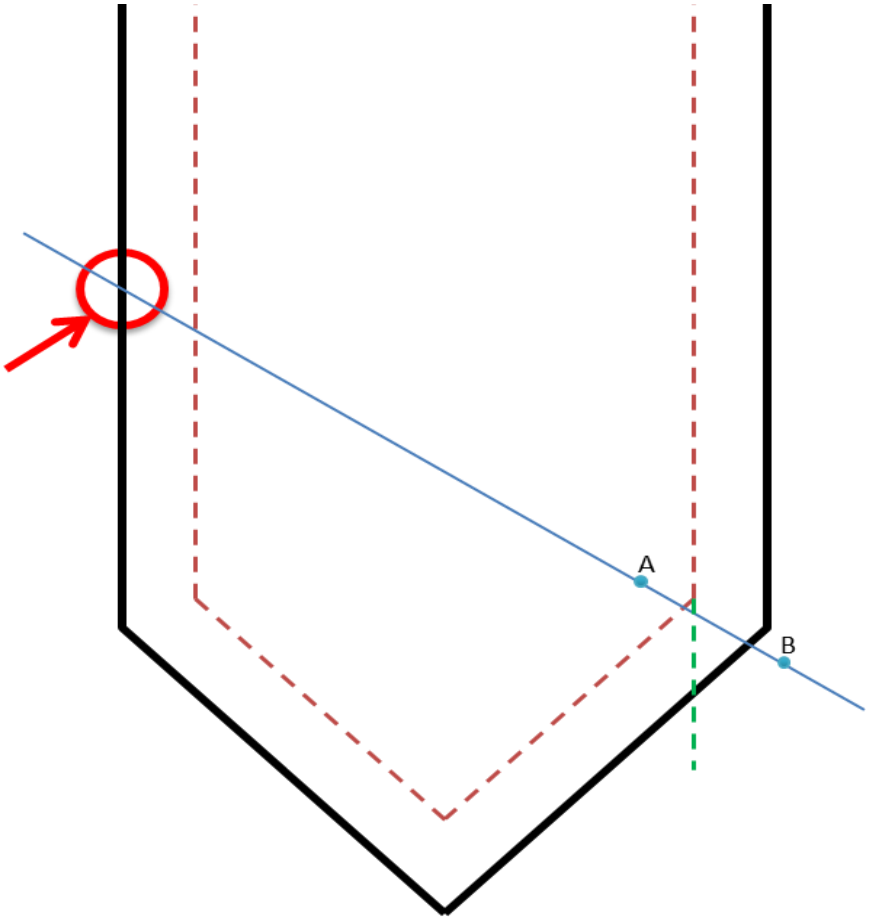

Figure 14 The way for special collision calculations

are points of the balls centers before collision (A) and after a collision (B).

The way how to validate solution and choose proper "t" value is effortless. Value of parameter "t" should be in range [0,1]. If "t" was higher than 1, the solution would be invalid, an exception would be called, and algorithm stopped. For the case with negative "t", the solution is marked on Figure 14.

If "t" is known, it can be put into z-dimension part of the line equation – equation (30). The result is presented in equation (41). The variable *p* was defined in equation (31).

$$z_{crossing\ point=CP} = z_A + p \cdot t \quad (41)$$

Then z_{CP} can be compared with *geometry bend z* variable. If z_{CP} is higher than *geometry bend z*, the type of collision is cylinder collision. If it is lower than *geometry bend z*, type of collision is cone collision.

In the case when z_{CP} and *geometry bend z* are equal, the collision is both with the cylinder and the cone. Then the algorithm calculates a cone's normal vector and a cylinder's normal vector. Afterward, it sums both normal vectors.

The pair of collisions normal vectors are not unit vectors, but it does not matter. Important is the only direction. Of course, it could be easily rescaled to the unit vector, but it is also not necessary.

3 Computer program

The solution of the given problem was not only the model. For generation any pebble bed there is need to create a computer program. Accordingly, it was developed in MATLAB [1] environment which has implemented collision model. The code of the program is straightforward and is described in this paragraph.

3.1 Initialization

As the first step, as it was shown in Figure 3, main world properties are initialized. All of them are described below.

- Ball radius – in general, mean ball's radius, now ball creation function is set to create balls with the same size. In the future function *set_ball_radius(BallRadius)* could be changed to generate random radius in a specific range.
- Ball mass – mean ball's mass, which value is assigned to mean ball's radius.
- Time step – more information in section 3.3.2
- Gravity – constant, unchangeable variable
- Energy Dissipation (matrix) – more info in 3.3.1
- Geometry (array) – parameters, which are characterizing the size of the reactor:
  - Reactor Diameter
  - Cylinder Height
  - Cone Height

The next step is to check the data correction. Function *DataCheck(BallRadius, BallMass, TimeStep, geometry, Gravity)* execute this part. Now the function is always set to return one. In the future, it will be fully completed. Now it is assumed that any user knows how to set a proper data.

```
 8      %% inicialize world
 9      %main features
10 -    BallRadius = 50*10^(-3);
11 -    BallMass = 1; % mean value
12 -    TimeStep = 10^(-3);
13 -    Gravity = 9.81;
14 -    EnergyDissipation = [0.7 0.9 ; 0.6 0.8];
15      % dissipation: first row consider dissipation rate of collisions between
16      % balls, and the second collisions with reactor walls. First column is
17      % connected to normal vector and the second to perpendicular vector
18
19      %geometry
20 -    ReactorDiameter = 2;
21 -    CylinderHeight = 0.5;
22 -    ConeHeight = 0.25;
23 -    geometry = [ReactorDiameter;CylinderHeight;ConeHeight];
24 -    if DataCheck(BallRadius, BallMass, TimeStep, geometry, Gravity) ~= 1
25 -        error('Error with DATA !!!');
26 -    end
27
```

Figure 15 Code realizing world initialization

After setting necessary parameters, the initial balls can be created. The initial balls are created like each layer. The process is described in Ball's creation chapter.

The last part of initialization is setting convergence factors like energy dissipation ratio and rest energy. The description of mentioned features is shown in the last paragraph of this chapter 3.3.4.

Just before calculations start, the program is paused and is waiting for user's acceptance. Script realizing of all above processes is presented in Figure 16.

```
28      %% Algoritm initialization
29 -    time = 0;
30 -    creation_completed = false;
31 -    Balls = [];
32 -    Balls = create_balls(Balls, BallRadius, BallMass, time, geometry, Gravity);
33 -    LayerSize = size(Balls,1);
34 -    quantity_of_balls = size(Balls,1);
35      %scatter3(Balls(:,1),Balls(:,2), Balls(:,3))
36
37      % convergence factors
38 -    Energy_Dissipation_ratio = (LayerSize-2)/LayerSize;
39 -    Energy_rest = 10^(-3);
40 -    MaxEnergy = max(abs(Balls(:,9)));
41 -    AvEnergy = mean(abs(Balls(:,9)));
42 -    disp('time: ');
43 -    disp(time);
44
45 -    fprintf('Program paused. Press enter to continue.\n');
46 -    pause;
47
```

Figure 16 Code realizing algorithm initialization

3.2 Main loop

A one loop process in this program is called bouncing balls calculation. This process repeats until variable *creation completed* is equally false. It only happens when three conditions are achieved. The first concerns Energy, the second the height of the created pebble bed. The last one is related to energy like the first but is more restricted.

The first and the second one is mostly connected to new balls creation. If all balls have small kinetic energy, the algorithm checks if the need is more balls or not. If a program needs new balls, it creates a new layer of balls in the reactor and drops them down. The created layer's height is equal to the highest ball height, plus a length of the ball's diameter.

The last one condition is the final check. If other conditions are fulfilled, algorithm calculating collisions, until the average kinetic energy of all balls is minimal. In the current version of code average ball's energy must be fitted to each case.

In the end, the program also returns "model time" of calculation. It means how many times the main loop was repeated, times the time step.

```
%% LOOP
while (creation_completed ~= true)
    time = time + TimeStep;
    if ((MaxEnergy < BallMass * Gravity * TimeStep) && (AvEnergy < ...
            BallMass * Gravity * TimeStep * Energy_Dissipation_ratio) )
        if (max(Balls(:,3)) < (CylinderHeight-2*BallRadius))
            [Balls] = create_balls(Balls, BallRadius, BallMass, ...
                time, geometry, Gravity);
        else
            if (AvEnergy < BallMass * Gravity * TimeStep * ...
                    Energy_rest * Energy_Dissipation_ratio)
                creation_completed = true;
                break;
            end
        end
    end
    Balls(:,6) = Balls(:,6) - Gravity*TimeStep;
    Balls = BoucingBalls(Balls, TimeStep, geometry, EnergyDissipation);
    MaxEnergy = max(abs(Balls(:,9)));
    AvEnergy = mean(abs(Balls((end-LayerSize+1):end,9)));
    disp(' ');
    Display = ['time: ', num2str(time)];
    disp(Display);
end
fprintf ('Total time: %d \n\n', time);
fprintf ('Pebble Bed created !!! \n\n');
```

Figure 17 Code realizing loop calculation

3.3 Convergence factors

This paragraph is describing converge factors, and influence of basic parameters like time step on calculation speed.

3.3.1 Energy Dissipation

It is one of the basic parameters of the model. It is a 2x2 matrix. The first row contains a value of collisions between balls dissipation rate and the second collisions with reactor walls. The first column of each row relates to a normal vector and the second with a perpendicular vector.

Energy dissipation matrix tells only how much energy balls loose on each collision. It makes each velocity vector smaller after each collision (collision after which ball must be iterated next time).

Practice and engineering sense set the values. If them would be too low, the solution was converged too fast, and the general result may be different from reality. If them would be too high, the solution was converged too slow and time of calculation would be too high for rational use of the code.

Because of above reason, it was concluded that the best values for energy dissipation variable would be in the range from 0.6 to 0.95.

3.3.2 Time step value

The primary parameter which characterizes each step of the iteration. It is hugely correlated with gravity because ball accelerates with speed of time step and gravity product. Another time step correlation is average ball's size. It means that if the lower time step is, the higher converge is.

Due to above facts, it was assumed to set acceleration of ball to natural – gravity. Then the time step has to be adjusted to ball's size. Above is shown reasoning how to get proper time step value for this model approximately.

At the first basic equation describing movement in the primitive mechanic model:

$$V_{MAX} \cdot \Delta t = R \quad => \quad V_{MAX} = \frac{R}{\Delta t}$$

It was assumed that maximum distance which ball can overcome is equaled to ball's average radius (R).

The next equation describes the primitive behavior of falling in the gravity field.

$$possible\, V_{MAX} = \sqrt{2 \cdot G \cdot (L + R \cdot \varphi)}$$

Expression *(L+R*φ)* means the highest possible height of created and dropped ball. Variable *φ* is the maximal variation of height in balls creation, and it must be higher than 1. More info about *φ* parameter in Ball's creation.

Now it can be concluded that:

$$possible\, V_{MAX} \leq V_{MAX}$$

$$\sqrt{2 \cdot G \cdot (L + R \cdot \varphi)} \leq \frac{R}{\Delta t}$$

$$\Delta t \leq \frac{R}{\sqrt{2 \cdot G \cdot (L + R \cdot \varphi)}}$$

Above equation shows a correlation between average ball's radius, gravity and time step. There are three variables and one equation, so it has two degrees of freedom. As it was mention before, gravity is set as natural. Ball radius is imposed by the user, like all geometry settings. Because of that, the only adjustable variable is time step.

Moreover, a time step value is mostly depended on cone height. If L is nearly or equal 0 then:

$$\Delta t \leq \frac{\sqrt{R}}{\sqrt{2 \cdot G \cdot \varphi)}}$$

It can be concluded that in this case time step could be much larger than in previous, even 10 or more. It happens, because of ball's creation algorithm. The above algorithm does not count cone. Due to this, the higher the cone is, the lower maximum time step is, hence the higher calculation time.

3.3.3 Number of ball's layers for collision checking

This parameter is not implemented into the current version of the code. However, application of this into script will be made in the future – section 7.1 A few layers calculation.

It is a parameter wholly depended on a balls creation algorithm. It means that whenever algorithm is changed, number of ball's layers for collision checking must be recalculated or even avoided. In general, this parameter is bounded by the rule: "How many

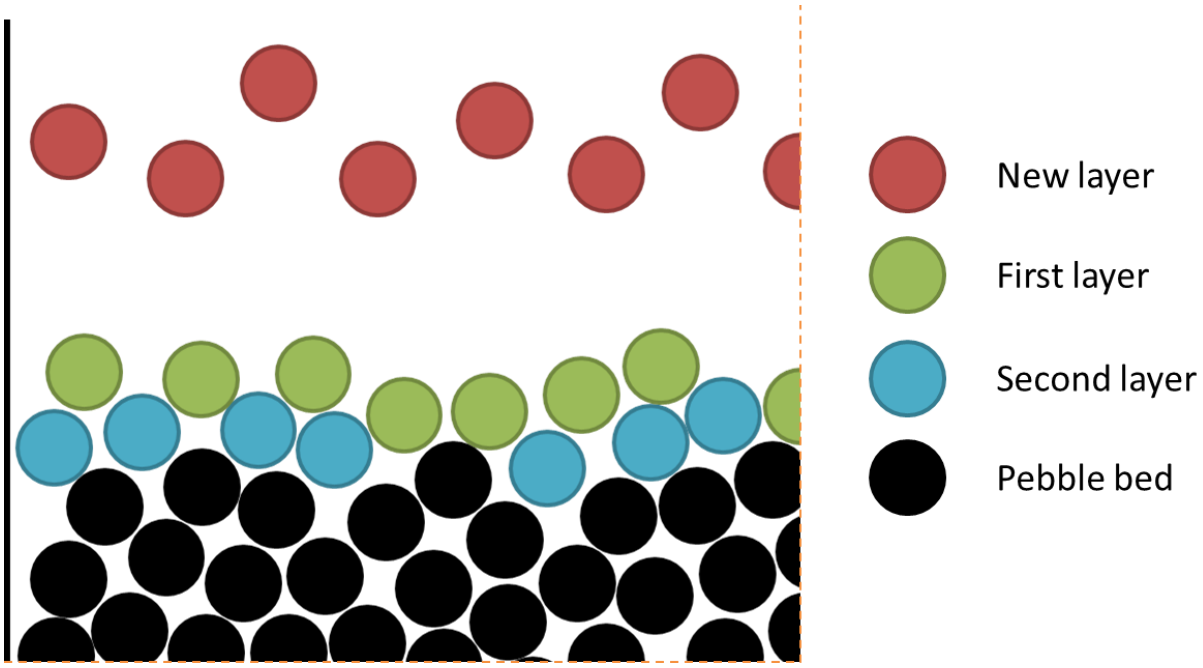

Figure 18 2D example of ball's distribution in the reactor

layers of newly created balls is needed for fully cover existing pebble bed." On the other side number of layers should not be so big, because the speed of calculation is exponentially depended on this variable.

In the current version where ball creation looks like in section 4, this feature should be higher than 3. The explanation for this is straightforward. In Figure 18 can be seen that in the 2D case is needed about two layers of balls to cover the previous layer fully. In the 3D case with current creation, the algorithm is needed minimum three layers. Due to possible "holes" in layers caused by randomization, it is better to use more layers than 3, for example, 4 or better 5.

At least it must be said that if this improvement were added, rest of balls would have to be “grounded.” It means that velocity of each ball in each direction would have to be equal to 0.

3.3.4 Energy dissipation ratio & Energy Rest

Both factors are used for stopping calculation. Energy dissipation ratio parameter is created to avoid situations when the number of new balls created is not higher than 20.

Energy Rest parameter is crucial. When enough balls are created, algorithms comparing a sum of ball's energy with energy rest value and if it is lower, the program is stopped. This parameter is depended on: time step value and the average mass of the ball. Each time when any mentioned features are changed, energy rest parameter must be adjusted.

4 Ball’s creation

4.1 Basic description

The ball’s creation algorithm is the most critical scrip. In this chapter is described current version of creation algorithm, where each ball has the same size and weight. It was done for model simplification.

4.1.1 Ball model representation

In the beginning, balls are described. A vector of size 9 represents each ball. First three elements are position in 3D (x, y, z). Unit of this part is a meter. Next three elements are velocity through each dimension (Vx, Vy, Vz), with unit meter per second. Last three elements are respectively: ball’s radius (meter), ball’s mass(kg) and ball’s energy (kg *m^2/s^2 = Joule).

All balls are placed into the two-dimensional matrix. Each row is a different ball. Each column is a different element.

4.1.2 Layer of balls

The way how algorithm creates balls is effortless. It creates a layer of balls and drops them all down. Rings of balls composite mentioned layers. Conception was taken from [2], where is an algorithm for creation 2D small circles in the bigger one. Conception was little modified – it does not create a ball in the middle of a layer to avoid a problem with the discontinuity at the top of the cone.

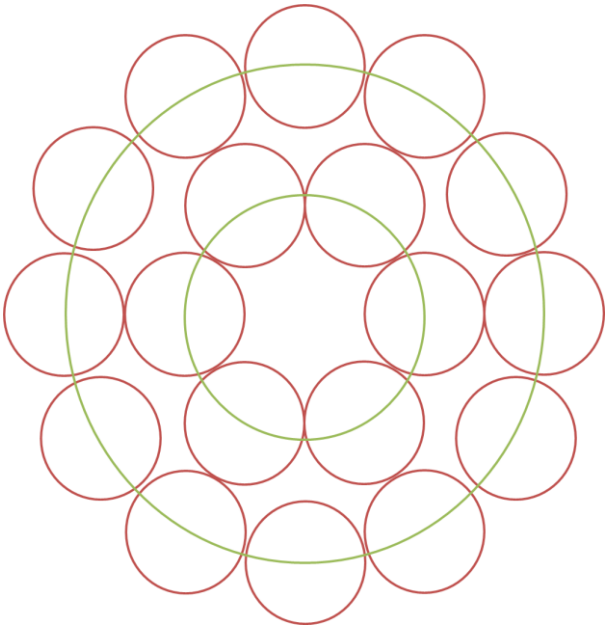

Figure 20 Thesis ball's distribution in the layer

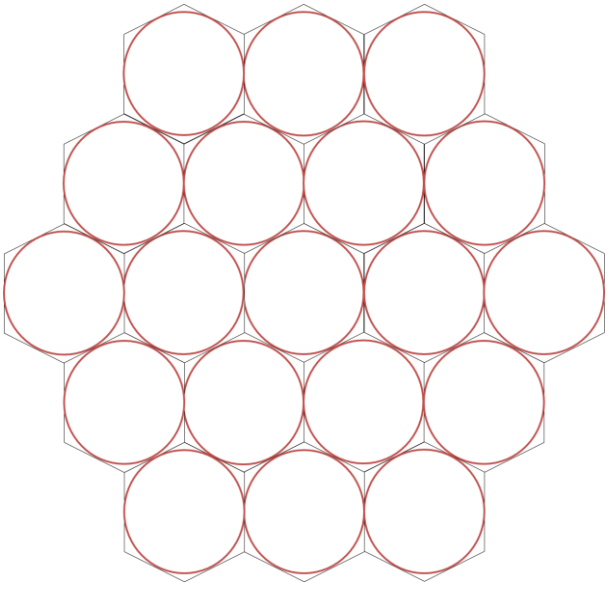

Figure 19 Hexagonal ball's distribution in the layer

The number of balls in each ring is equal to *ring_number* * *6*. It can be easily proved that in this system any balls do not override themselves. For example, if the hexagonal lattice is created, where a circle is inscribed into hexagonal, circles are more packed in the surface than in thesis's solution. If balls from hexagonal lattice are little more distributed in a big circle area, the solution is exactly like in thesis’s solution. Both solutions have the same number of balls inside. However, thesis’s solution has better distribution in the big circle.

4.2 Creation algorithm

Creation algorithm is shown in Appendix B – ball's creation. The script, more precisely layer creation is based on [2]. It must be mentioned that in the script is used name *num_layers* which refers to layer's rings, described in the previous paragraph in this chapter.

As the first script initialize some variables like:

- Radius_ratio – radius of circle area in a circle lattice is little more prominent than ball radius, due to increase randomization and for better distribution of balls. In this thesis, *radius_ratio* is equal to 1.2.
- Height_variance – each layer is not flat, each ball has distance from layer surface equal to *random * height_variance * ball_radius*, where random is a random real number in the range <0,1>.

The next step is making few calculations to find how many layers can exist. It is common that reactor diameter is not multiple of the ball's diameter. Due to this fact, all circle areas are equidistant from each other. In the case when reactor diameter is multiple of ball's diameter *radius_ratio* protects the balls from touching.

After finding proper geometry features, the script sets the position in the z-dimension. If the highest ball is below level 0, it means the whole ball is not in the cylinder, the new layer's height is set to level 0. Else if the highest ball is above level 0, the new layer's level is set just above that ball.

The last step is creating the new layer of balls. In this part is much randomization. The algorithm creates the rings of balls. A random eagle twists each ring. If each circle area is set, the algorithm creates ball randomly, fitting to a circle area. Additionally, each ball has a various height from the layer's level in the range <0; 4**ball_radius*> and small initial velocity towards cylinder axis. More information about randomization in this section in 0.

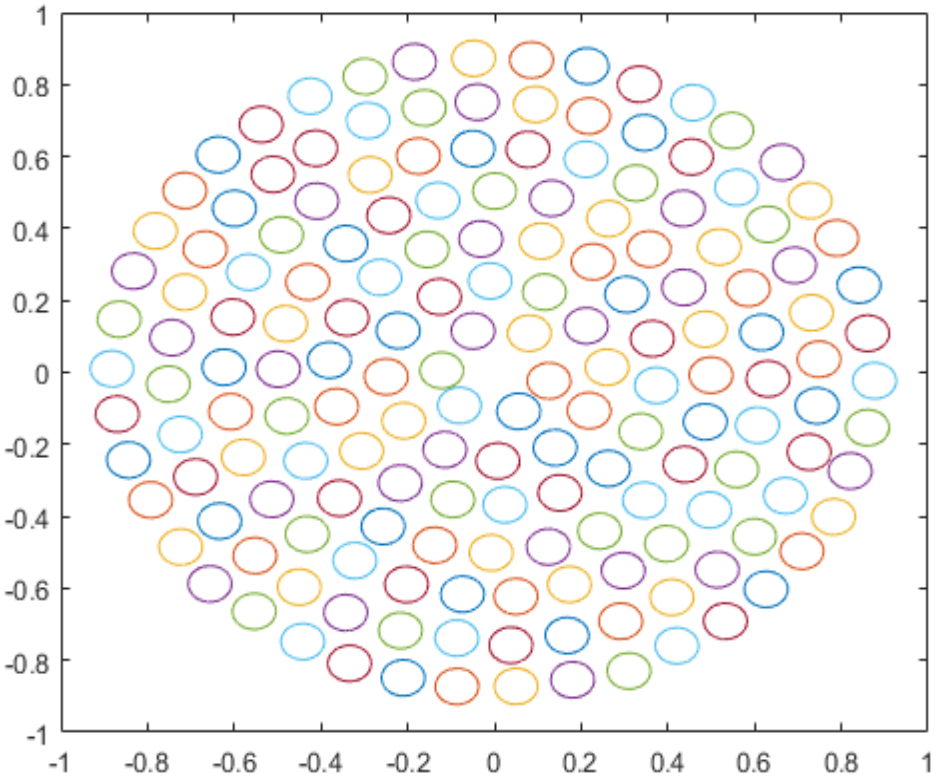

Figure 21 Example of the created layer on XY surface

5 Randomization

Randomization was added to create as chaotic as possible ball's distribution. It is essential because if it was not, the solution could be like a ball on a ball. Shortly, the solution would be far from reality.

5.1 Velocity randomization

Each velocity calculation has implemented little vector's direction randomization. It has to be done due to a situation when a ball has velocity vector only in z (vertical) direction. If it was not done, in algorithm could happen situation when one ball all the time reflect from another one ball. It is the unreal situation, and it had to prevent from this.

Above, on Figure 21 is shown velocity a randomization script. It is effortless. It creates a new vector by adding random value in the range (0; 0.1 * old length in a specific direction) to the old one multiplied by *rand_factor*. Of course, in a different direction of a vector is a different random value. In the end, a new vector is scaled to has the same length as the old one, but both have not the same direction.

```
1   function new_vector = vector_randomization(vector)
2   %little vector randomization
3
4   rand_factor = 0.9;
5   new_a = vector(1)*(rand_factor + rand/10);
6   new_b = vector(2)*(rand_factor + rand/10);
7   new_c = vector(3)*(rand_factor + rand/10);
8
9   new_vector = [new_a; new_b; new_c];
10
11  if norm(new_vector) ~= 0
12      new_vector = new_vector .* (norm(vector)/norm(new_vector));
13  end
```

Figure 22 Vector randomization code

5.2 Layer initiation randomization

Initiation randomization means randomization during the creation of new layers. There are few things which are randomized. Things and reasons why they are created are in points below. All features are created to avoid the situation when balls from all layers have an apparent pattern of distribution.

- Twist of each ring in each layer

  This solution is solving a problem with uniformity of the layer. Each layer in pebble bed is unique. This variable generating eagle between rings in a layer for better distribution during collisions.

- Distant from layer's level

  This variable mostly helps to avoid a situation when layers in pebble bed have an apparent pattern. When the cone is small and layers falling on each other, the layer could fall like one object. Prevention from this situation is giving each ball various distant from layer's level.

- Position in a small circle area

  The additional feature which helps make layer unique. Thanks to this each ring is unique. This variable generating eagle between balls in a ring, for better distribution during collisions.

6 Tests and results

The first parameter which should be described before examples presentation is real convergence time. This feature means how much time a computer needs for creating a pebble bed. Of course, each computer can calculate the same problem at the different time. Due to this fact reader should consider the only relative magnitude of convergence time. The home computer calculated all cases. The computer had following properties: laptop MSI GE70 2OC with processor Intel i7-4700MQ and 16 GB RAM.

The second parameter is virtual convergence time. This parameter is the sum of time steps needed to obtain a result. It was created because it is less dependent on the number of balls and non-dependent on the running machine.

Rest parameters were described in previous chapters. Geometry and calculations properties for each test are given in the next paragraphs.

6.1 Example: HTR-10 with twice larger ball radius

As an example, was chosen HTR-10 reactor. Geometry data was taken from [3], and it is presented in Figure 23. The reactor contains balls with average radius 0.03m, and the final number of balls would be 27 thousand. Due to the predicted number of balls, calculation time would be very high, so as an example was used in the same reactor, but with twice larger balls radius. The radius of balls in the example is equal to 0.06m.

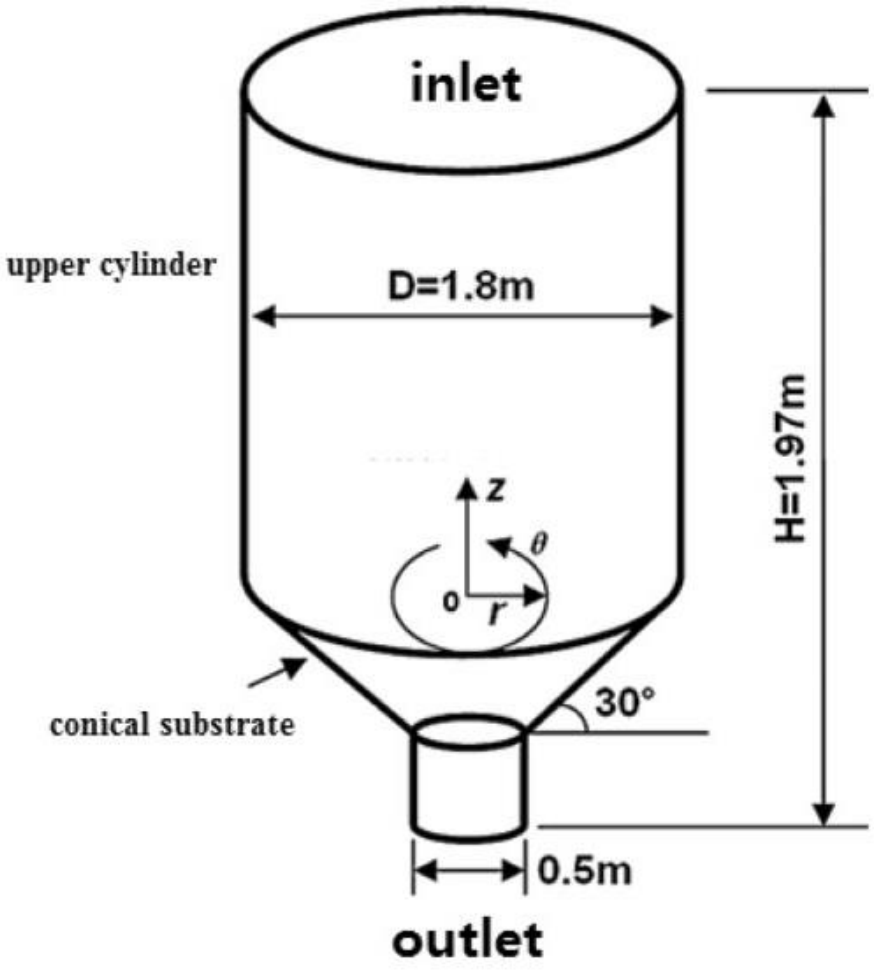

Figure 23 HTR-10 geometry (source: [3])

Despite the use of larger ball radius, calculations took about 98hours (4 days). Thesis's example does not include discharger showed in Figure 23. Table 1 is presented data used in thesis algorithm for HTR-10 pebble bed generation.

Table 1 calculation and result properties of HTR-10 with larger ball radius case

Reactor Geometry		
Diameter	1,8	m
Cone Height	0,52	m
Reactor Height	1,45	m

Ball properties		
Mass	1	kg
Radius	0,06	m

Calculations Properties		
Rings in Layer	5	-
Balls in Layer	90	-
Time step <=	1,55E-02	s
Chosen Time Step	1,00E-02	s

Result		
virtual time	69,94	s
real time	353713	s
	98,3	hour
balls created	2430	-
Av balls per real min	24,73	1/hour
Av balls per virtual s	34,74	1/s

Calculations took time very much. However, the result is satisfying. On the next pictures (Figure 24, Figure 25 and Figure 27) is presented the result of HTR-10 pebble bed with twice larger ball radius. Pictures are screenshots of generated 3D figure by [4] are presented on the next pages. Moreover, Figure 26 presents void fraction through the vertical axis of the reactor. The result of void fraction is nearly the same as in [3]. Whereby a little difference is caused by the larger size of balls and stiffness of balls in the model in [3].

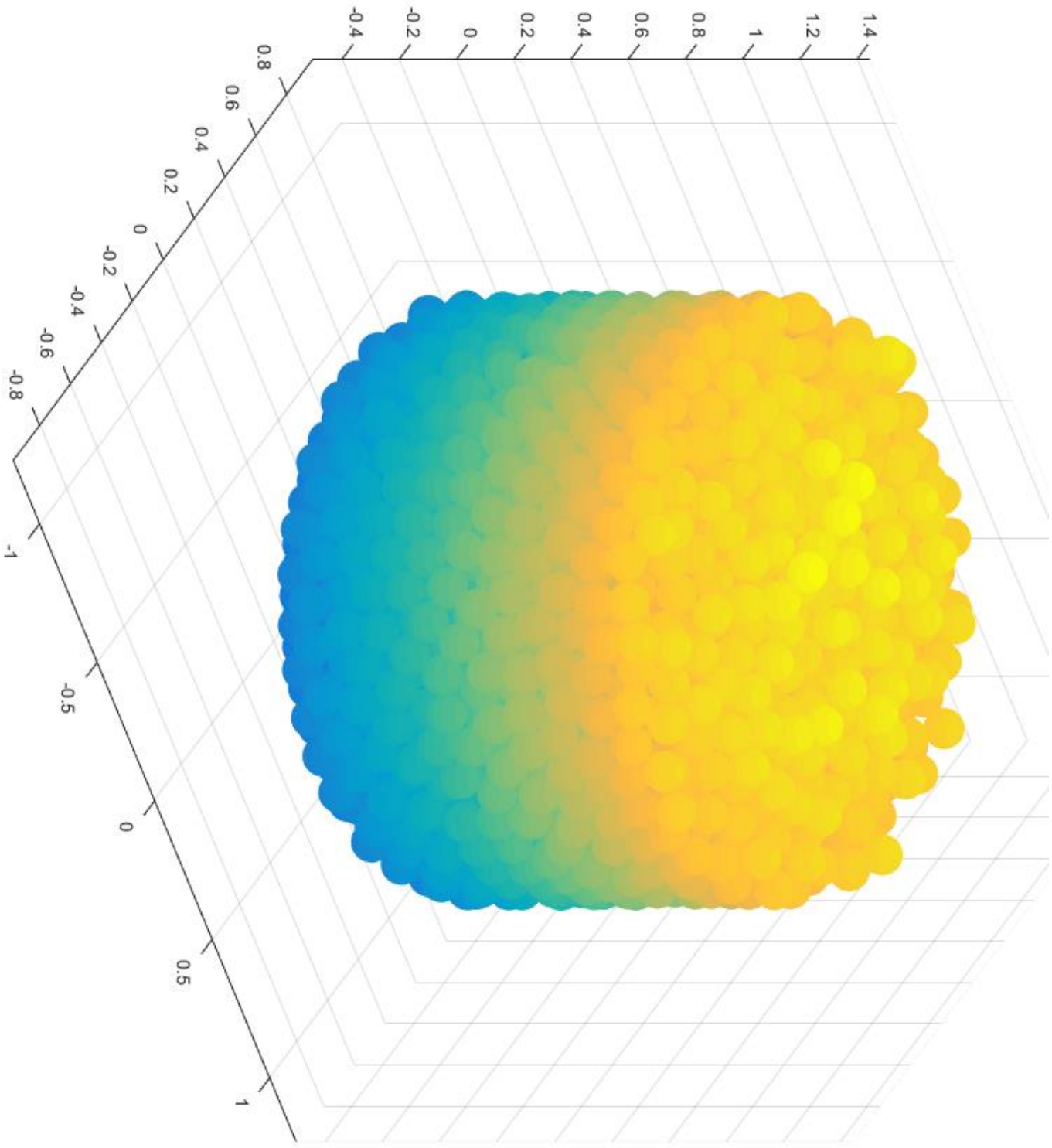

Figure 24 View of the created HTR-10 pebble bed with twice larger ball radius

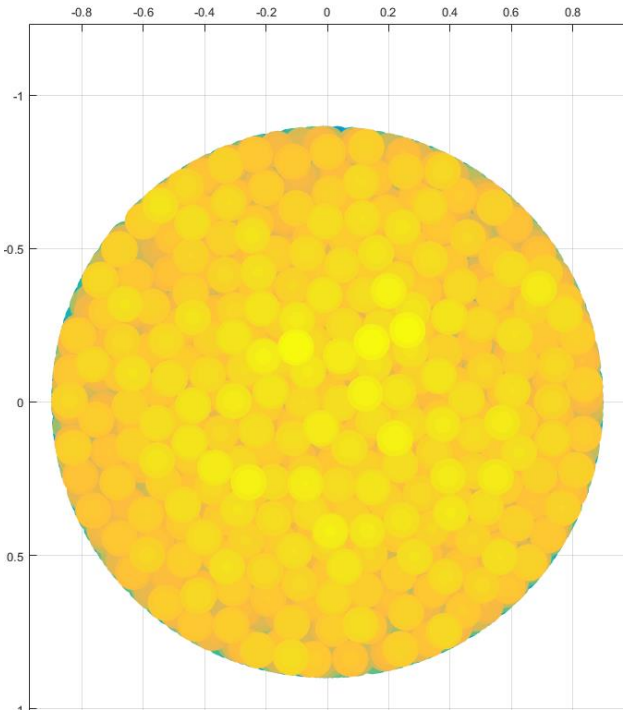

Figure 27 Plan view of the HTR-10 pebble bed with twice larger ball radius

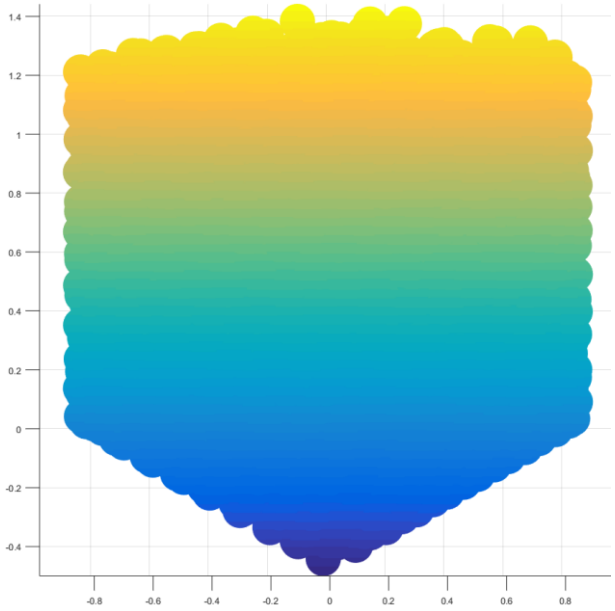

Figure 25 Side view of the HTR-10 pebble bed with twice larger ball radius

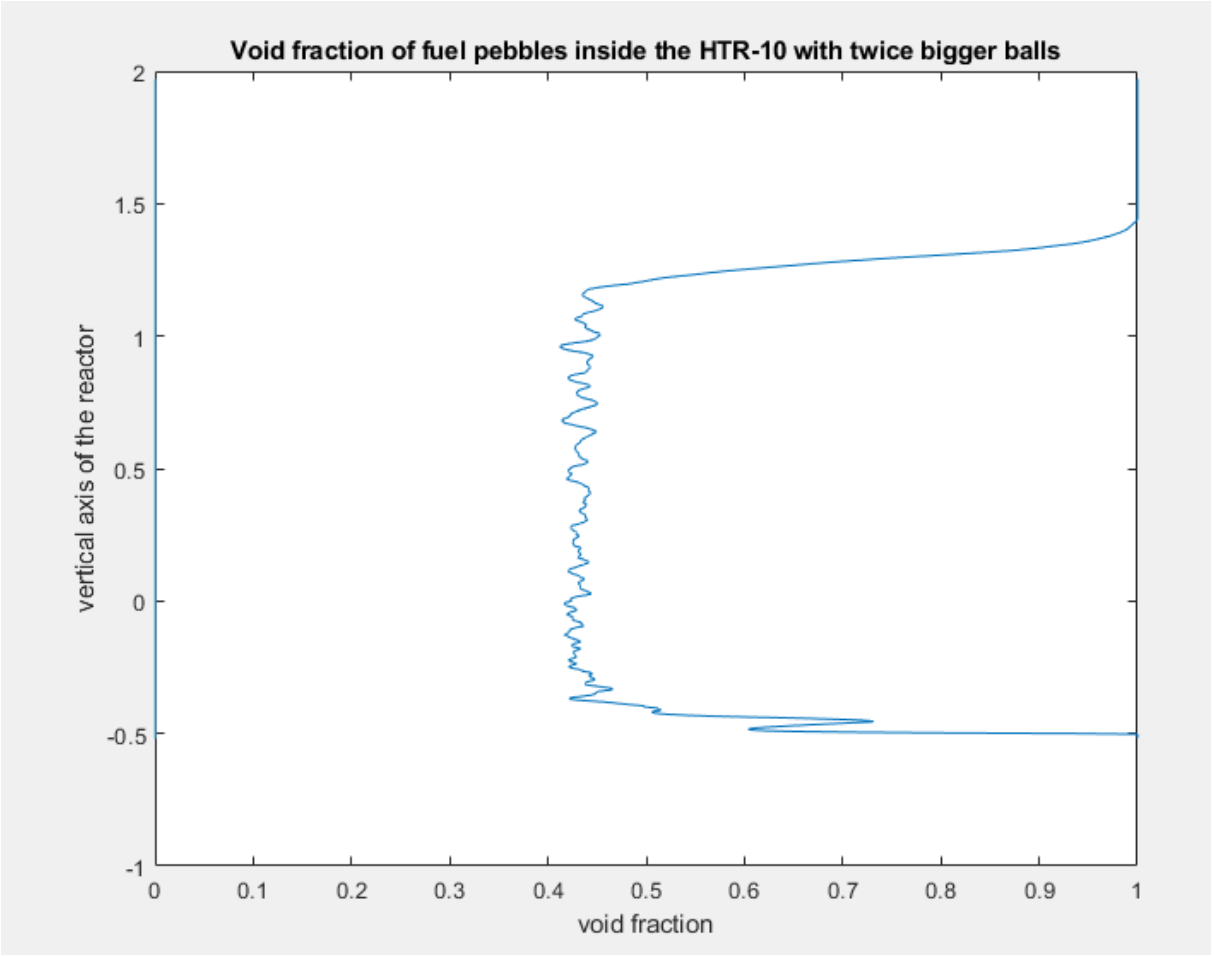

Figure 26 Void fraction of fuel pebbles inside the HTR-10 with twice bigger balls

6.2 Ball's size versus time

The first test is conducted to calculate time dependency on ball's radius. Reactor geometry was chosen to make trustful test – sufficient amount of balls for each case. Calculated maximum time step for each case is different. However, a single time step value was selected in each case for better result comparison. Cone height is relatively small due to maximize time step and thus minimize the calculation time. Above table contain geometry data for whole cases in the test.

Table 2 calculations properties for ball radius versus time case

Reactor Geometry		
Diameter	2	m
Cone Height	0,25	m
Reactor Height	1,5	m

It must be added that in each case is also the same mass of ball equal 1kg, which is the default value and the result does not depend on this variable. Next tables contain data as calculations properties, and main calculations result for each test in this case, as it was in the example.

Table 3 Calculation and result properties of ball radius test – (1) and (2)

Calculations Properties (1)		
Ball Radius	0,16	m
Rings in Layer	2	-
Balls in Layer	18	-
Time step <=	3,83E-02	s
Time Step	1,00E-02	s
Result: R = 0,16		
virtual time	24,89	s
real time	325	s
	5,4	min
balls created	126	-
Av balls per real min	23,23	1/min
Av balls per virtual s	5,06	1/s

Calculations Properties (2)		
Ball Radius	0,15	m
Rings in Layer	2	-
Balls in Layer	18	-
Time step <=	3,67E-02	s
Time Step	1,00E-02	s
Result: R = 0,15		
virtual time	27,62	s
real time	511	s
	8,5	min
balls created	144	-
Av balls per real min	16,90	1/min
Av balls per virtual s	5,21	1/s

Table 4 Calculation and result properties of ball radius test – (3) and (4)

Calculations Properties (3)		
Ball Radius	0,14	m
Rings in Layer	2	-
Balls in Layer	18	-
Time step <=	3,51E-02	s
Time Step	1,00E-02	s

Calculations Properties (4)		
Ball Radius	0,13	m
Rings in Layer	2	-
Balls in Layer	18	-
Time step <=	3,34E-02	s
Time Step	1,00E-02	s

Result: R = 0,14		
virtual time	35,75	s
real time	1026	s
	17,1	min
balls created	198	-
Av balls per real min	11,58	1/min
Av balls per virtual s	5,54	1/s

Result: R = 0,13		
virtual time	41,11	s
real time	2050	s
	34,2	min
balls created	252	-
Av balls per real min	7,37	1/min
Av balls per virtual s	6,13	1/s

Table 5 Calculation and result properties of ball radius test – (5) and (6)

Calculations Properties (5)		
Ball Radius	0,12	m
Rings in Layer	2	-
Balls in Layer	18	-
Time step <=	3,17E-02	s
Time Step	1,00E-02	s
Result: R = 0,12		
virtual time	49,01	s
real time	3575	s
	59,6	min
balls created	324	-
Av balls per real min	5,44	1/min
Av balls per virtual s	6,61	1/s

Calculations Properties (6)		
Ball Radius	0,11	m
Rings in Layer	3	-
Balls in Layer	36	-
Time step <=	2,99E-02	s
Time Step	1,00E-02	s
Result: R = 0,11		
virtual time	37,75	s
real time	5534	s
	92,2	min
balls created	432	-
Av balls per real min	4,68	1/min
Av balls per virtual s	11,44	1/s

Table 6 Calculation and result properties of ball radius test – (7) and (8)

Calculations Properties (8)		
Ball Radius	0,1	m
Rings in Layer	3	-
Balls in Layer	36	-
Time step <=	2,80E-02	s
Time Step	1,00E-02	s
Result: R = 0,10		
virtual time	43,15	s
real time	8856	s
	147,6	min
balls created	540	-
Av balls per real min	3,66	1/min
Av balls per virtual s	12,51	1/s

Calculations Properties (9)		
Ball Radius	0,09	m
Rings in Layer	4	-
Balls in Layer	60	-
Time step <=	2,60E-02	s
Time Step	1,00E-02	s
Result: R = 0,09		
virtual time	41,12	s
real time	22415	s
	373,6	min
balls created	840	-
Av balls per real min	2,25	1/min
Av balls per virtual s	20,43	1/s

Plots of the most important results are presented in Figure 29 and Figure 28. Additionally, on the second plot is added trend line, generated in MS Excel. The first figure does not contain a trend line, because result varies too much. It happens, because change a 1-dimensional ball radius cause a 3-dimensional change of a ball, even with the same time step value. It is a very complex dependency. In general, for a ball radius much smaller than the reactor diameter, virtual time is increasing with a ball radius size.

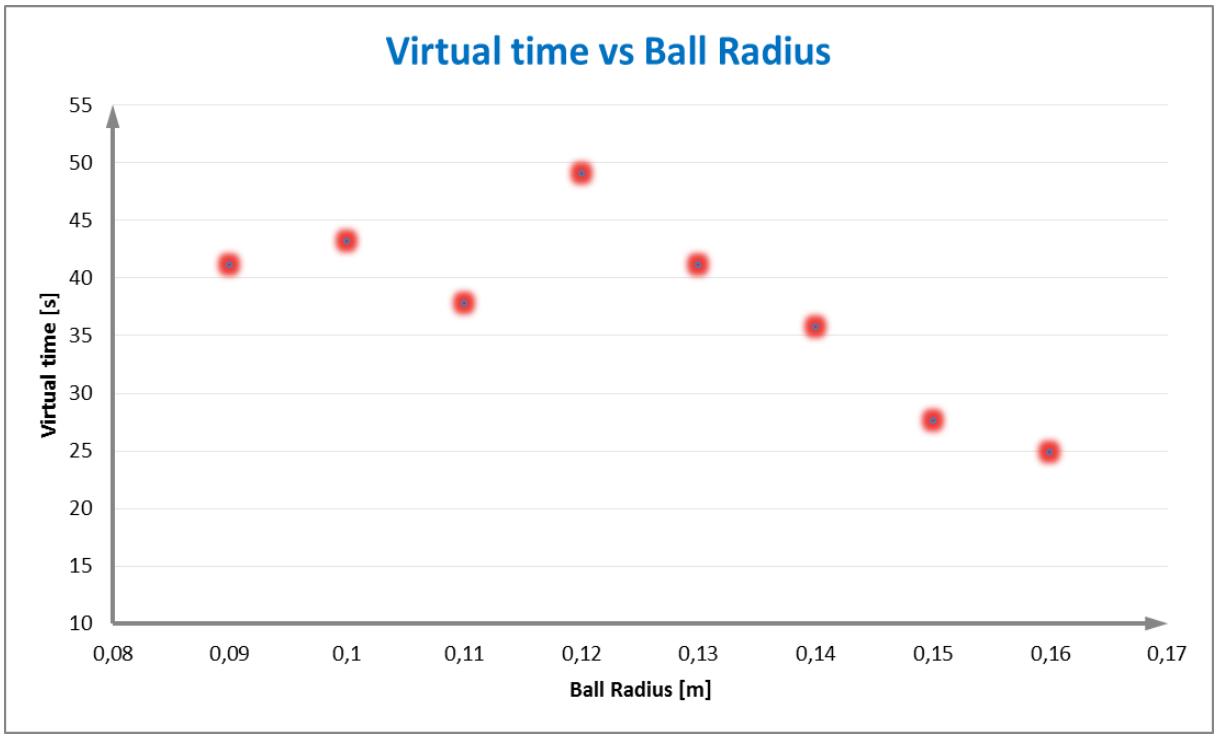

Figure 29 Virtual time versus ball radius

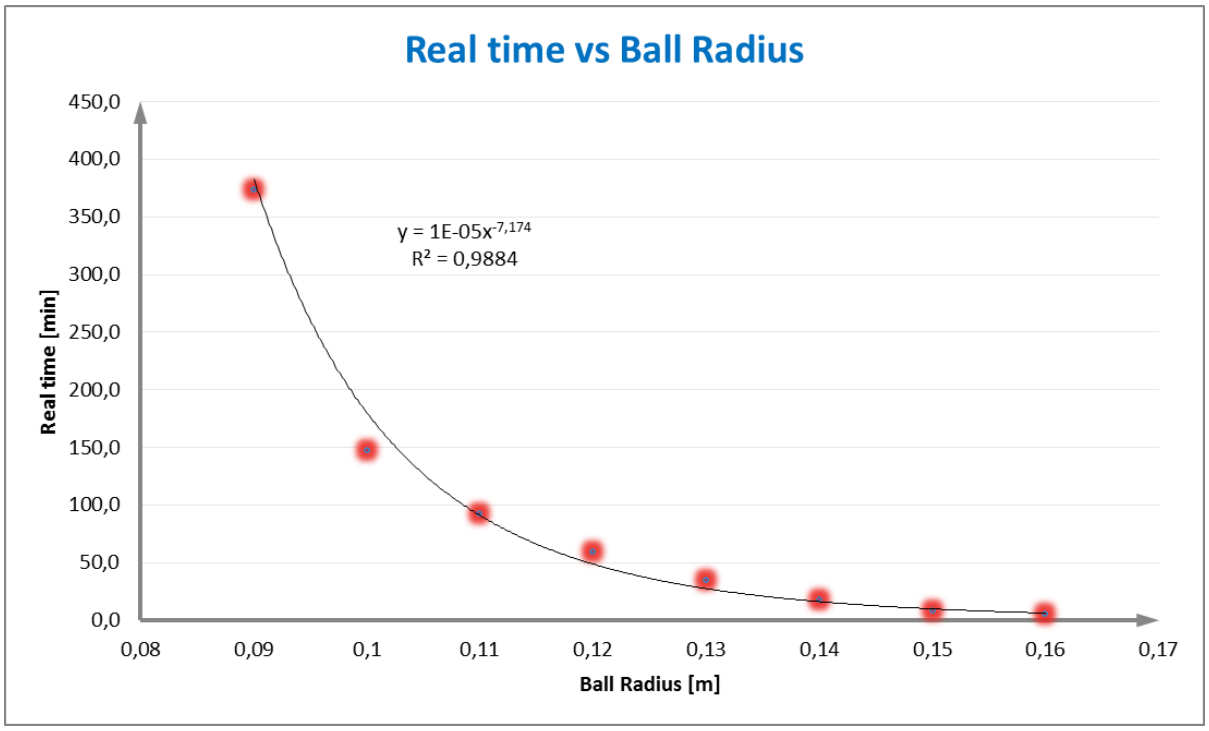

Figure 28 Real-time versus ball radius

On Figure 28 is presented real convergence time versus ball radius. The best fitting trend line is polynomial line with very high accuracy R^2 = 0.988, where real-time is the function of the ball size up to about seventh power. It is very close to right conclusion. The explanation for this is straightforward. Any change in ball radius cause number of balls changes up to the third power. An increase of the number of balls causes up to the second power more correlations between balls to check and nearly the same increase caused by the geometry collision checking (in tested range of ball radius). The increase caused by the geometry collision checking is nearly the same because the algorithm is making a*n operations (a is constant). However, these operations take more time than operations. According to this theory, calculation time is the function of the ball radius in the following form:

$$f(R) = (a * R^2 + b * R)^3$$

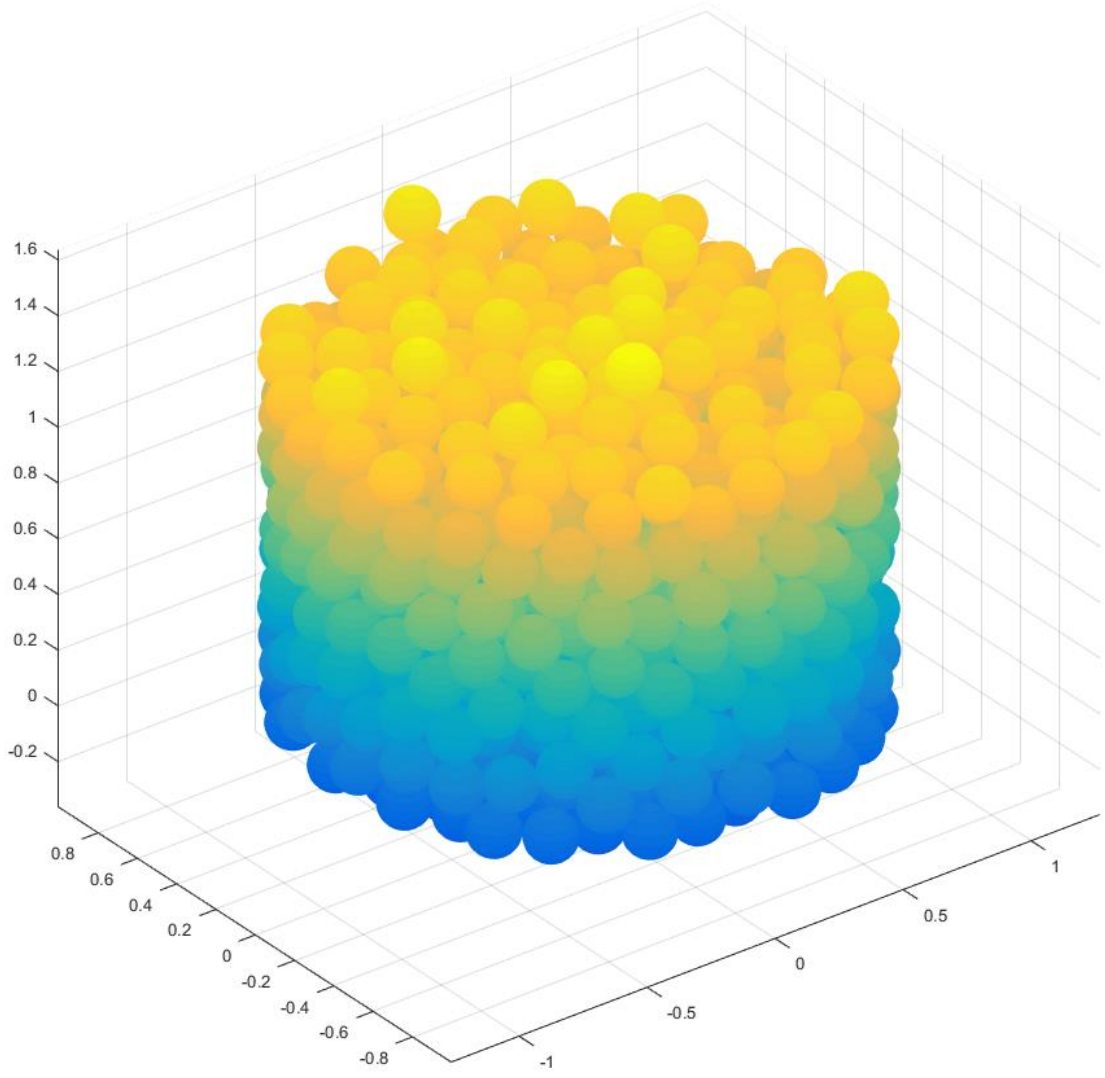

Figure 30 View of one of created pebble beds during ball radius test

6.3 Reactor height versus time

The last test is conducted to calculate time dependency on cylinder height parameter. The chosen ball radius is like the biggest one in 6.2 to minimalize calculation time for each case. Cone height is relatively small due to maximizing time step, and thus minimize the calculation time, same as in 6.2. An Above table (Table 7) contain data chosen for all cases in this test.

Table 7 calculations properties for reactor height versus time case

Reactor Geometry		
Diameter	2	m
Cone Height	0,25	m

Ball properties		
Mass	1	kg
Radius	0,16	m

Calculations Properties		
Rings in Layer	2	-
Balls in Layer	18	-
Time step <=	2,22E-02	s
Chosen Time Step	1,00E-02	s

The test was prosecuted for 8 cases, where the last one was a repetition test with 4 meters cylinder height. It was repeated because virtual time was lower than in the case with cylinder height equal 3.5 meters. Table 8 contains result data for conducted tests.

Table 8 The result of cylinder height test

Result: H = 1m		
virtual time	18,66	s
real time	114	s
	1,9	min
balls created	72	-
Av balls per real min	37,84	1/min
Av balls per virtual s	3,86	1/s

Result: H = 1,5m		
virtual time	24,89	s
real time	325	s
	5,4	min
balls created	126	-
Av balls per real min	23,23	1/min
Av balls per virtual s	5,06	1/s

Result: H = 2m		
virtual time	34,86	s
real time	828	s
	13,8	min
balls created	180	-
Av balls per real min	13,04	1/min
Av balls per virtual s	5,16	1/s

Result: H = 2,5m		
virtual time	52,72	s
real time	2372	s
	39,5	min
balls created	234	-
Av balls per real min	5,92	1/min
Av balls per virtual s	4,44	1/s

Result: H = 3m		
virtual time	48,71	s
real time	3452	s
	57,5	min
balls created	288	-
Av balls per real min	5,01	1/min
Av balls per virtual s	5,91	1/s

Result: H = 3,5m		
virtual time	58,76	s
real time	5504	s
	91,7	min
balls created	342	-
Av balls per real min	3,73	1/min
Av balls per virtual s	5,82	1/s

Result: H = 4m		
virtual time	58,53	s
real time	5773	s
	96,2	min
balls created	378	-
Av balls per real min	3,93	1/min
Av balls per virtual s	6,46	1/s

Result: H = 4m second		
virtual time	60,67	s
real time	6452	s
	107,5	min
balls created	378	-
Av balls per real min	3,52	1/min
Av balls per virtual s	6,23	1/s

The results of virtual time and real-time are presented in Figure 31 and the Figure 32. Moreover, on plots are added trend lines, generated in MS Excel. Due to randomization, each point has a relatively significant confidence interval. However, on the first figure trend line is a straight line as it could be predicted. It happens because virtual time is a sum of time steps, not iterations inside the script.

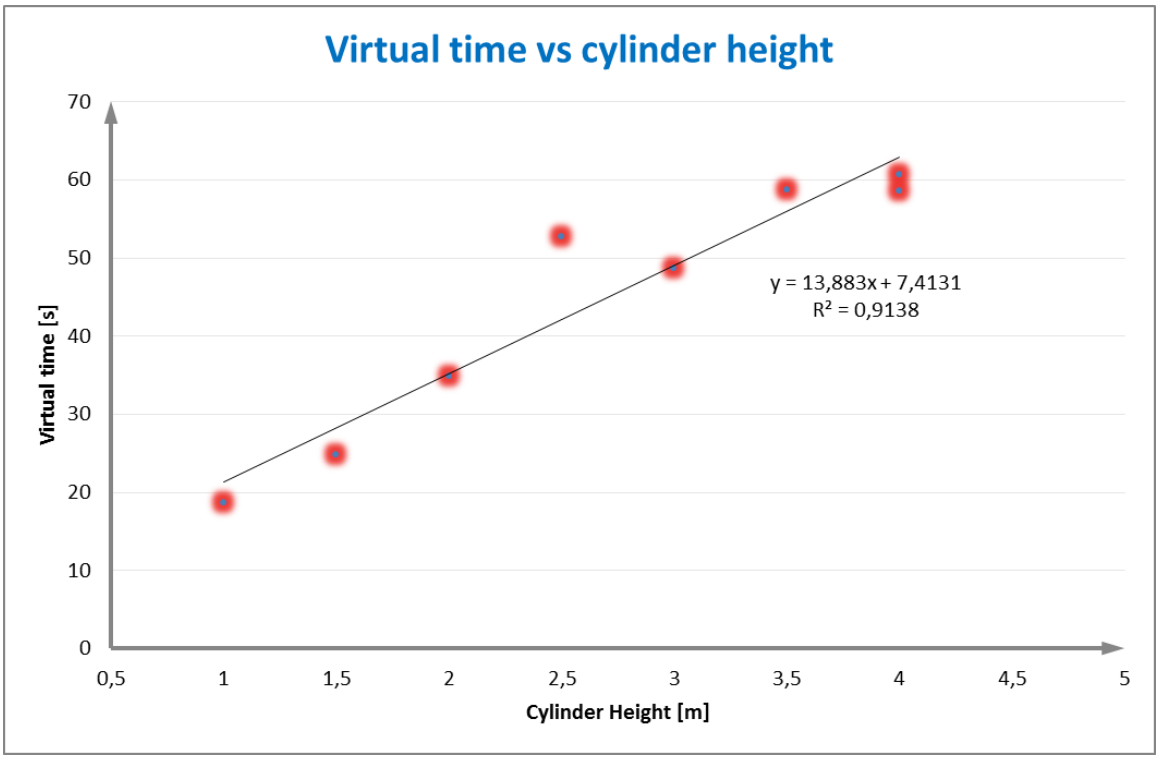

Figure 31 Virtual time versus cylinder height

On Figure 32 is presented real convergence time versus cylinder height. The best fitting trend line is a polynomial curve. The explanation is straightforward. For example, let's compare two reactors: one with height equals 1 meter and the second 2 meters. In higher one there are two times more balls. Because of that algorithm checking relationships between n balls, the algorithm must do $a*n^2$ operations, where a is a constant value. For small heights real time is a more exponential function (determination of the function equals 0.995), it is prompt by the existence of cone and checking for geometry collision.

The last figure (Figure 33) presented in this chapter is the result of created balls in one of 4-meter cylinder height, generated by [4].

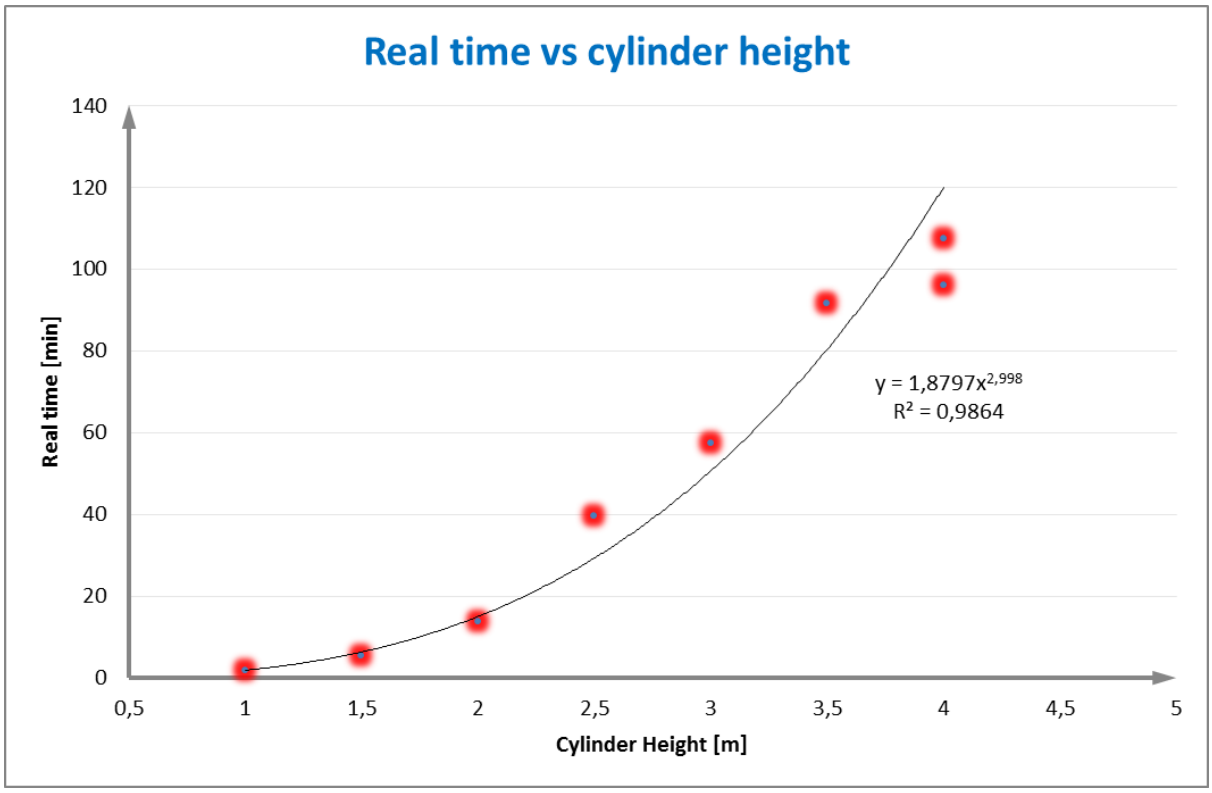

Figure 32 Real-time versus cylinder height

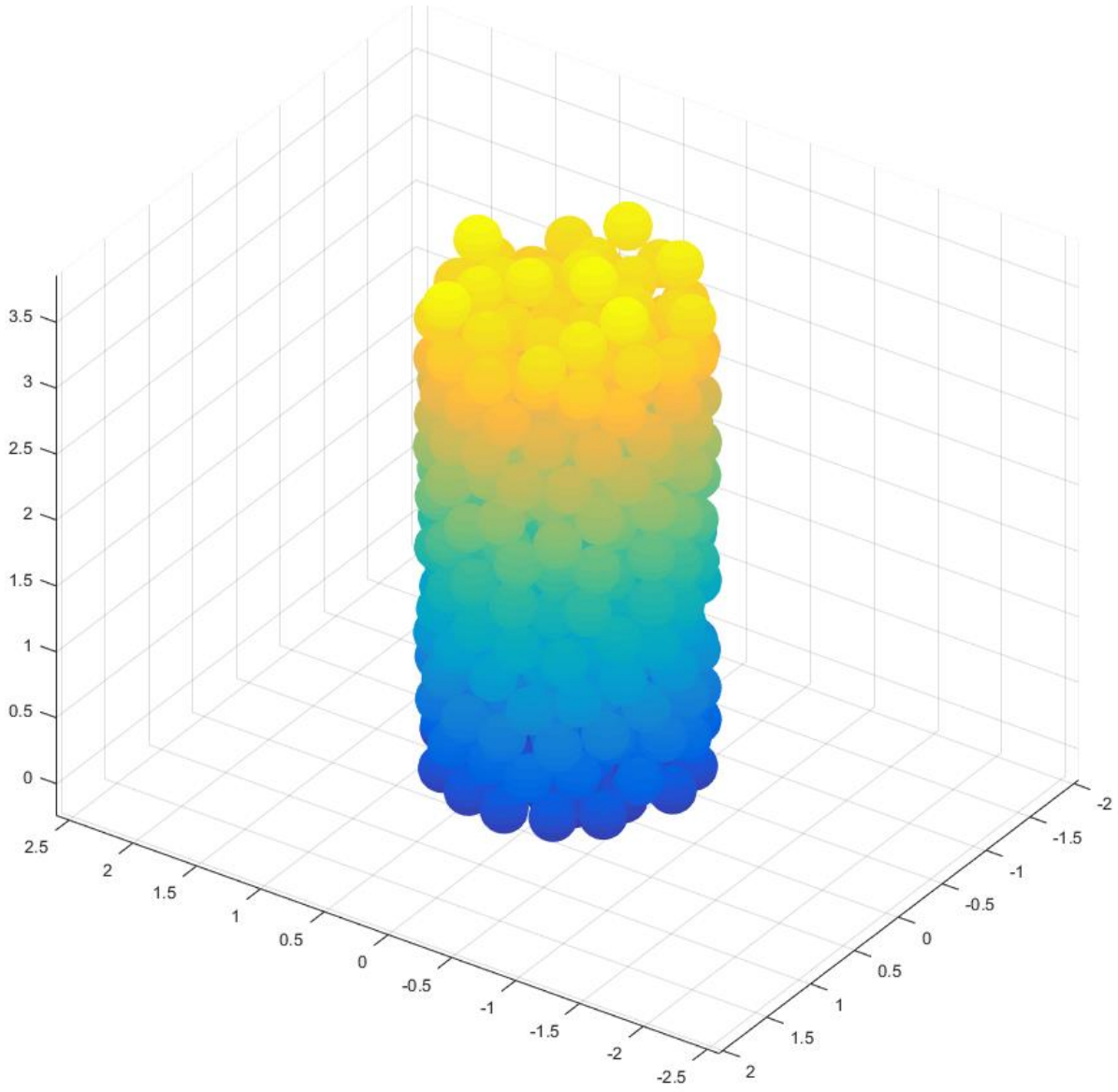

Figure 33 View of one of the created pebble beds during cylinder height test

6.4 HTR-10

Thanks to using a better computer – PC with processor intel core i7-8700, and script's correction, HTR-10 has been generated. The correction consisted in changing the way in which equations were written in MATLAB. Precisely the changes are in the way how program reads the data from the balls matrix. Thanks to the only correction, about one hundred faster calculation time was obtained. Moreover, speed improvement due to better processor was from four to five times. The total improvement was about five hundred times.

The data for the case is nearly the same as in the case from paragraph 6.1, except the cylinder height and the sphere radius, which is set as in real HTR-10 – 0.03 meters. The new reactor is also higher than previous one, because of probably mistake in Figure 23, taken from [3]. The cylinder height was set as 1.97 meters, as it can be deduced from tests in [3]. The simple result data was shown in Table 9.

Table 9 The result of HTR-10 case

Result HTR-10		
virtual time	~136	s
real time	~5	days
balls created	26208	-

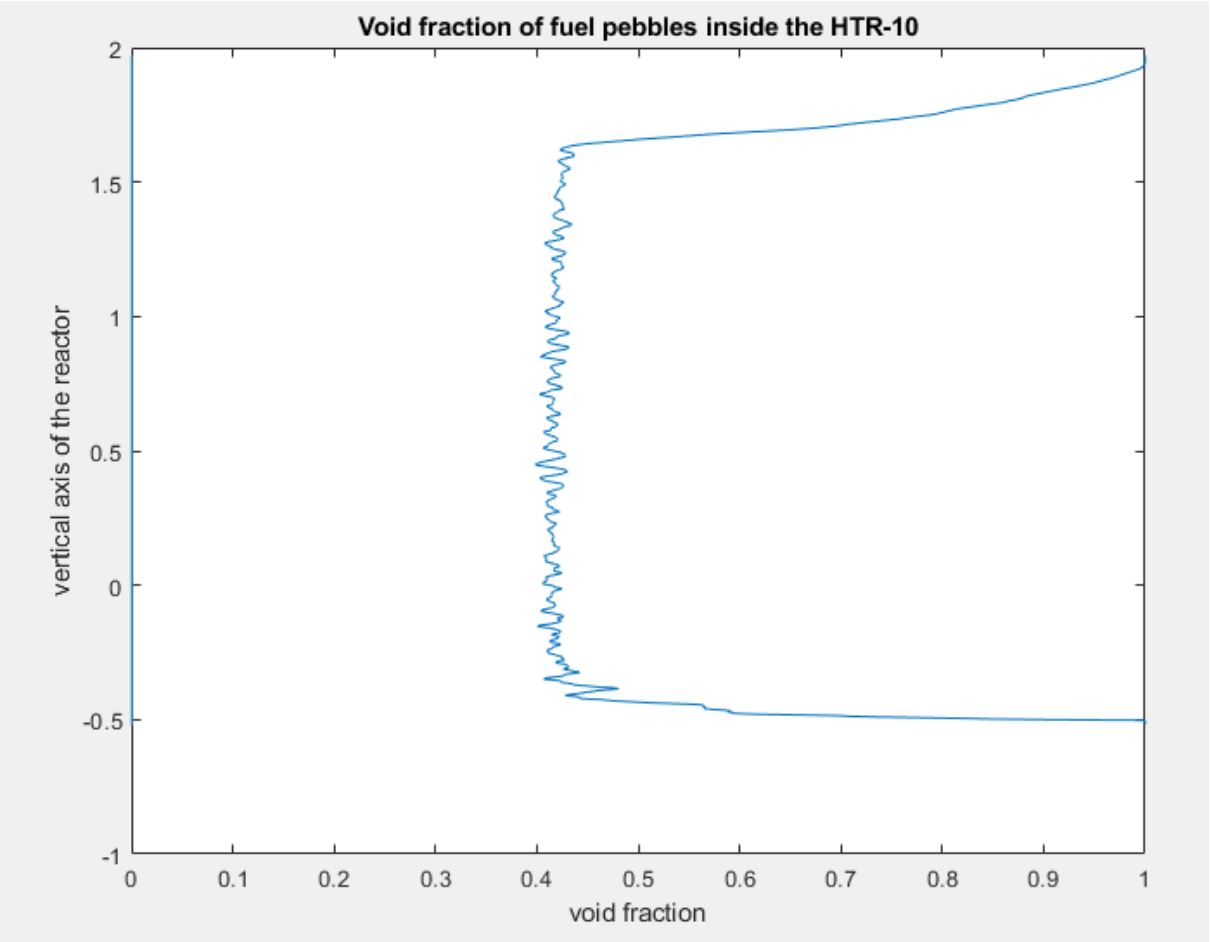

Figure 34 Void fraction of fuel pebbles inside the HTR-10

Pictures of created pebble bed were made by [4] and presented in Figure 35, Figure 36 and Figure 37. Moreover, void fraction of the reactor was calculated as the function through the vertical axis and presented in Figure 34.

The result is very similar to [3]. A number of balls are nearly the same. The difference is equal to 792. It is caused by the fact that thesis program creates pebble bed until the highest ball has not height equal to the height of the reactor minus two ball's radius. Additionally, the small difference occurred in a void fraction. It is caused by implemented stiffness of balls in the model used in [3].

Both programs: the old one and the improved one will be published on authors GitHub profile: "Klukov". The balls matrix containing all balls positions in the reactor will also be published there.

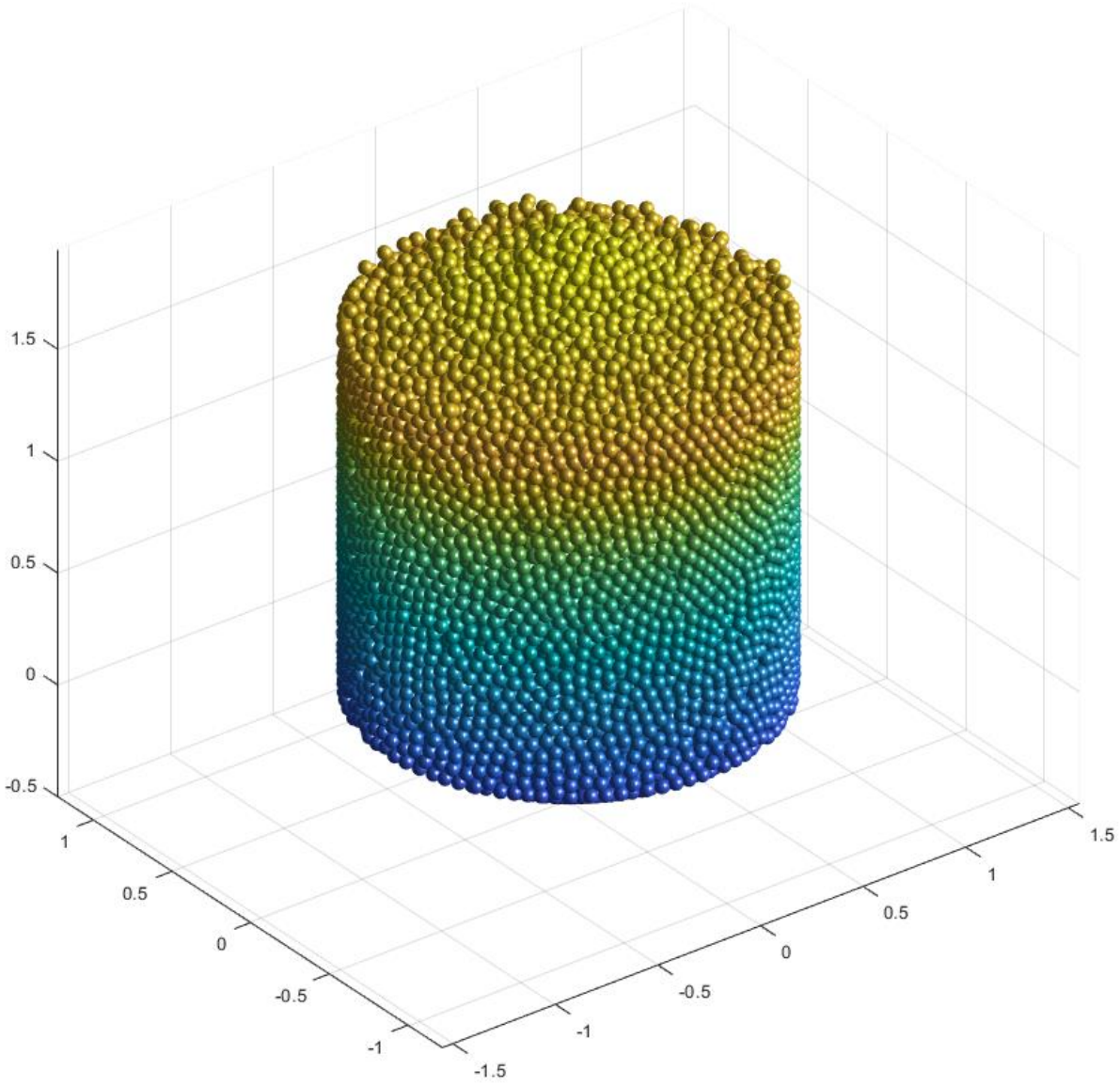

Figure 35 View of the created HTR-10 pebble bed

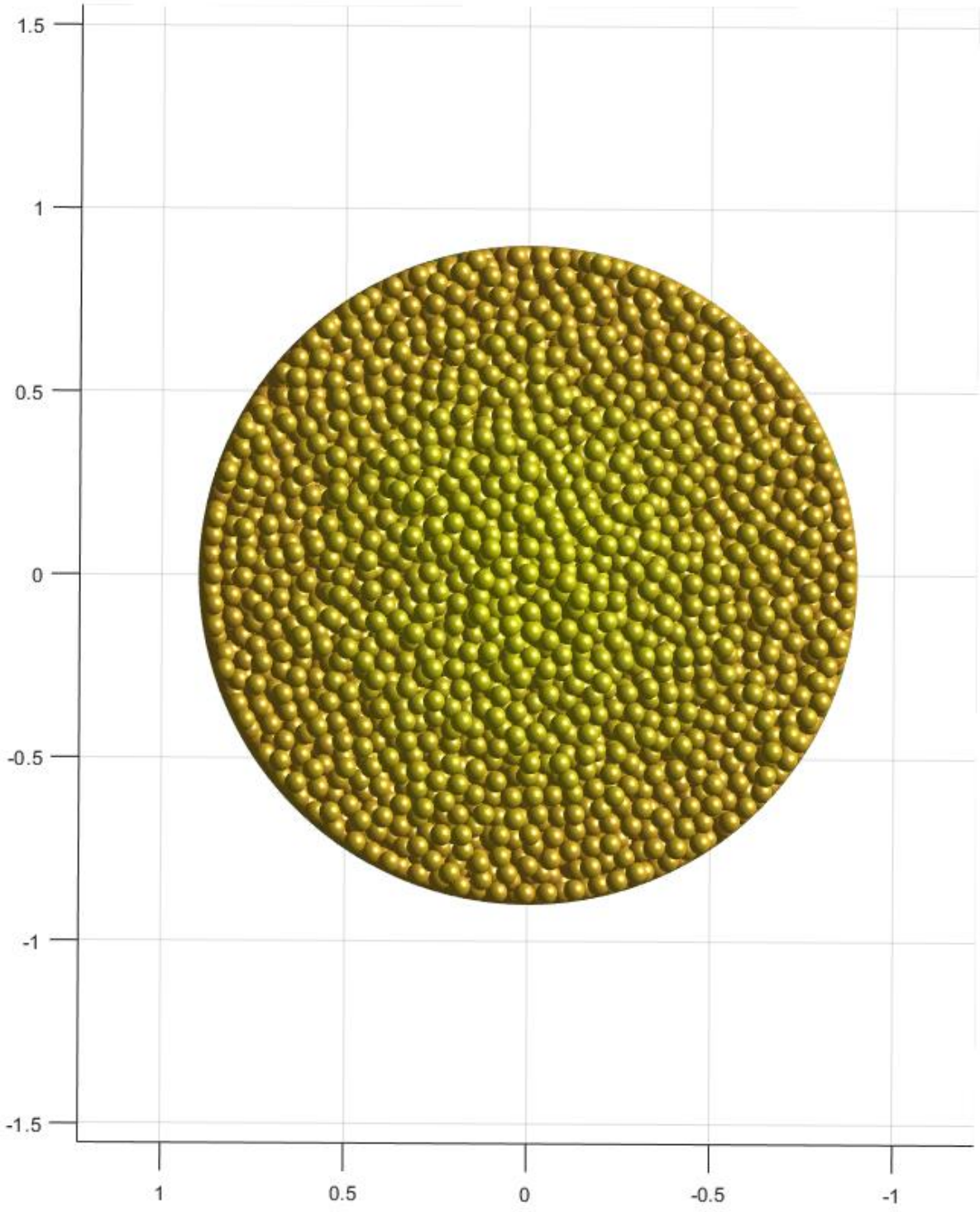

Figure 36 Plan view of the created HTR-10 pebble bed

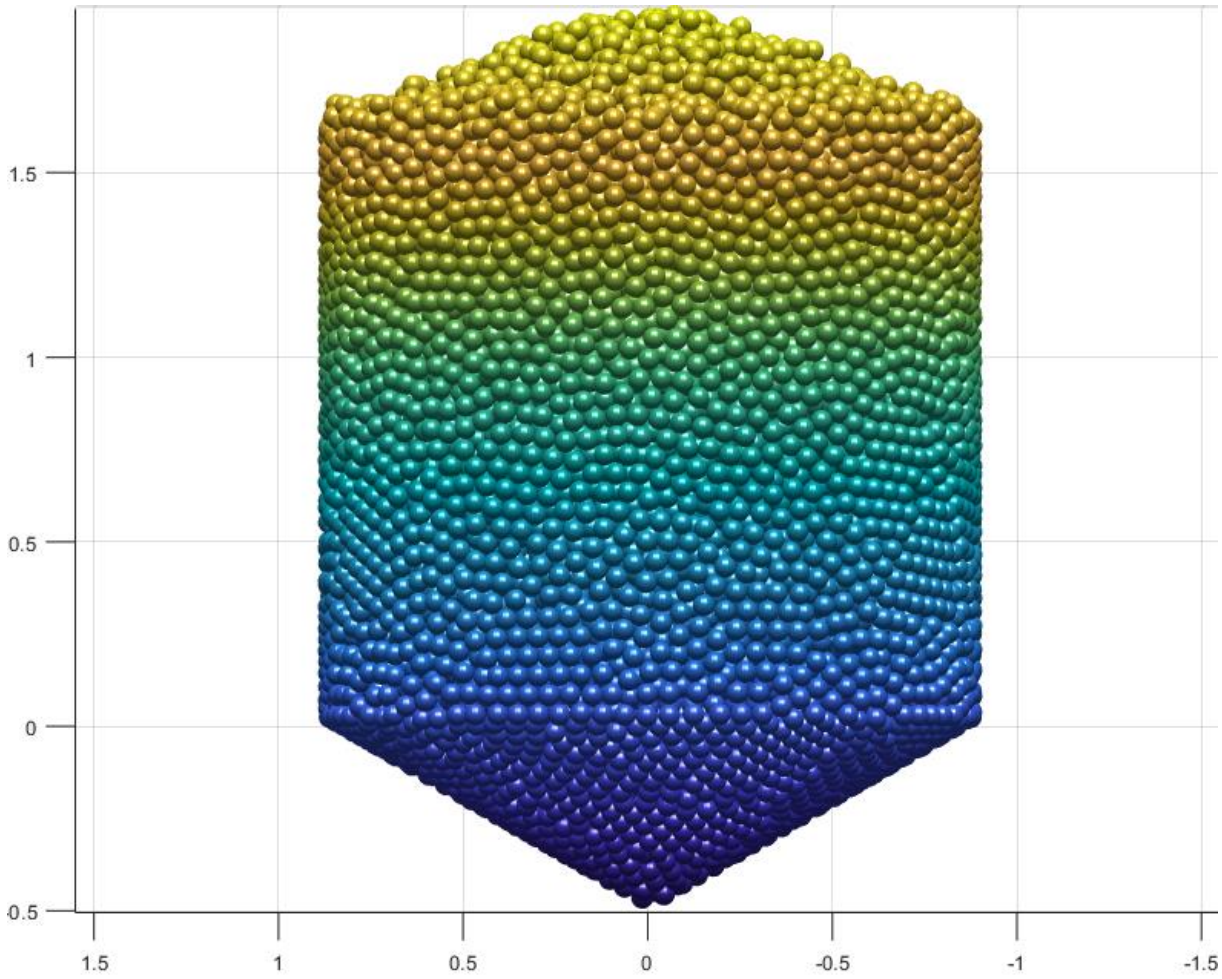

Figure 37 Side view of the created HTR-10 pebble bed

7 Possible improvements

7.1 A few layers calculation

As it was mentioned in 3.3.3 one of the best improvement is implementing a few layers calculation into the thesis. It can be implemented thanks to solution chosen in 2.1.2, which makes partially created pebble bed no destructible.

A few layers calculation can be realized by implementing parameter *number of layers for collision checking*, which is entirely depended on a balls creation algorithm. In the current version of the code, this new feature should be higher than 3, preferably 5. More information about this parameter in 3.3.3.

Speed improvement is dependent on reactor diameter to ball diameter ratio. The more significant ratio is, the more significant magnitude of improvement is reached.

7.2 Improvement of creation algorithm

In chapter 4 was described the algorithm for creating a new layer of balls. It creates the layer which perfectly suits the cylindrical reactor. However, in this case, the reactor has cone-cylindrical shape. Due to this fact, a new layer of balls could be very distant from the top of the cone. Regarding paragraph 3.3.2, it can cause a problem of significant *time_step* value, conducting into much longer calculation time, even few times.

One of the ideas is creating at the beginning few layers with the cone shape. Of course, it has to be tested and modified to reduce maximum distant from ball to the cone surface.

7.3 Automatically time step calculation

This simple function would be beneficial in the future. The user will not worry about proper *time_step* calculation. Now it is a little problem because if *time_step* is not correctly set and creation of pebble bed taking about 24h (very small balls), the user loses all day on waiting for an inconsistent solution or get an error.

The script which would realize this task is dependent on creation algorithm. Any change in creation script cause change of *time_step* calculation.

8 Conclusions

Developing algorithm for balls distribution in pebble bed reactor accomplished with full success. The program makes faultless, randomized pebble bed. Many tests were developed, in all of them did not occur any expected or unexpected error. The shown example and conducted tests are the proof of proper working of the algorithm. Also, a secondary goal was achieved. The HTR-10 pebble bed was generated, which will be used for Monte Carlo simulations for neutronics in Serpent environment.

Additionally, the correctness of the solution generated by algorithm proofs that model is correct and can be used in other applications, for example, HTR simulations. After little improvement, the program is now able to generate HTR pebble beds for academic purposes is not relatively long time (5 days). The important thing is that the improvement does not affect on relative values of calculation time obtained during tests.

The principal not mentioned advantage of this solution is a possibility of applying the algorithm in HTR transient states simulator. Where on each balls movement iteration would be calculated k_{eff} by Monte Carlo method.

Therefore, the reader may conclude that algorithm is not perfect, because five days for generating HTR pebble bed is not satisfying. However, computer program proofs that mathematical model is entirely correct, which is most important. The only problem is calculation time and, in this field, could be done with some mentioned improvements.

On the other hand, the time in pebble bed generation is not the primary target. The target is correctness. If the obtained solution is like the other ones, the results obtained from the program are correct. So that future users can generate different types of the pebble beds, and them could be sure that the result is correct.

The whole code and future done upgrades after thesis publication will be published on author's GitHub profile: "Klukov”.

Appendix A – main loop

```
%% HTR Pebble Bed reactor creator
% balls bouncing model
% by Piotr Klukowski, Warsaw Univesity of Technology
% metric in meters
% time in seconds
% weight in kg

%% inicialize world
%main features
BallRadius = 50*10^(-3);
BallMass = 1; % mean value
TimeStep = 10^(-3);
Gravity = 9.81;
EnergyDissipation = [0.7 0.9 ; 0.6 0.8];
% dissipation: first row consider dissipation rate of collisions between
% balls, and the second collisions with reactor walls. First column is
% connected to normal vector and the second to perpendicular vector

%geometry
ReactorDiameter = 2;
CylinderHeight = 0.5;
ConeHeight = 0.25;
geometry = [ReactorDiameter;CylinderHeight;ConeHeight];
if DataCheck(BallRadius, BallMass, TimeStep, geometry, Gravity) ~= 1
    error('Error with DATA !!!');
end

%% Algoritm initialization
time = 0;
creation_completed = false;
Balls = [];
Balls = create_balls(Balls, BallRadius, BallMass, time, geometry, Gravity);
LayerSize = size(Balls,1);
quantity_of_balls = size(Balls,1);
%scatter3(Balls(:,1),Balls(:,2), Balls(:,3))

```

```
36
37      % convergence factors
38 -    Energy_Dissipation_ratio = (LayerSize-2)/LayerSize;
39 -    Energy_rest = 10^(-3);
40 -    MaxEnergy = max(abs(Balls(:,9)));
41 -    AvEnergy = mean(abs(Balls(:,9)));
42 -    disp('time: ');
43 -    disp(time);
44
45 -    fprintf('Program paused. Press enter to continue.\n');
46 -    pause;
47
48      %% LOOP
49 -    while (creation_completed ~= true)
50 -        time = time + TimeStep;
51 -        if ((MaxEnergy < BallMass * Gravity * TimeStep) && (AvEnergy < ...
52                  BallMass * Gravity * TimeStep * Energy_Dissipation_ratio) )
53 -            if (max(Balls(:,3)) < (CylinderHeight-2*BallRadius))
54 -                [Balls] = create_balls(Balls, BallRadius, BallMass, ...
55                      time, geometry, Gravity);
56 -            else
57 -                if (AvEnergy < BallMass * Gravity * TimeStep * ...
58                          Energy_rest * Energy_Dissipation_ratio)
59 -                    creation_completed = true;
60 -                    break;
61 -                end
62 -            end
63 -        end
64 -        Balls(:,6) = Balls(:,6) - Gravity*TimeStep;
65 -        Balls = BoucingBalls(Balls, TimeStep, geometry, EnergyDissipation);
66 -        MaxEnergy = max(abs(Balls(:,9)));
67 -        AvEnergy = mean(abs(Balls((end-LayerSize+1):end,9)));
68 -        disp(' ');
69 -        Display = ['time: ', num2str(time)];
70 -        disp(Display);
71 -    end
72 -    fprintf ('Total time: %d \n\n', time);
73 -    fprintf ('Pebble Bed created !!! \n\n');
74
```

Appendix B – ball's creation

```
function New_Balls = create_balls(Balls, BallRadius, BallMass, ...
    time, geometry, Gravity)
% ball creation algorithm
% this will be first algorithm of ball creation
% Balls is a matrix num_Balls x [x,y,z,Vx,Vy,Vz,R,M,E]

radius_ratio = 1.2;
height_variance = 4; %times radius
Rcircle = radius_ratio * BallRadius; %initial value
D = geometry(1);

%additional layers, except centre
num_layers = fix((D/2-Rcircle)/(2*Rcircle));
if num_layers == 0
    error('Problem with BallCreation #1');
end

Rcircle = D/4/(num_layers+1); %new
Temporary_Balls = zeros(3*(num_layers+1)*num_layers, 9);
RadiusVariation = Rcircle - BallRadius;

%checking z0
if time == 0
    z0 = BallRadius;
else
    height = max(Balls(:,3)) + 2*BallRadius;
    if height < BallRadius
        z0 = BallRadius;
    else
        z0 = height;
    end
end

```

```
%create balls in circle
ball_number = 1;
for i = 1:num_layers
    balls_in_layer = i*6;
    twist_of_balls = rand()*2*pi;
    for j=1:balls_in_layer
        x0 = (2*Rcircle*i)*cos(2*pi/balls_in_layer*(j-1) + twist_of_balls);
        y0 = (2*Rcircle*i)*sin(2*pi/balls_in_layer*(j-1) + twist_of_balls);
        module = 0.9 * RadiusVariation*(rand()*2 - 1);
        %randim position in circle
        phi = 2*pi*rand();
        x = x0 + module*cos(phi);
        y = y0 + module*sin(phi);
        z = z0 + height_variance*BallRadius*rand();
        %setting up Velocity
        Velocity_randomization_factor = 0.5;
        Vz = (-1)*((2*Gravity*(2*BallRadius))^(1/2));
        phi = 2*pi*rand();
        if x ~= 0
            Vx = abs(Vz)*Velocity_randomization_factor*cos(phi);
        else
            Vx = 0;
        end
        if y ~= 0
            Vy = abs(Vz)*Velocity_randomization_factor*sin(phi);
        else
            Vy = 0;
        end
        %ball creation
        Radius = create_ball_radius(BallRadius, 1);
        Mass = create_ball_mass(BallMass, BallRadius, Radius);
        Energy = (1+2*Velocity_randomization_factor.^2).*...
            (Vz.^2)./2 .* Mass;
        Temporary_Balls(ball_number,:) = ...
            [x y z Vx Vy Vz Radius Mass Energy];
        ball_number = ball_number + 1;
    end
end

New_Balls = [Balls ; Temporary_Balls];

```

Appendix C – bounce check

```
function Balls_new = BoucingBalls...
    (Balls, TimeStep, geometry, EnergyDissipation)
% geometry = [ReactorDiameter; CylinderHeight; ConeHeight]
% Balls is a matrix num_Balls x [x,y,z,Vx,Vy,Vz,R,M,E]
m = size(Balls,1);
Balls_new = Balls;  %zeros(size(Balls))
number_of_collisions = 1;
a = geometry(1)/geometry(3)/2;
delta_z_ratio = 2*((geometry(3).^2 + ...
    (geometry(1).^2)/4).^(1/2))/geometry(1);
iteration = 0;

while number_of_collisions ~= 0
    %% Caclulate Balls position after TimeStep
    Balls_new(:,1) = Balls(:,1) + Balls(:,4) .* TimeStep;
    Balls_new(:,2) = Balls(:,2) + Balls(:,5) .* TimeStep;
    Balls_new(:,3) = Balls(:,3) + Balls(:,6) .* TimeStep;

    %% check collisions with geometry
    %geometry = [ReactorDiameter; CylinderHeight; ConeHeight]
    geometry_bounces = zeros(m,1);

    for i = 1:m
        %check bounce with cylinder and cone
        if ((((((Balls_new(i,1)).^2) + ((Balls_new(i,2)).^2)).^(1/2)) >=...
                (geometry(1)/2 - Balls_new(i,7))) || ...
        ((((Balls_new(i,3) + geometry(3) - ...
        Balls_new(i,7).*delta_z_ratio).^2).*((a).^2)) <= ...
        (((Balls_new(i,1)).^2) + ((Balls_new(i,2)).^2))) || ...
        ((Balls_new(i,3) + geometry(3) - ...
        Balls_new(i,7).*delta_z_ratio) <= 0))
            geometry_bounces (i,1) = 1;
            Balls_new(i,:) = calculate_geometry_collision(Balls(i,:), ...
                Balls_new(i,:), geometry, EnergyDissipation(2,:));
        end
    end

```

Appendix D – ball's bounce

```
function [Ball1_new, Ball2_new] = calculate_balls_collision...
    (Ball1, Ball2, EnergyDissipation)
%each parameter is an array 1x9
%Balls is a matrix num_Balls x [x,y,z,Vx,Vy,Vz,R,M,E]
Ball1 = Ball1';
Ball2 = Ball2';
EnergyDissipation = EnergyDissipation';
Ball1_new = Ball1;
Ball2_new = Ball2;

normal_vector = [(Ball1(1)-Ball2(1));(Ball1(2)-Ball2(2));...
    (Ball1(3)-Ball2(3))];
m1 = Ball1(8);
m2 = Ball2(8);
p1 = Ball1(4:6).*m1;
p2 = Ball2(4:6).*m2;
p1n = (dot(p1,normal_vector))/(norm(normal_vector)^2)*normal_vector;
p2n = (dot(p2,normal_vector))/(norm(normal_vector)^2)*normal_vector;
p1p = p1 - p1n;
p2p = p2 - p2n;
module_p1n = norm(p1n);
module_p2n = norm(p2n);
%%
%{...%}
%%
module_p1n_new = module_p1n;
module_p2n_new = module_p2n;
if norm(p1n) == 0
    p1n_new = p1n * (-1);
else
    p1n_new = p1n * (-1) * module_p1n_new / norm(p1n);
end
if norm(p2n) == 0
    p2n_new = p2n * (-1);
else
    p2n_new = p2n * (-1) * module_p2n_new / norm(p2n);
end
p1_new = p1n_new * EnergyDissipation(1) + p1p * EnergyDissipation(2);
p2_new = p2n_new * EnergyDissipation(1) + p2p * EnergyDissipation(2);
p1_new = vector_randomization(p1_new);
p2_new = vector_randomization(p2_new);
Ball1_new(4:6) = p1_new ./ m1;
Ball2_new(4:6) = p2_new ./ m2;
Ball1_new = Ball1_new';
Ball2_new = Ball2_new';
```

Appendix E – geometry collision

```
function Ball_new = calculate_geometry_collision...
    (Ball, FakeNewBall, geometry, EnergyDissipation)
% each parameter is an array 1x9
% geometry is an vector 3 x 1
% geometry = [ReactorDiameter; CylinderHeight; ConeHeight]
% Balls is a matrix num_Balls x [x,y,z,Vx,Vy,Vz,R,M,E]

%% We looking for new velocity vector - ONLY!
Ball = ((Ball)');
Ball_new = Ball;
FakeNewBall = FakeNewBall';
EnergyDissipation = EnergyDissipation';

R = Ball(7);
a = geometry(1)./geometry(3)./2;
delta_z_ratio = 2*((geometry(3).^2 + ...
    (geometry(1).^2)/4).^(1/2))./geometry(1);
geometry_bend_z = (geometry(1)./2 - R)/a - ...
    geometry(3) + delta_z_ratio .* R;

% normal vector calculation with normalization
if ( ((Ball(3)) > (geometry_bend_z)) && ...
        ((FakeNewBall(3)) > (geometry_bend_z)) )
    %bounce with cylinder
    normal_vector = ([Ball(1) Ball(2) 0]);
    normal_vector = normal_vector ./ norm(normal_vector);
elseif ( ( ((FakeNewBall(1).^2 + FakeNewBall(2).^2).^(1/2)) < ...
        (geometry(1)/2 - R) ) || ( ((Ball(3)) < (geometry_bend_z)) && ...
        ((FakeNewBall(3)) < (geometry_bend_z)) ) )
    %bounce with cone
    if ((Ball(1) == 0 && Ball(2) == 0))
        %only z velocity - different gradient
        normal_vector = [0 0 -1];
    else
        z = (Ball(1).^2 + Ball(2).^ 2).^(1/2).*(1/a); % + geometry(3)
        grad_z = (-1).*a.*a.*z;
        normal_vector = ([Ball(1) Ball(2) grad_z]);
        normal_vector = normal_vector ./ norm(normal_vector);
    end
else
    %bounce with cylinder or cone
```

```
41        %bounce with cylinder or cone
42 -      vect = FakeNewBall(1:3) - Ball(1:3);
43 -      qe_factors = [(vect(1).^2 + vect(2).^2);...
44            (2*(Ball(1).*vect(1) + Ball(2).*vect(2)) );...
45            (Ball(1).^2 + Ball(2).^2 - (geometry(1)/2-R).^2)];
46 -      qe_delta = qe_factors(2)^2 - 4.*qe_factors(1).*qe_factors(3);
47
48        %caclulating cross_z
49 -      if qe_delta < 0
50 -          error('Error with geometry collision #1');
51 -      elseif qe_delta == 0
52 -          line_factor = (-1)*qe_factors(2)/2/qe_factors(1);
53 -          if ((line_factor > 1) || (line_factor < 0))
54 -              error('Error with geometry collision #2');
55 -          end
56 -          cross_z = Ball(3) + vect(3)*line_factor;
57 -      else %qe_delta > 0
58 -          line_factor = ((-1)*qe_factors(2) + ...
59                qe_delta^(1/2))/2/qe_factors(1);
60 -          if ((line_factor > 1) || (line_factor < 0))
61 -              line_factor = ((-1)*qe_factors(2) - ...
62                    qe_delta^(1/2))/2/qe_factors(1);
63 -              if ((line_factor > 1) || (line_factor < 0))
64 -                  error('Error with geometry collision #3');
65 -              end
66 -          end
67 -          cross_z = Ball(3) + vect(3).*line_factor;
68 -      end
69
70        %checking cross_z
71 -      if cross_z > geometry_bend_z
72            % bounce with cylinder
73 -          normal_vector = ([Ball(1) Ball(2) 0]);
74 -          normal_vector = normal_vector ./ norm(normal_vector);
75 -      elseif cross_z < geometry_bend_z
76            % bounce with cone
77 -          if ((Ball(1) == 0 && Ball(2) == 0) && ...
78                    (Ball(4) == 0 && Ball(5) == 0))
79                %only z velocity - different gradient
80 -              normal_vector = [0 0 -1];
81 -          else
82 -              z = (Ball(1).^2 + Ball(2).^2).^(1/2).*(1/a); % + geometry(3)
83 -              grad_z = (-1).*a.*a.*z;
84 -              normal_vector = ([Ball(1) Ball(2) grad_z]);
85 -              normal_vector = normal_vector ./ norm(normal_vector);
86 -          end
87 -      else
```

```
87 -        else
88              %bounce with both cylinder and cone
89              %cone normal vector + cylinder normal vector
90 -            cylinder_normal_vector = ([Ball(1) Ball(2) 0]);
91 -            cylinder_normal_vector = cylinder_normal_vector ./ ...
92                  norm(cylinder_normal_vector);
93 -            z = (Ball(1).^2 + Ball(2).^2).^(1/2).*(1/a); %+ geometry(3)
94 -            grad_z = (-1).*a.*a.*z;
95 -            cone_normal_vector = ([Ball(1) Ball(2) grad_z]);
96 -            cone_normal_vector = cone_normal_vector ./ ...
97                  norm(cone_normal_vector);
98 -            normal_vector = cylinder_normal_vector + cone_normal_vector;
99 -        end
100 -   end
101
102 -   normal_vector = normal_vector';
103 -   p = Ball(4:6).*Ball(8);
104 -   pn = normal_vector.*(dot(p,normal_vector))./(norm(normal_vector));
105 -   pp = p - pn;
106 -   pn_new = pn.*(-1) .* EnergyDissipation(1);
107 -   pp_new = pp .* EnergyDissipation(2);
108 -   p_new = pn_new + pp_new;
109 -   p_new = vector_randomization(p_new);
110 -   Ball_new(4:6) = p_new./Ball(8);
111 -   Ball_new = Ball_new';
112
```

List of drawings

List of tables

Bibliography

[1] "Mathworks," [Online]. Available: https://www.mathworks.com/products/matlab.html.

[2] C. Kurniawan, "MATLAB Answers - circles packed with circles," [Online]. Available: https://www.mathworks.com/matlabcentral/answers/24614-circle-packed-with-circles.

[3] G. L. G. N. Y. X. T. J. J. S. Yin Xiong, "Effect of pebble size and bed dimension on the distribution of voidages in pebble bed reactor".

[4] A. Deoras, "Mathworks - Bubbleplot - Multidimensional scatter plots," [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/48005-bubbleplot-multidimensional-scatter-plots.

[5] "Atomic-Energy.ru," [Online]. Available: http://www.atomic-energy.ru/news/2012/10/09/36571.

[6] G. Strang, Introduction to Linear Algebra 4th edition.

[7] G. Strang, Linear algebra and its applications.

[8] "Pebble bed reactor - wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Pebble-bed_reactor.

[9] IAEA, "Advances in Small Modular Reactor Technology Developments".