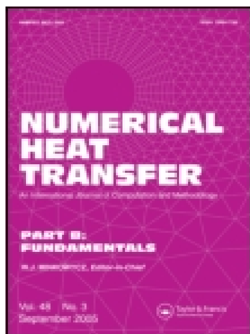

Taylor & Francis
Taylor & Francis Group

Numerical Heat Transfer, Part B: Fundamentals
An International Journal of Computation and Methodology

ISSN: 1040-7790 (Print) 1521-0626 (Online) Journal homepage: http://www.tandfonline.com/loi/unhb20

An Efficient CPU-GPU Implementation of the Multiple Absorption Coefficient Zonal Method (MACZM)

Boutros Ghannam , Maroun Nemer , Khalil El Khoury & Walter Yuen

To cite this article: Boutros Ghannam , Maroun Nemer , Khalil El Khoury & Walter Yuen (2012) An Efficient CPU-GPU Implementation of the Multiple Absorption Coefficient Zonal Method (MACZM), Numerical Heat Transfer, Part B: Fundamentals, 62:6, 439-461, DOI: 10.1080/10407790.2012.709418

To link to this article: http://dx.doi.org/10.1080/10407790.2012.709418

Published online: 09 Nov 2012.

Submit your article to this journal

Article views: 62

View related articles

Citing articles: 3 View citing articles

Full Terms & Conditions of access and use can be found at
http://www.tandfonline.com/action/journalInformation?journalCode=unhb20

Download by: [University of California Santa Barbara] **Date:** 07 January 2016, At: 18:24

Numerical Heat Transfer, Part B, 62: 439–461, 2012
Copyright © Taylor & Francis Group, LLC
ISSN: 1040-7790 print/1521-0626 online
DOI: 10.1080/10407790.2012.709418

Taylor & Francis
Taylor & Francis Group

Downloaded by [University of California Santa Barbara] at 18:24 07 January 2016

AN EFFICIENT CPU-GPU IMPLEMENTATION OF THE MULTIPLE ABSORPTION COEFFICIENT ZONAL METHOD (MACZM)

Boutros Ghannam[1], Maroun Nemer[1], Khalil El Khoury[1], and Walter Yuen[2]
[1]*MINES ParisTech, Center for Energy and Processes, Paris, France*
[2]*University of California Santa Barbara, Santa Barbara, California, USA*

The multiple absorption coefficient zonal method (MACZM) is an efficient radiative heat transfer modeling method in nonisothermal inhomogeneous media. The method is of high interest for dynamic applications because of its ability to asses semitransparent radiative heat transfer in very short computation time. In this work, an efficient algorithm for MACZM is implemented. A connectivity control study is presented for taking into account the connectivity considerations required by the method. An identified ray traversal algorithm corresponding to part of the MACZM implementation is then selected among three different approaches presented in the article, based on well-known ray traver sal algorithms, the 6-tripod line algorithm and the 6-parametric line algorithm. On the other hand, the MACZM is highly parallel and is implemented in CUDA, a parallel computing architecture that enables easy use of a powerful graphics processing unit (GPU). An efficient implementation is discussed consisting of an optimal solution for exploiting the method parallelism (threading) and the use of the memory resources available on the GPU. Speed-ups going from 300 to 600 times are achieved, using a NVIDIA Tesla C 1060 GPU and an Intel Xeon CPU E5507 at 2.27 GHz. Radiative heat transfer is then simulated in a steel reheating furnace using the optimized GPU implementation. The computation time is further reduced by using a multigrid approach.

INTRODUCTION

Applications where the radiative heat exchange is dominant are very challenging. Steel reheating furnaces are a good example of an application where the radiative heat exchange is dominant, because of the high operating temperatures. Moreover, a steel reheating furnace is filled with nonhomogeneous media due to combustion gases that circulate in it. Steel slabs have to be reheated to temperatures around 1,200°C. Many processes can be applied to the steel slabs when they leave the furnace. The quality of the final product is very sensitive to the temperature profiles of the slabs leaving the furnace. For example, the temperature of a slab has to be homogeneous when it enters a rolling process. Thus a good knowledge of the temperature profiles

Received 15 May 2012; accepted 21 June 2012.
Address correspondence to Boutros Ghannam, MINES ParisTech, Center for Energy and Processes, CNRS FRE 2861, 5, Rue Leon Blum, 91120 Paliseace, France. E-mail: boutros.ghannam@mines-paristech.fr

NOMENCLATURE

a	absorption coefficient of volume	L_t	transmission mean beam length
a_m	mean absorption coefficient	MBL	mean beam length
D	characteristic dimension of the surface and volume used in the definition of generic exchange factors	n_x	dimensionless x coordinate
		n_y	dimensionless y coordinate
		n_z	dimensionless z coordinate
dS	surface element	x, y, z	orthogonal coordinates
e	algebraic length		
F_{12}	view factor between surfaces A_1 and A_2	**Subscripts**	
		i, j, k	element index
g_1g_2	generic exchange factor between volumes V_1 and V_2	pp	parallel component
		pd	perpendicular component
GEF	generic exchange factor	x, y, z	direction index
L	length of line segment	xy	situated in the xy plane
L_a	absorption mean beam length	xz	situated in the xz plane
L_{em}	emission mean beam length	zy	situated in the zy plane

Downloaded by [University of California Santa Barbara] at 18:24 07 January 2016

inside the furnace is very important. From this comes the importance of a dynamic numerical simulation of the heat transfer. In additon, the energy consumption can be decreased when dynamic simulations are provided.

The high need for efficient modeling of radiative heat transfer in participating media has led to the development of many numerical methods. The multiple absorption coefficient zonal method (MACZM) is a recent method published by Yuen [1]. It is based on the concept of generic exchange factors (GEFs) and is a more sophisticated form of the zonal method. In the MACZM, a new definition of GEFs as superposition of partial generic exchange factors is given in order to make the method suitable in general three-dimensional and inhomogeneous media. In this way, the accuracy of the method was validated by Yuen on the mixing of hot molten fuel with water. In a separate work, Yuen [2] introduced the definition of a set of physical entities, the mean beam lengths (MBLs), as a basis for simple GEF correlations. The advantage of MBLs is the ability of their functional behavior to be accurately correlated by artificial neural networks (ANN) correlations. In a previous work [3], the fundamental MACZM and the definition of MBLs were combined and implemented together based on the ANNs generated by Yuen in [1]. The validity of the entire approach was then verified experimentally on a dynamic modeling of a steel reheating furnace. In addition to the good accuracy of the method, it was found to be very fast in comparison to existing methods, even with a nonoptimized C++ code. Because of the importance of the computation time, especially in dynamic modeling of radiative heat transfer, the purpose of this work is to present an efficient, fast algorithm for computing the MACZM. This is done in two steps by the combination of an optimized algorithm and the best material architecture.

First, the best algorithm for implementing the method in serial code is sought. As will be described in the body of the article, the MACZM computational time is spent mainly on two parts. First is the application of ANNs for computing MBLs and then GEFs, and a ray traversal or line discretization phase in which an important factor is computed, the mean absorption coefficient. As the application of ANNs is straightforward, attention will focus on the line discretization operation. The latter

is more complicated and is widely studied in the literature. Three-dimensional discrete lines are of special importance in graphics, because they are fundamental in building complex discrete objects [4, 5]. A 3-D discrete line also represents the path of a ray in 3-D discretized space, and tracing discrete rays is of high interest in graphics and in image processing because it is significantly more efficient in terms of calculation time than ray-tracing methods that are based on a geometric representation of 3-D scenes [6]. Therefore, many fast algorithms have been developed for generating 3-D discrete lines in voxel space [6–8]. In the present work, these algorithms are adapted to the MACZM and compared in order to find the fastest ray traversal algorithm that best fits the calculation of the mean absorption coefficient.

The MACZM is implemented in CUDA on a NVIDIA Tesla C 1060 GPU. Because of its high parallelism, the MACZM can give high speed-up with parallel implementation. The only difficulty is to choose which multiprocessor architecture best fits the application. Multicore architectures were born when the rise in CPU clock frequency had become limited, for example the Intel multicore CPUs, the Sony/Toshiba/IBM alliance's Cell Broadband engine [9], etc. Another powerful solution for parallel programming is GPU. Originally, GPUs were developed for fast graphics rendering. They were then used in general-purpose programming [10]. Nevertheless, before the appearance of CUDA, a good knowledge of graphics API was necessary for programming general-purpose GPUs, and thus only a few specialized programmers were able to develop efficient parallel codes on GPUs. CUDA was released in early 2007 by NVIDIA [11, 19]. CUDA is an extension of C/C++ associated with a hardware support added to the GPU. It allows the programmer efficient programming without passing by any graphics API. Starting in 2007, many scientific applications (especially in biological photography) have been developed in CUDA. High performance and speed-up have-been achieved, thus permitting much more advanced development in scientific research. In the present work, CUDA is used because of its single-instruction, multiple-data (SIMD) programming model, which can achieve high speed-up of the program compared to the CPU program, as demonstrated in this article.

The article is organized as follows. First the multiple absorption coefficient zonal method (MACZM) is explained briefly and the main parts of its algorithm are highlighted. Then the methods and algorithms for a one-thread implementation are discussed and the optimal ones are chosen (discretization, discrete lines, and ANNs). A brief introduction to CUDA is necessary for the discussion of a CUDA implementation for MACZM. An efficient MACZM implementation is then presented. Finally, the approach is applied to a steel reheating furnace in which a multigrid method is used to avoid time waste during computation.

MULTIPLE ABSORPTION COEFFICIENT ZONAL METHOD (MACZM) AND ALGORITHM

The multiple absorption coefficient zonal method MACZM [1–3] is based on the concept of generic exchange factors (GEFs). A generic exchange factor is the fraction of energy emitted from a radiating space element and absorbed by another space element. In the work of Yuen, space elements are cubical voxels. Artificial neural networks (ANNs) are generated accordingly to allow the computation of GEFs between

Downloaded by [University of California Santa Barbara] at 18:24 07 January 2016

any couple of voxels in a voxelized discrete space. The GEF depends on the distance between the two voxels, their absorptivities, and the transmissivity of the medium between them. The ANNs do not generate GEFs directly. Yuen defined six entities, the mean beam lengths (MBLs), that are generated directly by ANNs and are used in simple equations to give GEFs. MBLs are functions of the same variables as GEFs. Equations giving GEFs from MBLs are similar to the following:

$$\frac{g_1 g_2}{D^2} = F_{gg} = F_{12}(n_x, n_y, n_z)(1 - e^{-a_2 L_a})(1 - e^{-a_1 L_{em}})e^{-a_m L_t} \tag{1}$$

where $g_1 g_2$ is the volume–volume GEF; L_t, L_{em}, and L_a are, respectively, the transmission, emission, and absorption mean beam lengths; n_x, n_y, and n_z are the dimensionless distances between the two voxels; F_{12} is a form factor; and a_m is the mean absorption coefficient of the medium separating the two voxels.

The term volume–volume refers to the fact that the GEF gives the energy radiation exchanged between two voxels (cubical volumes). Similar definitions exist for GEFs giving the radiation heat exchange between a volume and a surface or between two surfaces.

Moreover, Yuen defined partial GEFs in order to improve the method accuracy in homogeneous and noncontinuous media. He demonstrated the accuracy of the method in [1] by calculating the radiative heat exchange between a high-temperature ($\sim$3,000 K) molten nuclear fuel and water. Partial GEFs are fractions of the GEF between two voxels, and they correspond to the fraction of the radiation emitted by the first voxel through one face only and received by the second voxel through only one face. Since a voxel is always exposed by three faces to any other voxel in the discrete space, nine partial GEFs exist. All GEFs can be deduced from the definition of two cases. The first case is when the two emitting and receiving "virtual" surfaces (voxel' faces) are parallel, and the second case is when they are orthogonal in space. Figure 1 shows the parallel and perpendicular components of partial GEFs. Parallel and perpendicular MBLs are defined correspondingly, and their expression to partial GEFs is then

$$\frac{(g_1 g_2)_{pp}}{D^2} = F_{12,pp}(n_x, n_y, n_z)(1 - e^{-a_2 L_{a,pp}})(1 - e^{-a_1 L_{em,pp}})e^{-a_m L_{t,pp}} \tag{2}$$

$$\frac{(g_1 g_2)_{pd}}{D^2} = F_{12,pd}(n_x, n_y, n_z)(1 - e^{-a_2 L_{a,pd}})(1 - e^{-a_1 L_{em,pd}})e^{-a_m L_{t,pd}} \tag{3}$$

where the subscripts *pp* and *pd* refer to parallel and perpendicular components, respectively. It is now clear that in this equation the form factors are form factors between two surfaces.

Looking back to Eq. (1), it is clear that all variables are known except the mean absorption coefficient a_m. In fact, the computation of a_m is an important part of the algorithm and will be detailed further in the article.

Consider now an enclosure with radiating objects and let MACZM be applied to compute the radiation heat exchange factors between elements of the scene. Three major steps are to be considered in the algorithm. The first step is the discretization of

Downloaded by [University of California Santa Barbara] at 18:24 07 January 2016

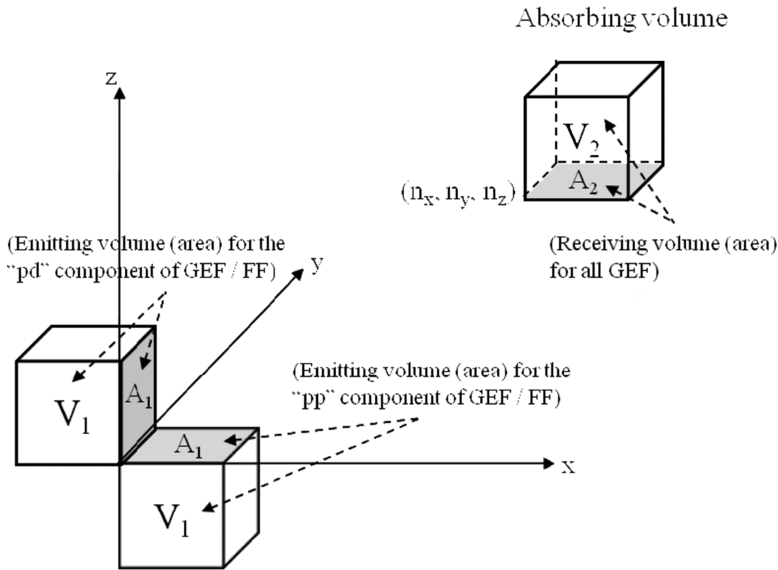

Figure 1. Partial GEFs' basic parallel and perpendicular components (*pp* and *pd*).

Downloaded by [University of California Santa Barbara] at 18:24 07 January 2016

the scene, by dividing the space into cubical voxels. The heat-exchange factor between any couple of objects is to be computed in the next two steps. The second step computes the mean absorption coefficients needed for the application of ANNs and the calculation of GEFs. The third step is the calculation of MBLs and then GEFs using ANNs and the equations described above. Finally, the summation of all GEFs between two objects or, in other words, GEFs between all their voxels, gives the direct radiativeheat-exchange factors. Many methods could be applied for the deduction of the total exchange factors, for example, the optimized plating algorithm of El Hitti et al. [12], but this is not within the scope of this article. In the following three sections, optimized methods for the three steps of the algorithm are discussed.

GRID GENERATION AND REPRESENTATION OF THE OBJECTS IN THE DISCRETE SPACE

As mentioned before, the first step of the algorithm is the application of uniform volume meshing (cubical voxels, $dx = dy = dz = D$) to the scene volume and objects. Let $\mathbb{Z}^3$ be the subset of $\mathbb{R}^3$ corresponding to all the elements of $\mathbb{R}^3$ whose coordinates are integers, and let us consider an orthogonal system of coordinates (O, x, y, z) in $\mathbb{Z}^3$. In a dimensionless representation, the dimension of a voxel is $D = 1$ and each voxel in (O, x, y, z) is identified by the coordinates of its lower left corner $(i, j, k) \in \mathbb{Z}^3$. The dimensionless distances between two voxels along the principal coordinates are (n_xD, n_yD, n_zD), where $(n_x, n_y, n_z) \in \mathbb{Z}^3$. The algorithm then stores the discrete scene absorption coefficients in a 3-D array, where each value in the array is the absorption coefficient value in the voxel assigned to this array position. The 3-D array is a direct projection of the meshed 3-D space.

Given the dimensions and the position of an object in the 3-D scene, the set of voxels corresponding to the object in the discrete space is found by the 3-D scan-conversion algorithms'' of Kaufman and Shimony [4]. These algorithms allow voxelization of 3-D quadratic objects such as cylinders, cones, and spheres as volumes or surfaces optionally. These basic geometric shapes are then used to construct the whole objects in the scene. The 3-D scan-conversion algorithms are very efficient; they generate voxels incrementally, using only additions, subtractions, and comparison tests, thus guaranteeing linear complexity in terms of the voxels' number.

MEAN ABSORPTION COEFFICIENT COMPUTATION BY RAY TRACING

This step of the algorithm computes a mean absorption coefficient over a ray or the line between the center points of two surfaces of two distant voxels. First, the ray (or the continuous line) is discretized (voxelization). Then, the mean absorption coefficient takes the value of the mean absorption coefficient over all voxels crossed by the ray. The value of the mean absorption coefficient is computed by the following equation:

$$a_m = \left(\sum_i a_i L_i \right) \Big/ L \tag{4}$$

where a_m is the mean absorption coefficient to be computed. The index i refers to voxels crossed by the ray or, in the algorithm, voxels that are in the discrete line corresponding to this ray. a_i is the absorption coefficient assigned to the voxel i. L_i is the length of the segment of the line (the ray) in the voxel i. L is the total length of the continuous line.

Connectivity of the Discrete Line

Consider now the discrete space represented by $\mathbb{Z}^3$, where a voxel is defined by the coordinates of its lower left corner. A line discretization function representing a line from $\mathbb{R}^3$ in $\mathbb{Z}^3$ is a function that associates the value 1 to each voxel of $\mathbb{Z}^3$ corresponding to the continuous line and the value 0 to other voxels. However, the definition of this function is not unique and the choice of voxels representing the line is flexible, depending on the connectivity between them. A discrete line is said to have the *containment property* if it contains all voxels pierced by the continuous line [13]. A voxel of the 3-D discrete space shares 6 faces with the adjacent voxels, 12 edges, and 8 corners. The connectivity between two voxels is defined accordingly: Two voxels are 6-connected if they have a face in common, 18-connected if they share a face or an edge, and 26-connected if they share a corner or an edge or a face. In the 2-D space, there are similarly 4- and 8-connected pixels. From here, an N-path in $\mathbb{Z}^3$ is a sequence of voxels where all consecutive pairs of voxels are N-connected ($N = 6$, 18, or 26). Consequently, an N-line is an N-path in $\mathbb{Z}^3$ corresponding to a continuous line in $\mathbb{R}^3$. Two 2-D discrete lines corresponding to the 4- and 8-connected lines are represented in Figure 2*a*. As we can see in Figure 2*a*, an 8-connected line does not have the containment property. Similarly, in 3-D space only the 6-line has the

Downloaded by [University of California Santa Barbara] at 18:24 07 January 2016

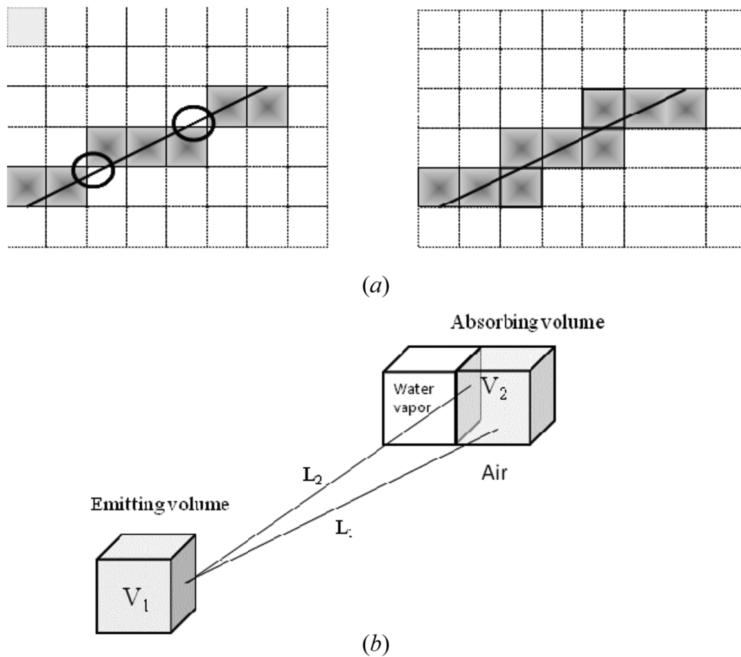

Figure 2. Connectivity control. (*a*) An 8-connected line (left) and a 4-connected line (right). (*b*) Difference in mean absorption coefficient over two different paths.

Downloaded by [University of California Santa Barbara] at 18:24 07 January 2016

containment property. Nevertheless, 18-connected and 26-connected lines are used in graphics for ray tracing. The reason is that the latter contain fewer voxels by a mean factor of 2, which implies a faster algorithm for computing lines [6–8, 12–14]. Some algorithms can even switch between connectivities in order to achieve maximum precision with minimum computation time.

Hence, a 6-connected discrete line algorithm clearly gives more precision at the cost of some added computation time. In the present case, the 6-line is used for computing the mean absorption coefficient from discrete lines, because of some particularities of the MACZM. First, the MACZM is accurate when relatively large mesh size is considered, as demonstrated in Yuen's work [1]. This implies that a low-resolution meshing is taken whenever possible, i.e., every time that the geometry inside the scene is not complicated. This means that a large portion of the continuous line can be omitted when using 18- or 26-connected discrete lines while estimating the mean absorbance coefficient, which gives a nonprecise value, given that the medium is not homogeneous. This privileges the use of the 6-line to give an acceptable precision for the estimation. In addition, the mean absorption coefficient is used later in the algorithm as an input to a system of artificial neural networks. Taking into account the error produced by the ANN, only a small error can be accepted on its inputs. Finally, considering the application of the MACZM in a noncontinuous medium, a single mean absorption coefficient between two voxels is replaced by 9 mean absorption coefficients in order to take into account accurately the presence of discontinuities in the intervening medium. An example is illustrated in Figure 2*b*, as it is

presented in Yuen's work [1]. From this example it can be deduced that omitting a voxel, or replacing a voxel by another one while computing the pass for a mean absorption coefficient, may result in an error that can be very significant, especially when a bigger mesh size is considered.

Three well-known 6-line algorithms are now reviewed and completed with the calculation of the length traversed by the continuous line inside each voxel of the corresponding discrete line. The issue is to select the fastest discrete line computation algorithm when combined with the ray-length calculation inside voxels.

Span-by-Span Algorithm

Using mathematical formulations, discrete lines are demonstrated to have global properties [15, 16]. Those properties are used to reduce the space interval over which the line is computed. For example, all discrete lines are demonstrated to have a point of symmetry, which simply reduces the computation to half. More generally, the whole discrete line is reduced to a word, the word being a small number of voxels representing a segment of the continuous line, and the line is demonstrated to be equal to the recurrence of this word excluding the beginning and the end of the line. The word is computed directly using Bresenham's algorithm [17]. Improvement to the word's computation time can be obtained using N-step algorithms [18].

The best span-by-span method reaches 20 times better computation time [15] over the basic Bresenham algorithm. Unfortunately, the overall gain of those algorithms is significant only for lines having a length of more than 100 voxels. However, it has been shown, on one hand, that the use of very long lines in MACZM, which is equivalent to a very fine meshing, would result in a long computation time for ANNs. On the other hand, and as could be observed in the computation of the length of the ray inside the voxels, there are no properties for length values that can be compared to the line properties. This implies that the method cannot be held for computing lengths at the same time, which means no gain in length-computation time that has to be computed separately. In fact, even with some precalculated values, the calculation of the length will at least take three multiplications per voxel, which is by itself more time-consuming than a whole step of the tripod algorithm and parametric algorithm, as will be seen in the next two subsections.

The Tripod 6-Line Algorithm

The tripod algorithm was presented by Kaufman and Cohen-Or [14] as an efficient 6-line algorithm that guarantees the containment property; its efficiency is very similar to that of the parametric algorithm. We will briefly describe the algorithm. First, consider a Cartesian coordinate system (O, x, y, z) and a positive ray AB connecting two points A and B in (O, x, y, z). A line is said positive when it has positive slopes in the three principal directions $\overrightarrow{Ox}$, $\overrightarrow{Oy}$, and $\overrightarrow{Oz}$. Based on the fact that in a discrete 6-line all voxels are face-adjacent, the tripod moves from voxel to voxel by tracking which face the ray pierces when leaving the voxel. In the case of a positive line, only three faces of the voxel can be pierced by the leaving ray, and the right face is determined by tracking the projections of the line on the three main axes' planes. This can be done using the midpoint technique as presented in [14]. Globally,

Downloaded by [University of California Santa Barbara] at 18:24 07 January 2016

the projections of the 3-D line on the three principalplanes (O, x, y), (O, x, z), and (O, y, z) are defined by the implicit equations:

$$e_{xy} = Ay - Bx - D_1 = 0 \tag{5}$$

$$e_{xz} = Az - Cx - D_2 = 0 \tag{6}$$

$$e_{zy} = Bz - Cy - D_3 = 0 \tag{7}$$

The direction to which the leaving face is orthogonal is determined by eliminating the two other directions using the following tests:

1. If $(e_{xy} < 0)$, then the face is not orthogonal to the y direction.
2. If $(e_{xz} < 0)$, then the face is not orthogonal to the z direction.
3. If $(e_{zy} < 0)$, then the face is not orthogonal to the y direction.

This results in the following pseudo-code:

```
Set n = delta_i + delta_j + delta_k (n = number of voxels in the discrete line)
WHILE n > 0
 IF e_xy < 0
  IF e_xz < 0
   e_xy += b2
   e_xz += c2
   indx += step_x
   La = Lc
   Lc = slope_x*step_x (length end of voxel)
   am += (Lc-La) * aijk (absorption coefficient)
  ELSE
   e_xz -= a2
   e_zy += b2
   indz += step_z
   La = Lc
   Lc = slope_z * step_z
   am += (Lc-La) * aijk
  END IF
 ELSE
  IF e_zy < 0
   e_xz -= a2
   e_zy += b2
   indz += step_z
   La = Lc
   Lc = slope_z * step_z
   am += (Lc-La) * aijk
  ELSE
   e_xy -= a2
   e_zy -= c2
```

Downloaded by [University of California Santa Barbara] at 18:24 07 January 2016

```
        indy += step_y
        La = Lc
        Lc = slope_y * step_y
        am += (Lc-La) * aijk
      END IF
    END IF
    DECREMENT n
ENDWHILE
```

Here, instructions are added to the fundamental algorithm for calculating the length of the ray inside each voxel as well as the fraction it adds to the mean absorption coefficient. Given the length L_a of the ray before entering the voxel and the calculated length L_c of the ray when leaving the voxel, the length of the ray inside the voxel is then evaluated as $L_c - L_a$, where L_c is calculated at each step by multiplying the slope of the direction to which the leaving face is perpendicular by the coordinate of the face in this direction, and L_a is the L_c of the previous step. Finally, the contribution of the voxel to the mean absorption coefficient is obtained by multiplying the length of the ray inside the voxel by the mean absorption coefficient a_{ijk} of the voxel:

$$(L_c - L_a) \times a_{ijk} \tag{8}$$

The tripod 6-line algorithm includes only integer operations. The voxel traversal takes only two sign tests and three additions by incremental step. The calculation of the length of the ray inside the voxel takes one multiplication and one more addition ($L_c - L_a$), by step, and the mean beam length takes equally one more multiplication and one more addition. As will be seen in the 6-parametric algorithm, the multiplication is avoided in the calculation of the length inside voxels, and thus some gain in computation time can be achieved. In summary, the tripod algorithm has two multiplications, five additions, and one auto-decrement branch by incremental step.

The Parametric 6-Line Algorithm

The parametric 6-line, well known as the 3D-DDA algorithm in graphics, was designed by Amananatides and Woo [8]. This algorithm is very efficient, guarantees the containment property, and is widely used in computer graphics. In Figure 3, the method is illustrated in the 2-D plane$(\overrightarrow{Ox}, \overrightarrow{Oy})$. In an initialization phase, the voxel where the ray origin is situated is considered. In this phase the ray length inside the origin voxel is calculated in the three main directions x, y, and z, and stored in three variables $tmax_x$, and $tmax_z$, respectively. The path lengths of the rays corresponding to the voxel width in each of the three main directions $tmax_y$ are also calculated and stored in $delta_x$, $delta_y$, and $delta_z$, respectively. The second phase of the algorithm is an incremental phase. In this phase, starting from the origin, variables $tmax_x$, $tmax_y$, and $tmax_z$ are compared. The smallest of them shows the direction perpendicular to the face from which the ray leaves the voxel. For example, if $tmax_x$ is the smallest, it will be increased with the value of the corresponding variable $delta_x$, which gives in $tmax_x$ the value corresponding to the length that the ray will have when it hits for

Downloaded by [University of California Santa Barbara] at 18:24 07 January 2016

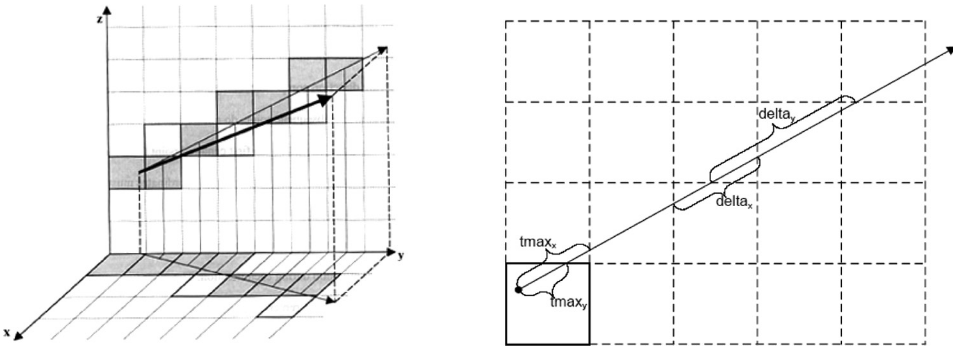

Figure 3. A parametric 6-line in two dimensions.

the next time a face perpendicular to the x direction. The same step is repeated until all voxels of the discrete line are traversed. The pseudo-code is written hereafter:

```
Setn = delta_i + delta_j + delta_k (n = number of voxels in the discrete line)
WHILE n > 0
  IF tmax_x < tmax_y
    IF tmax_x < tmax_z
      am += (tmax_x-Lca) * aijk (absorption coefficient)
      Lca = tmax_x
      tmax_x += delta_x
      indx += step_x
    ELSE
      am += (tmax_z-Lca) * aijk
      Lca = tmax_z
      tmax_z += delta_z
      indz += step_z
    END IF
  ELSE
    IF tmax_y < tmax_z
      am+ = (tmax_y-Lca) * aijk
      Lca = tmax_y
      tmax_y += delta_y
      indy += step_y
    ELSE
      am += (tmax_z-Lca) * aijk
      Lca = tmax_z
      tmax_z += delta_z
      indz += step_z
    END IF
  END IF
  DECREMENT n
ENDWHILE
```

Downloaded by [University of California Santa Barbara] at 18:24 07 January 2016

It can be seen that the voxel traversal in the basic 3D-DDA algorithm takes only two sign tests and two additions by step, which means an addition less than the tripod algorithm. Furthermore the calculation of the ray length at each step is given directly by the *tmax* variable corresponding to this step. The calculation of the length traversed by the ray inside the voxel, as well as the mean absorption coefficient, is similar to the previous algorithm. Finally, the parametric 6-line algorithm has only one multiplication, four additions, and one auto-decrement branch. Hence it has one multiplication and one addition less than the tripod algorithm by incremental step.

Comparison of the Ray Traversal and Mean Absorption Coefficient Computation Algorithms

Three cases are considered in comparisons: the algorithm based on the tripod 6-line ray traversal, which is executed with integer variables only; the algorithm based on the parametric 6-line ray traversal, which is executed in the first case with single floating-point variables and in the second case in integer mode. As demonstrated earlier, both the 6-tripod and 6-parametric line algorithms guarantee the property of containment of the line, thus they generate a 6-connected line. The precisions of the two algorithms are then similar, and both of them guarantee an accurate generation of the line. Hence, the choice of the mean absorption coefficient computation algorithm will be done based on its efficiency in computation time. Table 1 presents a comparison of the computation for three cases. In each case, the algorithm is run for the computation of 1 million rays having a mean length of 100 voxels. All computation times are recorded on a Xeon 2.33 GHz CPU. As expected, when executed in integer arithmetic, the parametric 6-line algorithm, which has the smallest number of operations, is the fastest. It is then chosen for the practical implementation.

Table 1. Computation time for 1 million rays having an average length of 100 voxels (Xeon 2.66 GHz CPU)

Algorithm	Time (s)	Properties
Tripod 6-line (Kaufman et al.)	2.26	6-connected line Integer arithmetic Nonsymmetric
Parametric 6-line (Amanatides and Woo) (floating-point arithmetic)	5.16	3DDDA (3-D discrete differential analyzer) 6-conneted line Floating-point calculations Symmetric
Parametric 6-line (Amanatides and Woo) (integer arithmetic)	1.89	3DDDA (3-D discrete differential analyzer) 6-connected line Integer arithmetic Symmetric

Downloaded by [University of California Santa Barbara] at 18:24 07 January 2016

ARTIFICIAL NEURAL NETWORK AND GEF SUPERPOSITION

ANNs are two-layer neural networks. An example of an ANN is:

$$(X) = [W_1(20 \times 5)] \times (P) + (B_1) \tag{9a}$$

$$Z = F(X) \times (W_2) + b_2 \tag{9b}$$

The inlet vector is $P = (n_x, n_y, n_z, a_m, a_1, a_2)$. (n_x, n_y, n_z) are the dimensionless distances, a_m the mean absorption coefficient, and a_1 and a_2 the absorption coefficients of voxels. The application of ANNs is straightforward. In the first layer, P is multiplied by a 2-D matrix W_1 and added to a vector B_1. A log-sigmoid function is then applied to the resulting vector. The second layer consists of multiplying the resulting vector by a vector W_2 and adding a constant b_2 to the result. Finally, we obtain an MBL, which is put in its equation to compute the GEF. The size of ANNs on the disk is about 4 KB, which means absolutely that matrices will be loaded to DRAM memory in order to have fast access to them. ANNs have to be applied nine times for each GEF computation, correspondingly to the nine partial GEFs. Between the nine partial GEFs, there are three cases where surfaces are in parallel planes and six cases where they are in orthogonal planes (Figure 1). Thus ANNs have only to be defined for two cases, one *parallel* case and one *perpendicular case*, resulting in three couples of *parallel* and *perpendicular* neural networks (Figure 1).

Hence, in order to compute the nine cases, the nine corresponding mean absorption coefficients along the lines between the center points of the nine surface couples are first computed. Then, the parallel and perpendicular cases are separated and geometric transformations to the dimensionless distances of each case are applied in order to obtain the same parallel or perpendicular configuration as in the definition of ANNs.

Finally, the superposition of partial GEFs gives GEFs that are in their turn summed over all the couples of voxels of two objects, which gives the direct radiative exchange factor between the two objects.

GPU CUDA PROGRAMMING MODEL OVERVIEW

Figure 4 shows the architecture of a CUDA-capable GPU, the NVIDIA Tesla C 1060 (used in this work). A GPU is presented as a set of highly threaded streaming multiprocessors (SMs). Each SM has a number of streaming processors (SPs) with a cached shared memory. A GPU has its own DRAM memory, a graphics double rate (GDDR), referred to as global memory. Thus, GPU DRAM has longer latency access than CPU DRAM. The longer latency of GPU DRAM is compensated by higher memory bandwidth and by running thousands of threads concurrently. Performance increases when increasing the number of parallel threads. All parallel threads execute on SPs. The Tesla C 1060 has 30 SM (8 SPs for each), and it supports up to 512 threads per SM, which sums to 15,360 threads. Each CUDA-capable GPU is equipped with a hardware control unit for managing thread execution. Threads execute in groups called tiles and warps (16 and 32 threads). Threads in a SM can exchange data via the shared memory, a limited memory resource with low-latency access. Threads in

Downloaded by [University of California Santa Barbara] at 18:24 07 January 2016

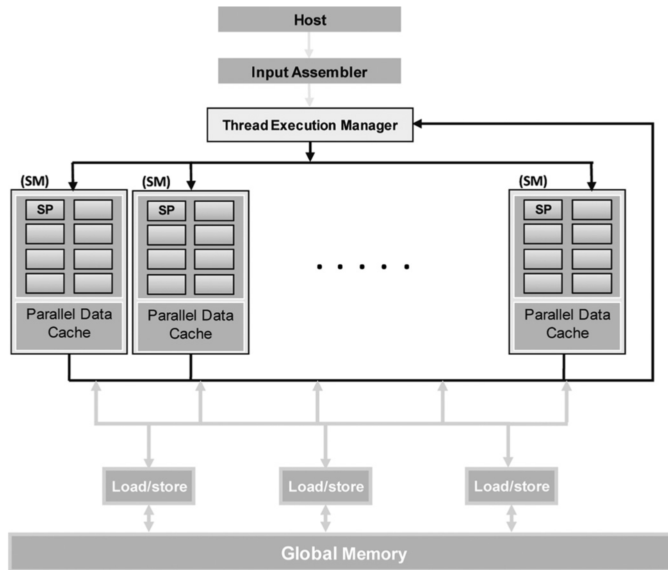

Figure 4. Architecture of a CUDA-capable GPU.

Downloaded by [University of California Santa Barbara] at 18:24 07 January 2016

a warp execute the same instruction at a time (using different data input). This technique of parallel execution is called single-instruction, multiple-thread (SIMT), which is a case of single-program, multiple-data (SPMD) programming.

PARALLEL IMPLEMENTATION OF MACZM IN CUDA

Since the MACZM is here for the first time implemented on a parallel processor, the choice of CUDA is first justified. As stated before, the GEF computation in the MACZM is carried out in two principal steps. The first is the computation of the corresponding mean absorption coefficient. The second step is the application of an ANN. The mean absorption coefficient for a GEF is computed using a discrete line algorithm applied to the line between the couple of voxels corresponding to the current GEF. This algorithm applies to any couple of voxels independently of the others. Thus, it is able to run in parallel. This is the same for ANNs that are applicable independently to the computation of any GEFs. Moreover, while computing GEFs between any voxels, the same ANNs apply for computing the partial GEFs, and since they are independent, they are executable in the same time and in the same chronological order. This gives one identical kernel to execute on different voxel couples. This is, consequently, a single-instruction, multiple-thread (SIMT) parallel programming model [19, 22]. Now that this is demonstrated, the CUDA implementation is done as follows.

CUDA Parallel Kernel

The computation of GEFs (mean absorption coefficient, ANNs) takes 99.8% of the execution time (34.98%, 65% respectively) of the optimized CPU implementation.

The rest of the computation time is for initializing the absorption coefficients in the scene array and for initializing the objects using the scan-conversion algorithms. Scan-conversion algorithms are incremental, and thus their execution is sequential. Their execution time is also very small. Thus, the scan-conversion is executed on CPU and then a GPU kernel is launched for computing GEFs. It should be remembered that the direct exchange factor between two objects is the sum of all GEFs between them. In other terms, GEFs have to be computed between each voxel of the first object and all voxels of the second one. The summation of all computed GEFs gives the exchange factor between the two objects. One CUDA kernel computes the exchange factor between two objects. The CUDA kernel launches one thread for the computation of each GEF. At the end of its execution, the kernel will have computed the exchange factor between the two objects. While the kernel is computing an exchange factor between two objects, the CPU sets data for the next kernel launch and waits for the current kernel to finish before launching the next one on the GPU. If, for example, there are two objects of 256 and 384 voxels, respectively, it gives 256×384 or around 100,000 threads to run in parallel for one kernel, which is large enough to take advantage of the GPU performance.

Computation of the Mean Absorption Coefficient

The first part of the kernel executes the computation of the mean absorption coefficient. The same algorithm of the CPU implementation, the parametric 6-line algorithm, does this. Unlike other discrete line algorithms, this algorithm has the property of being symmetric. In other words, the same instructions are applied for computing the discrete line independently of its position in the space (in which main direction the line slope is higher). This property avoids thread divergence due to the line positions, since all the threads are executing the same instructions. Generally, thread divergence occurs when not all threads of a warp execute the same instruction, due, for example, to an if–else instruction. Then, because of the SIMT execution technique, some threads have to wait for the others to finish so they continue their execution, which reduces the number of concurrent threads in a warp. Thus, avoiding thread divergence when possible is crucial. Divergence can also result when a loop of the kernel has a condition that does not achieve the loop at the same time in all threads of a warp. In the present case, this happens if the discrete lines in a warp have different lengths and consequently different numbers of voxels. Making the threads of the same warp compute mean absorption coefficients of neighbor voxel couples may solve this problem, and also reduce the memory access time.

Actually, the parametric 6-line algorithm determines the voxels of the discrete line incrementally and for each voxel it reads its absorption coefficient from the scene 3-D array. The execution time of the algorithm is limited by the memory access time, since the number of arithmetic operations in the algorithm is very small. Fortunately, global memory latency can be reduced with coalesced memory accesses. This is when a block of neighbor variables are accessed once all together (16 Bytes of memory), and it takes the same time as for accessing one variable in the block. When neighbor threads (threads of the same warp) compute neighbor discrete lines, they read neighbor voxels' absorption coefficients in the scene 3-D array simultaneously, and thus promote coalesced reads from the 3-D array in global memory. Coalesced

Downloaded by [University of California Santa Barbara] at 18:24 07 January 2016

reads, as well as equal-length lines, are promoted but cannot be guaranteed. Finally, the 3-D scene array contains only char variables because the execution of this algorithm is two times faster with char array than with float array. For this work, char is sufficient, and absorption coefficients in the interval $[10^{-2}, 1]$ are multiplied by 10^2 to obtain char numbers in the interval $[1, 10^2]$.

At the end, the CUDA parametric 6-line algorithm runs 150 times faster on the GPU compared to the one-thread CPU implementation. The hardware used here are a NVIDIA Tesla C 1060 GPU and an Intel Xeon CPU E5507 at 2.27 GHz.

Downloaded by [University of California Santa Barbara] at 18:24 07 January 2016

Artificial Neural Networks

As described in the ANNs section, the inlet vector to ANNs is $P = (n_x, n_y, n_z, a_m, a_1, a_2)$ [Eq. (9)]. At this point, the thread computes (n_x, n_y, n_z) from the kernel input vector, the mean absorption coefficient a_m is already computed in the previous step (discrete line), and the absorption coefficients of voxels read from the 3-D scene array. The only particularity of this phase is that all threads load the entire ANN matrixes. They could load ANNs by tiles from global memory to the shared memory. Nevertheless, it is a lot more efficient to transfer (from the CPU) ANNs to the constant memory and let the threads read them directly in the constant memory. Actually, this is possible because all ANNs occupy only 3.96 KB memory space, which is very much smaller than the GPU constant-memory space (64 KB). Constant memory is a read-only low-latency memory. When many threads access the same variable in constant memory, the read operation is very fast. Constant memory is usually used to store data to be read by threads while executing the kernel. ANNs, transferred as they are to the constant memory, lead the code to run not as fast as expected. This is because of the constant memory design. Actually, the constant memory is a cached global memory. Due to design cost, each cache is designed to store multiple consecutive words, which gives higher space, but with few cache entries. While different warps (threads in other blocks) could be using a different ANN (there are nine ANNs), this may require many entries altogether. In order to solve the problem, each ANN is regrouped in one data structure. Thus, each thread now accesses only one cache entry and cache entry traffic is avoided. Once the computation of GEFs is finished, all threads work together for completing the GEF summation in parallel.

Finally, as the present ANN CUDA implementation is tested, the code runs from 400 and up to 1,000 times faster than the CPU code, depending on the grid size.

As for the average time repartition of the MACZM CUDA implementation, the mean absorption coefficient computation is now the most time-consuming, with 71% of the total time and 29% for the ANN computation. The scan-conversion execution time is now hidden by the GPU time, since it is executed on the CPU during kernel run.

At the end, with MACZM CUDA implementation, the gains in computation time compared to the CPU implementation are 300 times for surface–surface exchange factors, 450 times for volume–surface exchange factors, and 600 times for volume–volume exchange factors.

APPLICATION: SIMULATION OF A STEEL REHEATING FURNACE

Description of the Steel Reheating Furnace and the Simplified Furnace

The furnace considered is asteel-slab reheating furnace. The steel slabs are introduced at the furnace entry at ambient temperature. They are positioned on four supporting rails and circulate slowly toward the furnace exit. During this operation, the steel slabs are heated to a uniform temperature of about 1,200°C. Then they can undergo different procedures at the exit (e.g., rolling). The quality of the final product is very sensitive to the output temperature of the steel slabs at the furnace exit and the temperature uniformity inside the steel slabs. This needs a real-time dynamic numerical simulation process. One more reason the dynamic numerical simulation is useful, is the reduction of combustible gas consumption.

The internal furnace dimensionsare 36,275 cm × 680 cm × 380 cm. The furnace can hold 44 slabs of steel laid on four rails. The slabs of steel have different widths but they all have a height of 20 cm and a length of 80 cm. An average distance of 20 cm is assumed to be separating the steel slabs over the length of the furnace. Figure 5 shows the repartition of the steel slabs in the furnace. The rails are insulated; they are assumed to have rectangular shape of dimensions 36,275 cm × 20 cm × 40 cm. Finally, burners are positioned in the upper and lower zones of the furnace. All burners are identical, and the combustion volumes are assumed rectangular with dimensions 80 cm × 400 cm × 80 cm.

Many mesh sizes are going to be considered in the simulations. Since the whole furnace simulation will be too time-consuming when very fine meshes are used, a simplified case is well be considered. The simplified case is an identical part of the furnace beginning from the inlet and going to a depth of 7,200 cm (Figure 5).

Simulation in MODRAY

We use MODRAY in this study in order to benchmark the results. Modray is a tool developed by the Center for Energy and Processes. It is based on the flux-plane approximation method for computing radiativeheat-exchange factors. The method was explained and validated in a previous work [3]. It is just recalled here that MODRAY decomposesthe surfacesof the scene and the objects it contains into a number of surface meshes, in order to compute the heat-exchange factors. For our simulations we considered a surfacedecomposition of 10 × 2 × 2 for the scene (internal

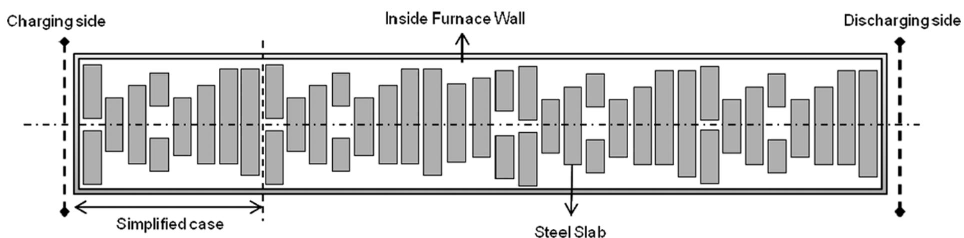

Figure 5. A charging plane of steel slabs in the steel reheating furnace.

Downloaded by [University of California Santa Barbara] at 18:24 07 January 2016

walls), $2 \times 4 \times 2$ for the combustion volumes, $4 \times 2 \times 2$ for the steel slabs, and $10 \times 2 \times 4$ for the rails.

The furnace is assumed to be filled with gases with an absorption coefficient of $0.1\,\mathrm{m}^{-1}$ for all simulations. Walls and rails are insulated, and they have an emissivity of 0.85. The emissivity of combustion volumes is assumed to be 0.7, and the emissivity of the steel slabs is assumed to be 0.9. As a result, the simulation of the whole furnace takes 800 s, and the simulation of the simplified case takes 55 s.

Simulation in MACZM

The simulation of the whole furnace and the simplified case (part of the furnace) are now carried out using MACZM with the same configuration and the same properties. In MACZM, uniform rectangular meshing is applied. First, a mesh of voxel volume size $10\,\mathrm{cm}^3$ is considered (Figure 6*a*), which gives over 9 million voxels. The simulation is too much time-consuming; only the simplified case can be totally simulated. Results are compared to MODRAY results by calculating the relative difference.

Simulations are repeated with voxel sizes of $20\,\mathrm{cm}^3$, $40\,\mathrm{cm}^3$, and $80\,\mathrm{cm}^3$, leading to 1.17 million, 0.15 million, and 18,000 voxels respectively (Figures 6*b*–6*d*). Their relative errors and computation times for all voxel sizes are summarized in Table 2.

Relative Error

We note here that the relative erroris calculated for each category of heat-exchange factors as a weighted average of all heat-exchange factors in this category, by the following equation:

$$\text{Relative error} = \sum_i (e_i \times v_i) \Big/ \sum_i v_i \tag{10}$$

where e_i and v_i are, respectively, the relative error and the value of the heat-exchange factor i.

Discussion and MACZM Multigrid

Consider now the results of MACZM with the finest mesh size, $10\,\mathrm{cm}^3$. Table 2 shows that all results are accurate. We note here that when the distance between objects is greater, results are less accurate. This is because ANNs are generated for a small numbers of voxels, since MACZM accuracy is goodwith largemesh size, as demonstrated by Yuen [1].

For 20-cm^3 mesh size, Table 2 shows that the relative error has become high for the heat-exchange factors between the slabs themselves and between slabs and rails. Then a 10-cm^3 mesh size is needed to compute the former two factors.

Because the computations of heat-exchange factors are independent from each other, different mesh sizes can be considered for the computation of each category. Looking more at the results and finding the highest accurate mesh size for each

Downloaded by [University of California Santa Barbara] at 18:24 07 January 2016

Downloaded by [University of California Santa Barbara] at 18:24 07 January 2016

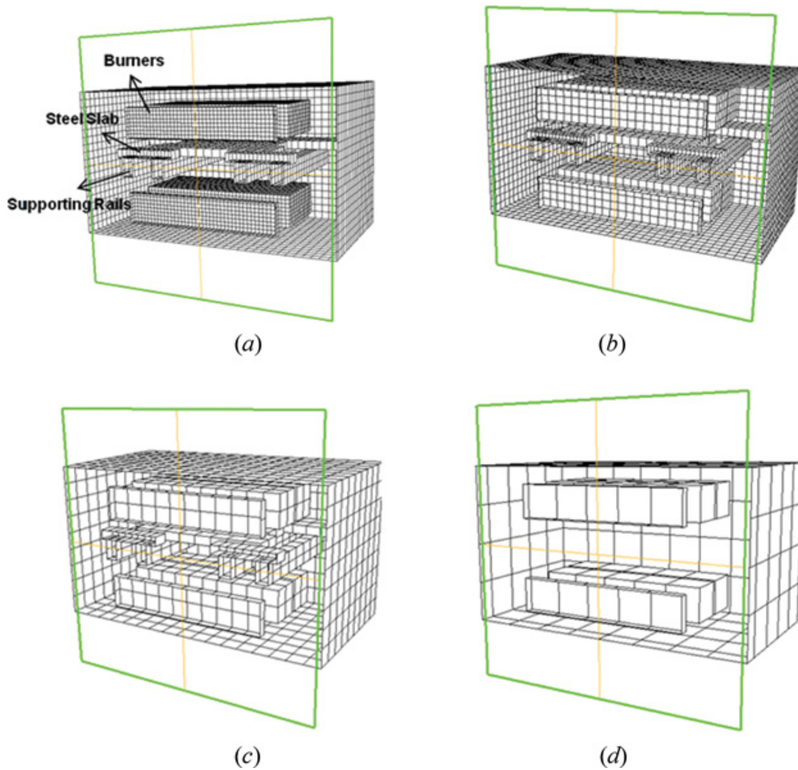

Figure 6. The steel reheating furnace simulation in MACZM. (*a*) Mesh size $D = 10$ cm. (*b*) Mesh size $D = 20$ cm. (*c*) Mesh size $D = 40$ cm. (*d*) Mesh size $D = 80$ cm (color figure available online).

heat-exchange factor category, we easily obtain the multigrid configuration. Actually, it is recognized that the heat-exchange factors between the walls and the steel slabs, the rails, and the combustion volumes, as well as the heat-exchange factors between

Table 2. Comparison of simulation results of the steel reheating furnace

Exchange factor	CPU time (s)				Relative error to MODRAY (%)			
Mesh size D (cm)	10	20	40	80	10	20	40	80
Walls–steel slabs	77.45	6.26	0.36	0.01	3.17	4.13	19.31	—
Walls–rails	89.90	5.01	0.31	0.06	2.41	2.60	21.11	156.97
Rails–steel slabs	8.19	0.72	0.03	0.00	5.18	11.95	29.73	—
Steelslabs–steel slabs	2.42	0.22	0.05	0.00	1.24	19.44	47.05	—
Rails–rails	5.26	0.37	0.05	0.02	4.51	7.72	59.45	552.13
Walls–walls	375	18.72	0.92	0.06	—	—	—	—
Burners–burners	32,459	527	5.90	0.08	4.28	4.83	6.29	13.49
Burners–steel slabs	731	21.96	1.34	0.00	6.31	7.58	—	—
Burners–rails	1628	38.55	1.16	0.06	6.99	8.59	20.84	310.55
Walls–burners	6341	186	5.26	0.13	4.85	5.46	6.09	11.82
Total computation time	41,717	805	15.36	0.42				

Table 3. MACZM multigrid summary for the steel reheating furnace

Exchange factor	Mesh size (cm)	Computation time, simplified case (s)	Computation time, whole furnace (s)	Relative error to MODRAY (%)
Walls–steel slabs	20	0.36	6.00	4.13
Walls–rails	20	0.31	5.88	2.60
Rails–steel slabs	10	8.19	289.00	5.18
Steelslabs–steel slabs	10	2.42	251.21	1.24
Rails–rails	20	0.37	10.56	7.72
Walls–walls	40	0.92	134.32	—
Burners–burners	40	5.90	86.57	6.29
Burners–steel slabs	20	21.96	67.50	7.58
Burners–rails	20	38.55	403.40	8.59
Walls–burners	40	1.16	7.20	6.09
Total computation time		80.14	1,162	

the combustion volumes and the rails, can be taken from the case of voxel size 20 cm^3. Finally, the heat-exchange factors between the combustion volumes themselves and between them and the walls, and the heat-exchange factors between the walls themselves, can be taken from the case of voxel size 40 cm^3. The voxel size of 80 $cm^3$is not accurate for any cases because it doesnot represent well the furnace geometry, as can be observed in Figure 6*d*. At the end, results of the multigrid are grouped in Table 3, and Figure 7 shows an idea of what the multigrid results in. As we see in Table 3, the multigrid has not only given accurate results, it has reduced the computation time efficiently. It avoids any waste in computation time. By adding computation times for all the exchange-factor computations, it was found that the simulation ofpart of the furnace in MACZM takes 80 s, while the simulation of the whole furnace takes 1,162 s.

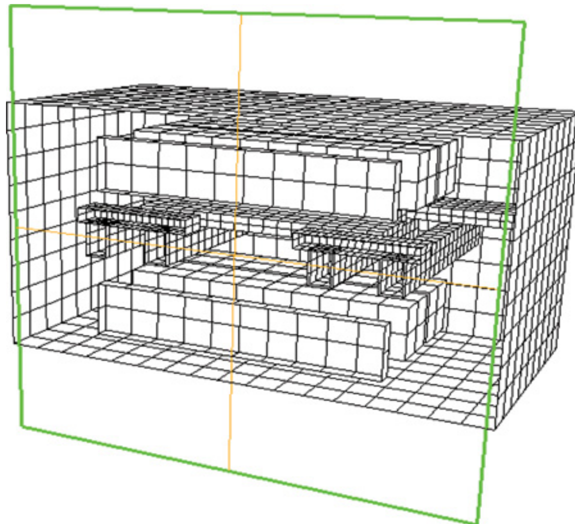

Figure 7. Multigrid configuration of the steel reheating furnace simulation (color figure available online).

Downloaded by [University of California Santa Barbara] at 18:24 07 January 2016

Table 4. MACZM multigrid comparison between CPU and GPU times for the whole furnace simulation

Exchange factor	CPU computation time (s)	GPU computation time (s)	Computation time ratio CPU/GPU
Walls–steel slabs	6.00	0.019	315
Walls–rails	5.88	0.018	326
Rails–steel slabs	289.00	0.996	290
Steelslabs–steel slabs	251.21	0.931	270
Rails–rails	10.56	0.033	320
Walls–walls	134.32	0.447	300
Burners–burners	86.57	0.130	665
Steelslabs – steel slabs	67.50	0.153	421
Burners–rails	403.40	0.85	474
Walls–burners	7.20	0.016	450
Total computation time	1,162	3.59	322

Table 4 shows comparisons of the computation times on the CPU and the GPU for the whole furnace simulation. The column next to the GPU computation times contains the ratio of the CPU time to the GPU time. The ratio is varying between 270 and 640, depending on the category of heat-exchange factors (surface–surface, volume–surface or volume–volume). The GPU computation time is as well influenced by the distance between the objects, because of the length of discrete lines. When the distance is longer, the discrete line is taller and the mean absorption coefficient computation slows up the GPU execution more than the CPU execution.

At the end, the total GPU computation time is 3.6 s, for the whole furnace. The GPU time is then 320 times faster than the CPU time.

The computation time of one simulation being reduced to a few secondsmakes the dynamic modeling possible.

CONCLUSIONS

A fast implementation of the multiple absorption coefficient zonal method (MACZM) in three dimensions has been presented. First, efficient methods and algorithms have been discussed for the main parts of the MACZM implementation. The first part is the scene voxelization and the objects inside the scene. This part was implemented using 3-D scan-conversion algorithms that have linear computation complexity in terms of object size and short computation time. The second part is the most important phase of the algorithm because it is the most time-consuming. In this phase a mean absorption coefficient has to be computed over ray traversals, taking into account the value of a mean absorption in each of the voxelsof a discrete line corresponding to the continuous ray. In this article the 6-tripod line and the 6-parametric line algorithmswere adapted and compared. From the topological point of view, both algorithms guarantee the containment property of the line. Hence, the 6-parametric line algorithm has been selected for its advantage in computation time.

A CUDA implementation of the method has then been described. The method has been demonstrated to be massively parallel in a SIMD mode. An efficient solution for the optimization between the method parallelism (threading) and the use of the memory resources available on the GPU was discussed. Speed-up achieved

Downloaded by [University of California Santa Barbara] at 18:24 07 January 2016

on the GPU ranges from 300 to 600 times, using a NVIDIA Tesla C 1060 GPU and anIntel Xeon CPU E5507 at 2.27 GHz.

The MACZM has then been applied to a steel reheating furnace in order to compute the radiative heat transfer inside the furnace. A multigrid approach has been presented in order to minimize the computation times while keeping an acceptable accuracy level for all radiativeheat-exchange factors. A computation time of few seconds is achieved using the GPU implementation, with an average speed-up of 320 times. This short computation time will allow dynamic numerical real-time simulation of the reheating furnace, and will guarantee the product quality (product of the reheating furnace) and less energy consumption.

The speed of computations achieved using the MACZM will be very useful in many applications whereradiative heat transfersneed to be calculated. The short computation time and the accuracy of MACZM in nonisothermal, nonhomogeneous media give the possibility of simulating new radiative problems as well as optimization problems that were impossible before because of their long computation times.

Downloaded by [University of California Santa Barbara] at 18:24 07 January 2016

REFERENCES

1. W. W. Yuen, The Multiple Absorption Coefficient Zonal Method (MACZM), an Efficient Computational Approach for the Analysis of Radiative Heat Transfer in Multidimensional Inhomogeneous Nongray Media, *Numer. Heat Transfer B*, vol. 49, pp. 89–103, 2006.
2. W. W. Yuen, Definition and Evaluation of Mean Beam Lengths for Applications in Multidimensional Radiatve Heat Transfer: A Mathematically Self-Consistent Approach, *J. Heat Transfer*, vol. 130, no. 11, pp. 114507, 2008.
3. B. Ghannam, M. Nemer, K. El Khoury, and W. Yuen, Experimental Validation of the Multiple Absorption Coefficient Zonal Method (MACZM) in a Dynamic Modeling of a Steel Rehating Furnace, *Numer. Heat Transfer A*, vol. 58, pp. 1–19, 2010.
4. A. Kaufaman, and E. Shimony, 3D Scan-Conversion Algorithms for Voxel Based Graphics, *Proc. Workshop on Interactive 3D Graphics*, ACM Press, New York, 1986, pp. 45–75.
5. D. Cohen and A. Kaufaman, Scan-Conversion Algorithm for Linear and Quadratic Objects, in A. Kaufman (ed.), *Volume Visualization*, IEEE Computer Society Press, Los Alamitos, CA, pp. 280–301, 1991.
6. R. Yagel, D. Cohen, and A. Kaufman, Discrete Ray Tracing, *IEEE Comput. Graphics Appl.*, vol. 12, no. 5, 19–28, 1992.
7. A. Fujimoto, T. Takayu, and K. Iawa, ARTS: Accelerated Ray_Tracing System, *IEEE Comput. Graphics Appl.*, vol. 6, no. 4, 16–26, 1986.
8. J. Amanatides and A. Woo, A Fast Voxel Traversal Algorithm, in G. Marechal (ed.), *Proc. Eurographics 87*, Elsevier Science, Amsterdam, pp. 3–9, 1987.
9. J. Kahle, M. Day, H. Hofstee, C. Johns, T. Maeurer, and D. Shippy. Introduction to the Cell Multiprocessor, *IBM J. Res. Dev.*, vol. 49, no. 4/5, pp. 589C604, 2005.
10. J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. E. Lefhon, and T. J. Purcell, A Survey of General-Purpose Computation on Graphics Hardware, Eurographics 2005, *State of the Art Reports*, pp. 21–51, Aug. 2005.
11. NVIDIA, CUDA Technology, www.nvidia.com/CUDA, 2007
12. G. El Hitti, M. Nemer, K. El Khoury, and D. Clodic, The Re-plating Algorithm for Radiation Total Exchange Area Calculation, *Numer. Heat Transfer B*, vol. 57, pp. 110–125, 2010.

13. J. R. Mitchel, A Comparison of Line Integral Algorithm, Med. Phys., pp. 166–172, 1990.
14. D. Cohen and A. Kaufaman, 3D Line Voxelizationand Connectivity Control, *IEEE Comput. Graphics Appl.*, vol. 17, no. 6, pp. 80–87, 1997.
15. V. Boyer and J.-J. Bourdin, Fast Lines: A Span by Span Method, *Proc. Eurographics 99*, *Comput. Graphics Appl.*, vol. 18, no. 3, pp. 337–384, 1999.
16. V. Boyer and J.-J. Bourdin, A Faster Algorithm for 3D Discrete Lines, *Eurographics 98*.
17. J. E. Bresenham, Algorithm for Computer Control of a Digital Plotter, *IBM System J.*, vol. 4, no. 1, pp. 25–30, 1965.
18. V. Boyer and J.-J. Bourdin, Auto-Adaptive Atraight-Line Algorithm, *IEEE Comput. Graphics Appl.*, vol. 20, no. 5, pp. 67–69, 2000.
19. NVIDIA. CUDA Programming Guide, http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf, 2011.
20. M. J. Atallah, Algorithmic Techniques for Networks of Processors, in *CRC Handbook of Algorithms and Theory of Computation*, pp. 46 1–19, 1998.
21. D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, 2nd ed., Morgan Kaufmann, San Francisco, p. 751, 1998.

Downloaded by [University of California Santa Barbara] at 18:24 07 January 2016