Hindawi
International Journal of Aerospace Engineering
Volume 2018, Article ID 4827810, 12 pages
https://doi.org/10.1155/2018/4827810

Research Article

Combined Acceleration Methods for Solid Rocket Motor Grain Burnback Simulation Based on the Level Set Method

Ran Wei, Futing Bao, Yang Liu, and Weihua Hui

Science and Technology on Combustion Internal Flow and Thermal-Structure Laboratory, School of Astronautics, Northwestern Polytechnical University, Xi'an 710072, China

Correspondence should be addressed to Ran Wei; metorm@outlook.com

Received 16 October 2017; Accepted 2 April 2018; Published 30 April 2018

Academic Editor: Hikmat Asadov

Copyright © 2018 Ran Wei et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

A detailed study of a set of combined acceleration methods is presented with the objective of accelerating the solid rocket motor grain burnback simulation based on the level set method. Relevant methods were improved by making use of unique characteristics of the grains, and graphical processing unit (GPU) parallelization is utilized to perform the computationally intensive operations. The presented flow traced the expansion of burning surfaces, and then Boolean operations were applied on the resulting surfaces to extract various geometric metrics. The initial signed distance field was built by an improved distance field generating method, and a highly optimized GPU kernel was used for estimating the gradient required by the level set method. An innovative Boolean operation method, thousands of times faster than ordinary ones, was ultimately proposed. Performance tests show that the overall speedup was close to 15 on desktop-class hardware, simulation results were proven to converge to analytical results, and the error boundary was 0.25%.

1. Introduction

The working process of a solid rocket motor (SRM) is always accompanied by dramatic changes in the shape of its grain. As a result, burnback data are required in various SRM simulations. Ordinary grains usually burn uniformly, so the minimum distance method (MDM) [1, 2] can be used to calculate burnback data efficiently. For those SRMs in which erosive burning is encountered, a general surface tracing method is required to handle the arbitrarily distributed burnback speed. Specifically for dual-propellant grains, since the burnback speed distributes discontinuously, the tracing method has to be capable of strong discontinuity.

The level set method (LSM) [3] is a widely used and well-tested method for tracing general evolving surfaces. It has been proven to work well in SRM simulation [4–10]. The LSM processes geometry using implicit representation; that is, an $\mathbb{R}$-dimensional geometry is represented by an isosurface in $\mathbb{R}+1$ dimensional space. Implicit representation provides the capability to handle complex topological changes but also requires a larger data structure and more computation [11]. Full three-dimensional (3D) LSM burnback simulation may consume hours of computation time. In specific tasks, such as optimization [4] or data fitting, the simulation has to be repeated for hundreds or thousands of times to obtain the required data, making computational efficiency even more crucial in these tasks.

The flow of burnback simulation based on a level set can be summarized as follows: (1) generating the initial signed distance field (SDF) on a computation grid, (2) tracing the geometry formed by the expansion of burning surfaces via the LSM, and (3) applying Boolean operations to produce corresponding burnt grain or other required geometries. The reason why we choose tracing the expansion of burning surfaces rather than tracing the burning grain directly is that such flow causes less numerical dissipation, evades extending the definition of the burning rate, and allows one to easily distinguish burning surfaces from nonburning ones. The issue of Boolean operation will be further discussed in Section 2. With the aim of boosting the simulation speed,

accelerating any of the above subflows makes sense. Here, the subflows are discussed one by one.

An initial SDF is required to start the LSM. Constructing a SDF using a brute-force method on high-resolution grids may take hours. Previous research [12, 13] provides efficient algorithms for this task. In [1], the algorithm introduced by [12] is used to generate the required SDF. The method discretizes any geometry into sufficiently small triangles and calculates the SDF using OpenGL rendering functions. On the other hand, the algorithm relies on the OpenGL shading language, making it difficult to improve or run in server environments. In [13], a scanning-based SDF-generating method was proposed. This method is not bound to any specific architecture and thus can be easily improved. In this paper, the method of [13] is transplanted to NVIDIA's CUDA architecture to make use of the computing power provided by modern GPUs. Since the follow-up process requires two SDFs to work, the method is improved to generate all required SDFs simultaneously using a reduced level of computation.

Previous efforts to accelerate the tracing process of the LSM can be divided into two categories: narrowband methods [14] and avoiding reinitialization [15–18]. In [14], the authors proposed a narrowband method, allowing one to skip the computation of the nodes that are far from the target surface. Narrowband methods do significantly accelerate LSM simulation in many cases. However, the technique lost its role in this paper due to the usage of a cuboid mesh (see the end of Section 2), so the technique is not adopted here. In [15], a method was proposed to keep the implicit representation as an exact signed distance function during the evolution. Reinitialization is thus totally eliminated from the simulation flow. However, the method cannot handle dual-propellant grains because its accuracy is extremely sensitive to the error of gradient estimation of the evolving speed field, but in such grains, the gradient of discontinuous speed fields cannot be accurately estimated. In [16–18], various methods were proposed to avoid reinitialization in an active contour method for image segmentation. While working well in image segmentation, these energy-function-based methods cannot be used for burnback simulation, because the propagation speed and time components in such methods cannot be mapped to physical solid rocket grains. To summarize, the standard LSM is currently irreplaceable in the field of burnback simulation.

From another perspective, the acceleration of the LSM can still be achieved by improving computing power. Preliminary tests show that the performance bottleneck of LSM lies in estimating the gradient of the implicit representation. In order to handle the discontinuity caused by dual-propellant grains, in this work, we used weighted essentially nonoscillatory (WENO) schemes [19] to estimate the gradient. There have been studies on implementing WENO schemes on GPUs [20, 21]. Both of the latter studies provide a significant speedup compared to the CPU version. The major difference between them lies in their memory accessing model: in [20], source data are fetched from texture memory, while in [21], the data are preloaded into shared memory. In this paper, considering that memory access in WENO is complex but not random, the shared-memory scheme is adopted. The proposed computing pattern fuses the computation of three axes, further reducing memory transfers without increasing shared-memory occupancy.

The last step in the simulation is extracting the resulting geometries and geometric metrics. Depending on the ultimate purpose of the burnback simulation, there may be various geometric metrics to extract. For simple 0D interior ballistics analysis, we need to compute the total burning area in each simulation step. In 1D interior ballistics analysis, the burning area in each discretized segment is needed. For 3D computational fluid dynamics (CFD) simulation or visualization, the shape of the fluid boundary is required. In previous research [5, 7], a method proposed in [22] is commonly used to retrieve the area and volume of geometries from implicit representations. While the method is easy to implement and works reliably, it is not adopted in this paper due to the fact that it is unable to provide explicit geometry in CFD simulations and that it introduces $O(1)$ error in multidimensional cases [23]. Owing to accuracy and flexibility concerns, in this paper, a triangular surface mesh of the resulting geometry is first generated using the marching cube (MC) method [24], and then other geometric operations (mainly Boolean ones) are applied to this surface mesh to obtain the required metrics. Boolean operations on triangular meshes are usually computationally expensive, but an innovative Boolean operation method is proposed in this paper to solve the issue. The proposed Boolean operation uses SDF to reduce the complexity compared to corresponding ordinary operations, can be embedded into the flow of the MCs, and can take advantage of the hardware-accelerated interpolation of GPU to further improve its efficiency.

Our main contributions in this work can be summarized as follows: (1) an improved SDF-generating technique is presented, (2) a highly optimized GPU kernel is designed to estimate the gradient required by LSM, and (3) an extremely fast Boolean operation method is proposed to obtain the needed geometries from the LSM output. All of the above methods are combined to construct an efficient, accurate, and flexible framework for the solid rocket motor grain burnback simulation. Results of the simulation flow are proven to converge to corresponding analytical results, and the overall acceleration is significant.

2. Level Set Method

In this section, we use a finocyl SRM grain to demonstrate the simulation framework. Extension to other types of grains is easy. Figure 1 shows two sectional views of the grain. The thick black lines denote the nonburning surfaces of the grain, the thick red lines mark the burning surfaces, and the thin blue grid is the computational mesh. We use uniform Cartesian grids in the simulation. Cylindrical meshes may seem to better fit the shape of grains, but they would require time-consuming transformation and provide no obvious advantage in describing complex shapes (e.g., fins or stars). In Figure 1, each point on the burning surfaces burns and moves inwards at the speed of the local burning rate.

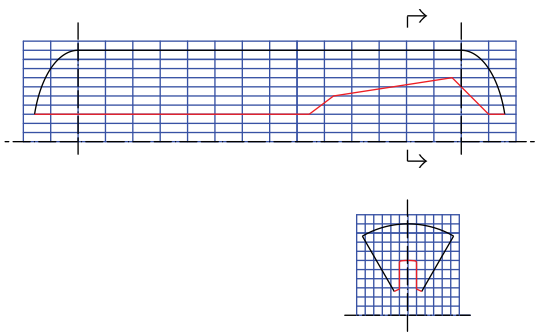


Figure 1: Computation grid.

The straightforward practice is to simply trace the shrinking (i.e., burning) process of the grain. However, such an approach is not optimal due to the following disadvantages:

(1) Since burning rate is defined on the burning surface instead of on the grid nodes, its definition has to be extended to the grid nodes via a partial-differential-equation- (PDE-) based method [25], which is obviously time-consuming.

(2) Nonburning and burning surfaces are blended in an implicit representation. Therefore, extracting geometric metrics of burning surfaces would be difficult.

(3) As the burnback continues, sharp corners may emerge near fin-like structures (see Figure 2(a)). Numerical dissipation near these sharp edges is often more severe than that in any other part of the grain.

A simple solution for the above issue is to trace the expansion of the burning surfaces. Despite the fact that the flame actually only spreads inwards on the grain, we imagine that the flame starts from initial burning surfaces and spreads to all directions like waves; the space swept by the flame can be defined as the "flame domain," which is shown by red lines in Figure 2(b). The expansion speed of the flame domain is naturally defined by the burning rate of local propellant, so there is no need to expand the definition. The expansion rate outside the grain can be simply set to the slowest burning rate among that of all propellants. Whenever the shape of burnt grain is needed, the shape can be easily obtained by applying a Boolean operation, that is, subtracting the produced flame domain from the unburnt grain (see Figure 3). Burning and nonburning surfaces can be distinguished by their source geometry. Since the flame domain keeps expanding during simulation, few sharp edges would emerge (see Figure 2(b)).

The flame domain expansion process can be easily handled by the LSM. We use LSM following (1), where $\phi(\vec{x})$ is a SDF defined on a uniform Cartesian grid around a SRM grain and V_n is the local burning rate. We use the WENO5 scheme to estimate the gradient and Godunov's scheme [26] to determine the corresponding domain of dependence. The total variation diminishing Runge–Kutta (TVD-RK) scheme is used for time discretization. Other foundational details regarding the LSM will not be described in this article for simplicity.

$$\phi_t + V_n \cdot |\nabla\phi| = 0. \tag{1}$$

An obvious characteristic of SRM grains is that most have a large length-radius ratio and that the shape-changing rate along a grain's axial direction is more moderate than that along the radial direction. Therefore, using a finer mesh in the radial direction than that in the axial direction is reasonable. In the proposed framework, all of the combined methods allow the mesh grid to be cuboid, so we can set a larger grid length in the axial direction to balance accuracy and efficiency.

One side effect of the cuboid mesh is that a narrowband technique [14] is no longer effective. A narrowband technique requires roughly 10 grids to be normally calculated around the zero level set, so the width of the narrow band would be roughly 2×10 times the axial grid spacing (assumed as L_x). Since L_x can be several times larger than the grid spacing in the radial direction, a narrow band of $20L_x$ width covers dozens of grid nodes in the radial direction. Unfortunately, less than 128 radial grids already provide enough accuracy in most cases. To summarize, if a cuboid mesh is used in the simulation, most of the grid points will appear in the band, rendering the narrowband technique ineffective.

3. Implementation

3.1. Efficient SDF Generation. Our framework requires SDFs of both the initial burning surface and the entire model. The SDF of the initial burning surface will be used in LSM evolution, and that of the entire model will be used to accelerate the Boolean operation (see Section 3.3).

To construct a SDF, we need the minimum distance field (MDF) value and sign information for each grid node. Calculating the MDF via a brute-force method results in $O(G \cdot E)$ complexity, where G is the number of grid nodes and E is the number of geometric elements. In [13], an efficient scanning technology called DiFi was proposed to calculate the MDF of any model. We have improved DiFi so it generates the two required SDFs simultaneously.

The flow of DiFi in the 2D case is illustrated in Figure 4. Expanding to the 3D case is straightforward. A scanning line moves in one direction and divides the geometric elements (which can be of any type) into three categories: approaching, intersecting, and receding. The approaching group is shown in blue and contains the elements not reached by the scanning line. The intersecting group is shown in red and holds the elements that have been passed by the scanning line and thus may contribute to the MDF near the scanning line. If an element no longer contributes to the MDF near the moving scanning line, it will be moved to the receding group, which is shown in green. During each scanning step, we iterate through the intersecting group and search the shortest distance from each element to each nearby node. By performing two scans at opposite directions (head to end, then end to head) and taking the minimum result, we can obtain

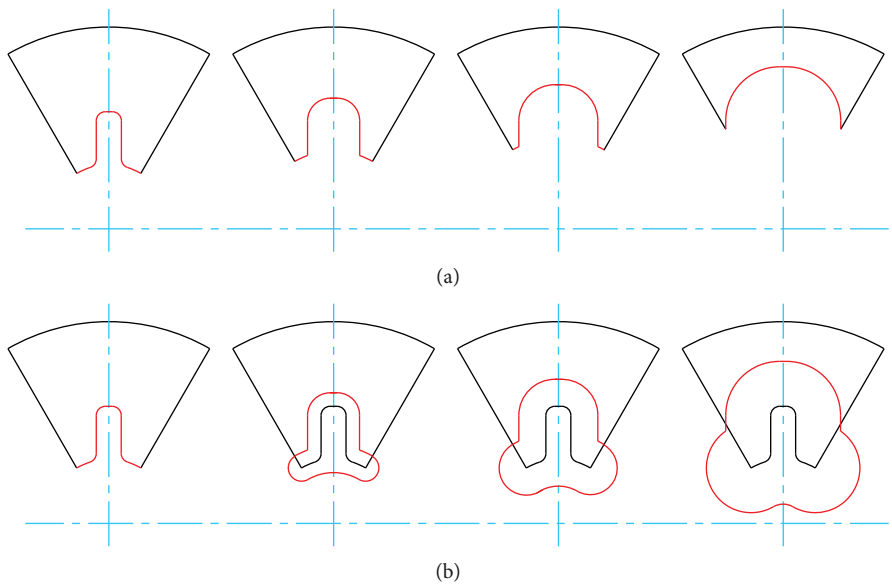


Figure 2: Avoiding sharp edges.

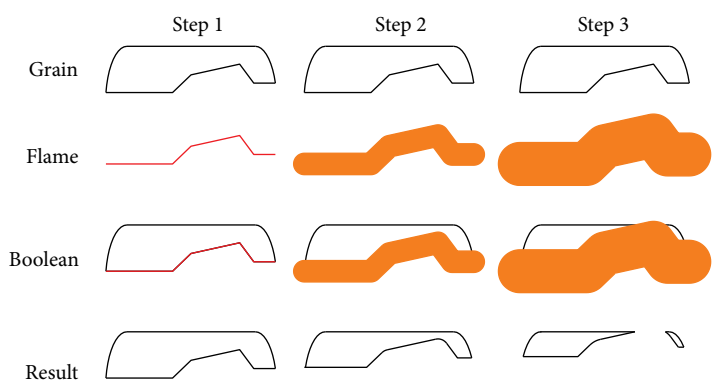


Figure 3: Boolean subtraction operation.

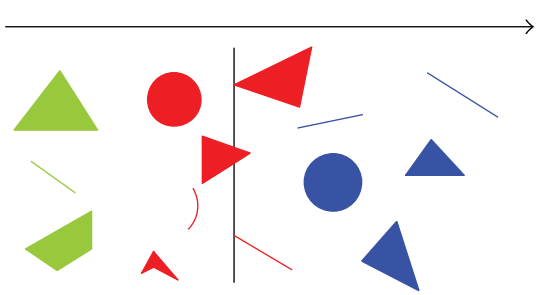


Figure 4: DiFi demonstration.

an exact MDF value. A detailed procedure and proof can be found in [13].

Noting that each element of the burning surface has an identical copy in the corresponding grain model, it is unnecessary to process these elements using two independent scans. During the abovementioned iteration among the intersecting group, the improved DiFi implementation maintains two required MDFs simultaneously: if the element being processed belongs to the burning surface (so it must also belong to the grain), the minimum result is written to the MDFs of both geometries; otherwise, the result is written to the MDF of the grain only. By the end of scanning, both MDFs are ready for follow-up procedures.

On the base of the generated MDFs, sign information on each node is still required to finally build a SDF. Using the CPU library CGAL [27], it is easy to determine whether a point is inside the grain and to assign negative signs to inside nodes. Noting that generating a MDF value and a determining sign is two independent operations carried out on different hardware, we can launch the two operations in parallel to improve efficiency. Moreover, the procedure for determining each node is also independent, and thus, CGAL can be run across multiple CPU cores to further improve efficiency.

3.2. Weighted ENO Scheme on a GPU. In this paper, we use the WENO5 scheme to estimate the gradient of level set fields, and in this section, we focus on accelerating the WENO5 computation using a GPU.

The WENO5 computation on each node is independent of that on other nodes, so it is easy to distribute the task to multiple GPU cores. However, to achieve the best performance, the memory access pattern of the program must be carefully designed. Unlike CPUs, the time spent to access the GPU RAM (or global memory) is quite long. Since the WENO5 computation contains stencils that include three neighboring nodes on both sides, if all values are read directly from the global memory, each element would have been loaded repeatedly seven times on each axis. There are two ways to reduce memory transmission: fusing the computation of the three axes and avoiding the redundant memory access.

To estimate the gradient on a specific node, the numerical difference values on all three axes are required. The practice detailed in [20] is to launch three independent GPU kernels for the three respective axes. Accordingly, a similar loading procedure must be repeated at least three times. In the LSM, however, only the module of the gradient is needed. This reminds us to use one giant kernel to compute the three numerical differences. The obtained values are then fed to Godunov's scheme and modulo operation. Because all of the subtasks are fused, there is no need to save or fetch intermediate results to or from the global memory, and thus, global memory transmission is minimized.

In addition, shared memory can be used to eliminate redundant accessing of the global memory. Compliant shared-memory access is much faster than global memory access. To reduce global memory access, we can use a block of GPU threads to preload corresponding elements into the shared memory. Each GPU thread then reads its input from the shared memory instead of the global memory. As the capacity of the shared memory is limited, preloading the entire array into the shared memory is impossible, so the field is divided into a plurality of blocks and processed one by one. The problem is that the dependence region of one specific block is not well aligned, as shown in Figure 5. Assuming that the size of the CUDA thread block is $[n, n, n]$, to calculate the numerical difference on all three axes, $n^3 + 6 \times 3n^2$ values are needed. It is difficult to design an efficient access pattern on such a cross-shaped region.

Noting that the numerical difference on each axis is independent, we do not have to keep all of the nodes shown in Figure 5 in the shared memory all the time. Cache and calculation can be performed in an axis-by-axis manner. In this instance, only $n^2(n+6)$ shared memory is required, and the dependence region is aligned on two axes with the block. Moreover, less shared-memory usage leads to more active threads on the GPU and higher overall performance. In most cases, $n > 6$, so $n^2(n+6) < n^3$, meaning that we can load all required data within two loading cycles using n^3 threads. Figure 6 illustrates the proposed loading flow on one row of the thread block. For the first axis to be loaded, both steps in Figure 6 must be performed. For the latter two axes, as the elements corresponding to the first step are already in the shared memory, only the second step must be performed.

The above discussion can be summarized as follows: In the simplest method, neither caching nor fusing is applied. In such a method, on each axis, each thread has to load seven elements and write two results (positive and negative numerical differences). Then, each thread for Godunov's scheme and modulo operation loads 3×2 intermediate results and writes one final result. Fusing the computations eliminates the saving and loading of intermediate results. Simply caching the entire dependence region costs $n^3 + 3 \times 6n^2$ shared memory but largely reduces loading operations. By computing axis by axis and rearranging the shared memory, shared-memory usage can be reduced. A comparison of these methods is shown in Table 1. Other details in GPU implementation are straightforward and will not be discussed here for simplicity.

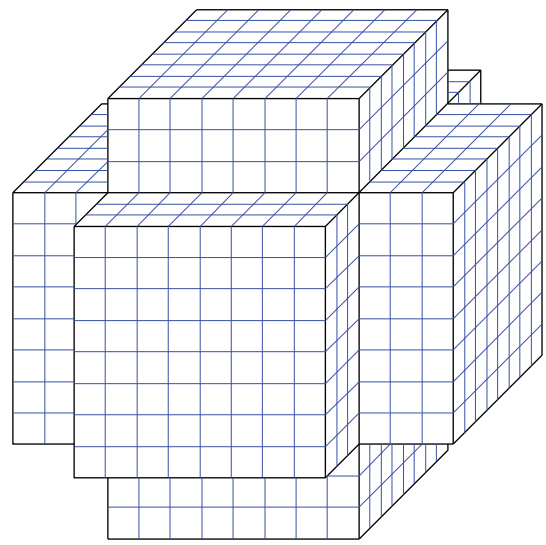


Figure 5: Dependence region of an 8^3 block.

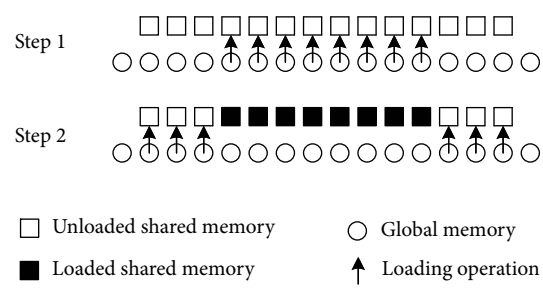


Figure 6: Loading flow on one axis.

3.3. Efficient Boolean Operation. As discussed in Section 2, a Boolean operation is required to acquire the real geometry of the grain. A straightforward way of accomplishing this is to build a triangular mesh representation of the grain via the MC method and then to perform the Boolean operation on the obtained mesh. Assuming that the triangular mesh representing the flame domain generated by MC is S and the original grain is G, the burning surfaces can then be expressed by $S \cap G$, the burnt grain can be expressed by $G - S$, and the 1D discretized burning area data requiring 1D interior ballistics simulation can be obtained by $B_i \cap (S \cap G)$, where B_i is a series of cuboids arranged along the x-axis.

The CGAL library provides robust Boolean algorithms on triangular meshes. However, Boolean operations on triangular meshes are usually slow. According to profile data,

Table 1: Complexity of all calculation methods on GPU n^3 nodes in a cubic region, with n^3 threads.

Method	Shared-memory usage	Loading cycles	Loading transmission	Writing transmission
No cache, not fused	0	27	$27n^3$	$7n^3$
No cache, fused	0	21	$21n^3$	n^3
All cache, fused	$n^3 + 18n^2$	4	$n^3 + 18n^2$	n^3
Partial cache, fused	$(n+6)n^2$	4	$n^3 + 18n^2$	n^3

the cost of a CGAL-based Boolean operation is close to that of LSM evolution. In [1], the SDF of G (assumed as ϕ_G) is used to quickly estimate $S \cap G$. Their practice is to use the sign of ϕ_G to determine whether each of the grid nodes is outside G. Then, in the follow-up MC process, all the cubes that contain any outside nodes are excluded, and the triangles generated from the remaining cubes are used as the estimation of $G \cap S$. Such a practice is fast but introduces error that can cause slight oscillation in the resulting burning surface area (see Section 4.1). Since the average area of the discarded triangles is proportional to the grid surface area, the error introduced is proportional to ${L_g}^2$, where L_g is the scale of computation grids. However, we can improve the method to eliminate the error without downgrading its performance.

Here, we present the process for obtaining $S \cap G$ to demonstrate the proposed SDF-based Boolean operation; expansion to other types of Boolean operations is straightforward. Considering Figure 7, where the blue curve is the boundary of G and the red triangles are part of S, points A and B are outside G while C, D, and E are inside G. Therefore, the boundary of $G \cap S$ in Figure 7 is $C\ \widehat{OPQR}\ E$, which can be estimated using the polyline $COPQRE$. From the definition of the SDF, it is obvious that $\phi_G(O) = \phi_G(P) = \phi_G(Q) = \phi_G(R) = 0$. As the coordinates of A, B, C, D, and E are already known, $\phi_G(A)$, $\phi_G(B)$, $\phi_G(C)$, and $\phi_G(D)$ can be easily acquired using linear interpolation, and then the coordinates of O, P, Q, and R can be calculated using another linear interpolation. Finally, by discarding outside triangles and dividing the quadrilaterals $OPDC$ and $QRED$ into triangles, a triangular estimation of $G \cap S$ can be obtained. Noting that the above operations on each triangle are independent, in order to further speed up the SDF-based Boolean operation, the discard and divide operations can be embedded into a standard MC method and be performed cube by cube.

Among the above flows, two operations that introduce error are the curve estimation $\widehat{OPQR}$ using the polyline $OPQR$ and linear interpolation. The error introduced (assumed as E_m) is obviously larger than that of the CGAL-based operation (assumed as E_{CGAL}). However, similar estimations are already included in a standard MC method, and therefore, the error contained in S is of the same order as E_m. As a result, the overall error of the SDF-based Boolean operation is E_m, while the overall error of the MC- and CGAL-based Boolean operations in a series is $E_m + E_{\mathrm{CGAL}} \approx E_m$. In other words, using a SDF-based Boolean operation to replace the CGAL-based one does not degrade overall accuracy.

The above process contains many random reads and trilinear interpolations, which happen to be the strengths of texture memory. Using the hardware-based acceleration for random read and interpolation operations of texture memory, performance of the SDF-based Boolean operation can be further improved.

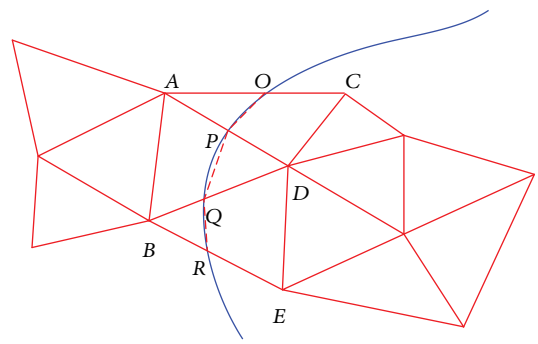


Figure 7: SDF-based Boolean operation.

The accuracy and complexity of various Boolean operations are shown in Table 2. The row labeled "Triangle filtering" refers to the method used in [1]. It is clear that the proposed SDF-based Boolean operation exhibits the best overall performance.

4. Results

4.1. Regression Test. Since the LSM is a mature method, we can verify the result of our implementation by comparing it with previous implementations. In this subsection, we use several commonly used samples to verify the LSM evolution module of our implementation. The reference results are produced using ToolboxLS [28]. In addition, two sample grains are used to verify the entire framework, and the referenced results are analytical results.

We use the maximum regression error, which is defined by (2) (where Δx denotes the scale of mesh grids), to measure the error of our LSM implementation with respect to ToolboxLS.

$$E_{mr} = \max_{\left\{\vec{x}_i \mid \phi\left(\vec{x}_i\right) < 5\Delta x\right\}} \left|\phi\left(\vec{x}_i\right) - \phi_{\mathrm{ToolboxLS}}\left(\vec{x}_i\right)\right|. \quad (2)$$

The maximum regression errors of three common samples are shown in Table 3: "constant convection" describes a sphere moving in a constant direction at constant speed; "linear convection" describes a sphere moving along an arc at constant speed; and "star reinitialization" starts from an inaccurate 2D SDF of a star and regulates it to an exact SDF. The "resolution" column shows the resolution of the computation grid on one axis. From the table, it is clear that

Table 2: Complexity and accuracy of various Boolean operations: n_S and n_G triangles in S and G, with n_T threads and n_C cubes.

Method	Hardware	Error	Time complexity
MC & CGAL	CPU	E_m	$O(n_S n_G)$
Triangle filtering	GPU	$E_m + O(L_g{}^2)$	$O\frac{n_C}{n_T}$
SDF-based	GPU	E_m	$O\frac{n_C}{n_T}$

Table 3: Regression errors at various resolutions.

Resolution	Constant convection	Linear convection	Star reinitialization
32	$8.386e-6$	$1.234e-5$	$9.244e-6$
64	$6.439e-6$	$6.051e-6$	$7.256e-6$
128	$2.245e-6$	$2.638e-6$	$3.157e-6$

the results produced by our implementation fit well with the reference results in all cases.

The above results indicate that our GPU implementation correctly traces the target geometry. As long as the "flame domain-Boolean operation" mechanism is validated, it can be confirmed that the framework gives correct results. We use sample SRM grains that have a clear analytical burnback solution to perform the validation. The two chosen sample grains are a nonuniform tube grain and a uniform star grain (see Figure 8). The normal propagation speeds V_n (see (1)) of the two sample grains are defined by (3), where collections S and F are the slow- and fast-burning regions, respectively, and $R_{sf} \in (0, 1)$ is the ratio of the burning speed of the slow- and fast-burning propellant.

$$V_{n|\text{Tube}}(\vec{x}) = \begin{cases} R_{sf}, & \vec{x} \in S, \\ 1, & \vec{x} \in F, \\ R_{sf}, & \vec{x} \notin S, \vec{x} \notin F, \end{cases} \tag{3}$$

$$V_{n|\text{Star}}(\vec{x}) \equiv 1.$$

The analytical solution of the sample grains can be summarized by burning area-propagation distance functions. Equation (4) [29] and Figure 9 demonstrate the analytical solution of a nonuniform tube grain, where A_s and A_f are the burning areas of slow- and fast-burning propellants, respectively, and other symbols are defined by Figure 9.

$$\begin{aligned} \alpha &= \sin^{-1} R_{sf}, \\ e_s &= e_f R_{sf}, \\ A_s(e_s, e_f) &= 2\pi(r_0 + e_s)\left[L_s - (e_f - e_s)\tan\alpha\right] \\ &\quad + \pi(e_s + e_f)\frac{(e_f - e_s)}{\cos\alpha}, \\ A_f(e_f) &= 2\pi(r_0 + e_f)L_f. \end{aligned} \tag{4}$$

Equation (5) [30] describes the analytical solution of star grains, where L is the length of the grain, and other symbols are defined by Figure 10.

$$\begin{aligned} T_e &= R - l - r, \\ h &= \frac{\sin((\epsilon\pi)/n)}{\cos(\theta/2)} l - r, \\ \frac{A(e)}{2L} &= \begin{cases} h + r + l(1-\epsilon)\frac{\pi}{N} + (e+r)\left(\frac{\pi}{2} + \frac{\pi}{N} - \frac{\theta}{2} - \cot\frac{\theta}{2}\right), & 0 \le e \le \min(h, T_e), \\ l(1-\epsilon)\frac{\pi}{N} + (e+r)\left[\frac{\pi}{N} + \sin^{-1}\left(\frac{1}{e+r}\sin\frac{\epsilon\pi}{N}\right)\right], & \min(h, T_e) < e \le T_e, \\ (e+r)\left[\cos^{-1}\frac{(e+r)^2 + l^2 - R^2}{2(e+r)l} - \frac{\pi}{2} + \frac{\epsilon\pi}{N} - \cos^{-1}\left(\frac{l}{e+r}\sin\frac{\epsilon\pi}{N}\right)\right], & e > T_e. \end{cases} \end{aligned} \tag{5}$$

We measure the accuracy by the average relative error of the predicted burning surface area to the average burning surface area (defined by (6)).

$$E = \frac{\sum_{e_i} \left| A_{\text{LSM}}(e_i) - A_{\text{Analytical}}(e_i) \right|}{\sum_{e_i} A_{\text{Analytical}}(e_i)}. \tag{6}$$

The error data of the two samples at various grid resolutions are shown in Table 4. The "fast" and "slow" columns in Table 4 mean the fast- and slow-burning parts, respectively, in the nonuniform grain. It is clear from Table 4 that the results correctly converge to the analytical results as the simulation resolution increases.

In Figure 11, we have presented the SDFs of the sample grains, which remain constant during evolution. Figure 12

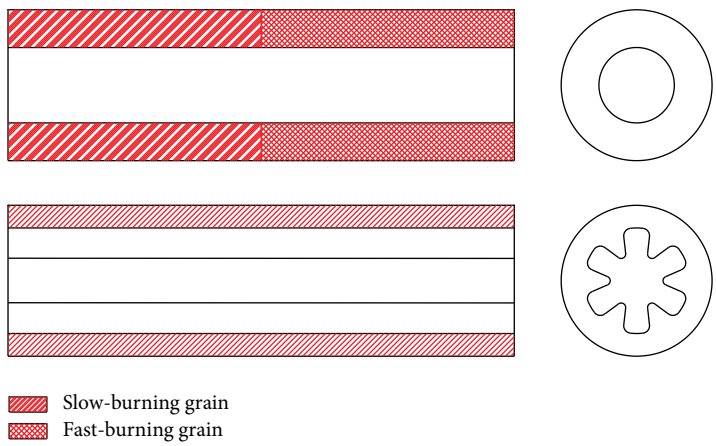


Figure 8: Sample grains.

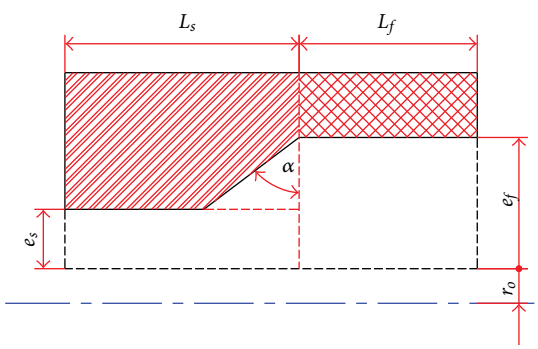


Figure 9: Burnt nonuniform tube grain.

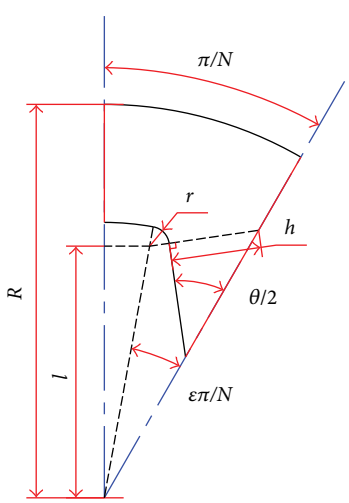


Figure 10: Geometric parameters of star grain.

Table 4: Average relative error at various resolutions.

Resolution	Tube		Star (%)
	Fast (%)	Slow (%)	
[96,48,48]	0.4276	0.5002	0.2287
[128,64,64]	0.2359	0.2802	0.1965
[256,96,96]	0.1912	0.2101	0.1127

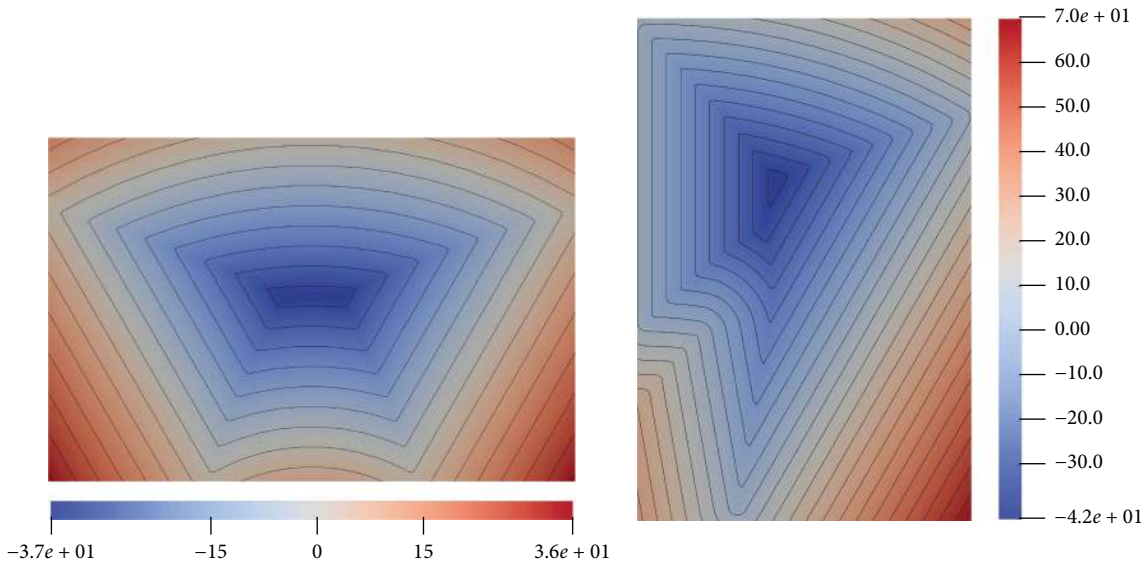
demonstrates the transient contours of flame domains during evolution. Figure 12(a) shows the contour near the interface of slow- and fast-burning propellants. The α angle measured from Figure 12(a) is in good agreement with the prediction made by (4). Figure 12(b) shows 3 radial sections belonging to the slow-burning propellant, the interface, and the fast-burning propellant, respectively. It can be seen from Figure 12(b) that V_n outside the grain equals to the burn speed of the slow-burning grain. Figure 12(c) shows the transient contours of the star grain. Apparently, LSM degrades to the minimum distance method when V_n is uniform across the field. All above results imply that our framework works well on SRM grains.

The superiority of a SDF-based Boolean operation compared to a triangle filtering method is shown by the star sample. In Figure 13, an enlarged view (the first third of the curve) of the results produced by both methods [1] is shown. It is clear that the output of the SDF-based Boolean operation is smoother and more accurate than that of triangle filtering.

4.2. Performance Test. In this subsection, we analyze the performance of our framework by profiling each GPU kernel and equivalent CPU implementation. The following referenced CPU LSM code is from ToolboxLS [28], and the CPU Boolean operation code is from CGAL [27].

The averaged execution profile data are shown in Table 5. The kernel WENO5-Module computes $|\nabla\phi|$ using the WENO5 scheme and Godunov's scheme, the kernel DiFi-Slice is repeatedly called in the SDF-generating stage to calculate the distance field on one slice of the field, and the kernel MC-Boolean generates the current burning surface and calculates its area using the MC- and SDF-based Boolean methods proposed in Section 3.3.

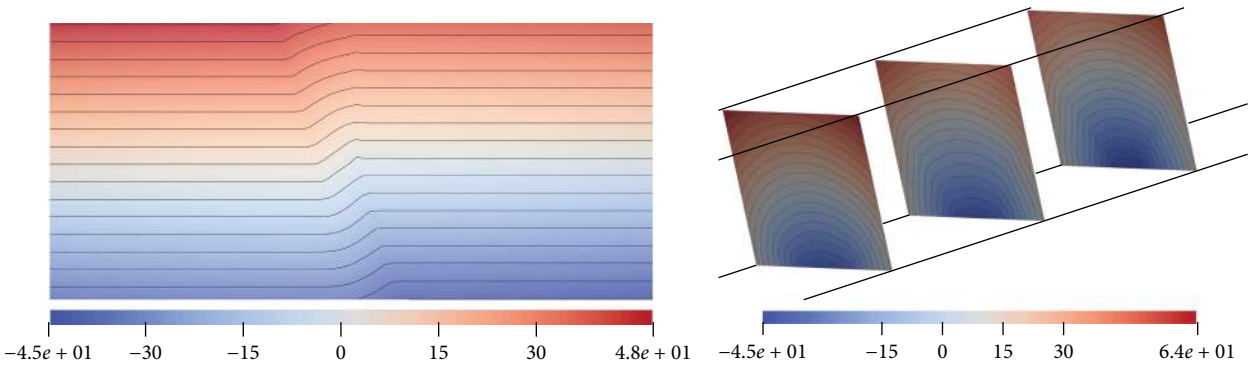
It is clear from the table that all subtasks benefit from GPU parallelization. Since the NVIDIA K2200 has poor double-precision computation capability (40 GFLOPS nominal peak), the acceleration would be much more significant on professional computing GPUs such as Tesla. The acceleration in the "MC-Boolean" column is amazing because the SDF-based Boolean operation has highly reduced complexity compared to the CGAL-based one. The speedup rate of MC-



(a) 1/6 of the tube grain

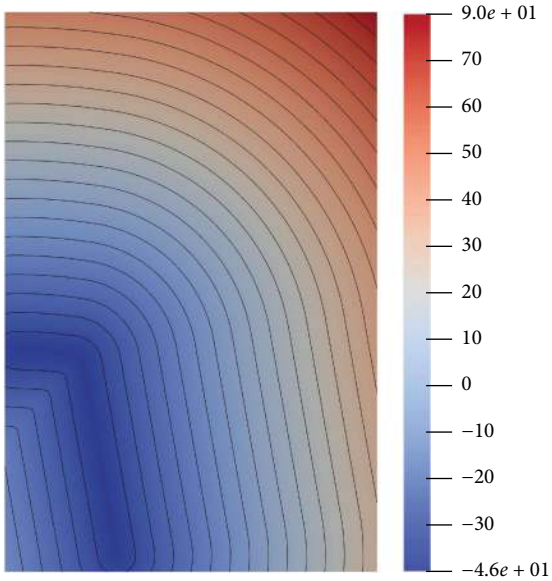
(b) 1/12 of the star grain

Figure 11: Contour of SDF of sample grains.



(a) Tube grain: axial section at $e_f = 45$

(b) Tube grain: radial sections at $e_f = 45$



(c) Star grain: radial section at $e = 46$

Figure 12: Transient contour of SDF of flame domains.

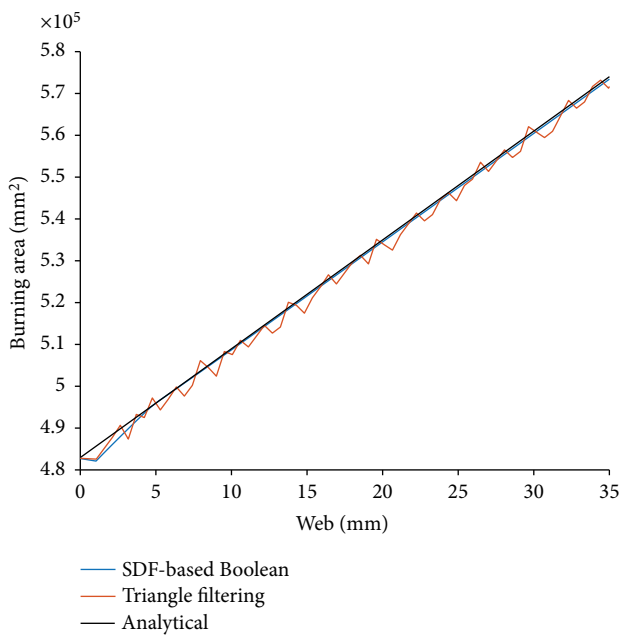


Figure 13: Result of star grain.

Table 5: Comparison between CPU and GPU execution time (per call). CPU, Intel i54570; GPU, NVIDIA K2200.

Grid	CPU time	GPU time	Speedup
		WENO5-Module	
64, 32, 32	$1.208e-2$	$1.183e-3$	10.21
96, 48, 48	$3.662e-2$	$3.935e-3$	9.306
128, 64, 64	$9.511e-2$	$9.305e-3$	10.22
256, 96, 96	0.4286	$4.180e-2$	10.25
		DiFi-Slice	
64, 32, 32	$4.911e-4$	$6.443e-5$	7.623
96, 48, 48	$1.750e-3$	$2.597e-4$	6.739
128, 64, 64	$5.443e-3$	$7.566e-4$	7.194
265, 96, 96	$1.119e-2$	$1.553e-3$	7.204
		MC-Boolean	
64, 32, 32	0.7985	$1.965e-4$	4064
96, 48, 48	1.775	$4.703e-4$	3774
128, 64, 64	2.618	$9.110e-4$	2874
256, 96, 96	4.762	$3.886e-3$	1226

Table 6: Comparison between CPU and GPU execution times (per simulation): CPU: Intel i54570; GPU: NVIDIA K2200.

Kernel	CPU		GPU	
	Time	Portion (%)	Time	Portion (%)
WENO5-Module	1070	58.29	104.3	83.73
DiFi-Slice	5.707	0.3110	0.7920	0.6358
MC-Boolean	742.9	40.47	0.6060	0.4865
Other	17.03	0.9277	18.87	15.15
Total	1836	100.0	124.6	100.0

Boolean slowly declines as the grid grows finer, due to the fact that the computation growth of two implementations is not proportional. Time spent by the SDF-based implementation is close to that by a simple MC operation. However, the percentage of MC in the execution time of CGAL-based implementation decreases as the grid density grows, meaning that the acceleration of the SDF-based Boolean operation would be covered up. Fortunately, the speedup remains significant within a reasonable grid scale.

Further, we run the framework at [256,96,96] resolution to analyze the overall performance gain and locate the bottleneck. The profile data are shown in Table 6. The "other" item in Table 6 covers the execution time of all other operations (other element-by-element computation, input and output, simple CPU computation, etc.).

The overall speedup ratio is 14.74. The WENO5-Module and MC-Boolean play a major role in the overall speedup. The WENO5-Module contains many floating-point operations and is repeatedly called by the TVD-RK scheme. As a result, the WENO5-Module occupies a large portion of the total execution time, and the corresponding acceleration contributes greatly to overall speedup. For the MC-Boolean, the GPU version takes nearly no time, while the time spent by the CPU version is close to that by the WENO5-Module.

From the "GPU" column in Table 6, it is easy to see that for the current GPU implementation, the only performance-critical kernel is the WENO5-Module. Analysis of WENO5 and the Godunov scheme shows that the minimum required computation is 400 FLOPS per result element. Using the data provided in Table 5, we can easily calculate that the average performance of the WENO5-Module is 22.44 GFLOPS, which corresponds to 56.10% of the nominal peak. Considering that there are inevitable extra computations (e.g., computing memory index) and branch code, we conclude that the WENO5-Module is well optimized.

5. Conclusions

In order to accelerate the LSM-based SRM burnback simulation, we have developed or improved the following combined methods that jointly improve the overall efficiency: (1) the concept of flame domain is introduced to reduce numerical dissipation and therefore allows a relatively coarser grid; (2) cuboid grids are allowed, so a coarser grid can be used on a specific axis when necessary; (3) the improved DiFi module generates all required SDFs within one scan; (4) the WENO5 module estimates the gradient of level set fields using well-designed memory accessing patterns; (5) the MC module is fused with the innovative SDF-based Boolean operation; and (6) all computation-intensive operations are performed on a GPU to further improve the performance.

Surface tracing regression testing shows that the proposed framework gives exactly the same result as previously published LSM code. Further regression testing shows that the "flame domain-Boolean operation" mechanism works as intended. The overall error on sample grains is less than 0.25%. Efficiency testing shows that the overall performance is roughly 15 times higher than that using equivalent CPU code. In particular, the fused MC- and SDF-based

Boolean operation is thousands of times faster than an ordinary method.

The implemented code executes within minutes on desktop-class hardware, making it suitable for adoption in iterative execution roles such as in optimization systems. Further work will focus on embedding CFD simulation features into the current framework.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] Y. Liu, F. Bao, Q. Cai, H. Hu, and J. Du, "GPU computing architecture-based discrete voxelized burning surface calculation method for complex SRM grains," *Journal of Solid Rocket Technology*, vol. 34, no. 1, pp. 18–22, 2011.

[2] M. A. Willcox, M. Q. Brewster, K. C. Tang, and D. S. Stewart, "Solid propellant grain design and burnback simulation using a minimum distance function," *Journal of Propulsion and Power*, vol. 23, no. 2, pp. 465–475, 2007.

[3] S. Osher and J. A. Sethian, "Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations," *Journal of Computational Physics*, vol. 79, no. 1, pp. 12–49, 1988.

[4] W. Dong-Hui, F. Yang, H. Fan, and Z. Wei-Hua, "An integrated framework for solid rocket motor grain design optimization," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 228, no. 7, pp. 1156–1170, 2014.

[5] C. Yildirim and H. Aksel, "Numerical simulation of the grain burnback in solid propellant rocket motor," in *41st AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit*, Tucson, AZ, USA, 2005American Institute of Aeronautics and Astronautics (AIAA).

[6] K. Albarado, A. Shelton, and R. Hartfield, "SRM simulation using the level set method and higher order integration schemes," in *48th AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit*, Atlanta, GA, USA, 2012American Institute of Aeronautics and Astronautics (AIAA).

[7] W. Sullwald, G. J. Smit, A. J. Steenkamp, and C. W. Rousseau, "Solid rocket motor grain burn back analysis using level set methods and Monte-Carlo volume integration," in *49th AIAA/ASME/SAE/ASEE Joint Propulsion Conference*, San Jose, CA, USA, 2013American Institute of Aeronautics and Astronautics (AIAA).

[8] D. Ribereau, P. Le Breton, and E. Giraud, "SRM 3D surface burnback computation using mixes stratification deduced from 3D grain filling simulation," in *35th Joint Propulsion Conference and Exhibit*, Los Angeles,CA, USA, 1999American Institute of Aeronautics and Astronautics (AIAA).

[9] E. Cavallini, *Modelling and Numerical Simulation of Solid Rocket Motors Internal Ballistics, [Ph.D. thesis]*, Sapienza University of Rome, Rome, Italy, 2010.

[10] A. P. Lorente, *Study of Grain Burnback and Performance of Solid Rocket Motors. phdlorente2013study, [M.S. thesis]*, Universitat Politècnica de Catalunya, Barcelona, Spain, 2013.

[11] Q. Li, G.-q. He, P.-j. Liu, and J. Li, "Coupled simulation of fluid flow and propellant burning surface regression in a solid rocket motor," *Computers & Fluids*, vol. 93, pp. 146–152, 2014.

[12] K. E. Hoff, J. Keyser, M. Lin, D. Manocha, and T. Culver, "Fast computation of generalized Voronoi diagrams using graphics hardware," in *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '99*, pp. 277–286, Los Angeles, CA, USA, 1999, Association for Computing Machinery (ACM).

[13] A. Sud, M. A. Otaduy, and D. Manocha, "DiFi: fast 3D distance field computation using graphics hardware," *Computer Graphics Forum*, vol. 23, no. 3, pp. 557–566, 2004.

[14] D. Peng, B. Merriman, S. Osher, H. Zhao, and M. Kang, "A PDE-based fast local level set method," *Journal of Computational Physics*, vol. 155, no. 2, pp. 410–438, 1999.

[15] J. Gomes and O. Faugeras, "Reconciling distance functions and level sets," in *5th IEEE EMBS International Summer School on Biomedical Imaging, 2002*, p. 15, Berder Island, France, 2002, IEEE.

[16] V. Estellers, D. Zosso, Rongjie Lai, S. Osher, J. Thiran, and X. Bresson, "Efficient algorithm for level set method preserving distance function," *IEEE Transactions on Image Processing*, vol. 21, no. 12, pp. 4722–4734, 2012.

[17] K. Zhang, L. Zhang, H. Song, and D. Zhang, "Reinitialization-free level set evolution via reaction diffusion," *IEEE Transactions on Image Processing*, vol. 22, no. 1, pp. 258–271, 2013.

[18] C. Li, C. Xu, C. Gui, and M. D. Fox, "Level set evolution without re-initialization: a new variational formulation," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, pp. 430–436, San Diego, CA, USA, 2005.

[19] X.-D. Liu, S. Osher, and T. Chan, "Weighted essentially non-oscillatory schemes," *Journal of Computational Physics*, vol. 115, no. 1, pp. 200–212, 1994.

[20] F. Wermelinger, B. Hejazialhosseini, P. Hadjidoukas, D. Rossinelli, and P. Koumoutsakos, "An efficient compressible multicomponent flow solver for heterogeneous CPU/GPU architectures," in *Proceedings of the Platform for Advanced Scientific Computing Conference on ZZZ - PASC '16*, pp. 1–10, Lausanne, Switzerland, 2016, ACM Press.

[21] H. M. Darian and V. Esfahanian, "Assessment of WENO schemes for multi-dimensional Euler equations using GPU," *International Journal for Numerical Methods in Fluids*, vol. 76, no. 12, pp. 961–981, 2014.

[22] S. Osher and C.-W. Shu, "High-order essentially nonoscillatory schemes for Hamilton–Jacobi Equations," *SIAM Journal on Numerical Analysis*, vol. 28, no. 4, pp. 907–922, 1991.

[23] A.-K. Tornberg and B. Engquist, "Numerical approximations of singular source terms in differential equations," *Journal of Computational Physics*, vol. 200, no. 2, pp. 462–488, 2004.

[24] W. E. Lorensen and H. E. Cline, "Marching cubes: a high resolution 3D surface construction algorithm," *ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4, pp. 163–169, 1987.

[25] S. Chen, B. Merriman, S. Osher, and P. Smereka, "A simple level set method for solving Stefan problems," *Journal of Computational Physics*, vol. 135, no. 1, pp. 8–29, 1997.

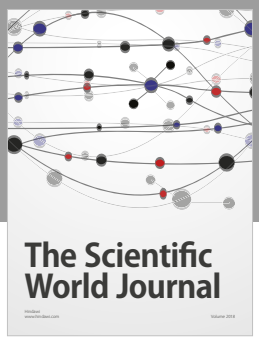
[26] S. K. Godunov, "A difference method for numerical calculation of discontinuous solutions of the equations of hydrodynamics," *Matematicheskii Sbornik*, vol. 89, no. 3, pp. 271–306, 1959.

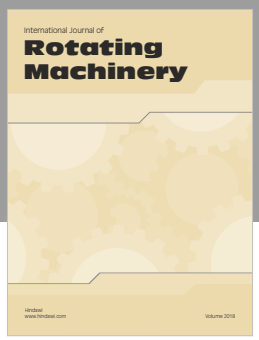
[27] The CGAL Project, *CGAL User and Reference Manual*, CGAL Editorial Board, 4.10.1 edition, 2017.

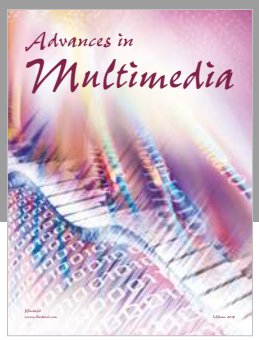
[28] I. M. Mitchell, "The flexible, extensible and efficient toolbox of level set methods," *Journal of Scientific Computing*, vol. 35, no. 2-3, pp. 300–329, 2008.

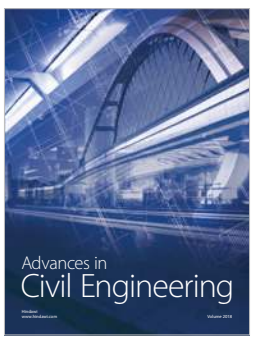
[29] X. Lin, B. Wang, and S. Jin, "A method for the interior ballistics calculations of solid rocket motors with dual burning rate propellant," *Journal of Solid Rocket Technology*, vol. 4, p. 4, 1991.

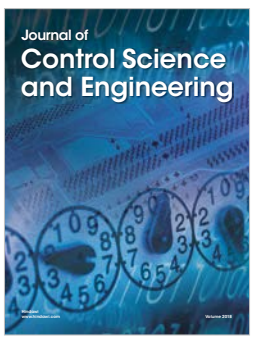
[30] B. Fu-ting and X. Hou, *Solid Rocket Motor Design*, China Astronautic Publishing House, Beijing, 2016.

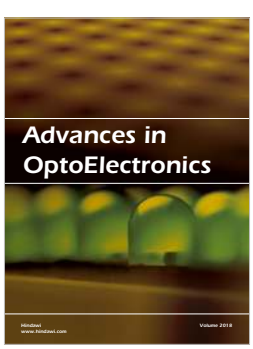
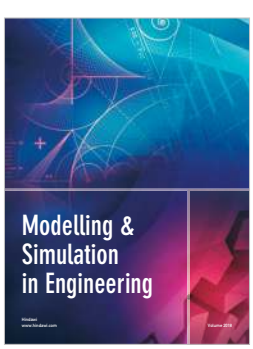
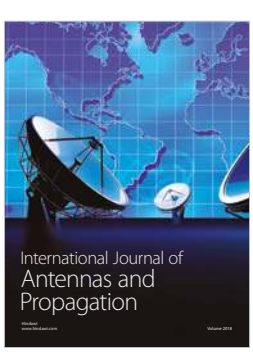
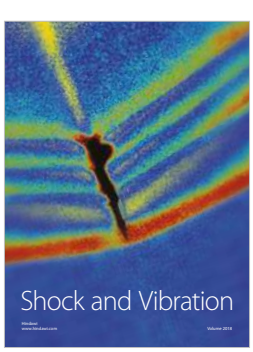
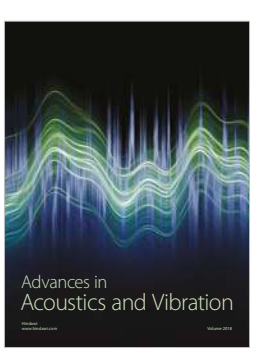










Hindawi

Submit your manuscripts at
www.hindawi.com