In this paper, we developed a GPU parallelized 3D mesh deformation based on the radial basis function (RBF) interpolation using NVIDIA CUDA C++. The RBF mesh deformation method interpolates displacements of the boundary nodes to the whole mesh, which can handle large mesh deformations caused by translations, rotations, and deformations. However, the computational performance of RBF mesh deformation depends on the quantity of grids. For 3D mesh deformation, especially for mesh with large number of boundary nodes, RBF interpolation has been verified computationally intensive. By introducing GPU acceleration, the computational performance of RBF mesh deformation code improved significantly. Several benchmark test cases were carried out to show the feasibility and efficiency of GPU parallelization. In summary, the GPU parallelized RBF interpolation shows the potential to become an alternative way to deal with 3D mesh deformation problems in an efficient way.

1. Introduction

In recent decades, a numerical simulation method such as Computational Fluid Dynamics (CFD) has become an efficient way to solve complex scientific and engineering problems. For simulations such as fluid-structure interaction and aerodynamic shape optimization which typically involve moving boundaries of fluid domain, mesh deformation is needed to adapting the computational grid to the new domain.

Mesh deformation based on the radial basis function (RBF) interpolation method was first developed in 2007 [Boer et al. [1]]. Compared to traditional mesh deformation methods such as the spring analogy [Batina (1989); Farhat et al. [2]; Degand and Farhat [3]] and transfinite interpolation (TFI) method [Dubuc et al. [4]], RBF interpolation has many advantages: (1) As a point-to-point interpolation method, RBF interpolation requires no grid connective information and thus is much easier to implement; (2) RBF interpolation is more flexible and can be applied to both structured and unstructured grids; and (3) RBF interpolation can handle large deformations of a mesh while maintaining an acceptable geometrical accuracy and mesh quality [Boer et al. [1]]. In the past few decades, RBF interpolation has been applied in many numerical simulation procedures such as aerodynamic shape optimization [Allen and Rendall [5]; Morris et al. [6]], CFD simulation of flapping wings [Bos et al. [7]], and fluid-structure interaction problems [Rendall and Allen [8]]. RBF interpolation has been stated as more computationally efficient than other mesh deformation method in literature [Boer et al. [1]; Morris et al. [6]]; however, this conclusion is established in the case of mesh with small number of grid nodes. As the grid quantity grows, the computational cost increased drastically as presented in Section 4, especially for 3D mesh deformation with a large number of grids. Due to this reason, RBF mesh deformation has not been popularized in actual numerical simulations.

Since the introduction of RBF interpolation mesh deformation method, a lot of efforts has been taken to further improve its efficiency. These works can be divided into two categories: algorithm and hardware perspective. Rendall and Allen developed an RBF
interpolation with the data reduction algorithm [Rendall and Allen [9]]. In a study [Rendall and Allen [8]], a data reduction RBF mesh deformation method based on the greedy algorithm was proposed. As demonstrated in Section 2, a system of linear equations consists of \( n_b \) equations which need to be solved in the process of RBF interpolation, where \( n_b \) is the number of boundary nodes. For data reduction RBF mesh deformation, only a small subset of boundary nodes is selected as control points, so the computational cost reduced drastically. However, this method brought in interpolation errors and geometry precision declines [Gillebaart et al. [10]; Niu et al. [11]]. There are also some works to improve the computational efficiency of RBF mesh deformation by the parallel algorithm with message passing interface (MPI) and run RBF codes on multiple CPU cores [Chao et al. [12]; Fang et al. [13]].

In recent decades, the computational performance of graphic processing units (GPU) has reached to a great extent [Owens et al. [14]], coupled with the rapid development of its computing platform and application programming interface such as CUDA [Shuai et al. [15]]. CUDA is a general purpose parallel computing platform introduced by Nvidia company in 2006, which employs GPUs to solve various computationally intensive problems more efficiently than multiple CPU cores [Nvidia [16]]. Nowadays, GPU computing has extended its application from computer vision to many fields like numerical simulation of fluid flow [Harris [17]; Rogers et al. [18]; Liu et al. [19]] and structure deformation [He et al. [20]; He and Lei [21]] problems. However, to our knowledge, there are few works that have attempted to accelerate the mesh deformation process on the GPU platform.

The motivation of this paper is to develop a GPU-accelerated 3D mesh deformation algorithm based on RBF interpolation. The paper is organized as follows: In Section 2, we introduced RBF formulation used in this paper, including the schemes and equations. In Section 3, the details of GPU parallelization with CUDA programming are presented. Test cases are carried out in Section 4 to validate the reliability of used RBF interpolation and the efficiency of GPU parallelization. Conclusions and future works are given in Section 5.

2. RBF Interpolation Formulation

Radial basis functions (RBF) refer to a series of functions in the form:

\[
\Phi (x) = \varphi (\| x \|),
\]

where \( \| x \| \) is Euclidean distance between two points, which refers to the spatial distance between two grid nodes of the mesh. There are various forms of radial basis functions which can be used to interpolating the data, and they can be divided into two groups: (1) functions with compact support and (2) functions with global support. Here, we chose the functions with compact support, and the distance \( \| x \| \) is normalized by support radius \( r: \xi = \| x \|/r \). Then (1) can be transformed into the form as follows:

\[
\Phi (\xi) = \left\{ \begin{array}{ll}
\varphi (\xi), & 0 \leq \xi \leq 1, \\
0, & \xi > 1.
\end{array} \right.
\]

When a radial basis function with compact support is applied, the influence of control points will be constricted inside a circle (for 2D) or sphere (for 3D) of compact radius \( r \). The commonly used forms of \( \varphi (\xi) \) are listed in Table 1. In this paper, we choose the function with form of CP \( C^0 \).

In the process of mesh deformation based on RBF interpolation, the displacements of internal grid nodes \( s(x) \) can be approximated by the sum of basis functions as

\[
s(x) = \sum_{j=1}^{n_b} a_j \varphi \left( \left\| x - x_b \right\| \right) + p(x),
\]

where \( x_{bj} = [x_{bj}, y_{bj}, z_{bj}] \) are the coordinates of control points, which refers to boundary nodes. \( n_b \) is the total number of boundary nodes. \( \| x - x_b \| \) is the distance between an internal grid point and \( j \)-th boundary node. The coefficients \( a_j \) polynomial \( p(x) \) are determined by solving the linear equations as

\[
\begin{bmatrix}
M & P_b \\
P_b^T & 0
\end{bmatrix}
\begin{bmatrix}
a \\
b
\end{bmatrix}
= \begin{bmatrix}
d_i \\
0
\end{bmatrix},
\]

where \( M \) is a square matrix of dimension \( n_b \times n_b \) with the form as

\[
M = \begin{bmatrix}
\varphi_{b_1} & \varphi_{b_1b_2} & \cdots & \varphi_{b_1b_{n_b}} \\
\vdots & \vdots & \ddots & \vdots \\
\varphi_{b_{n_b}b_1} & \varphi_{b_{n_b}b_2} & \cdots & \varphi_{b_{n_b}b_{n_b}}
\end{bmatrix},
\]

and \( \varphi_{b_ib_j} = \varphi(\| x_b - x_i \|) \), which is radical basis function of distance between the \( i \)-th and \( j \)-th boundary node. \( P_b \) is a matrix of dimension \( n_b \times 4 \) which has the form as follows:

\[
P_b = \begin{bmatrix}
1 & x_{b_1} & y_{b_1} & z_{b_1} \\
\vdots & \vdots & \vdots & \vdots \\
1 & x_{b_{n_b}} & y_{b_{n_b}} & z_{b_{n_b}}
\end{bmatrix}.
\]

<p>| Table 1: Radial basis functions with compact support. |
|-----------------|-----------------|</p>
<table>
<thead>
<tr>
<th>No</th>
<th>Name</th>
<th>( f (\xi) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CP ( C^0 )</td>
<td>((1 - \xi)^2)</td>
</tr>
<tr>
<td>2</td>
<td>CP ( C^2 )</td>
<td>((1 - \xi)^3/(4\xi + 1))</td>
</tr>
<tr>
<td>3</td>
<td>CP ( C^4 )</td>
<td>((1 - \xi)^6 (35/3\xi^2 + 6\xi + 1))</td>
</tr>
<tr>
<td>4</td>
<td>CP ( C^6 )</td>
<td>((1 - \xi)^8 (32\xi^2 + 25\xi^2 + 8\xi + 1))</td>
</tr>
<tr>
<td>5</td>
<td>CTPS ( C^0 )</td>
<td>((1 - \xi)^2)</td>
</tr>
<tr>
<td>6</td>
<td>CTPS ( C^1 )</td>
<td>(1 + 80/3\xi^2 - 40\xi^2 + 15\xi^4 - 8/3\xi^5 + 20\xi^2\log(\xi))</td>
</tr>
<tr>
<td>7</td>
<td>CTPS ( C^2 )</td>
<td>(1 - 30\xi^2 + 10\xi^2 - 6\xi^5 - 6\xi^6 - 6\xi^4\log(\xi))</td>
</tr>
<tr>
<td>8</td>
<td>CTPS ( C^3 )</td>
<td>(1 - 20\xi^2 + 80\xi^2 - 45\xi^2 - 16\xi^5 + 60\xi^6\log(\xi))</td>
</tr>
</tbody>
</table>
\[ \alpha = \begin{bmatrix} \alpha_{b_1} \\ \vdots \\ \alpha_{b_n} \end{bmatrix}, \]

and vector \( \beta \) contains the coefficients of polynomial \( p(x) \). Vector \( d_b \) is the prescribed displacement of boundary nodes:

\[ d_b = \begin{bmatrix} d_{b_1} \\ \vdots \\ d_{b_n} \end{bmatrix}. \]

3. CUDA Implementation of RBF Mesh Deformation

In Section 2, we introduced the RBF formulation for mesh deformation. Here, we present the GPU parallelization of RBF interpolation using CUDA C++. The serial version CPU code is also developed as a reference. The procedure of RBF mesh deformation can be decomposed to following subroutines: (1) importing grid points from mesh file; (2) constructing the matrix \( M \), as described in Section 2; (3) solving the linear system of (4); (4) calculating the desired moving vector of inner grid points; and (5) updating the coordinates of boundary and inner grid nodes.

For a CUDA program, the CPU is regarded as host to perform the task schedule, and the GPU is regarded as a device to deal with the parallel computing part of the program. Usually a CUDA program starts from the host side and copies data from the host side to the device side, then the host calls the kernel function executed on the device. By analyzing the algorithm of RBF interpolation and the computational cost of subroutines measured from the serial CPU program, we found that most of the subroutines in RBF interpolation can be parallelized and executed on GPU. The flow chart of CUDA implementation of RBF deformation in this work is shown in Figure 1.

In this work, we adopted four GPU kernel functions to accelerate the RBF interpolation. The first one is for constructing the matrix \( M \), the second one is for solving the linear equation, the third one is for computing the displacement of inner grid nodes, and the fourth one is for updating coordinates of the inner and boundary nodes. The pseudo code of the kernel functions are illustrated in Algorithms 1–4.

4. Results and Discussion

In this section, we will study two test cases for 2D and 3D mesh deformation based on the RBF interpolation algorithm described in Section 2. Firstly, mesh deformation around an 2D airfoil with relatively smaller number of grid nodes is carried out. Then, we presented the mesh deformation of a 3D wing with large number of grid nodes. The computational cost of serial CPU and GPU parallelized code are compared for these test cases with different
quantities of inner grid nodes and boundary nodes to verify the capability of GPU acceleration. In this work, the serial C++ code runs on Intel core(M) i7-8700K while the CUDA C++ code runs on NVIDIA GeForce RTX 2080Ti graphic processing units.

4.1. Mesh Deformation of Flow Field around an Airfoil. In this subsection, we perform a series of test cases for mesh deformation of flow field around an NACA0012 airfoil with chord length \( c = 1.0 \), and the boundary of far field is a circle of radius \( R = 5.0 \), as shown in Figure 2. During the process of
mesh deformation, grid nodes on the upper and lower chamber of the airfoil move according to

\[ \Delta y = \alpha x^2, \quad (9) \]

where \( \Delta y \) is the displacement along the \( y \) direction of boundary grid nodes on the airfoil, \( x \) is the \( x \) coordinate of these nodes, and \( \alpha = 0.8 \) is a prescribed coefficient. The deformation is carried out for 10 steps which leads to
maximum displacement $\Delta y_{max} = 8.0$ at the trailing edge of the airfoil. There are four meshes of different resolutions which are investigated, as listed in Table 2. The initial and deformed mesh with number of grid nodes $n_t = 1.0 \times 10^4$ and boundary nodes $n_b = 200$ are shown in Figure 2. As shown in the figure, the mesh is deformed smoothly without negative volume cells. The change of mesh quality after deformation is not examined in this paper since it has been studied in many literatures [Boer et al. [1]; [22]].

In Table 2, the execution time of GPU-accelerated code and the corresponding CPU code for different movement types and resolutions are presented, where $n_b$ is the number

<table>
<thead>
<tr>
<th>Mesh no</th>
<th>$n_b$</th>
<th>$n_t$</th>
<th>$\Delta t_C$ (s)</th>
<th>$\Delta t_G$ (s)</th>
<th>Speed up</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50</td>
<td>$1.0 \times 10^4$</td>
<td>0.0102</td>
<td>0.1900</td>
<td>0.053×</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>$1.0 \times 10^3$</td>
<td>0.0099</td>
<td>0.1950</td>
<td>0.051×</td>
</tr>
<tr>
<td>3</td>
<td>200</td>
<td>$1.0 \times 10^4$</td>
<td>0.2794</td>
<td>0.1970</td>
<td>1.51×</td>
</tr>
<tr>
<td>4</td>
<td>400</td>
<td>$1.0 \times 10^4$</td>
<td>0.2850</td>
<td>0.2045</td>
<td>1.39×</td>
</tr>
<tr>
<td>5</td>
<td>400</td>
<td>$1.0 \times 10^3$</td>
<td>18.7416</td>
<td>2.0172</td>
<td>9.29×</td>
</tr>
<tr>
<td>6</td>
<td>800</td>
<td>$1.0 \times 10^5$</td>
<td>19.0651</td>
<td>1.9834</td>
<td>9.61×</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subroutine no</th>
<th>CPU (s)</th>
<th>GPU (s)</th>
<th>Speed up</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.4032</td>
<td>0.1982</td>
<td>7.080×</td>
</tr>
<tr>
<td>2</td>
<td>3.8577</td>
<td>1.2035</td>
<td>3.205×</td>
</tr>
<tr>
<td>3</td>
<td>13.5858</td>
<td>0.5219</td>
<td>26.031×</td>
</tr>
<tr>
<td>4</td>
<td>0.2154</td>
<td>0.0598</td>
<td>3.602×</td>
</tr>
</tbody>
</table>

Figure 3: Deformation of the mesh around a 3D wing. (a) Cross-section of the initial mesh. (b) 3D view of the initial mesh. (c) Final mesh with deflection. (d) Final mesh with translation and rotation.
of boundary nodes, \( n_b \) is the total number of grid nodes, and \( \Delta t_c \) and \( \Delta t_G \) are the measured execution time spent on CPU and GPU for single deformation step, respectively. As shown in Table 2, in the case of small quantity of grid nodes (around \( 1.0 \times 10^4 \)), the execution time of CUDA code is longer than the CPU code. We analyse the reason that GPUs cannot be fully exploited in this situation, and the execution time of memory copy shares a large proportion. When the total number of grid nodes increases to the magnitude of \( 1.0 \times 10^5 \), the GPU code becomes faster than the CPU code. As the total number of grid nodes grows to the level of \( 1.0 \times 10^5 \), more than nine times acceleration was attained for the GPU code. Also, we found that both the CPU and GPU code are not sensitive to the number of boundary nodes in these cases.

In Table 3, the efficiencies of CPU subroutines and corresponding GPU kernel functions for 2D mesh deformation (\( n_b = 500, n_i = 1.0 \times 10^5 \)) are compared, in which subroutine No. 1 to No. 4 corresponding to the algorithms are presented in Section 3.

### 4.2. Deformation of the Mesh around a 3D Wing

In this subsection, we investigate test cases for mesh deformation of flow field around a 3D wing, the configuration of the wing is extruded from a NACA1410 airfoil with chord length \( c = 1.0 \), and the boundary of far field is a cylinder of radius \( R = 5.0 \) and length \( L = 10.0 \). As presented in Table 3, a series of test cases with different movement types and resolutions are performed. Figure 3 shows the initial and final deformed meshes.

In Table 4, we compared the efficiencies of GPU-accelerated code and the corresponding CPU code for different movement types and resolutions, where \( n_b \) is the number of boundary nodes, \( n_i \) is the total number of grid nodes, and \( \Delta t_c \) and \( \Delta t_G \) are the measured execution time spent on CPU and GPU for the single deformation step. As shown in the table, for these 3D mesh deformation cases, tens to hundreds of times acceleration is achieved via GPU parallelization. We also found that for these 3D mesh deformation cases, as the number of grid nodes grows, the execution time of CPU codes becomes sensitive to the number of boundary nodes while the corresponding GPU codes do not.

In Table 5, the efficiencies of CPU subroutines and the corresponding GPU kernel functions for 3D mesh deformation (\( n_b = 5000, n_i = 5.0 \times 10^5 \)) are compared, in which subroutine No. 1 to No. 4 corresponding to the algorithms are presented in Section 3. As presented in Table 5, the CPU computational cost of subroutine-2 increased drastically, and the GPU kernel function can accelerate this subroutine by more than 500 times for this case.

### 5. Conclusion and Future Works

In this paper, we developed a GPU-accelerated mesh deformation method based on radial basis function interpolation. The GPU parallelized code is designed with CUDA C++ provided by NVIDIA, and the implementation of GPU parallelization is described in detail. Two test cases are carried out to validate the reliability of prescribed radial basis function interpolation and the performance of GPU-accelerated mesh deformation code. Over hundreds of times speed up has been attained for the GPU parallelized code corresponding to the serial CPU code. The code is proved to be efficient and robust. In future, the authors will integrate data reduction radial basis function in the GPU parallelized code to further improve the performance of mesh deformation.

### Data Availability

The source code used to support the findings of this study are currently under embargo while the research findings are commercialized. Request for data, 6 months after publication of this article, will be considered by the corresponding author.

### Conflicts of Interest

The authors declare that they have no conflicts of interest.

### Acknowledgments

JH wishes to acknowledge the partial financial support from the Natural Science Foundation of Inner Mongolia under the grant number 2020BS01010.

### References


