Chapter 16
Numerical Solution of BVP on GPU with Application to Path Planning

Lumír Janošek, Martin Němec, and Radoslav Fasuga

Abstract The problem of path planning in a virtual environment is a widely researched area, which finds application in fields such as robotics, simulations, and computer games. This article focuses on a comparison of numerical methods for solving partial differential equations with BVP on the GPU with NVIDIA CUDA, used in the path planning of virtual characters using the potential fields. The most commonly used methods for computing the potential fields on the GPU are compared in this article in terms of time consumption.

Keywords Path-planning • Agent • Iteration methods • Potential fields

16.1 Introduction

The original purpose of a graphic processing unit (GPU) was primarily for image data processing. Programming of graphical chips was not a simple matter. It was necessary to use an application programming interface (API) to access the graphic processor such as Direct3D® or OpenGL®. The release of NVIDIA CUDA in 2007 changed the approach to the programming of graphic processors [1].

This article focuses on a comparison of the implementation of iterative methods for solving partial differential equations on a GPU in the agent path-planning domain. This article is not intended to present new approaches, but only to show the differences in iterative methods implemented on the GPU, which are used in potential field-based path planning. In this article the most widely used methods for the generation of potential fields used for agent navigation are compared in terms of time consumption.

The problem of path planning is widely applied in areas such as robotics and computer games. Path finding can generally be understood as finding the optimal path from an arbitrary position in a virtual world to a goal. In practical applications, there is often the requirement that the methods must be able to find paths in real time. Currently, the A* algorithm is still widely used for path planning [2], falling among graph-oriented algorithms. An alternative to graph-oriented algorithms are

L. Janošek (✉) • M. Němec • R. Fasuga
Department of Computer Science, VŠB-Technical University, Ostrava, Czech Republic
e-mail: lumir.janosek.st@vsb.cz; martin.nemec@vsb.cz; radoslav.fasuga@vsb.cz

Y. Cai, S. See (eds.), *GPU Computing and Applications*,
DOI 10.1007/978-981-287-134-3_16

methods for path planning using the potential fields. These methods are traditionally used in robotics. The application of potential field-based path planning can also be found in computer games [3]. Application of the BVP path planning may not be limited just to 2D. In [4] a method for the new application of BVP path planning on the surface of a 3D object is presented.

The idea of BVP path planning is using the interplay between repulsion from obstacles and attraction to a target position to create the expected behavior. Potential fields are obtained from the class of partial differential equations (PDE) called the boundary value problem (BVP) [5]. BVP-based path planning can create realistic-looking complex humanlike behavior similar during the agent's movement toward to the goal. Implementation of the numerical solution of the BVP on the GPU then enables the application of these methods in multi-agent real-time applications [6].

This chapter is structured as follows: Sect. 16.2 summarizes the problems of BVP in the path-planning domain, Sect. 16.3 describes the iterative methods used for solving the partial differential equations, Sect. 16.4 presents the implementation of the listed methods on the GPU, Sect. 16.5 summarizes the achieved results during the implementation, and the final section presents our conclusion and future work.

16.2 Harmonic Potential Field

One of the most widely used methods for generating a potential field for agent navigation in a virtual environment is the numerical solution of a partial differential equation based on the boundary value problem (BVP). One of the first steps in this area was undertaken by Connolly and Grupen [7]. In their work they presented a method for the generation of potential fields, which do not have local minima. Such a local minimum may be the reason why the agent can end up trapped in local minima. In their work, Connolly and Grupen proposed a method for generating a potential field through a solution to the Laplace equation:

$$\nabla^2 u = 0, g(x,y) = \begin{cases} 1, \text{obstacle} \\ 0, \text{goal} \end{cases}, (x,y) \in \partial\Omega \tag{16.1}$$

called harmonic function. The property of the Laplace equation is that it does not present local minima. This property is based on the so-called maximum principle, which the Laplace equation satisfies [8].

Equation (16.1) is solved with preset values on the boundaries. This type of boundary condition is called the Dirichlet boundary condition in the terminology of the BVP given by $g(x,y)$. In the case of obstacle space, the potential values at the obstacles are preset to a higher value, while in the goal area the values are preset to zero. The resulting potential field is used to find the agent's path to the goal by gradient descent. Higher values of the obstacles repel the agent to prevent collision.

On the other hand, zero values of the goal create an attraction force. Because there is only one minimum defined in the goal area, there exists exactly one path from any point on the map to the goal [9].

16.3 Iterative Methods

In general, there exist two methods for solving the boundary value problem, classified as direct methods and iterative methods. Direct methods lead to an exact solution to the problem with the use of a finite sequence of operations. In contrast to direct methods are iterative methods, in which the solution is obtained by a number of iterations [10]. A typical procedure is to determine the initial solution, on the basis of which the new values are calculated. This procedure is repeated until the convergence reaches the desired solution. This is usually determined by some criterion of convergence.

The iterative solution of elliptic equations most commonly uses the following methods: Jacobi, Gauss-Seidel, or Successive Overrelaxation (SOR).

In the Jacobi method, the dependent variable at each grid point is solved using the initial values of the neighboring points or previously computed values [10]:

$$u_{i,j}^{(k+1)} = \frac{1}{4}\left[u_{i-1,j}^{(k)} + u_{i+1,j}^{(k)} + u_{i,j-1}^{(k)} + u_{i,j+1}^{(k)}\right]$$

where k denotes the values computed in the previous iteration and i, j denotes the grid point.

The Gauss-Seidel method is a modification of the Jacobi method. To compute the value of a dependent variable in the current iteration, the values from the previous and current iteration are used. This will certainly increase the convergence rate dramatically over the Jacobi method [10]. The iteration formula for the Gauss-Seidel method has the following form:

$$u_{i,j}^{(k+1)} = \frac{1}{4}\left[u_{i-1,j}^{(k+1)} + u_{i+1,j}^{(k)} + u_{i,j-1}^{(k+1)} + u_{i,j+1}^{(k)}\right]$$

where k denotes the values computed in the previous iteration, k +1 denotes the values computed in the current iteration, and i, j denotes the grid point.

Better convergence can be achieved with the Successive Overrelaxation (SOR) method. The main idea behind the SOR algorithm is to compute a better approximation to the true solution by forming a linear combination of the current updated solution k +1 and solution k from the previous iteration [11]. The iteration formula for SOR method is defined as:

$$u_{i,j}^{(k+1)} = (1-\omega)u_{i,j}^{(k)} + \frac{\omega}{4}\left[u_{i-1,j}^{(k+1)} + u_{i+1,j}^{(k)} + u_{i,j-1}^{(k+1)} + u_{i,j+1}^{(k)}\right] \tag{16.2}$$

where ω denotes the relaxation parameter and i, j denotes the grid point. The optimal value of ω should be in the range $1 < \omega < 2$. If $0 < \omega < 1$, this is so-called under-relaxation [12]. In the case of $\omega = 1$, the SOR algorithm is reduced to Gauss-Seidel.

16.4 Implementation

With access to today's NVIDIA CUDA-enabled GPU, it is possible to significantly accelerate the methods of numerical solution of elliptic equations using parallel implementation. With the parallel performance of the GPU, which is provided by the CUDA interface, it is possible to solve many complex computational problems with more efficiency than on the CPU. GPU is suitable for solving problems which require the parallel processing of large amounts of data.

Not all iterative methods for solving elliptic equations are suitable for implementation on the GPU. For parallel implementation and performance comparison of the numerical solution of elliptic equations on the GPU, the Jacobi, Jacobi Red-Black, and SOR Red-Black methods were chosen. The sequential implementation of the Gauss-Seidel uses two values from the current iteration and two values from the previous iteration to calculate the current cell. In the implementation of this method on the GPU, it is necessary to have some synchronization, which can lead to performance degradation [13]. Gauss-Seidel is an effective method for implementation on the CPU. Due to the need for synchronization, the Gauss-Seidel method is not best suited for parallel implementation on the GPU, and therefore was not taken into account for the implementation of iterative methods on the GPU.

As mentioned in the introduction, the methods presented in this article are focused on agent navigation in a virtual world. A virtual environment contains a number of obstacles, which the agent tries to avoid on the way to the goal. Before the start of the potential field calculation, it is necessary to discretize the virtual environment into a fixed homogeneous grid representation. Each grid cell (i,j) is associated with a small region of the real environment and maintains the potential value $u_{i,j}$, which holds information about whether the given cell is an obstacle or free space. Cells defined in place of the obstacles have the initial potential set to 1, while cells containing a goal have the potential value set to 0. Such a manner of setting the initial values corresponds to the Dirichlet boundary conditions [14].

With such a defined initial boundary condition, the values of all other cells are computed using a certain number of iterations. In order for the method to converge to the correct solution, a sufficient number of iterations must be specified. The number of iterations varies depending on the used method. One option of how to control the number of iterations is assessment of some convergence criteria based

on which the calculation is terminated. Such criteria could be check of the error that occurred during the iterations, for instance. The iteration is terminated once the error is less than the given tolerance [11]. An alternative way is to specify a fixed number of iterations at the beginning of the algorithm. [12] shows that the required number of iterations can be determined by an analytical formula. The number of iterations r required to reduce the error by a factor 10^{-p} , for the Jacobi method, is defined as:

$$r \approx \frac{1}{2} p J^2 \tag{16.3}$$

J^2 denotes the number of grid points.

Using the Red-Black method in conjunction with the Jacobi method, it is possible to achieve certain optimization [11]. The Red-Black method divides grid points into odd and even, symbolically expressed by red-black coloring. The coloring of the grid points is done so that no point is directly adjacent to a point of the same color. The red point values from the previous iteration are utilized during the calculation of the values of the black points. This step is identical to the Jacobi iteration, applied to all black points. Updated black point values are used in the next step in the computation of the red points, which is identical to the Gauss-Seidel iteration. The Red-Black method is thus composed of one Jacobi iteration and one Gauss-Seidel iteration. As mentioned in the previous Sect. 16.3, the Gauss-Seidel method uses values computed in the previous iteration to compute the current values, thus significantly contributing to speeding up the convergence rate. The number of iterations for the Jacobi Red-Black method can then be defined practically as well as for the Gauss-Seidel method, for which it is defined as [12]:

$$r \approx \frac{1}{4} p J^2 \tag{16.4}$$

J^2 denotes the number of discrete grid points. The GPU implementation of the Red-Black methods uses two kernels, one for computation of the red points and one for computation of the black points. The number of black or red points on the y-axis of the grid is half. This can reduce the number of threads in each kernel on the y-axis by half. Reducing the number of threads leads to a certain optimization of the iterative process.

Compared to the Jacobi or Jacobi Red-Black, the SOR method leads to much faster convergence. As already stated, the SOR method uses the values from the previous iteration and the values from the current iteration to compute the current point, similarly as the Gauss-Seidel method, see (16.2). The parallel GPU implementation of the SOR method is enabled using the Red-Black ordering [15]. Updated values of the black points, i.e., values of the current iteration, are used to compute the red points. Updated values of the red points, i.e., values of the previous iteration, are used to compute the black points. The number of required iterations needed in order to reduce error by factor 10^{-p} is given by [12]:

$$r \approx \frac{1}{3}pJ \tag{16.5}$$

Comparing the number of iterations of the SOR method with the number of iterations of the Jacobi method (16.3) and Jacobi Red-Black (16.4), it is obvious that the optimal number of iterations of the SOR method is in the order of J, compared with J^2 of the Jacobi and Jacobi Red-Black method. The weak point of the SOR may be the choice of overrelaxation parameter ω. In [12] the following equation is stated, which can be used to estimate the overrelaxation parameter:

$$\omega \approx \frac{2}{1 + \frac{\pi}{J}}$$

In general, finding the correct value of ω is not an easy task. In many cases experimentation is the only possible way to determine the correct value of parameter ω.

16.5 Results

Implementation of the Jacobi, Jacobi Red-Black, and SOR Red-Black methods was compared in terms of time performance. These methods were tested on GeForce GTX 560 and GeForce GTX 670 graphics cards.

A map of static obstacles is copied into the device memory before the start of the actual iterative procedure. Since the obstacle map is read only, it is copied into the texture memory of the GPU before the calculation. The texture memory is optimized for a 2D spatial locality, so threads of the same warp that read texture addresses that are close together will achieve the best performance [16]. The map of obstacles only holds information about the position of the obstacles and walkable spaces. For this reason, the 8-bit data format was chosen for maximum reduction of the memory requirements.

In practical applications of these numerical methods in the field of path finding and agent navigation in a virtual environment, such as in [17], it is necessary to change the global obstacle map only in case of adding new obstacles or removing existing ones. Due to the individual approach to the implementation of the global obstacle map, the data transfers from the host to the device were not taken into account during the speed comparison of the methods.

Maps of different sizes were used to compare the speed of these methods. The resulting time difference of the method is shown in Fig. 16.1. The most optimal performance was achieved with SOR Red-Black when compared with the Jacobi and Jacobi Red-Black. For each method the number of iterations was determined based on equations (16.3) for Jacobi, (16.4) for Jacobi Red-Black, and (16.5) for SOR Red-Black.

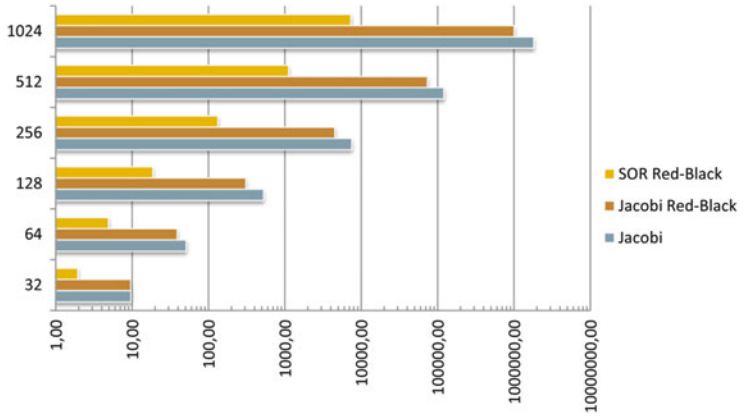

Fig. 16.1 Speed differences (in milliseconds) of the GPU computation of the Jacobi, Jacobi Red-Black, and SOR Red-Black methods. The comparison was made for input grid size $32^2 - 1024^2$

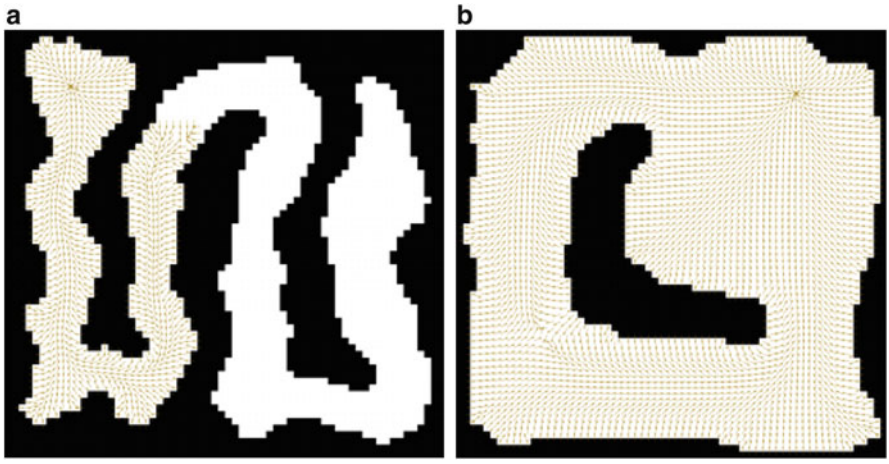

Fig. 16.2 Picture 1.2a shows the resulting gradient of the potential field. Picture 1.2b illustrates the failure of the calculation in confined space due to a lack of real number precision

Implementation of the tested methods was performed in the double-precision floating-point format. Potential field computation was tested in such obstacle configurations which simulated the cramped spaces. These configurations were often the cause of the loss of the potential value in locations too far from goal, because of insufficient accuracy of the real number. One such situation is illustrated in Fig. 16.2b. Values in this potential field were rounded to 1 due to insufficient accuracy of the real number. Final computation of the gradient cannot then be achieved in these cases. The potential field gradient illustrated on Fig. 16.2a and

fig. 16.2b with size of 64^2 was computed using the Jacobi Red-Black method. The number of iterations required to obtain a valid solution was determined using equation (16.4).

Conclusion

In this chapter the implementations of the numerical methods for solving elliptic equations using CUDA with application on BVP path planning were compared. The Jacobi, Jacobi Red-Black, and SOR Red-Black methods were compared in terms of time complexity. Using the SOR Red-Black, we reached the fastest convergence, in comparison with Jacobi and Jacobi Red-Black. These methods were applied to the obstacle configuration simulating a real environment. It was shown that the configuration of obstacles simulating cramped spaces, such as underground caves, does not provide sufficient freedom for the convergence of methods. The information is lost due to insufficient accuracy of the real number during the convergence to the final potential field.

In [18] the methods of BVP path planning were combined with the Full Multigrid method, which solves elliptic equations using a hierarchical strategy. The hierarchical approach overwhelms the speed of convergence of the original SOR method.

In the previous section, an error caused by insufficient accuracy of the real number, leading to early rounding to 1, was described. One option of solving this problem is described in [19]. Future development of this work will focus on finding an alternative way to solving the problem with insufficient accuracy of the real number and to optimizing the convergence in cramped spaces. This would then allow the application of BVP path planning for space-limited interiors.

Acknowledgment This work was partially supported by the SGS in VSB Technical University of Ostrava, Czech Republic, under the grant No. SP2013/185.

References

1. Kirk, D.B., Hwu, W.-W.: Programming Massively Parallel Processors: A Hands-on Approach, 1st edn. Morgan Kaufmann Publishers Inc, San Francisco, CA (2010)
2. Cui, X, Shi, H.: A*-based pathfinding in modern computer games. IJCNIS **11**(1), 125–130 (2011)
3. Silveira, R., Fischer, L., Jos' e AntônioSalini F., Prestes, E., Nedel, L.: Path-planning for RTS games based on potential fields. In: Proceedings of the Third international conference on Motion in games, MIG'10, pp. 410–421. Springer, Heidelberg (2010)
4. Fischer, L., Fischer L.: Semi-automatic navigation on 3d triangle meshes using bvp based path-planning. In: 24th SIBGRAPI Conference on Graphics, Patterns and Images (Sibgrapi), pp. 33–40, (2011)

5. Marcelo, T., Idiart, M.A., Edson, P., Engel, P.M.: Exploratory navigation based on dynamical boundary value problems. J. Intell. Robotics Syst. **45**(2), 101–114 (2006)
6. Fischer, L.G., Silveira, R., Nedel, L.: Gpu accelerated path-planning for multi-agents in virtual environments. In: VIII Brazilian Symposium on Games and Digital Entertainment (SBGAMES), pp. 101–110, (2009)
7. Connolly, C.I., Grupen, R.A.: On the applications of harmonic functions to robotics. J. Robot. Syst. **10**, 931–946 (1993)
8. Strauss, W.A.: Partial Differential Equations: An Introduction. Wiley, New York, NY (1992)
9. Dapper, F., Prestes, E., Idiart, M.A.P., Nedel, L.P.: Simulating pedestrian behavior with potential fields. In: Proceedings of the 24th international conference on Advances in Computer Graphics, CGI'06, pp. 324–335. Springer, Heidelberg (2006)
10. Klaus, A.: Hoffmann and Steve T Chiang. Computational fluid dynamics vol.i - hoffmann.pdf. Int. J. Comut. Fluid. Dyn. **126**(2), 581–594 (2000)
11. Zhu, J.: Solving Partial Differential Equations on Parallel Computers. World Scientific Publishing Co. Inc., River Edge, NJ (1994)
12. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes 3rd Edition: The Art of Scientific Computing, 3rd edn. Cambridge University Press, New York, NY (2007)
13. Gomes, G.A.A.: Linear solvers for stable fluids: GPU vs CPU. In: 17th EncontroPortugues de ComputacaoGrafica (EPCG09), pp. 145–153 (2009)
14. Dapper, F., Prestes, E., Nedel, L.P.: Generating Steering Behaviors for Virtual Humanoids Using BVP Control. In: Proc. of CGI, pp. 105–114 (2007)
15. Konstantinidis, E., Cotronis, Y.: Graphics processing unit acceleration of the red/black SOR method. Concurr Comput. **25**(8), 1107–1120, (2012)
16. NVIDIA. CUDA C BEST Practices Guide (2012)
17. Fischer, L.G., Silveira, R., Nedel, L.: Gpu accelerated path-planning for multi-agents in virtual environments. In: VIII Brazilian Symposium on Games and Digital Entertainment (SBGAMES), pp. 101–110, (2009)
18. Silveira, R., e Silva, E.P., Jr., PorcherNedel, L.: Fast path planning using multi-resolution boundary value problems. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, 18–22 October 2010, Taipei, Taiwan, pp. 4710–4715. IEEE (2010)
19. Renato, S., Fbio, D., Edson, P., Luciana, N.: Natural steering behaviors for virtual pedestrians. Vis. Comput. **26**(9), 1183–1199 (2010)