Accelerated GRECO Based on a GPU

[1] Yang ZhengLong, [1] Jin Lin, and [2] Li WeiQing

[1] Nanjing Research Institute of Electronics Technology, China
[2] Computer Science and Tech. Institute, Nanjing University of Science. and Technology, China

Abstract – For obtaining the electromagnetic scattering characteristic of a complex target efficiently, GRECO (Graphical Electromagnetic COmputing) is implemented by a programmable pipeline of a modern GPU (Graphics Processing Unit). The speed of the simulation can be improved up to 20 times compared with the raw GRECO. The ray tracing algorithm based on a GPU is implemented to obtain the multiple reflection contribution of a target with concave structure. This approach will redound to research works such as radar target identification and Inverse Synthetic Aperture Radar (ISAR) imaging.

Key words – EM scattering, GRECO, and GPU.

I. INTRODUCTION

GRECO (GRaphics Electromagnetic COmputing) is an effective method for computing the high-frequency radar cross section (RCS) of complex targets based on physical optics (PO), and physical theory of diffraction (PTD) [1]. In this paper, an accelerated version of the GRECO method is implemented by the programmable pipeline of a modern GPU (Graphics Processing Unit), the speed of the simulation can be improved up to 20 times compared with the base GRECO. Furthermore, the ray tracing algorithm based on the GPU is implemented to obtain the contribution of multiple reflection of a target.

Compared with the raw GRECO, the GPU accelerated GRECO has higher efficiency and enhanced ability to simulate the multiple reflection of a complex target with concave structure.

With the development of GPU and the creation of the new feature of programmability, researchers begin to transfer some of the processing stages in the graphics output pipeline or some graphics algorithms from the CPU (Central Processing Unit) to the GPU. Except for those graphics-only applications, GPU finds applications in general purpose computations in other fields, and it has become a hot topic for research in recent years. In the electromagnetics filed, the FDTD method has been implemented based on the GPU for higher efficiency [2].

In some applications such as computational electromagnetics and signal processing, the speed of the CPU can not meet the requirement of efficiency. One can use other high-speed processing unit like DSP (Digital Signal Processing) or HPC (High Performance Cluster) system, but DSP or HPC system is very expensive and limited in application. By contrast with a general CPU, a GPU consists of higher-bandwidth memory and more floating-point hardware units. For example, current GPU such as the Nvidia 6800 Ultra has a peak performance of 40 Gflops and a memory bandwidth of 35.2 Gbytes per second, compared to 6.8 Gflops and 6 Gbytes per second for a 3-GHz Pentium 4 CPU. Furthermore, GPU performance for graphics applications throughput has been increased from 2 to 2.5 times a year. This growth rate is faster than Moores law as it applies to CPUs, which corresponds to about 1.5 times a year. In a GPU, there are several vertex pipelines using MIMD (Multiple Instructions Multiple Data), and fragment pipelines using SIMD (Single Instruction Multiple Data) to provide the ability for high-speed parallel data processing. So the GPU can be treated as a parallel vector machine that is suitable for some kinds of numerical computations [3], [4].

The function of the fixed pipeline of graphics hardware used in raw GRECO is to obtain the shadow of different parts of a complex target in the rendering process. Based on the raw GRECO, the programmable pipeline of a GPU can be applied to implement GRECO method without a rendering process. The customized vertex and fragment shaders for RCS computing can be compiled and linked into the GPU pipeline to substitute some functions of the fixed-pipeline [5]. Since most of the time-consuming computation in raw GRECO is used to obtain the scattering contribution of the small facets represented by the pixels on the screen, thus it can be implemented by the parallel fragment shader to accelerate the simulation. In this paper, the vertex and fragment shaders are applied to the raw GRECO method based on prior work to obtain the mono and bistatic RCS of complex targets [6]. The GPU-based ray tracing algorithm is implemented to obtain the contribution of multiple reflection of a target with concave structure. The paper is focused on the combination of GRECO and GPU programming, the GRECO and related techniques will not be discussed here, its details can be found in [1] and [7].

Compared with the raw GRECO, the main advantages of GPU-based GRECO are:

1) Higher efficiency, where the speed can be improved

up to 20 times compared with the raw GRECO.
2) Ability to simulate multiple reflection of the complex targets (see section II.***C***), this feature is not involved in raw GRECO.

The development of GPU-based GRECO is part of the work for the target echo simulation for radar target identification and Inverse Synthetic Aperture Radar (ISAR) imaging. High speed simulation is required for obtaining the wide-band and wide-angle scattering response of many complex targets, it is the main motivation of the work.

II. METHODOLOGY

In GRECO, the procedures of RCS prediction are:
1) Read the 3D model files created by CAD software.
2) Render the 3D model in the frame-buffer of a graphics card.
3) Obtain the depth information of each pixel.
4) Obtain the surface normal of each pixel using two different lighting configurations.
5) Map the depth information to the real depth of the target.
6) Obtain the scattering contribution of each pixel using PO/PTD.
7) Obtain the total scattering contribution by accumulating the contribution of each pixel coherently.

In the procedures mentioned above, normal vector computation, depth mapping, scattering simulation of each pixel, and final accumulating are done in the CPU. There are three massive data exchanges between main memory and video memory: two for color information of two different lighting configurations and one for depth information. In the CPU, massive floating point operations are needed for computing the normal vector and scattering contribution of each pixel on the target surface based on serial processing mechanism, that is, many loop operations are needed in the raw GRECO. Using the programmable pipeline of the GPU, the normal vector and depth can be accessed directly by using built-in variables of shading languages such as Cg (C for Graphics, released by Nvidia) and GLSL (Graphics Library Shading Language) [8], so the two different lighting configurations and depth mapping are needless. Thus scattering contribution of each pixel can be obtained rapidly based on parallel processing mechanism of fragment shader, and the final total scattering contribution can be obtained by a parallel reduction process in the GPU [9]. The detailed procedures are explained as follows.

A. PO Simulation by Shader

Figure 1 represents the procedures of the GPU's fixed-pipeline (solid line) and the programmable pipeline (dashed line). Some functions of the fixed-pipeline can be replaced by the programmable pipeline using vertex shaders and fragment shaders. Vertex shaders can be used to specify a general sequence of operations to be applied to each vertex and its associated data, and the fragment shaders can be used to specify the operations on fragment values and its associated data.

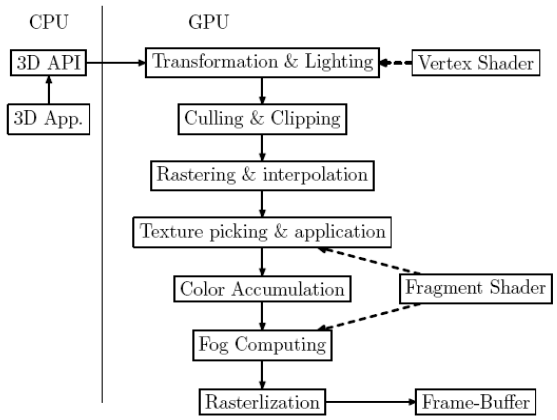

Fig.1. GPU pipeline.

In GRECO, the main time for the RCS prediction is spent on the electromagnetic computation, while the geometric model manipulations are left to the graphics hardware. Raster element is applied to discrete the target surface natively, and automatic culling technique is used to remove the shadowed parts of the target. With the rapid development of graphics hardware especially the programmable pipeline of the GPU, GRECO can be implemented entirely in the GPU. The key procedures of GPU accelerated GRECO are:
1) Write the user-defined vertex shaders and fragment shaders for RCS computation based on PO/PTD.
2) Compile and link the shaders, and then embed the shaders to the GPU pipeline.
3) Start up the general drawing process and store the scattering results of each pixel in the frame buffer.
4) Obtain total scattering contribution by the reduction technique that will be described in section II.***D***.

For GPU accelerated GRECO, the 3D geometrical transformation, including normal transformation can be implemented in a vertex shader. The scattering results of each pixel can be obtained directly in a fragment shader by equation (5) in [1], then it can be written into the R component and G component of RGB (Red, Green, Blue) by render-to-texture technique, where R and G components represent the real part and imagery part, respectively. Finally, the reduction technique can be applied to obtain the total scattering contribution.

B. Diffraction of Edge

In [1], the Element Edge Wave (EEW) is applied

to obtain the edge diffraction contribution. The geometrical parameters such as the normal vectors of two facets that construct the edge, edge inner angle and the direction of the edge should be obtained correctly. The method used in [1] can obtain the edge information on the condition that two facets are all illuminable. It will fail when one of the facets that construct the edge is shadowed.

In [10] and [11], the complete edge information is obtained from the model information stored in the 3D model file based on some principles of computer graphics. It is found that the edge diffraction can be obtained by the GPU programming. The primary issue is how to obtain the edge information with the shadow between model and edges must be considered, and the second issue is how to pass the edge information to the fragment shader for diffraction computation using PTD or ILDC (Incremental Length Diffraction Coefficient) [7].

The 3D facet model is constructed by a number of triangles with a certain topological relationship. For a regular 3D model, the common edge exists in the adjacent facets. If the angle between two normal vectors of two facets is larger than the predefined threshold, the common edge needs to be considered for diffraction; otherwise, the two facets are treated as locating on the smooth surface. This is similar to normal averaging in computer graphics [8].

Through the preprocessing of the model information, edge information such as normal vectors, edge direction, and inner angle of each edge can be obtained for edge diffraction computation later. These parameters are dependent, the edge direction and inner angle can be obtained by the cross product and dot product of two normal vectors respectively. The normal vector of one illuminable facet, edge direction and inner angle are sufficient for edge diffraction computation. In paper [11], three display lists [8] are used to store the normal vector, edge direction and inner angle respectively. In order to eliminate the shadowed edge, the "dark" model (r,g,b=0,0,0) can be rendered with lighting disabled before the edges are rendered. In this paper, only one display list is used to store the three parameters for edge diffraction by eliminating the shadowed edges and pass this edge information to the fragment shader.

In OpenGL, the main color and secondary color can be assigned for each vertex of a 3D model and each color has four components named RGBA (Red, Green, Blue, and Alpha). In the rendering procedure of an edge, the RGBA of the main color can be used to store the normal vector $\boldsymbol{n}$ of illuminated facet and the inner angle α, that is, R = $\boldsymbol{n}$.x, G = $\boldsymbol{n}$.y, B = $\boldsymbol{n}$.z, and A = α, while the RGB of the secondary color can store the edge direction. Eliminating the shadowed edges can be implemented by the "dark" model mentioned above.

Figure 2 illustrates the rendering result of a missile model with shadowed edges that are eliminated. The smooth part of the model, such as the fuselage and wings, is full dark as the background, while the edges of the wings are rendered with the geometrical parameters passed to the fragment shader by main color and secondary color for diffraction computation. When all information for diffraction computation is available, the EEW method can be implemented in the fragment shader for edge diffraction.

Fig. 2. Rendering result of edges.

C. *Multiple Reflections*

Multiple reflections play an important role in the scattering of complex targets. The Shooting and Bouncing Ray (SBR) technique has been developed for RCS prediction for a target with concave structure [12]. The software Xpatch based on SBR has been released by ASIC Inc. Ray tracing is the core algorithm of SBR. In order to obtain multiple reflection contribution, the ray propagation paths of incident wave and reflected wave need to be recorded to obtain the amplitude and phase of each ray that bounced between different parts of the target surface. All contributions from scattering and iterative multiple reflection should be accumulated in the direction in which the receiver is located.

Conventional ray tracing algorithm computes light intensity and color components of the scene. The coherence of light is not considered in conventional ray tracing because the phase of the light is not important for rendering scene in computer graphics. However, it is as important as the amplitude in the EM scattering of complex targets. Thus the modifications should be applied to conventional ray tracing algorithm for obtaining the multiple reflections' contribution. The differences between conventional ray tracing and SBR in EM scattering are:

1) Conventional ray tracing calculates the amplitude of light. As for the EM scattering, both the amplitude and phase are to be calculated.

2) The light amplitude in a conventional ray tracing is obtained by the Phong lighting model, while the amplitude and phase of the EM scattering are obtained by physical optics, geometrical optics, and PTD.

3) Refraction must be considered in conventional ray tracing while there is no refraction contribution from metal target surface for EM scattering computation.

To obtain the multiple reflections' contribution, the propagation paths bounced between different parts of the complex target should be recorded including the sequences of the intersections between the radar beam and the facets of the target. It is very time-consuming because massive intersection tests are needed to be computed.

In order to accelerate the ray tracing by GPU, Purcell mapped the vertices of the complex model to three textures and constructed a texture representing the linked list which stores the triangles of the model surface [13]. Thus, the ray tracing algorithm can be implemented by the GPU. The GPU accelerated ray tracing for EM scattering is implemented in the fragment shader based on Purcell's work in this paper. The algorithm flow chart is shown in Fig. 3.

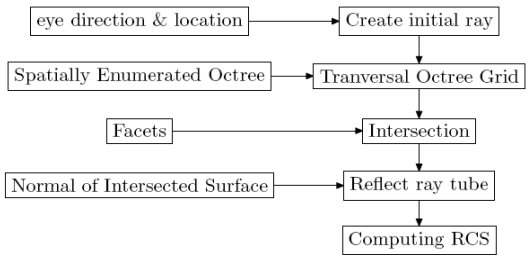

Fig. 3. Ray tracing for multiple reflection.

The algorithm can be divided into five parts and implemented by fragment shader, except the ray tube initialization is preprocessed in the CPU. The method provided by Didier Badouel is applied to obtain fast ray-polygon intersection [14] and the Proximity Clouds algorithm is applied to scene traversal [15]. The performance of the 3D traversal is important to the efficiency of the algorithm. BSP (Binary Space Partition) tree, Adaptive octree, KD tree, and SEADS (Spatially Enumerated Auxiliary Data Structure) [16] etc., can be adopted to store the 3D scene data for acceleration of traversal [17]. In this paper, the SEADS method is applied to fast traversal due to the following reasons:

1) It is simple for parallel processing.
2) The time for each data access is constant and with linear time complexity.
3) Easy code for hardware implementation.

The ray tracing algorithm is very complicated; it is a hot topic in computer graphics, the detailed procedures of the ray tracing accelerated by the GPU is not described here. It should be noted that if the depth of tracing is reduced to 1, the algorithm degenerates to GRECO.

D. Reduction

When shaders for PO/PTD and multiple reflections are applied to scattering computing, the contribution of each pixel is stored in RG components of the current texture and has to be accumulated to obtain the total scattering contribution. Traditionally, the RGB components can be read back to the main memory and then accumulated by CPU. It is time-consuming because of long time loop operations for accumulation and massive data exchanges between video memory and main memory, for example, if the viewport is 1024 by 1024, this means that there is 1024×1024 = 1048576 accumulation operations that are needed to obtain the final total contribution. Additionally, it is slow to read the RGB components from video memory to main memory. If the accumulation can be implemented in GPU without the massive data exchanges and loop operations, higher execution efficiency will be obtained.

After investigating the parallel mechanism of the fragment shader, it is found that the parallel reduction technique is suitable for acceleration of accumulation [9] in GPU. After several reduction processes, only one complex number that represents the total contribution is needed to be read back to the main memory resulting in no massive data exchange.

In computer graphics, reduction technique is mainly applied to obtain the maximum value or accumulation of the floating point numbers stored in texture. Here, texture can be treated as a 2D array that stores the scattering contribution of each pixel. The maximum value in a 2D array can be obtained by the procedure shown in Fig. 4.

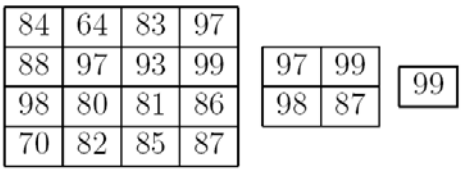

Fig. 4. Reduction for obtaining the maximum.

For obtaining the maximum of the 4 by 4 array, the maximums of 4 subregions with the elements {84, 64, 88, 97}, {83, 97, 93, 99}, {98, 80, 70, 82}, and {81, 86, 85, 87} should be first obtained, and then a new array can be created with the elements {97, 99, 98, 87} that are the maximums of 4 subregions.

The same procedure can be applied to the new array for obtaining the final maximum of the array, that is, 99. For obtaining the accumulation result of the array, similar procedure can be applied.

In the implementation of reduction by the GPU, the accumulation can be applied to a 2 by 2 subregion of the texture, then a new texture can be constructed

with 1/4 the size of the current texture. Iteratively, the final accumulated result can be obtained. This operation limits the size of the texture to the integer number that is power of 2 but it is suitable for parallel processing in fragment shader. In this paper, the size of the texture is set to 2^{10} = 1024, only 10 reduction operations are needed to obtain the total contribution without massive data exchanges from video memory to main memory and large amount of loop operations (up to 1048576).

Jinwook Kim provided a reduction example on the web and helped us to implement the reduction easily [18]. The reduction procedure for computing the RCS of a missile model is illustrated as shown in Fig. 5.

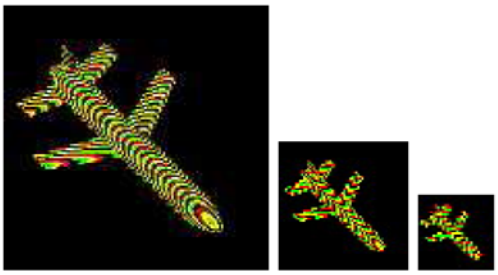

Fig. 5. Reduction for RCS accumulation.

The R and G components of the images in Fig. 5 represent the real and imaginary parts of the scattering contribution of each pixel, respectively. The size of the first image is 1024×1024. After one reduction operation, the size of the image is reduced to 512×512 and the accumulation results of each 2 by 2 subregion are obtained. The final accumulation result is obtained after 10 iterative reduction operations and stored in a 1×1 array.

III. EXAMPLES

In order to compare the simulation speed, the RCS of a scaled missile model (1:8) is simulated by raw GRECO and the GPU accelerated GRECO. The view port for computing is 1024 by 1024, f = 10GHz, aspect angle is from 0◦ to 360◦ with an angle step of 0.25◦, that is, 1441 RCS results are calculated. The CPU in our platform is an Intel Pentium 4 with clock frequency 2.8 GHz and the GPU is provided by Nvidia GeForce 6600 GT graphics card. The time for raw GRECO is 390 s and that for GPU accelerated GRECO is only 19 s. The speed of the simulation is improved up to 20 times. The RCS of the model is also measured by CATR (Compact Antenna Test Range) system and the results that are smoothed by 10-point adjacent average are shown in Fig. 6.

For illustration of multiple reflection contributions, the RCS of a dihedral constructed with two 1m × 1m metal planes is simulated with the depth of tracing set to 2. The result shown in Fig. 7 agrees well with that shown in [7]. The time for computing the RCS in the aspect angle range [-60◦, 60◦] with step 1◦ is about 100s.

Furthermore, the wide-band, wide-angle scattering data of complex targets are simulated by the GPU accelerated GRECO to obtain the high resolution range profile and ISAR image. Figure 8 is the turntable ISAR image of a Boeing 737 model obtained by the simulated data at X-band with bandwidth 300 MHz.

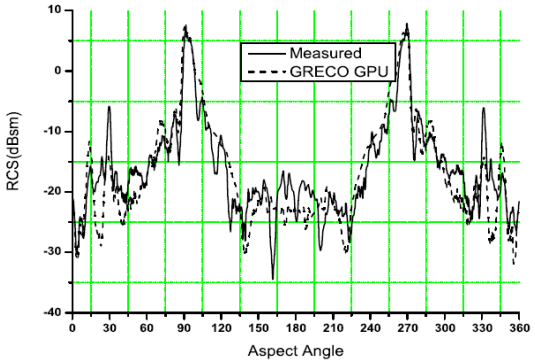

Fig. 6. Measured and simulated RCS of the missile model.

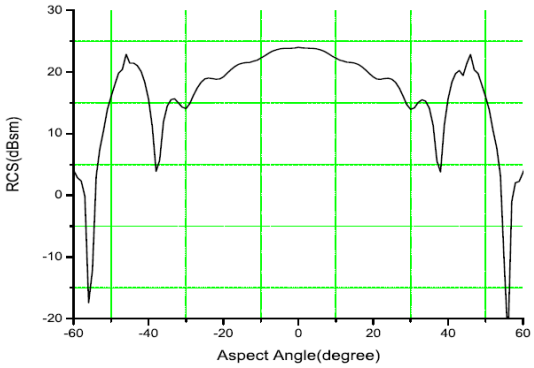

Fig. 7. Simulated RCS of dihedral.

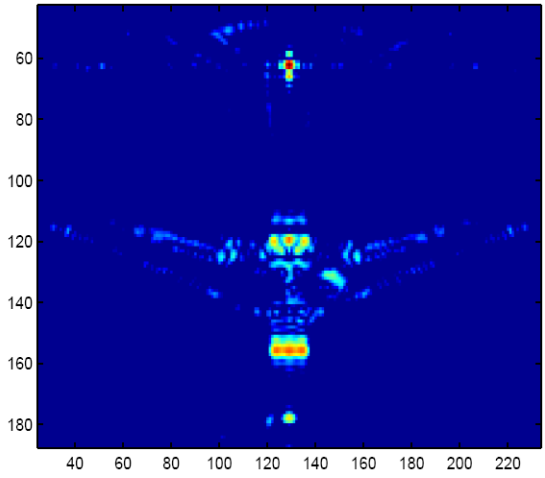

Fig. 8. Turntable ISAR image of a Boeing 737 model.

IV. CONCLUSION

The programmable pipeline of a modern GPU is applied successfully to the implementation of a GPU accelerated GRECO method with multiple reflection contribution included. The speed of simulation is improved up to 20 times compared with the raw GRECO. The GPU accelerated GRECO method has been used to simulate radar echo for different radar systems and the wide-band wide-angle scattering data of different targets for constructing the database for the radar target identification. Further improvement on simulation speed can be obtained by a more powerful GPU and better algorithms with the rapid development of computer graphics.

REFERENCES

[1] J. M. Rius, M. Ferrando, and L. Jofre, "High - frequency RCS of complex radar targets in real-time," *IEEE Trans. on Antennas and Propagat*., vol. 41, no.9, pp. 1308–1318, Sep. 1993.

[2] M. M. Okoniewski, S. E. Krakiwsky, L. E. Turner, "Graphics processor unit (GPU) acceleration of finite-difference time-domain (FDTD) algorithm," *ISCAS 2004, IEEE*, pp. 265–268, 2004.

[3] D. Goddeke, *GPGPU–basic math tutorial*, Technical report, FB Mathematik, Universitat Dortmund, http://www.mathematik/ unidortmund. de/~goeddeke/gpgpu, Nov. 2005.

[4] W. Enhua, "State of the art and future challenge on general purpose computation by graphics processing unit," *Journal of Software*(In Chinese), vol. 15, no. 10, pp. 1493–1504, Oct. 2004.

[5] R. J. Rost, *OpenGL Shading Language*, Addison Wesley, 2004.

[6] Y. ZhengLong, J. Lin, N. JingLing, and F. Dagang, "Bistatic RCS calculation of complex target by Greco," *Journal of Electronics* (In Chinese), vol. 32, no. 6, pp. 1033–1035, June 2004.

[7] E. F. Knott, J. F. Shaeffer, and M. T. Tuley, *Radar Cross Section*, Artech House, 1985.

[8] R. S. Wright and Jr. B. Lipchak, *OpenGL Super Bible*, Third Edtion. Sams Publishing, 2005.

[9] R. Fernando, *GPU GEMS, Programming Techniques, Tips, and Tricks for Real-Time Graphics*, Pearson Education, Inc., 2005.

[10] Y. ZhengLong, F. DaGang, and L. TieJun, "Modification of software for computing EM scattering of complex targets," *System Engineering and Electronics* (In Chinese), vol. 24, no. 4, pp. 86–89, April 2002.

[11] Q. DeHua, W. BaoFa, and L. TieJun, "Improvements of edges detecting and diffraction field computing in GRECO," *Journal of Electronics* (In Chinese), vol. 31, no. 8, pp. 1160–1163, Aug. 2003.

[12] H. Ling and R. Bhalla, "Three-dimensional scattering center extraction using the shooting and bouncing ray technique," *IEEE Trans. on Antennas and Propagat*., vol. 44, no. 11, pp. 1445–1453, Nov. 1996.

[13] T. J. Purcell, I. Buck, W. R. Mark, and P. Hanrahan, "Ray tracing on programmable graphics hardware," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 703–712, July 2002.

[14] D. Badouel, *An Efficient Ray-Polygon Intersection*, Academic Press Professional, Inc., San Diego, CA, USA, pp. 390–393, 1990.

[15] D. Cohen and Z. Sheffer, "Proximity clouds-an acceleration technique for 3D grid traversal," *The Visual Computer*, vol. 11, no. 1, pp. 27–38, 1994.

[16] A. Fujimoto, T. Tanaka, and K. Iwata, "Arts: Accelerated ray-tracing systems," *IEEE Comput. Graph*., vol. 6, no. 4, pp. 16–26, 1986.

[17] D. F. Rogers, *Prodedural Elements for Computer Graphics*, 2nd Edition. McGraw-Hill Press, 1998.

[18] J. Kim, *GPGPU reduction example*, http://web. imrc. kist.re.kr/~jwkim/GLSL/reduction.zip, 2005.

Yang ZhengLong was born in Gansu province, China. He received Ph.D. in electromagnetics and microwave technology at Nanjing University of Science and Technology (NUST) in 2002, then joined Nanjing Research Institute of Electronics Technology. His main research interests are computational electromagnetics and radar imaging.

Jin Lin was born in Jiangxi province, China. He received the master degree and Ph.D. in electronic engineering at Beijing University of Aeronautics and Astronautics (BUAA) in 1989 and 2003, respectively. He is the supervisor of R&D Center of Nanjing Research Institute of Electronics Technology and the member of IEEE and senior member of CIE (Chinese Institute of Electronics). His main research interests are radar system and microwave antenna.

Li WeiQing was born in Hebei province, China. He received Ph.D. in Computer Sci. and Tech. Inst. at NUST. His main research interests are computer graphics, virtual reality and system simulation etc.