Progress In Electromagnetics Research, PIER 81, 1–19, 2008

FROM CPU TO GPU: GPU-BASED ELECTROMAGNETIC COMPUTING (GPUECO)

Y. B. Tao, H. Lin, and H. J. Bao

State Key Laboratory of CAD&CG
Zhejiang University
Hangzhou 310027, P. R. China

Abstract—In this paper, we provide a new architecture by using the programmable graphics processing unit (GPU) to move all electromagnetic computing code to graphical hardware, which significantly accelerates Graphical electromagnetic computing (GRECO) method. We name this method GPUECO. The GPUECO method not only employs the hidden surface removal technique of graphics hardware to identify the surfaces and wedges visible from the radar direction, but also utilizes the formidable of computing power in programmable GPUs to predict the scattered fields of visible surfaces and wedges using the Physical Optical (PO) and Equivalent Edge Current (EEC). The computational efficiency of the scattered field in fragment processors is further enhanced using the Z-Cull and parallel reduction techniques, which avoid the inconsistent branching and the addition of the scattered fields in CPU, respectively. Numerical results show excellent agreement with the exact solution and measured data and, the GPUECO method yields approximately 30 times faster results.

1. INTRODUCTION

The prediction of the high-frequency radar cross section (RCS) of electrically large and complex targets is usually very complex. One popular and effective method is the high-frequency approximations [1–6], such as Geometrical Optics (GO), Physical Optics (PO) [7–11], Shooting and bouncing rays (SBR) [12–14], and other diffraction or scattering methods [15, 16]. The visibility computing that detects the surfaces or wedges of the target visible from the radar direction is a complex and time-consuming part of the high-frequency techniques [17, 18], while Graphical electromagnetic computing (GRECO) [3] method solves this problem effectively using the Z-Buffer of the

workstation graphics hardware, which is one of the most popular hidden surface removal algorithms in computer graphics. The Physical Optical (PO) and Physical Theory of Diffraction (PTD) are used to compute the first-order far field scattered from visible surfaces and wedges in the GRECO method, respectively. Besides the monostatic RCS, the GRECO method is also extended to predict the bistatic RCS of arbitrary shapes by the Physical Optical (PO) and Incremental Length Diffraction Coefficients (ILDC) [19].

Recent rapid development of graphics hardware has made significant impact on graphical processing, especially the programmable graphics processing unit (GPU) makes the graphics hardware more flexible and effective for rendering and other computationally demanding tasks [20]. As the graphics hardware based visibility computing has accelerated the RCS prediction in the GRECO method, it would be straightforward to move the electromagnetic computing code to graphics hardware through the programmable GPU, to further reduce the computation time of the RCS prediction.

In this paper, a new architecture of the GRECO method based on the programmable GPU is proposed for both monostatic and bistatic RCS prediction of electrically large and complex targets, which we called GPUECO. The identifications of visible surfaces and wedges are processed separately, and then the Physical Optical (PO) and Equivalent Edge Current (EEC) are applied to calculate the scattered fields of visible surfaces and wedges of the target respectively in several passes. The proposed approach greatly improves the accuracy and efficiency of the RCS prediction. This paper is organized as follows: Section 2 reviews the programmable pipeline of modern commodity graphics hardware, Section 3 describes the GPUECO method in detail, and numerical results and discussions are presented in Section 4.

2. PROGRAMMABLE PIPELINE OF GRAPHICS HARDWARE

With the rapid development of graphics hardware, especially the programmability of GPU, commodity graphics hardware, for instance NVIDIA GeForce 6 and 7 series, provides large memory bandwidth and high computing power in general-purpose processing, which is known as GPGPU (General-Purpose processing on the GPU) [20]. For example, the NVIDIA GeForce 6800 Ultra can obtain 35.2 GB/sec of memory bandwidth and over 53 GFLOPS of computing power, while the theoretical peak for the SSE units of a dual-core 3.7 GHz Intel Pentium Extreme Edition 965 is 8.5 GB/sec and 25.6 GFLOPS [20–22].

The current rendering pipeline is shown in Fig. 1. The underlying architecture of the programmable graphics hardware is called Single Instruction Multiple Data (SIMD), i.e., many parallel processors execute the same instruction on different data at a time. The vertex processor and fragment processor are programmable using the high-level shading languages like Cg and the OpenGL shading language. For example, Cg provides instructions including numeral operation, texture sampling, flow control, etc., and supports multiple built-in data types, such as boolean, integer, single floating point, matrix, vector of other types, structure, and array [23].

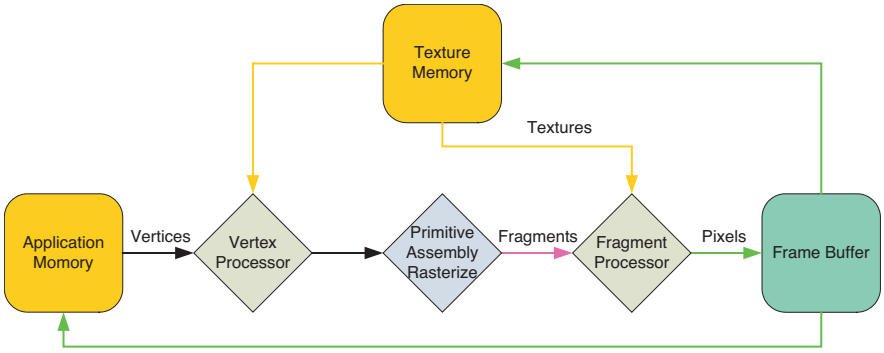

Figure 1. The programmable pipeline of graphics hardware.

The rendering pipeline starts with sending the geometric objects, for instance triangles or quadrangles, to graphics hardware from CPU. The geometric objects are described by 3D coordinates (vertices), connectivity information, and other numerical information (i.e., texture coordinates). In the first stage of the pipeline, multiple vertex processors execute the same instruction of the vertex program in parallel on different vertices. In general, the positions of vertices are transformed from object coordinates to camera coordinates. In the camera coordinate system, lighting is performed according to the transformed position and lighting information.

In the next stage of the pipeline, the transformed vertices are grouped into primitives, which are points, lines, or triangles, according to the connectivity information. The positions of vertices are then projected from camera coordinates to screen coordinates. Per-primitive operations are used to remove primitives that aren't visible at all and to clip primitives that intersect the view frustum. After performing edge and plane equation setup on the vertices, the primitives are rasterized into discrete points (fragments), each of which has related screen position, depth, color, and other numerical

information (texture coordinates) by interpolating the values of vertices.

Similar to the vertex processors, multiple fragment processors operate in concert applying the fragment program to each fragment independently, and produce the final result, which is usually the color information. The texture memory (in the following we use the texture for short) could be accessed in fragment processors through texture coordinates to obtain additional information for generating the final result.

In the final non-programmable stage, the color information of the fragment is recorded to the relevant pixel on the frame buffer. If multiple fragments are mapped to the same pixel position, the depth-buffer test (Z-Buffer hidden surface removal algorithm) decides which one is written into the frame buffer, by comparing the depth values of fragments. The data format of the frame buffer is not restricted to 8 bits of each RGBA color component, but it could be also a single floating point. In addition, the multiple-render-targets technique allows graphics hardware to render at most 4 render targets (multiple frame buffers) in one pass, totally 16 floating-point numbers, and the render-to-texture technique allows the frame buffer as the input texture in the next rendering pass if necessary.

3. GPUECO

The GRECO method makes use of parametric surfaces to represent the target geometry. Parametric surfaces must be tessellated into planar geometric objects before rendering, which is an error-prone conversion, while triangles are friendly to modern graphics hardware and could be processed more effectively as described in Section 2. Additionally, targets in terms of facets and wedges are universal in general processing and also common in the RCS prediction. Therefore, this paper uses a collection of facets and wedges to model the target geometry.

Since the scattered field of each rendered pixel is calculated independently, we employ the data-parallel method of the GPU computational mapping concepts [24] to transfer the electromagnetic computing to graphics hardware. The procedure of the GPUECO method and all fragment shaders are illustrated in Fig. 2. The graphical processing (visibility computing) and electromagnetic computing in the GPUECO method are both processed in programmable GPUs through several passes. On each pass, fragment processors are associated with one fragment shader (the computational kernel), which is a set of software instructions and is analogous to the CPU inner loop. By using the multiple-render-targets and render-to-texture techniques, the outputs

of the fragment shader are floating-point textures, which are equal to CPU arrays. All these textures have the same size to ensure the one-to-one mapping in electromagnetic computing. The scattered fields of surfaces and wedges of the target are processed separately in the left and right procedures and finally, the total scattered field is given by summing the contributions of each element through the Reduce Shader.

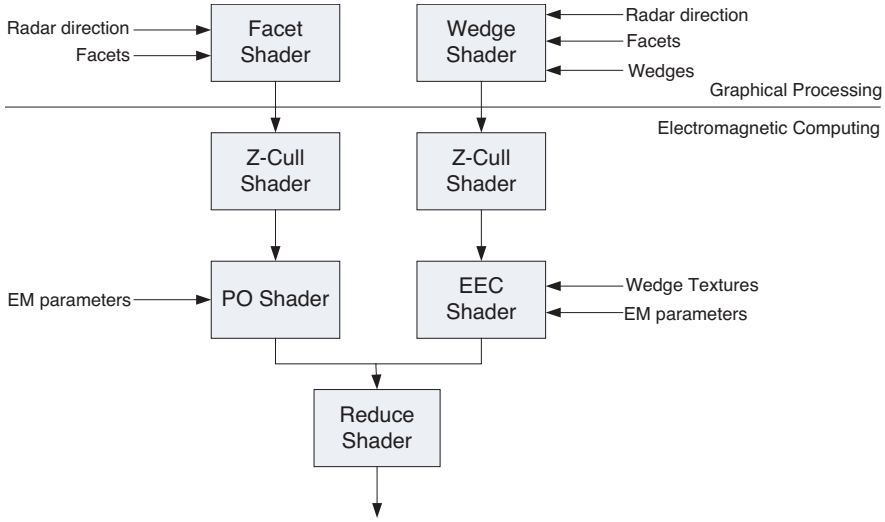

Figure 2. The procedure of GPUECO for the RCS prediction.

3.1. Scattered Filed of Surfaces of the Target in GPUECO

As shown in the left of Fig. 2, the procedure for predicting the scattered fields of surfaces of the target is composed of three shaders: the Facet Shader, the Z-Cull Shader, and the PO Shader. The data diagram of this procedure is illustrated in Fig. 3. The position and normal of visible facets are rendered into two textures in the Facet Shader, and then the Z-Cull Shader processes the background pixels and marks the depth values of these pixels. Hence, the PO Shader only calculate the scattered fields of rendered pixels and record the scattered fields into the corresponding pixels of two field textures. Note that all these floating-point textures have the same size. The details of these three shaders are discussed in the following subsections.

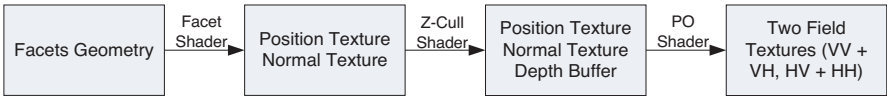

Figure 3. Data diagram for the scatted field prediction of surfaces of the target.

3.1.1. The Facet Shader

The Facet Shader takes as input the vertex coordinates and two associated texture coordinates, which are the position and normal of the vertex. In the stage of vertex processors, the position of the vertex is transformed into the coordinate system of the radar direction, lighting is canceled out, and the texture coordinates remain unchanged. The texture coordinates of fragments inside the facet are interpolated through these of vertices during the rasterization. In fragment processors, the Facet Shader outputs the texture coordinates as the color information to two textures separately. Finally, whether the color information should be written into the corresponding pixel of the texture depends on the result of depth-buffer test. After the Facet Shader, the pixels of these two textures are the position and normal of each point of the target visible from the radar direction, respectively.

The GRECO method sets up the light directions subtly to obtain the unit normal and needs rendering the target twice, i.e., positive and negative lighting axis directions, to obtain the positive and negative value of the normal separately, while the GPUECO method cancels out the lighting and renders the target only once through the programmable GPU, which is much simple and effective. The numerical precision of the position and normal in the GPUECO method is more precise compared to the GRECO method, where the normal components are discretized to [0, 255] in 8 bits.

Figure 4 shows two images of an aircraft rendered through the Facet Shader. The color components (R, G, B) are equal to the position (x, y, z) of visible surfaces in the Fig. 4(a), and the normal components correspond to the color components in the Fig. 4(b). The background color is changed from black to white to illustrate the part that each component of the color is less than zero.

3.1.2. The Z-Cull Shader

With the knowledge of the position and normal of each rendered pixel of the target, it is ready to calculate the scattered field using the PO integral. If we directly apply the PO shader in the data-

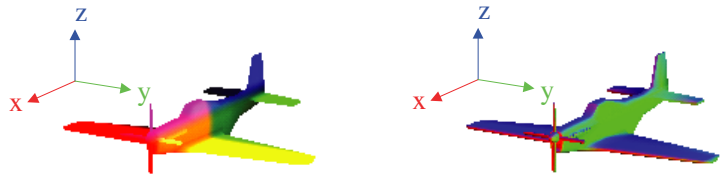

Figure 4. Rendered images of an aircraft, (a) position image, the color components are the 3D position of visible surface in the radar direction, (b) normal image, the normal components correspond to the color components.

parallel method, when the pixels of different types, i.e., the rendered pixels and background pixels, are simultaneously processed in fragment processors in the SIMD fashion, divergence in the branching will slow the performance. The reason is that both sides of the branching must be evaluated in the divergent branching, hence, the fragment processors that process the background pixels execute the same instruction as the other processors, but output zero at last.

This divergent branching can be avoided through the Z-Cull technique, which is one effective technique of moving branching up the pipeline [20]. Z-Cull is implemented in modern commodity graphics hardware and operates during the rasterization stage. When the pixel positions covered by each primitive have been determined, Z-Cull can quickly discard the fragments that fail the depth-buffer test. Only fragments that pass the depth-buffer test are processed in the subsequent stages of the pipeline and the outputs are recorded in the frame buffer. Hence, if the depth buffer could be marked in the way that only rendered pixels can pass the depth-buffer test, Z-Cull can make fragment processors only dealing with rendered pixels in the coherent branching and subsequently, greatly improves the utilization of fragment processors and the computational efficiency of the PO integral.

The Z-Cull Shader is designed to mark the depth values in the position of background pixels with the minimum depth value. The depth buffer is cleared with the maximum depth value (usually 1) as usual, and then the Z-Cull Shader is invoked by rendering a rectangle with the minimum depth value (usually 0). The rectangle covers the exact size of the frame buffer and its texture coordinates corresponds to the range of the normal texture. This ensures the one-to-one mapping between the pixel in the frame buffer and the pixel in the normal texture. In the Z-Cull Shader, the normal of the fragment can be

accessed in the normal texture through texture coordinates. The fragments with the zero normal (the position of background pixels) can pass the fragment processor and the corresponding depth value is replaced with the minimum depth value, while the pixels with the non-zero normal (the position of rendered pixels) are discarded and left the PO Shader to process, and the relevant depth values remain the maximum value.

3.1.3. The PO Shader

With the marked depth buffer, the scattered field of each rendered pixel of the target can be calculated effectively in the PO Shader. The PO Shader is invoked using the same rectangle and texture coordinates of the Z-Cull Shader, except that the depth value of the rectangle is the middle depth value, say, 0.5. The fragments corresponding to the background pixels are skipped by Z-Cull before the fragment processor, because the corresponding depth values in the depth buffer are smaller than these of current fragments. As a result, only the fragments in the position of rendered pixels continue to be processed in the PO Shader.

In the PO Shader, the position and normal of each rendered pixel are obtained from the position texture and normal texture through the well-designed texture coordinates. Besides the position and normal, the PO Shader also receives the parameters of the pixel size, the frequency, the direction of incidence and observation, and the direction of incident and observational polarization. Each processed fragment in the PO Shader corresponds to a small parallelogram, which we called patch for simplicity. The center and normal of the patch are the position and facet normal of the pixel, respectively. The four vertices of the patch can be calculated using the position, the normal, and the incident direction.

As the monostatic RCS is a special case of the bistatic RCS that the observational direction is equal to the incident direction, we employ the formula of the bistatic PO to calculate the scattered fields of the surfaces. The contribution to far field of the planar PEC patch can be approximated by the PO integral is expressed as [25]:

$$\sqrt{\sigma} = \frac{-\hat{n}\cdot(\hat{e}_r \times \hat{h}_i)}{\sqrt{\pi}T^2} e^{-jk\vec{r}_0\cdot\vec{w}} \sum_{n=1}^{4} (\hat{p}\cdot\vec{a}_n) e^{-jk\vec{r}_n\cdot\vec{w}} \frac{\sin(\frac{1}{2}k\vec{a}_n\cdot\vec{w})}{\frac{1}{2}k\vec{a}_n\cdot\vec{w}} \quad (1)$$

where σ is the bistatic RCS of the patch, $\hat{n}$ is the unit normal of the patch, $\hat{e}_r$ is the observational direction of polarization, $\vec{r}_0$ is the source position, $\vec{w} = \hat{i} - \hat{s}$, $\hat{i}$ is unit direction of incidence, $\hat{s}$ is the unit direction of observation, $\vec{a}_n$ is the nth edge vector of the patch, $\vec{r}_n$ is the center

position of nth edge of the patch, T is the projected length of $\vec{w}$ on the patch, $\hat{p} = \hat{n} \times \vec{w}/\hat{n} \times \vec{w}$.

Equation (1) is used to calculate the scattered field of each rendered pixel in the PO Shader for both the vertical and horizontal polarization, and the final results are 12 floating-point numbers. In order to reduce the number of outputs, the complex results of vv, vh, hv, and hh polarization are produced, and the 8 floating-point numbers can be stored in two field textures as shown in Fig. 3.

3.2. Scattered Field of Wedges of the Target in GPUECO

Besides the scattered fields of visible facets, the contribution of visible wedges must be considered for a realistic RCS prediction. The wedge detection method in the GRECO method has the limitation that the wedge across the neighbor pixels with the discontinuous depth values or on the contour could not be identified effectively using just the position and normal of rendered pixels. Hence, in the GPUECO method, a separate procedure, which includes the Wedge Shader, the Z-Cull Shader, and the EEC Shader as shown in the right of Fig. 2, is proposed for the visible wedge detection and the scatted field prediction of visible wedges.

The data diagram of this procedure is illustrated in Fig. 5. In the preprocessing stage, the actual wedges of the target are all identified and the associated information is stored into three static textures, here static means these textures will not be changed during the RCS prediction. In the runtime, the position and index of actual wedges are rendered into the wedge texture in the Wedge Shader, and then the Z-Cull Shader in the similar way processes the background pixels and marks the depth values of these pixels. Finally, the EEC shader calculated the scattered fields of rendered pixels using these three static textures and the wedge texture, and the scattered field is recorded in the corresponding pixels of two field textures. The wedge texture is the same size as the normal texture, and the two field textures are actually the same field textures in Fig. 3.

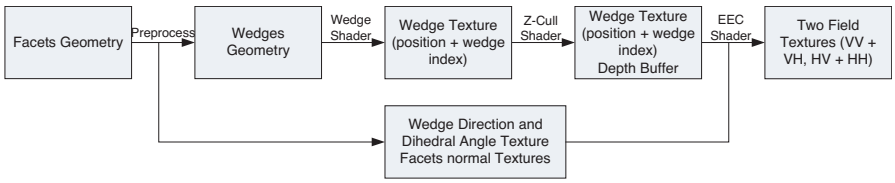

Figure 5. Data diagram for the scatted field prediction of wedges of the target.

3.2.1. The Preprocessing Stage

In the preprocessing stage, it is necessary to identify all actual wedges of the target for the following visibility computing. The facets are parsed to construct the adjacent facet information of all wedges, and then the dihedral angles of the wedges can be easily deduced from the adjacent facet information. For the boundary wedges, the dihedral angle of the wedge is assumed to be zero. The wedge due to the facet noise can be removed by checking whether the dihedral angle of the wedge is larger than the user-defined angle, and the remaining wedges are actual wedges in the target.

The computational parameters, which are needed in the EEC Shader, are packed into three static floating-point textures. As shown in Fig. 6, the content of one texture is the unit direction along the wedge and the dihedral angle of wedges, and the others are the unit normal of adjacent faces of actual wedges. These textures contain all computational parameters for the EEC integral for all actual wedges. Note that the size of these static textures is different with the wedge texture.

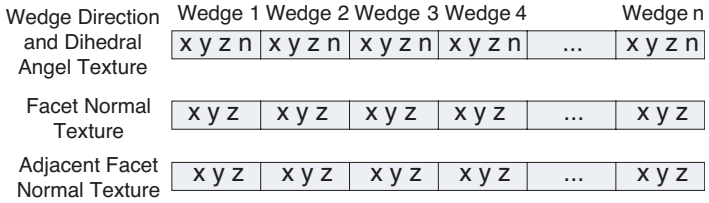

Figure 6. Texture information of actual wedges, including the direction of the wedge, the dihedral angle and the normal of adjacent facets of the wedge.

3.2.2. The Wedge Shader

The visible wedges are generated in two rendering passes. The first pass renders the facets of the target, but writing into the wedge texture is disabled. This implies that the first pass just renders the depth values into the depth buffer. In the second pass, the actual wedges that are detected in the preprocessing stage are rendered with the depth buffer of the first pass, and now writing into the wedge texture is allowed. In this manner, the shadowed wedges are removed, because the depth value of the invisible wedge is larger than the corresponding depth value in the depth buffer generated in the first pass. Hence, only the visible wedges are rendered into the wedge texture.

In the second pass, the texture coordinates are used to transfer the wedge position and wedge index to the programmable GPU and the Wedge Shader outputs the texture coordinates as the color information to the wedge texture. The visible wedge position are recorded in RGB components of the pixel in the wedge texture, and the A component is the corresponding the wedge index number, which starts with 1 to differ from the black color of background. It can be noted that the GPUECO method bypass the limitation of visible wedge detection in the GRECO method by rendering the wedge index into the wedge texture.

As shown in Fig. 7, an image of actual wedges of an aircraft rendered by the Wedge Shader is presented. The color components (R, G, B) are equal to the 3D position (x, y, z) of visible wedges, and the black color is due to that each component of the position (x, y, z) is less than zero.

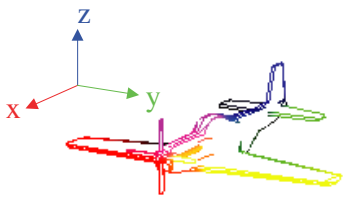

Figure 7. Rendered image of actual wedges of an aircraft. The color components (R, G, B) are the 3D position (x, y, z) of visible wedges in the radar direction. The background is changed from black to white to illustrate the part that each component of the position (x, y, z) is less than zero.

3.2.3. The EEC Shader

In the next EEC Shader, the position and index of visible wedges can be obtained from the wedge texture, and the unit direction along the wedge and the dihedral angle, and the unit normal of adjacent facets can be accessed from three static textures through the wedge index. The size of pixel, the frequency, the direction of incidence and observation, and the direction of incident and observational polarization are parameters as input for the EEC Shader.

The Ufimtsev PTD coefficients could be obtained using a very simple linear approximation in the GRECO method [3]. However, the PTD coefficients are only valid in observational direction of the Keller

cone. Mitzner developed the incremental length diffraction coefficients (ILDC) [26], which is valid for bistatic diffraction calculation. In this paper, Michaeli's physical theory of diffraction equivalent edge currents (EEC) [27, 28] will be considered.

According to the EEC theory, a high-frequency approximation to the fringe wave (FW) field is calculated from a line integral along the visible part of the wedge. The formulation in terms of diffraction coefficients can be found in the literature [27, 28]. The final formula for the wedge crossing the neighborhood pixels is defined as

$$\begin{aligned}\vec{E}^d \;=\; & \frac{\exp(-jks)}{2\pi s} dl \left[(D_m - D'_\perp)\hat{e}^s_\perp \cos\gamma - (D_e - D'_\parallel)\frac{\sin\beta}{\sin\beta'}\hat{e}^s_\parallel \sin\gamma \right. \\ & \left. -(D_{em}\sin\beta' - D'_x)\frac{\sin\beta}{\sin\beta'}\hat{e}^s_\parallel \cos\gamma \right] \end{aligned} \tag{2}$$

where dl is the length of the visible wedge in the neighbor pixels, which can be calculated from the wedge direction and the incident direction, $\hat{e}^s_\perp$ and $\hat{e}^s_\parallel$ are the unit vectors of the local coordinate systems as defined in [25, 29], γ is the angle subtended by the incident electric field and the normal to the plane of incidence, β and β' are the angles between the wedge and the incident and observational direction, respectively, and the D are the diffraction coefficients, the detail formulas are explained in [25, 29].

The EEC Shader applies the Eq. (2) to calculate the scattered field of each rendered pixel of visible wedges in the data-parallel method similar to the PO Shader, and produces the results of vv, vh, hv, and hh polarization. These results are added to the scattered fields of the two field textures generated in the PO Shader in the corresponding position.

3.3. Data Reducing in GPUECO

When the PO Shader and EEC Shader are finished, the scattered fields of vv, vh, hv, and hh polarization, total 8 floating-point numbers, are available in two field textures, and should be summed up to obtain the total scattered field.

One straight way is that reading the data of these two textures back to CPU and summing up the scattered filed in CPU. However, if the resolution of the texture is 1024×1024, each direction needs read back 32 M ($1024 \times 1024 \times 8 \times 4$) data, and CPU has to traverse these memories to obtain the total scattered field, which reduces the performance of the GPUECO method.

The more effective method is to reduce the large vector of scattered fields to 8 floating-point numbers in GPU and read the 8 floating-point

numbers back to CPU, which is called a parallel reduction [20]. The parallel reduction avoids the read back of large texture to CPU and simultaneously reduces the processing time of CPU.

The parallel reduction is implemented in the Reduce Shader, which will be invoked in multiple passes. On each pass, the size of the rectangle is half of the rectangle of last pass. The rectangle of the first pass is equal to the size of the field texture. In the fragment processor, the Reduce Shader reads the data of a 2×2 block from the field texture (previous results), sums up the data of the block, and outputs the result to the corresponding pixels of the field texture (new results). The Reduce Shader is applied in general $O(\log n)$ passes, where n is the size of the field texture. The final pass output two pixels, which are the total scattered fields. Fig. 8 shows the addition reduction on a texture with one floating-point component.

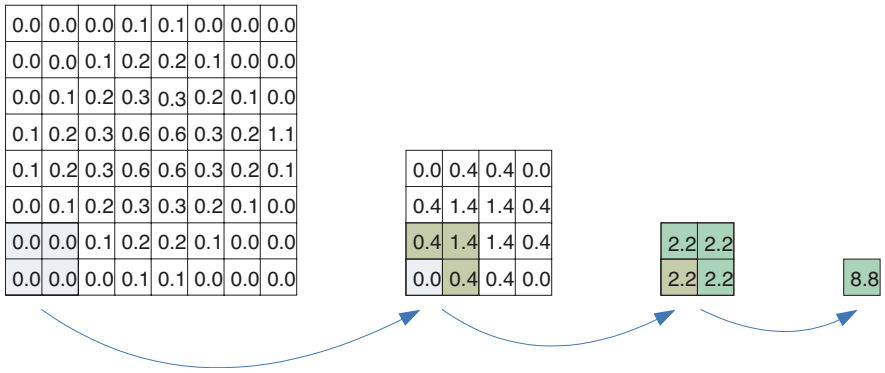

Figure 8. Addition reduction preformed with multiple passes.

4. NUMERICAL RESULTS

In order to validate the accuracy and computational time of the GPUECO method, several numerical examples are presented. The calculations were performed using a 2.8 GHz Pentium(R) D CPU and NVIDIA GeForce 7950 GT graphics card. The maximum resolution of the frame buffer is usually 4096×4096, while the 1024×1024 resolution is used for the following predictions in order to balance the discretization error [30] and computation time.

The first example is a sphere with radius 1 m for 180-degree bistatic RCS calculation in 181 equal-spaced incident directions at 1 GHz frequency. Fig. 9 shows the RCS comparison of both vv-polarization and hh-polarization of the result of GPUECO method

(solid line) and result of MLFMM (dot line). The aspect angle of zero degree means the direction of observation is parallel to the incident direction. The result of GPUECO for the zero degree is 5.07 dBsm, which is great agreement with the analytical solution $10 \lg \pi r^2 = 4.97$ dBsm. When the aspect angle is larger than 90 degree, there are some deviations from the MLFMM data. This is due to that PO assumes the zero induced current in the shadowed regions. However, the induced current in shadowed regions could not be omitted when the region is visible from the observational direction.

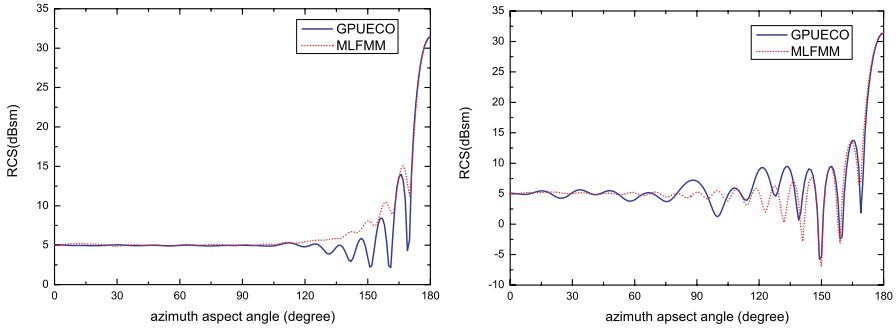

Figure 9. Comparison of our results (solid line) and MLFMM data (dot line) for a sphere at 1 GHz, (a) vv-polarization, (b) hh-polarization.

The second simulation is used to verify the efficiency of visible wedge detection and the accuracy of EEC. The cube with side length 1 m is used to predict the 90 degree monostatic RCS in 91 equal-spaced incident directions at 3 GHz frequency. Both vv-polarization and hh-polarization RCS of the GRECO result (dot line), the GPUECO result (solid line), and MLFMM result (circle) are illustrated in Fig. 10. The GPUECO result shows an excellent agreement with the MLFMM data. As the GRECO method could not detect the wedge on the contour, the deviation of the GRECO result from the MLFMM data is larger than the deviation of the GPUECO result and MLFMM data, especially the vertical polarization between 30° and 60°.

Besides the relatively simple objects, the generic complex missile is used to validate the efficiency of the GPUECO method for general complex objects. Fig. 11 shows the geometry of the generic missile ($1.1 \times 0.25 \times 0.2$ m) and the horizontal polarization result (solid line) compared with the measured data (dot line). The monostatic RCS is calculated from $\phi = 0°$ to $\phi = 360°$ in 361 equal-spaced incident

directions at 7.5 GHz frequency. Very good agreement is found between these two results. The deviation near 180° is because there is a cavity on the tail of the missile, however, the GRECO method does not consider the multi-scattered terms.

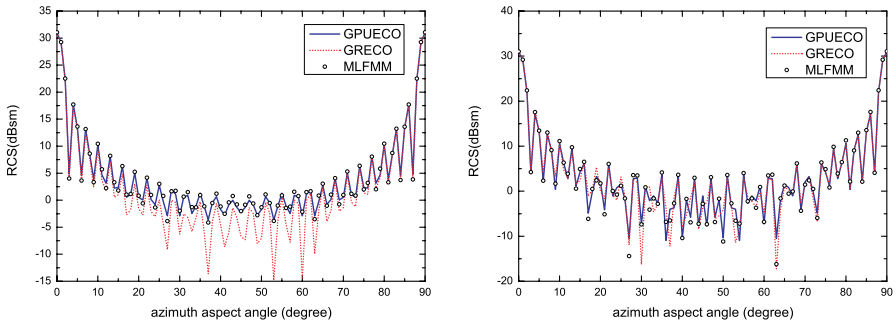

Figure 10. Comparison of our results (dot line), GRECO result (solid line) and MLFMM data (circle) for a cube at 3 GHz, (a) vv-polarization, (b) hh-polarization.

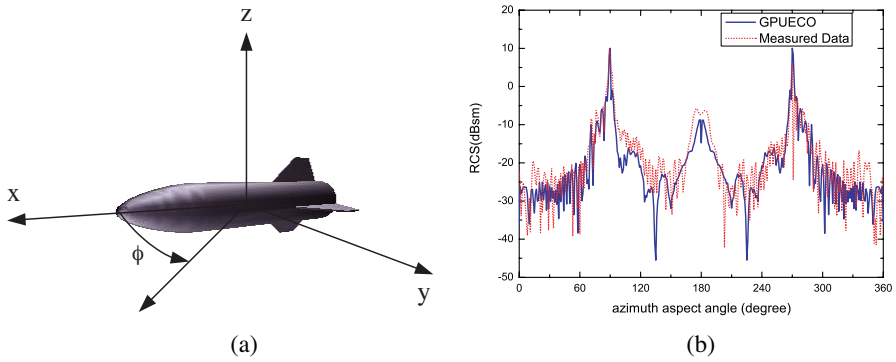

Figure 11. (a) The geometry of generic complex missile, (b) comparison of our results (solid line) and the measured data (dot line) of the missile at 7.5 GHz, hh-polarization.

The GPUECO method employs graphical hardware not only to remove the shadowed surfaces and wedges from the radar direction, but also to calculate the scattered field in the data-parallel fashion. The multiple fragment processors provide the powerful computing power for the electromagnetic computing. In addition, the Z-Cull

technique allows fragment processors dealing with the same branching and the parallel reduction technique further improves the efficiency of the RCS prediction. Table 1 show how the computation time reduces with these different accelerated techniques. The CPU-GRECO read the frame buffer of rendered pixels back to CPU and calculate the scattered field in CPU. The GPUECO I calculate the scattered fields of rendered pixels in GPU without using the Z-Cull and parallel reduction techniques, while the GPUECO II only use the Z-Cull technique and the GPUECO III make use of both the Z-Cull and parallel reduction techniques. From Table 1, we can conclude that significant speedups can be achieved by applying the Z-Cull and parallel reduction techniques in GPUECO, and the GPUECO method is at least 30 times faster than the CPU-GRECO method.

Table 1. Comparison of computation time (sec). The CPU-GRECO and GPUECO calculate the scattered field in CPU and GPU without using the Z-Cull and parallel reduction techniques, respectively. The GPUECO II is accelerated using the Z-Cull technique only and both Z-Cull and parallel reduction are used in GPUECO III.

Models	CPU-GRECO	GPUECO I	GPUECO II	GPUECO III
Sphere	412.43	9.307	7.459	1.804
Cube	317.20	6.064	4.256	1.411
Missile	120.37	23.77	14.97	3.57

5. CONCLUSION

It has been shown that thanks to the rapid development of graphics hardware, the GPUECO method moves all electromagnetic computing code to graphics hardware. Besides the hidden surface removal technique, the programmability of the powerful GPUs in conjunct with the Z-Cull and parallel reduction techniques significantly improves the computational efficiency of the RCS prediction. In addition, the floating-point precision and exact wedge detection enhance the accuracy of the scattered field effectively. Numerical results show excellent agreement with the exact solution and the measured data and demonstrate that the GPUECO method can greatly reduce the computational time.

ACKNOWLEDGMENT

The authors would like to thank Professor Tiejun Cui from South East University for providing their MLFMM method used in this paper. This paper is supported by China 863 Hi-Tech Research and Development Program (2002AA135020) and China 973 Program (2002CB312102).

REFERENCES

1. Youssef, N. N., "Radar cross section of complex targets," *Proc. IEEE*, Vol. 77, No. 5, 772–734, 1989.
2. Bouche, D. P., F. A. Molinet, and R. A. J. Mittra, "Asymptotic and hybrid techniques for electromagnetic scattering," *Proc IEEE*, Vol. 81, No. 12, 1658–1684, 1993.
3. Rius, J. M., M. Ferrando, and L. Jofre, "High frequency RCS of complex radar targets in real time," *IEEE Transaction on Antenna and Propagation*, Vol. 41, No. 9, 1308–1318, 1993.
4. Chen, X. J. and X. W. Shi, "Backscattering of electrically large perfect conducting targets modeled by NURBS surfaces in halfspace," *Progress In Electromagnetics Research*, PIER 77, 215–224, 2007.
5. Zhong, X. J., T. J. Cui, Z. Li, Y. B. Tao, and H. Lin, "Terahertz-wave scattering by perfectly electrical conducting objects," *Journal of Electromagnetic Waves and Applications*, Vol. 21, No. 15, 2331–2340, 2007.
6. Li, X. F., Y. J. Xie, and R. Yang, "High-Frequency method analysis on scattering from homogenous dielectric objects with electrically large size in half space," *Progress In Electromagnetics Research B*, Vol. 1, 177–188, 2008.
7. Perez, J. and F. Cátedra, "Application of physical optics to the RCS computation of bodies modeled with NURBS surfaces," *IEEE Transaction on Antenna and Propagation*, Vol. 42, No. 10, 1404–1411, 1994.
8. Chen, M., Y. Zhang, and C. H. Liang, "Calculation of the field distribution near electrically large NURBS surfaces with physical optics method," *Journal of Electromagnetic Wave Applications*, Vol. 19, No. 11, 1511–1524, 2005.
9. Zhang, P. F. and S. X. Gong, "Improvement on the forwardbackward iterative physical optics algorithm applied to compute the RCS of large open-ended cavities," *Journal of Electromagnetic Wave Applications*, Vol. 21, No. 4, 457–469, 2007.

10. Zhao, Y., X. W. Shi, and L. Xu, "Modeling with nurbs surfaces used for the calculation of RCS," *Progress In Electromagnetics Research*, PIER 78, 49–59, 2008.
11. Hemon, R., P. Pouliguen, H. He, J. Saillard, and J. F. Damiens, "Computation of EM field scattered by an open-ended cavity and by a cavity under radome using the iterative physical optics," *Progress In Electromagnetics Research*, PIER 80, 77–105, 2008.
12. Ling, H., R. C. Chou, and S. W. Lee, "Shooting and bouncing rays: Calculating the RCS of an arbitrarily shaped cavity," *IEEE Transaction on Antenna and Propagation*, Vol. 37, No. 2, 194–204, 1989.
13. Jin, K. S., T. I. Suh, S. H. Suk, B. C. Kim, and H. T. Kim, "Fast ray tracing using a space-division algorithm for RCS prediction," *Journal of Electromagnetic Waves and Applications*, Vol. 20, No. 1, 119–128, 2006.
14. Bang, J. K., B. C. Kim, S. H. Suk, K. S. Jin, and H. T. Kim, "Time consumption reduction of ray tracing for RCS prediction using efficient grid division and space division algorithms," *Journal of Electromagnetic Waves and Applications*, Vol. 21, No. 6, 829–840, 2007.
15. Mallahzadeh, A. R., M. Soleimani, and J. Rashed-Mohassel, "RCS computation of airplane using parabolic equation," *Progress In Electromagnetics Research*, PIER 57, 265–276, 2006.
16. Wang, N., Y. Zhang, and C. H. Liang, "Creeping ray-tracing algorithm of UTD method based on NURBS models with the source on surface," *Journal of Electromagnetic Waves and Applications*, Vol. 20, No. 14, 1981–1990, 2006.
17. Wang, N. and C. H. Liang, "Study on the occlusions between rays and NURBS surfaces in optical methods," *Progress In Electromagnetics Research*, PIER 71, 243–249, 2007.
18. Liang, C. H., Z. Liu, and H. Di, "Study on the blockage of electromagnetic rays analytically," *Progress In Electromagnetics Research B*, Vol. 1, 253–268, 2008.
19. Yang, Z. L., L. Jin, J. L. Ni, and D. G. Fang, "Bistatic RCS calculation of complex target by GRECO," *ACTA Electronica Sinica*, Vol. 32, No. 6, 1033–1035, 2004.
20. Owens, J. D., D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell, "A survey of general-purpose computation on graphics hardware," *Eurographics. Computer Graphics Forum*, Vol. 26, No. 1, 80–113, 2007.

21. Buck, I., "GPGPU: General-purpose computation on graphics hardware-GPU computation strategies & tricks," *ACM SIGGRAPH Course Notes*, August 2004.
22. "Intel processors product list," http://www.intel.com/products/processor, 2006.
23. Mark, W. R., R. S. Glanville, K. Akeley, and M. J. Kilgard, "Cg: A system for programming graphics hardware in a C-like language," *ACM Transactions on Graphics*, Vol. 22, No. 3, 896–907, 2003.
24. Harris, M., "Mapping computational concepts to GPUs," *GPU Gems 2*, Chapter 31, 493–508, Addison Wesley, 2005.
25. Knott, E. F., J. F. Shaeffer, and M. T. Tuley, *Radar Cross Section*, Artech House, New York, 1985.
26. Mitzner, K. M., "Incremental length diffraction coefficients," *Aircraft Division Northrop Corp. Tech. Rep. AFAL-TR-73-296*, Apr. 1974.
27. Michaeli, A., "Equivalent edge currents for arbitrary observation," *IEEE Transaction on Antenna and Propagation*, Vol. 32, No. 3, 252–258, 1984.
28. Michaeli, A., "Elimination of infinities in equivalent edge currents-Part I: Fringe current components," *IEEE Transaction on Antenna and Propagation*, Vol. 34, No. 7, 912–918, 1986.
29. Knott, E. F., "The relationship between Mitzner's ILDC and Michaeli's equivalent currents," *IEEE Transaction on Antenna and Propagation*, Vol. 33, No. 1, 112–114, 1985.
30. Rius, J. M., D. Burgos, and A. Cardama, "Discretization errors in the graphical computation of Physical Optics surface integral," *Applied Computational Electromagnetics Society (ACES) Journal*, Vol. 13, No. 3, 255–263, 1998.