Real Time Global Illumination Solutions to the Radiosity Algorithm Using Hybrid CPU/GPU Nodes

UMBC REU Site: Interdisciplinary Program in High Performance Computing

Oluwapelumi Adenikinju[1], Julian Gilyard[2], Joshua Massey[1], Thomas Stitt[3]

Graduate assistants: Jonathan Graf[4], Xuan Huang[4], Samuel Khuvis[4]

Faculty mentor: Matthias K. Gobbert[4], Clients: Yu Wang[1] and Marc Olano[1]

[1]CSEE, UMBC, [2]Wake Forest University, [3]Penn State University, [4]Math and Stat, UMBC

Problem

In computer graphics, an integral portion of the rendering timeline is solving for the light distribution of a scene. When realism is important, a solution involving global illumination – allowing light rays to bounce before hitting the viewer – is desired. One method that awards particularly good results for diffuse surfaces is the radiosity algorithm, despite the fact that this method is computationally expensive – using it for near-real to real time rendering is not generally practical. Starting with the existing computational radiosity solver `rrv`, we determine how runtimes can be reduced through a combination of multi-core CPU to massively-parallel GPU architectures available in the cluster maya of the UMBC High Performance Computing Facility (`www.umbc.edu/hpcf`).

Mathematical Background

Radiosity is defined as the total energy leaving a surface (the sum of emitted and reflected energy) and is given by:

$$B_i = E_i + \rho_i \sum_{j=1}^{N} B_j F_{i,j} \qquad \text{for } i = 1, 2, \ldots, N \qquad (1)$$

- B_i (radiosity) – the total energy leaving the surface (radiosity) of the ith patch (energy/unit time/unit area)
- E_i (emission rate) – the rate of energy leaving the ith patch (energy/unit time/unit area)
- ρ_i (reflectivity) – the reflectivity of the ith patch (unitless); the reflectivity depends on the wavelength of light
- $F_{i,j}$ (form factor) – the fraction of energy emitted from patch j that reaches patch i (unitless)
- N – the number of patches

The form factors $F_{i,j}$ are properties of the scene. The formulation can be found in HPCF-2014-15.

We want (1) as a linear system so that we can solve it computationally. Rewriting (1), we have

$$b = e + F\,b \quad \Longleftrightarrow \quad (I - F)\,b = e$$

and after substituting $A = I - F$, (1) can be rewritten as a linear system

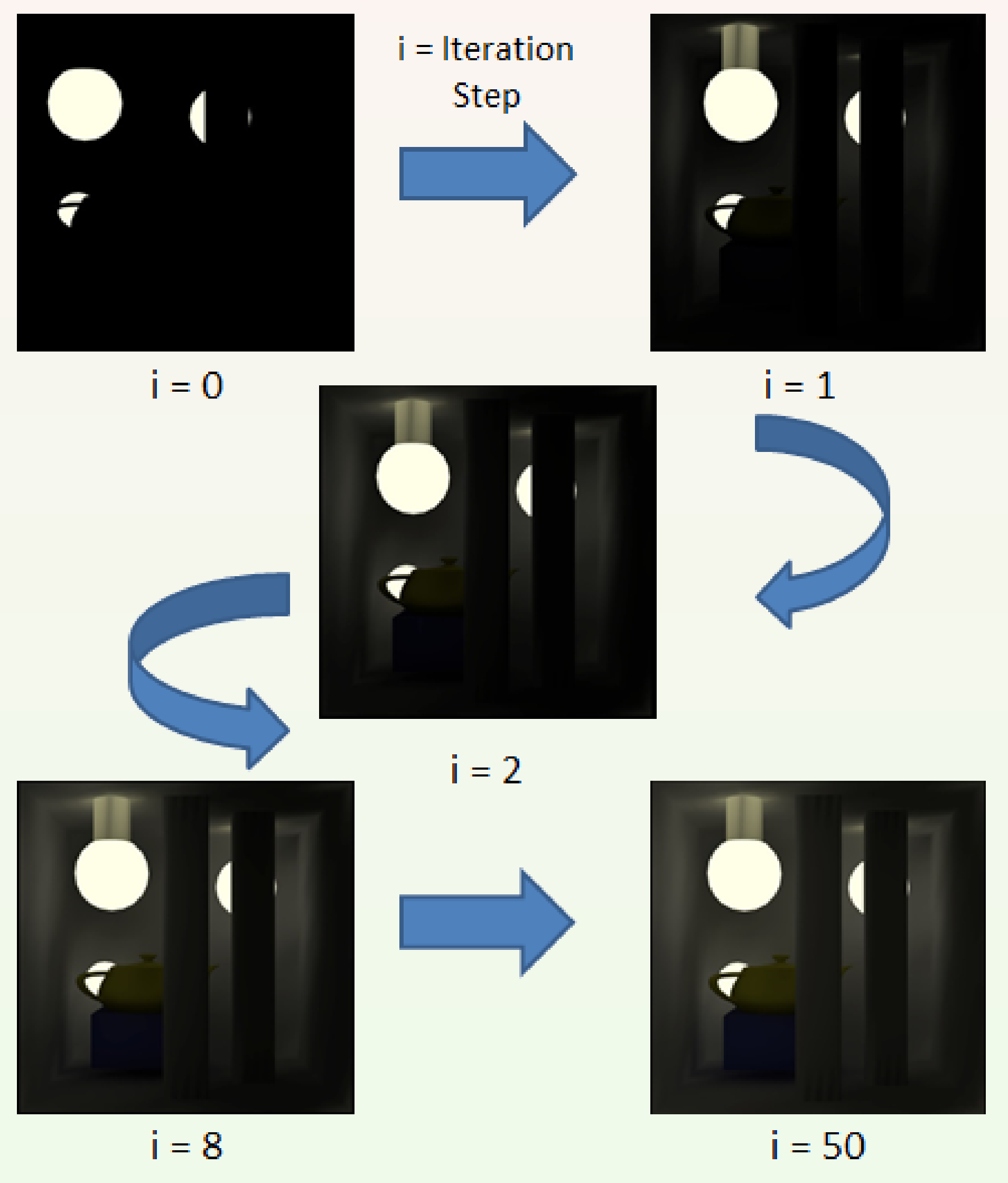
$$Ab = e \qquad (2)$$

which can be expanded to

$$\begin{bmatrix} 1-\rho_1 F_{1,1} & -\rho_1 F_{1,2} & \cdots & -\rho_1 F_{1,N} \\ -\rho_2 F_{2,1} & 1-\rho_2 F_{2,2} & \cdots & -\rho_2 F_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ -\rho_N F_{N,1} & -\rho_N F_{N,2} & \cdots & 1-\rho_N F_{N,N} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_N \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_N \end{bmatrix}.$$

Computational Methods

Jacobi – The original code in `rrv` solves for b in (2) using a fixed number of iterations in the Jacobi method. This can be physically thought of as one light bounce per iteration, as illustrated visually in the following flow chart:



To provide certainty that the method has actually converged and to save iterations potentially, we reimplement the Jacobi method to compute only until an equilibrium bouncing of light is reached as measured by a tolerance of 10^{-6} on the relative residual.

BiCG-STAB – The Jacobi method can be slow to converge. The BiCG-STAB method is an alternative iterative method whose convergence speed increases with iterations. It requires twice as much work per iteration, hence might take less time for complex problems that require a large number of iterations with the Jacobi method.

Runtimes in Seconds

Runtime results were obtained for 4 scenes of varied patch count and complexity. We see that the scenes used were not complex enough to make apparent the advantages of BiCG-STAB. Parallelization on the other hand showed marked improvement with run times dropping over an order of magnitude.

Computing Methods Jacobi vs. BiCG-STAB:

	Patch	Jacobi		BiCG-STAB	
Scene ID	count	iter	runtime	iter	runtime
1	1312	8	0.009	3	0.010
2	3360	28	0.045	16	0.058
3	9680	36	0.410	17	0.435
4	17128	32	0.993	17	1.157

Distributive Computing with Jacobi Methods:

Scene ID	Original	Serial	CUDA	OpenMP
1	0.028	0.031	0.006	0.009
2	0.857	0.677	0.115	0.045
3	10.072	6.973	1.209	0.410
4	27.855	19.394	3.415	0.993

Distributed Computing

CUDA – CUDA is NVIDIA's library for leveraging the massively parallel architecture of GPUs (graphics processing units). GPUs are designed for SIMD (single instruction multiple data) applications like those used in solving (2). CUDA has an efficient package (cuBLAS) for solving linear systems, but since the system is composed of vector elements with multiple values (arrays of structures), these methods cannot be used. We implement a dot-product and matrix-vector product for vectors of this type using binary tree reduction and a simple axpby ($a\,x + b\,y$) operation for vector scaling and addition.

OpenMP – OpenMP (Open Multi-Processing) is an application programming interface that provides simplified and cross-platform shared-memory parallelization. Memory is considered "shared" when each worker (thread) shares it with all other workers, unless marked worker-private – which is beneficial here since the matrix A is large. With OpenMP, portions of serial code are flagged to be run in parallel by work distribution among available threads. We utilize parallelization much like with CUDA to speed up the matrix-vector, dot, and axpby functions.

Conclusions

Both computational methods and parallelization were explored with the aim of near-real to real time global illumination solutions to the radiosity algorithm. Given the scenes we tested, OpenMP and CUDA both show substantial runtime improvements while the change from the Jacobi method to the BiCG-STAB method actually resulted in increased runtime due to the method's complexity. It appears that global illumination problems are not in general best suited for mathematical reformulations, but taking advantage of work distribution seems favorable no matter the scene complexity.

References

- For more results, see full technical report HPCF–2014–15 at `www.umbc.edu/hpcf` > Publications.
- `rrv` – Radiosity Renderer and Visualizer, DUDCA.cz: `http://dudka.cz/rrv`

Acknowledgments

- REU Site: `www.umbc.edu/hpcreu`
- NSF, NSA, UMBC, HPCF, CIRC