A parallel system for non-deterministic ray tracing

Michał Radziszewski, Witold Alda
AGH University of Science and Technology, al. Mickiewicza 30, 30-059 Kraków, Poland
mradzisz@student.agh.edu.pl, alda@agh.edu.pl

Figure 1. Difference between results of graphic card like algorithm (left), classic ray tracing (middle) and full global illumination (right). All guessed lighting terms (e.g. ambient light) were disabled deliberately, to show what given technique actually computes.

Abstract. Physically based global illumination algorithms produce images of unparalleled quality. These algorithms have wide range of possible applications – from modern entertainment (e.g. film, computer games or virtual reality applications) to CAD tools. However, the price for physical correctness is a very long rendering time. Our solution proposed in this paper deals with it by means of careful choice and modification of existing rendering algorithms, as well as parallel execution environment. The main purpose of the described project is a design and implementation of platform supporting non-deterministic ray tracing algorithms. The most important goals are effectiveness and clarity of design, which enables easy implementation and, with minor restrictions, automatic parallelization.

1. Introduction. Physically correct global illumination rendering is a powerful tool for rendering 3D scenes. The striking difference between results of global illumination and other techniques are shown in Figure 1. Until recently these algorithms were restricted to powerful supercomputers. However, nowadays clusters of PCs or multiprocessor machines can produce very good quality images in a reasonable time. This paper presents a platform for global illumination realized by Monte Carlo ray tracing. The basic intention is a full support of geometric optics based phenomena. The platform provides a clear, well defined interface between objects, cameras and rendering algorithms, as well as transparent parallel environment. Our model differs from previous approaches in numerous ways. First, it describes objects and cameras only by sampling. Second, it provides full spectral rendering. These features allow simulating arbitrary geometric optic effects with any physically correct algorithm that can be expressed as a sequence of sampling operations. The platform also benefits from symmetry of light transport. This provides consistent interfaces given to both camera and object, which greatly simplifies implementation of bidirectional algorithms. It also allows storing images in arbitrary format, i.e. raw sample data can be saved instead of RGB matrix, which enables huge freedom in enhancing images by sophisticated statistical analysis and other post processing algorithms.
2. Previous Work. The idea of ray tracing [12] is not new. Since its introduction in 1980, much effort has been spent to make this method faster and more accurate. The first approach that catches soft shadows and reflection by means of Monte Carlo sampling has been Distributed Ray Tracing [1]. However, this algorithm still finds difficulties with energy conservation. The first solution of global illumination problem which can be thought as the first physically correct rendering algorithm is due to Kajiya [4]. However, this method is often slow and well suited only for certain input scenes. The algorithms invented by Veach and Guibas [10] were designed to mitigate these problems. However, all the provably correct MC methods lead to artifacts which appear as distracting high frequency noise, unless very long computations are performed. There are also many known solutions which use different kinds of approximation. Most popular are Irradiance Caching [11] and Photon Mapping [3], in which error appears as less distracting excessive blur of indirect lighting.

There have also been many attempts to make ray tracing run in real time. This can be done by using GPUs [8] and/or due to parallelization [6]. However, currently available hardware does not allow to achieve correct global illumination in real time yet. Some simplifications which affect both flexibility of software and accuracy of solution have to be done.

Today there exist many frameworks for ray tracing, e.g. [5] and systems for global illumination, e.g. [7]. Some works [2] articulate importance of using full spectral rendering instead of RGB model. We have found this issue very important, especially with dispersion and non-white illumination.

3. Framework. Our idea is to make a framework based entirely on sampling, which allows greatest possible flexibility with Monte Carlo methods. The platform should also be physically correct, support full spectrum and be friendly to effective parallelization.

The framework specifies an interface mainly between 3D objects on a scene and a rendering algorithm. It establishes details such as sampling methods, color representation, random number generation, and ray intersection data. There were similar approaches in past, however designed mainly for classic ray tracing. Our method, on the other hand, is designed to be most clear and effective with Monte Carlo version of ray tracing, and therefore is based entirely on sampling.

3.1. Basics. The interface is based on a few structures: photon event, spectral color and hash data. These structures, with some other elements, are arguments of sampling routines. Photon event describes intersection of ray with object geometry and photon scattering. It is filled by intersection function, and can be read by subsequent function calls. Spectral color describes, depending on the routine, emission spectrum, camera sensitivity or scattering value. The hash data is used for generating pseudorandom numbers necessary for sampling.

3.2. Pseudorandom Numbers. Pseudorandom numbers are used instead of theoretically valid true random ones. In fact, despite not being strictly correct without theory modification, this allows reproduction of any sequence of operations, and sometimes better convergence. There are two kind of random number generators. First are generators which preserve state, with initialization routine and ‘next’ routine. These generators can produce numbers only sequentially, thus random access to the middle of sequence is extremely inefficient. Second kind of generators are so called explicit, hash functions or low discrepancy sequences. They allow immediate access to any pseudorandom number. The low discrepancy sequences allow better convergence of MC integration, but only for low-dimensional functions. Algorithm generating good quality high dimensional low discrepancy sequence is still an unsolved task.

The presented system needs random access sequence with theoretically unbounded dimensionality. It uses for this purpose a complex generator made from two basic ones. First two dimensions (sampling image plane) comes from 2D low discrepancy Halton sequence, due to its high quality. Further dimensions are generated by three 64-bit congruent RNG steps separated by translation of high order bits to mix with lower ones. The simple function works quite good, but is inferior to 2D Halton sequence. However, Halton sequence efficiency deteriorates when increasing sample dimensionality. Implementations of all interface objects must use the provided stateless random number generator, passing obtained hash data as its parameter.

3.3. Sampling. Sampling interface allows very elegant and concise way of specifying what
could be done with cameras and 3D objects. The sampling methods are general enough to implement majority of ray tracing algorithms. Basic sampling of a function is selecting arguments of function \( y = f(x) \) at random, with probability depending on its shape. Argument of each function is a variable in some space \( \Omega_1 \). Again, \( y = f(x) \) is not necessarily a real number, and then \( f \) transfers \( x \in \Omega_1 \rightarrow y \in \Omega_2 \). In our sampling there can be distinguished three basic operations:

- sampling the argument \( x \) (e.g. direction of scattering);
- evaluation of sampling probability density (e.g. what is the probability of scattering ray in given direction \( \omega \)); the result often is difficult to compute exactly and can be approximated, however crude approximation hurts performance;
- evaluation of \( y = f(x) \) value for given \( x \).

All the sampling routines can be grouped in three different categories listed below.

1. Queries about a point on a surface – in objects these queries randomize emission points, while in cameras they select a point on the lens; the \( y \) value denotes spectrum, which describes spatial emission or sensitivity respectively. In point lights or pinhole cameras \( y \) is a delta distribution coefficient. The probability is measured with respect to the lens or light source area. These query functions are used by ray tracer to start a path.

2. Queries about transmission – such as ‘find the nearest ray intersection with object or lens’ or ‘randomize a point of interaction’ if object contains volumetric effects. The argument \( x \) represents a point in a volume or a point on a surface, while the \( y \) value is spectrum describing attenuation on a ray path. The probability is measured with respect to the distance.

3. Queries about scattering – direction in which ray scatters after collision with object; the argument \( x \) is direction vector while the \( y \) value is BSDF represented as a spectrum. Probability is measured with solid angle for volumetric scattering, projected solid angle for surface scattering or is delta distribution for perfect specular surfaces. These functions are used only with respect to the points acquired by one of two previous methods – either surface sampling or transmission.

Eventually the basic interface contains nine functions altogether. However, to make the implementation more efficient, the query about transmission attenuation is divided into two functions. One tests ray with respect to surfaces only, thus returning a binary value if ray hits a surface or not, while another calculates volumetric attenuation completely ignoring surfaces. The interface contains one more function – routine for freeing temporary memory, since most implementations require storing data between calls which generate data between points and queries about scattering.

3.4. Colors. Correct simulation of colors is very important in physically correct rendering system. The most commonly used RGB model, despite being computationally efficient, is inappropriate in global illumination. The errors caused by using it are most distracting with dispersion and non-white illumination. Figure 2 shows the difference between RGB and our full spectral representation in dispersive refraction on prism.

![Figure 2. Separate dispersion lines in RGB model (left) versus full spectrum (right). For clarity refraction coefficients are exaggerated.](image)

The implemented system uses point sampled spectrum with randomly selected wavelengths. This solution works well with non-deterministic ray tracing, and, what is more important, is strictly correct. The system solves also numerous issues resulting from acquiring spectral data and converting physically based output to final RGB images for standard computer displays.

3.5. Parallelization. The parallel mode combines two basic parallelization techniques. First, it uses shared memory model on multi CPU machines, and second, message passing, implemented by custom RPC library, in distributed environment. The shared memory computations are usually more effective, but lack scalability. On the other hand, the RPC scheme allows connecting large number of machines at the price of communication delays. In Figure 3 there is presented speedup from parallelization during test
rendering of image from Figure 4. The speedup for two processors is nearly ideal, because of shared memory model. Using more processors, in our hardware configuration, requires connecting multiple 2-CPU machines. In the latter case speedup decreases with increasing image resolution because image preview was enabled, i.e. on master machine partially rendered image can be previewed and is updated approx. every five seconds, which cause network traffic and slow-downs. Disabling preview results in almost linear speedup, at least for simple scenes. The preview feature is very important during as interactive as possible composition of rendering scene, and in research towards real-time full global illumination.

![Figure 3. Parallelization efficiency.](image)

### 3.5.1. Shared Memory

On machines with more than one CPU, shared memory model is used. All the data structures, except the frame buffer, are carefully designed to be read only, which saves time needed for synchronization. The actual parallelization is done only for rendering, which is major time consuming process. Currently, parsing scene description, building kd-trees, etc. are performed sequentially. These activities are unlikely to make significant impact on overall image generation time.

We assume that each processor on the same machine has roughly the same power. This assumption allows not bothering with load balancing. The rendering task is subdivided into as equal as possible subtasks, and each of them is performed on single thread. The implementation is very simple – it effectively contains only fork at the beginning and join at the end of rendering. Each thread executes exactly the same code, but renders different range of samples.

### 3.5.2. Message Passing

The parallelization in distributed environment is implemented by master-slave model using a custom LibRPC library. This library is easily portable to Windows and Linux platforms and designed for maximum speed. It can work on machines with different operating systems and mixed 32/64 bit architecture as well, however we assume that they all run processors from the same family. It’s because little-endian processors cannot reliably communicate with big-endian. The library has a simple IDL which is compiled to socket based code with half automatic memory management. Each process creates $2^n$ threads, where $n$ is CPU number on a machine. The main thread gives a command line control, the second thread is RPC server thread, and the remaining $n$ threads perform computations. The parallelization is done by task farm method. The master provides a pool of tasks with size much larger than the number of machines. Each slave asks for a new task whenever it is ready. This provides a simple and effective load balancing. The master cannot become a bottleneck because a task identification and other RPC call data are tiny.

### 3.6. Extensions

The basic system code provides no 3D object implementations, which are necessary to render anything. This basic functionality is provided by extension mechanism. Extension is a simple dynamic library, which provides implementation of at least one of platform interfaces. One extension enables rendering of geometry encoded in ASCII Wavefront *.obj files. The extension provides also our script language for material description, described in details elsewhere. The language is computationally complete and therefore enables creating any desired surface appearance. Moreover, the integrated 2D image library allows usage of compressed textures in JPEG format, with decoding only the requested image parts on the fly. This feature greatly reduces memory consumption at slight loss of speed. This tradeoff is excellent especially in parallel environment, when there is much more computational power, but no more memory.

### 4. Algorithms

Physically correct, general and efficient algorithms are most important part of any global illumination system. Currently only non-deterministic ray tracing algorithms are correct and general enough to render realistic and convincing images. Great deal of effort is spent to make them more efficient. In this paper we present some classic algorithms, with modifications made in order to fit them into our platform design and to parallelize them.

#### 4.1. Path Tracing

The Path Tracing algorithm [4] is first correct solution to the global illumina-
nation with geometric optics in vacuum. The algorithm recursively constructs path starting from the viewer towards light sources. The original variant of this method creates final step of paths by selecting emission points on light sources directly. The improved method [7], allows also random intersection of light sources and combines both results by multiple importance sampling [10]. Unfortunately, both of these methods are not very reliable, and are well suited only to particular scenes. Path tracing fails utterly if scene contains either strong indirect illumination or caustics. Figure 5 presents difficulties when rendering caustics with Path Tracing.

4.2. Bidirectional Path Tracing. This algorithm [10] was invented to cope with inefficiencies of Path Tracing. It constructs paths starting both from viewer and light sources. Resulting subpaths are then connected in the middle of the scene. However, the number of such connections and resulting visibility tests can become large (roughly proportional to square of an average path length). Original algorithm address this issue by skipping some connections using so called efficiency optimized Russian roulette. Unfortunately this technique assumes reading from pixel-based frame buffer, and therefore is unsuitable for our framework. Our optimization instead ensures that only one full path with given length is concocted, which is enough to force number of paths to be linearly dependent on subpath lengths, and make computations effective. The bidirectional algorithm handles caustics much better, but still fails when caustics are viewed indirectly through mirror (Figure 6).

4.3. Photon Mapping. This algorithm [3] is designed to eliminate high frequency noise produced by unbiased Monte Carlo ray tracing. The noise is removed by caching photons and estimating indirect illumination by means of interpolation. Unfortunately, huge numbers of photons are necessary to avoid excessive blur-ring. This makes implementation of full spectral rendering extremely memory demanding. The implementation modifies basic algorithm making photon map structure dynamic, which allows progressive refinement of initial blurry image at the cost of slight performance loss. The Figure 7 presents image with superior quality, but for more complex scenes memory requirements can be prohibitive.

4.4. Frame Buffer. Most commonly used approach for storing partially rendered image uses rectangular grid of predefined resolution. However, storing all samples individually in sequential data stream allows for more sophisticated post-processing algorithms. The simplest techniques, which can greatly increase image quality, are nonlinear filters for noise reduction (modified version of [9]), and glare effects around bright image parts for enhanced realism.

5. Results and Conclusions. All realistic image synthesis algorithms require substantially more power than any contemporary machine can offer. This probably will be true in future, since as computing power grows, people want much more complex images to be created. However, recent development trends in modern hardware avoid creating very fast processors in favor of parallel computations. This trend affects not only graphic processors, but also main ones. Today it cannot be predicted if a GPU implementation of ray tracing (e.g. [8]) will ever be better than CPU one. But at least one thing is certain. Efficiency of sequential ray tracers cannot be substantially improved by hardware progress, and the future belongs to parallel implementations, designed either for CPUs or for GPUs.

The final effectiveness of platform depends on both – used algorithms and parallelization. However, the gains from them are quite different. We have found that parallelization of all algorithms implemented within the platform gives almost linear speedup with shared memory and asymptotically linear (with respect to increasing the image quality requirements) with message passing. In the latter case the sequential parts are: preprocessing (i.e. trees are built while reading scene description independently on each machine) and gathering of the final result. The speed-up from parallelization is almost independent on what scene represents, while the more sophisticated algorithms are designed to handle special, trickier cases. The rendering time reduction due to usage of better algorithms range from nothing to theoretically unbounded value, and is strongly varying on particular scene materials and geometry. In our test cases Bidirectional Path Tracing is sometimes better than ordinary Path Tracing ranging from several to hundreds times.

The most important contributions of our work are new sampling-based framework and optimizations for existing rendering algorithms. The framework is designed for easy implementation of ray tracing based global illumination, arbi-
trary rendered image storage method (not necessarily pixel based) which allows using sophisticated post-processing techniques, and automatic parallelization of Path Tracing and Bidirectional Path Tracing. New optimizations include reducing number of visibility tests in Bidirectional Path Tracing without image pixel dependence and dynamic Photon Map, which enables adding new photons while rendering. The latter technique enables using Photon Mapping not only as two pass technique, but also as algorithm that progressively improves image quality, until user decides that result is satisfying.

Bibliography.