A Lighting Model for Fast Rendering of Forest Ecosystems

Robert Geist* Jay Steele†

Clemson University
Clemson, SC

ABSTRACT

Real-time rendering of large-scale, forest ecosystems remains a challenging problem, in that important global illumination effects, such as leaf transparency and inter-object light scattering, are difficult to capture, given tight timing constraints and models that typically contain hundreds of millions of primitives. This paper proposes a new lighting model, adapted from a model previously used to light convective clouds and other participating media, together with a distribution of ray processing across multiple GPUs, in order to achieve these global illumination effects while maintaining near real-time performance. The lighting model is based on a lattice-Boltzmann method in which reflectance, transmittance, and absorptance parameters are taken from measurements of real plants. The lighting model is solved as a pre-processing step and requires only seconds on a single GPU. The ray tracing engine uses the well-known *short-stack* algorithm, due to Horn, Sugerman, Houston, and Hanrahan. Both the pre-processing step and the ray tracing engine make extensive use of NVIDIA's Compute Unified Device Architecture (CUDA).

Index Terms: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing; I.3.1 [Computer Graphics]: Hardware Architecture—Graphics Processors

1 INTRODUCTION

Real-time rendering of large scale, high-density, plant ecosystems is a topic of growing interest and wide application. There are two standard approaches to this task that continue to receive the attention of the research community. One is image-based and relies on conventional rasterization using billboard clouds [2], and the other is geometry-based and relies on ray tracing [4]. Ray tracing generally gives superior visual results, but until recently it has been too slow to provide high quality images at interactive frame rates for scenes that potentially require billions of polygons. Hardware improvements, in particular, improvements in CPU speed, have only partially ameliorated the problem because reduction in memory latency has not kept pace with the reduction in CPU clock cycle time. Instead, the application of large clusters of computing cores to an inherently parallel problem, together with careful management of the database that comprises the targeted plant ecosystem, has led to the emergence of ray tracing as a competing technique. Rendering performance is closely tied to careful integration of the plant database with the spatial partitioning of the scene into k-d trees that are used for hierarchical ray-surface intersection testing.

The goal of this effort is to extend the ray tracing technique to include important effects that are absent from other treatments, in particular, diffuse leaf transparency and inter-object light scattering, while maintaining at least near real-time rendering for scenes that comprise hundreds of millions of primitives. The fundamental approach is to adapt and apply a lattice-Boltzmann lighting model [7, 8], originally designed for lighting participating media, to large-scale forest ecosystems and then ray trace using CUDA [19] across multiple NVIDIA GPUs. The overall technique is similar, in spirit, to both pre-computed radiance transfer [23] and photon mapping [13], in that a pre-processing step is used to compute and store lighting information within the scene itself. This pre-processing step is, comparatively, very fast.

Subsequent sections will cover background, including related work and the basic illumination model [7], modifications to the model required to capture leaf transmittance and reflectance, the structure of the CUDA-based ray tracer, example results, including images and performance timings, and conclusions.

*e-mail: rmg@cs.clemson.edu
†e-mail: jesteel@cs.clemson.edu

2 BACKGROUND

2.1 Related Work

Interactive ray tracing on GPUs has drawn significant interest at least since the work of Purcell et al [21], who used a uniform grid as an acceleration structure. Foley et al [5] improved upon this by using a k-d tree acceleration structure and designing an important, stackless tree traversal algorithm called *kd-restart*. This early work was significantly hampered by tight GPU instruction limits. Horn et al [11] took advantage of improved hardware, in the form of an ATI X1900 XTX, and introduced a new algorithm, *short-stack*, to achieve better results. Short-stack is used herein and shown in pseudo-code in Figure 1. Short-stack uses a stack of bounded size during traversal and falls back to the stackless algorithm on underflow. Horn et al [11] used 4-wide ray packets per fragment, and they were able to achieve 15 - 18 million primary rays per second on test scenes.

Dietrich et al [4] combined the OpenRT real-time ray tracing engine [25], Xfrog plant models [20], geometry instancing, and adaptive transparency control to achieve interactive rendering of large, high-density plant ecosystems. The adaptive transparency control was in response to the structure of the Xfrog models, wherein leaves are represented by coarse triangles within which leaf shape is determined by an alpha channel. Ray-triangle intersections may then simply generate an additional forward ray, rather than a reflected value. Their test scene contained more than 365,000 plants, of 68 distinct species, with a total of approximately 1.5 billion triangles. With 32 CPUs they achieved 6 fps on a 640×480 scene. They did not attempt to account for global illumination effects such as leaf transparency and inter-object light scattering.

Wang et al [26] achieved beautiful results in rendering small collections of plant leaves using carefully constructed bidirectional reflectance and transmittance functions that were based on measurements of real plants. Their method is computationally intensive, with large memory requirements, and so as yet unsuitable for real-time rendering of large-scale, high-density ecosystems.

Luft et al [16] were able to capture ambient occlusion in rendering foliage through the addition of a pre-processing step in which overall tree geometry was simplified to an implicit surface, i.e., a density field, using Marching Cubes [15]. The ambient coefficient in a standard, local illumination model was then modified by a transfer function that was exponentially decreasing in the field density. They also realigned leaf normal vectors to match the implicit surface in order to reduce lighting noise.

```
t_min = t_max = scene_min
t_hit = ∞
while(t_max < scene_max):
    if stack.empty:
        node = root
        t_min = t_max
        t_max = scene_max
        pushdown = TRUE
    else
        (node,t_min,t_max) = stack.pop
        pushdown = FALSE
    while(not node.is_leaf):
        a = node.axis
        t_split = (node.value - ray.origin[a])/(ray.dir[a])
        (one,two)=order(ray.dir[a],node.left,node.right)
        if(t_split≥t_max)
            node=one
        else if(t_split≤t_min or t_split<0)
            node=two
        else
            stack.push(two,t_split,t_max)
            node=one
            t_max=t_split
            pushdown = FALSE
        if pushdown
            root = node
    for triangle in node.triangles:
        t_hit = min(t_hit,intersect(ray,triangle))
    if t_hit < t_max
        return t_hit
return t_hit
```

Figure 1: Short-stack algorithm for k-d tree ray tracing

2.2 Illumination Model

The technique of [7] provided a new solution to the standard volume radiative transfer equation for modeling light in a participating medium:

$$(\vec{\omega}\cdot\nabla+\sigma_t)\,L(\vec{x},\vec{\omega}) = \sigma_s\int p(\vec{\omega},\vec{\omega}')L(\vec{x},\vec{\omega}')d\vec{\omega}' + Q(\vec{x},\vec{\omega}) \quad (1)$$

where L denotes radiance, $\vec{\omega}$ is spherical direction, $p(\vec{\omega},\vec{\omega}')$ is the phase function, σ_s is the scattering coefficient of the medium, σ_a is the absorption coefficient of the medium, $\sigma_t = \sigma_s + \sigma_a$, and $Q(\vec{x},\vec{\omega})$ is the emissive field in the volume [1]. The solution technique proposed was based on a lattice-Boltzmann (LB) method. Lattice-Boltzmann methods are computational alternatives to finite-element methods for solving coupled systems of partial differential equations. The LB methods have recently provided significant successes in modeling fluid flows and associated transport phenomena [9]. The methods simulate transport by tracing the evolution of a single particle distribution through synchronous updates on a discrete grid. A complication of LB methods in three dimensions is that isotropic flow requires that all neighboring lattice points of any site be equidistant. A standard approach, due to d'Humières, Lallemand, and Frisch [3], is to use 24 points equidistant from the origin in 4D space and project onto 3D. The points are:

$$\begin{array}{lll}(\pm1,0,0,\pm1) & (0,\pm1,\pm1,0) & (0,\pm1,0,\pm1)\\ (\pm1,0,\pm1,0) & (0,0,\pm1,\pm1) & (\pm1,\pm1,0,0)\end{array}$$

and projection is truncation of the fourth component, which yields 18 directions. Axial directions then receive double weights. Representation of several phenomena, including energy absorption and energy transmission, is facilitated by adding a direction from each lattice point back to itself, which thus yields 19 directions, the non-corner lattice points of a cube of unit radius. The key quantity of interest is the per-site photon density, $f_m(\vec{r},t)$, which is the density arriving at lattice site $\vec{r}\in\Re^3$ at time t in cube direction $\vec{c}_m$, $m\in\{0,1,...,18\}$.

The entire LB system simulation then amounts to a nearly trivial, synchronous update of the lattice. If the lattice spacing is λ and the time step is τ, then the update is:

$$f_m(\vec{r}+\lambda\vec{c}_m,t+\tau) - f_m(\vec{r},t) = \Omega_m\cdot f(\vec{r},t) \quad (2)$$

where Ω_m denotes row m of a 19×19 matrix, Ω, that describes scattering, absorption, and (potentially) wavelength shift at each site. If $\rho(\vec{r},t)=\sum_m f_m(\vec{r},t)$ denotes total site density, then a derivation in [7] shows that the limiting case of (2) as $\lambda,\tau\to0$ is the diffusion equation

$$\frac{\partial\rho}{\partial t} = D\nabla^2_{\vec{r}}\rho \quad (3)$$

where the diffusion coefficient

$$D = \left(\frac{\lambda^2}{\tau}\right)\left[\frac{(2/\sigma_t)-1}{4(1+\sigma_a)}\right] \quad (4)$$

This is consistent with previous approaches to modeling multiple photon scattering events [12, 24], which invariably lead to diffusion processes.

For any LB method, the choice of Ω is not unique. Standard constraints are conservation of mass, $\sum_m(\Omega_m\cdot f)=0$, and conservation of momentum, $\sum_m(\Omega_m\cdot f)\vec{v_m}=\tau\vec{F}$, where $\vec{v_m}=(\lambda/\tau)\vec{c_m}$ and $\vec{F}$ represents any site external force. In [7, 8], for the case of isotropic scattering, Ω was chosen as follows:
For row 0:

$$\Omega_{0j} = \begin{cases} -1 & j=0\\ \sigma_a & j>0\end{cases} \quad (5)$$

For the axial rows, $i=1,...,6$:

$$\Omega_{ij} = \begin{cases} 1/12 & j=0\\ \sigma_s/12 & j>0,\quad j\neq i\\ -\sigma_t+\sigma_s/12, & j=i\end{cases} \quad (6)$$

For the non-axial rows, $i=7,...,18$:

$$\Omega_{ij} = \begin{cases} 1/24 & j=0\\ \sigma_s/24 & j>0,\quad j\neq i\\ -\sigma_t+\sigma_s/24, & j=i\end{cases} \quad (7)$$

Entry i,j controls scattering from direction $\vec{c}_j$ into direction $\vec{c}_i$, and directional density f_0 hold the absorption/emission component. On update, i.e., $\Omega\cdot f$, fraction σ_a from each directional density will be moved into f_0. The entries of Ω were then multiplied by the density of the medium at each lattice sight, so that a zero density yielded a pass-through in (2), and a density of 1 yielded a full scattering.

With modifications, this lighting model can be used to capture leaf transparency and inter-object light scattering for forest ecosystems.

3 Capturing Leaf Transparency and Scattering

The overall illumination model begins with a standard, local illumination model that captures direct lighting effects in the usual way. Local diffuse lighting is of the form $k_d(N\cdot L)$, where L is sun direction, N is leaf/wood normal, and k_d is the combined sun color and sampled texture color. The sun is modeled as a black body radiator at 6500K. Local specular lighting is of the form $k_s(V\cdot R)^s$, where V is the viewer position vector, R is the sun reflection vector, s controls highlight dissipation, and k_s is the combined sun color and leaf/wood specular color. Indirect global illumination is captured

from an LB lighting grid then modulated by the texture color and added to the local, direct illumination.

A two-level, hierarchical LB lighting grid can be imposed upon an instanced forest system. Solution of the higher-level grid over the entire forest, using iterations of (2), simply provides a per-node scale factor for lighting intensity that determines the initial boundary conditions for each lower-level, instanced per-plant/tree grid. Each node in the higher-level grid has a density factor estimated from the plant/tree count within the associated cell. Each lower-level, plant/tree grid, here of size 128^3 nodes, has a per-node density (biomass) factor estimated from local leaf count and leaf area within the associated cell. If a significant portion of the biomass is wood, rather than leaf, the density is classified as "brown" rather than "green", so that scattering may be restricted to backward only.

Unlike the case of lighting atmospheric clouds, where absorption is extremely small ($\sigma_a < 0.01$), plants absorb a significant fraction of the visible light reaching them, and this energy is not re-radiated within the visible spectrum. Further, absorption, reflection, and transmission are heavily wavelength-dependent. It is natural to conjecture that these components are also heavily species-dependent, but surprisingly, this is not the case. Knapp and Carter [14] measured leaf optical properties, in particular, reflectance, transmittance, and absorptance of 26 species of plants from widely varying habitats. They concluded that the lack of variability across species was remarkable, given the broad habitat range and unusual anatomical characteristics of several of the species included in the study. Thus a single set of wavelength-dependent model parameters should suffice in determining σ_s and σ_a.

Scattering is, of course, anisotropic and wavelength-dependent. Anisotropic scattering is incorporated by multiplying σ_s that appears in entry $\Omega_{i,j}$ by a normalized phase function:

$$pn_{i,j}(g) = \frac{p_{i,j}(g)}{\left(\sum_{i=1}^{6} 2p_{i,j}(g) + \sum_{i=7}^{18} p_{i,j}(g)\right)/24} \tag{8}$$

where $p_{i,j}(g)$ is a discrete version of the Henyey-Greenstein phase function [10],

$$p_{i,j}(g) = \frac{1-g^2}{(1-2g\vec{n}_i \cdot \vec{n}_j + g^2)^{3/2}} \tag{9}$$

Here $\vec{n}_i$ is the normalized direction, $\vec{c}_i$. Parameter $g \in [-1,1]$ controls scattering direction. Value $g > 0$ provides forward scattering, $g < 0$ provides backward scattering, and $g = 0$ yields isotropic. Mie scattering [6] is generally considered preferable, but the significant approximations induced here by a relatively coarse grid render the additional complexity unwarranted. Note that (8) differs from the treatment in [7], in that setting $\sigma_a = 0$ and $g = 1$ now yields an effect that is identical to a pass-through.

Wavelength-dependence is limited here to three color components. The model does not attempt to account for total leaf absorption as expressed in [14], since this represents almost all incident light energy. (The minimum is 72%, which occurs at 550 nm.) Instead, the absorptance values from [14] are scaled by a single, experimentally determined factor (here 0.125) to yield per-component model absorption coefficients, σ_a^X, for X = R, G, B. The per-component model scattering coefficients are then given by $\sigma_s^X = 1 - \sigma_a^X$, again for X = R, G, B. Per-component transmittance and reflectance ratios from [14] are used to determine forward and backward scattering components, fs^X and bs^X, by the constraint $fs^X + bs^X = \sigma_S^X$. Finally, values of the phase function parameter, g, are chosen as:

$$g^X = \frac{fs^X - bs^X}{fs^X + bs^X} \quad \text{for } X = R, G, B \tag{10}$$

Thus identical transmittance and reflectance values for color component X would yield $fs^X = bs^X$, and scattering would be isotropic ($g^X = 0$). If a node is classified as "brown," rather than "green," $g^X = -1$ for $X = R, G, B$.

Boundary conditions for the grid (initial values for directional densities) are determined so that directions having positive dot products with the sun direction receive photon densities that are proportional to these dot products. This differs from [7], where a minimum orthogonal collection of direction vectors with maximal dot products was selected. The present treatment offers smooth transitions with sun position change.

The pre-processing step then amounts to iterating (2) to steady-state, which yields a total photon density per component at each grid node. The significant loss of light energy from the visible spectrum is modeled by zeroing out component f_0 at each "green" node on each iteration. During ray tracing, at each leaf intersection point, an indirect illumination value is interpolated from the surrounding grid values and added to the local, direct illumination at that point.

4 Implementation

4.1 CUDA

NVIDIA's Compute Unified Device Architecture (CUDA) combines specific hardware, such as the G80-based Quadro FX 5600, used herein, together with drivers, libraries, and C language extensions, in order to provide access to the GPU hardware without the constraints imposed by traditional GPU programming techniques, which typically rely on a graphics API. Code is organized around *kernels*, which are functions that are invoked from the CPU (the host) but execute on the GPU (the device). The multi-processors of the GPU execute these kernels most efficiently in SIMD mode, but standard C control flow is available. In addition to invoking the device kernels, the host is responsible for managing device memory, which is segmented into multiple memory spaces of varying capabilities. Management of the memory hierarchy strongly impacts performance.

Kernels are invoked simultaneously on many threads. Internally, CUDA schedules the execution of threads to maximize performance. Threads are organized into *blocks*. Blocks are further organized as two-dimensional *grids*. Threads in a block share certain device resources, such as the small, fast, on-chip shared memory, and can synchronize with one another. Inside a thread block, threads are further arranged into groups of size 32 called *warps*. A warp can be considered the minimum collection of threads for SIMD execution, in that control flow divergence within a warp extracts a significant performance penalty.

4.2 Indirect illumination

Before the indirect illumination is computed, as detailed above, each tree model is converted to a volume density by intersecting the model's triangles with a 128^3 grid. A simple heuristic is implemented to avoid computing area coverage of all triangles intersecting each grid node. Each grid node is refined into a $3 \times 3 \times 3$ subgrid. If any triangle intersects a node of the subgrid, then that subgrid node is marked as intersected. The density of the grid node is then the ratio of the number of its intersected subgrid nodes to 27.

The indirect illumination computation is implemented as a CUDA kernel. This computation maps well to CUDA, as each iteration of (2) requires computing new values for each grid node that are independent of the new values of every other grid node. One device thread is invoked per grid node. The indirect illumination due to each wavelength (color component) is computed separately, due to memory constraints.

4.3 Ray tracing

The structure of the ray tracing engine follows the general techniques described in [4]. Tree models are instanced multiple times to

compose larger scenes. For each scene, a high level kd-tree is constructed from each instance's world space, axis aligned bounding box (AABB). Lower level kd-trees are constructed from the triangles of each model. All kd-trees are constructed using the surface area heuristic (SAH) introduced by MacDonald and Booth [17]. Each instance in the scene stores its world-space transformation and a reference to its model's kd-tree.

All ray tracing computations (ray/triangle intersections and shading computations) are implemented with CUDA. Before invoking the ray tracing kernels, all relevant scene data is transferred from the host to the device, with care taken to store data in the appropriate memory space. Maximizing read performance from a device's global memory space, which is not cached, requires that threads in a warp follow certain memory access patterns. Unfortunately, threads in a warp quickly diverge when traversing a scene's multiple kd-trees. Because of this divergence, it becomes increasingly difficult to maintain the best access patterns for global memory space. Instead, scene data that is accessed during kd-tree traversal is stored in one of two memory spaces, texture memory space or constant memory space, which are cached and, therefore, perform better with the memory access patterns inherent to kd-tree traversal. Both memory spaces come with limits. Constant memory space is relatively small (64 KB). Textures must be declared global in scope, and arrays of textures are not supported. Thus, all kd-trees are stored consecutively in one texture, with offsets stored elsewhere for proper indexing. This technique is used for all similar data stored in the texture memory space.

Ray tracing is performed by executing multiple kernels on the device. One device thread is created per ray. Since all threads within a block share resources, the resource constraints of traversing a two-level kd-tree hierarchy restrict the primary ray kernel and shadow ray kernel to 64 threads per block. For both kernels, care is taken to prevent, to the extent possible, threads within the same warp from traversing different paths in the kd-tree hierarchy. This is accomplished by arranging each thread block to trace rays in an 8×8 tile. Within this tile, each warp in a thread block traces rays in an 8×4 tile. A variety of block sizes and tile sizes were tested, and this layout was found to provide the best performance for both the primary ray kernel and the shadow ray kernel.

As previously mentioned, kd-tree traversal is accomplished by the short-stack algorithm. The stack is implemented using the fast, on-chip, shared memory, and the stack size is 5. The ray/triangle intersection algorithm is the fast, minimal storage technique described by Möller and Trumbore [18]. Since leaf shape is determined by the alpha channel of a leaf texture, each primary ray may result in multiple secondary rays being generated. Performance-enhancing adaptive transparency control, suggested in [4], was tested, but it has not been incorporated here, since it was found to noticeably degrade image quality.

Shading commences after invoking a primary ray kernel and shadow ray kernel. Both kernels follow the same traversal strategy. Ray/scene intersections are determined by first traversing the high level, scene kd-tree. During this traversal, if any instance's AABB is intersected, the corresponding model's low level kd-tree is then traversed. After invoking the above kernels, a shading kernel is then executed. If an intersection point is not shadowed, the local illumination is computed as the combination of diffuse and specular lighting. To produce the final color for a ray, this local illumination at the intersection point, which is derived from sun direction, leaf/wood normal, viewer position, sun color, and texture color, is combined with the pre-computed indirect illumination, whose value at the intersection point is interpolated from the values at the surrounding nodes of the lighting grid. Though 3D in its implicit structure, each model's indirect illumination grid is stored in a 1D texture on the device. This is necessary since the production release of CUDA does not yet support 3D texture access. Interpolation is performed according to the 3D texture interpolation algorithm found in [19]. Each kernel stores its results to an array in the global memory space of the device.

A final kernel handles tone mapping and, if needed, down-samples the results of the previous kernels to produce the final image. The tone mapping algorithm implemented is due to Reinhard et al [22]. Down-sampling occurs if more than one ray is generated for each pixel in the final image.

OpenMPI enables rendering across a GPU-based cluster. Each cluster node, which is responsible for rendering a portion of the final image, contains one NVIDIA Quadro FX 5600 GPU. Each node is sent a portion of the image to render. No effort was made to achieve load balancing. The workload is simply split along the vertical axis of the target image. Results are gathered by the root node and displayed to the user.

5 RESULTS

Though subtle at distances, the effects of forward scattering in the proposed illumination model are fairly dramatic at close range. Figure 2 shows a nearby view of a Southern Catalpa tree [20] rendered with the proposed technique 2(a) and (for comparison) with only local illumination 2(b) and with local illumination augmented by a constant ambient 2(c), where the ambient value is chosen to match average image luminance with that of 2(a). The figure also shows a volume visualization of the indirect illumination from the LB scattering grid used in this rendering. Thus image 2(a) can be regarded as the sum of image 2(b) and a texture-modulated image 2(d).

Figure 3(a) shows an example rendering of a high-density, forest ecosystem, constructed by taking a landscape image from [8], replacing all of the real trees with synthetic ones, and then lighting and rendering using the proposed technique. Note that the synthetic clouds were lighted with the original LB technique, described in section 2.2. For this scene, a higher-level LB grid was deemed unnecessary, since all trees have approximately equivalent exposure. For comparison, the same scene is shown again in Figure 3(b), except that every beech tree has been replaced with a pine tree. Specific lighting model parameters are shown in Table 1, and scene composition for Figure 3(a) is shown in Table 2.

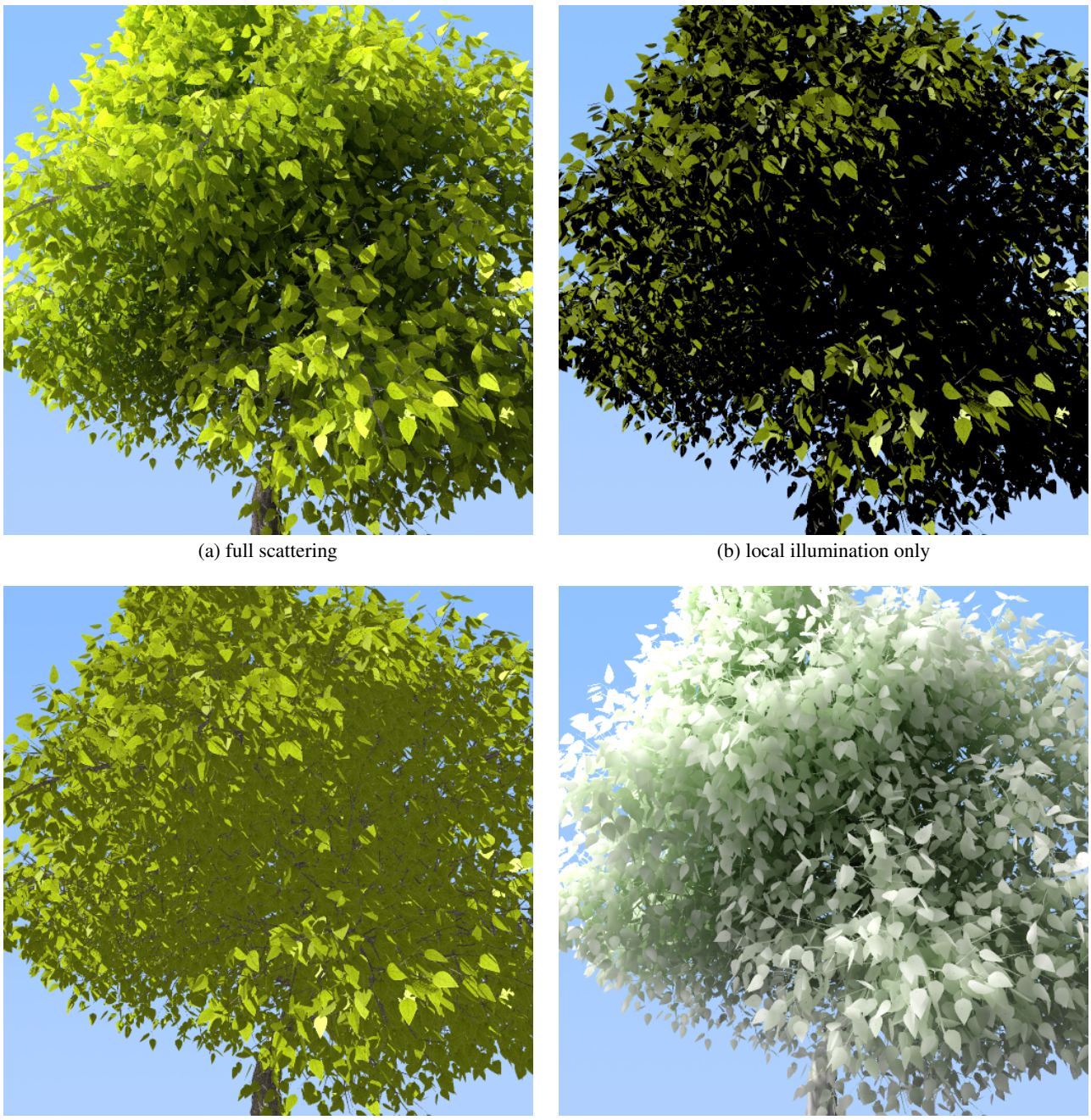
Table 1: Lighting model parameter values

$\sigma_a = (\sigma_a^R, \sigma_a^G, \sigma_a^B)$	(0.109, 0.091, 0.118)
$\sigma_s = (\sigma_s^R, \sigma_s^G, \sigma_s^B)$	(0.891, 0.909, 0.882)
$fs = (fs^R, fs^G, fs^B)$	(0.055, 0.150, 0.020)
$bs = (bs^R, bs^G, bs^B)$	(0.070, 0.125, 0.040)
$g = (g^R, g^G, g^B)$	(-0.120, 0.091, -0.333)

Table 2: Composition of beech tree scene

species	instances	triangles/instance
red maple	12	115,529
ohio buckeye	285	168,520
paper birch	291	372,896
southern catalpa	206	155,342
american beech	168	496,719
total scene	962	273,376,528

Execution times for ray tracing the scene of Figure 3(a) at resolution 896×448 pixels with 1 ray per pixel and 4 rays per pixel are shown in Table 3 for both a single G80 and an early engineering



(a) full scattering

(b) local illumination only

(c) local illumination plus ambient to match luminance

(d) volume visualization of LB solution

Figure 2: Rendering comparison: nearby view of Southern Catalpa tree.

Table 3: Execution times for ray tracing and LB lighting

platform	1 ray/pixel	4 rays/pixel	LB lighting
G80	2.277 s	8.044 s	32.1 s
G200 EES	1.151 s	-	15.9 s

Table 4: Execution times for ray tracing across multiple G80 GPUs

GPU count	execution time (1 ray/pixel)
2	1.162 s
4	0.666 s
8	0.351 s
16	0.170 s

sample of an NVIDIA GeForce GTX 280 (G200) with 240 cores clocked at 1.08 GHz. Also shown there are the LB lighting model pre-processing times. Each of these is the average of the times for five solutions, one per species, on a 128^3 grid. For such models, the number of iterations required to achieve convergence to steady-state is approximately twice the longest edge dimension. Table 4 compares the performance obtained when ray tracing this same scene at 1 ray/pixel across multiple G80 GPUs, all Quadro FX 5600s.

6 Summary and Conclusions

Real-time rendering of large-scale, forest ecosystems remains a challenging problem when global illumination effects, such as leaf transparency and inter-object reflection, which are important to visual realism, must be incorporated. One approach to achieving such effects, suggested herein, is through the use of a lattice-Boltzmann lighting model to approximate the indirect illumination. The LB model can be executed as a pre-processing step to generate lighting values that are stored on a three-dimensional scene grid. The ray tracing engine can then combine local, direct illumination at any ray/object intersection point with an indirect value obtained by interpolating values from the embedded LB grid. Near real-time performance is obtained by mapping the ray tracing engine, as well as the LB lighting model, to NVIDIA's Compute Unified Device Architecture and then distributing across multiple GPUs.

As seen in Table 4, 16 GPUs delivered a rate of 6 fps for a resolution 896×448 pixels on a scene containing 273M triangles. Further, it is well-known that GPU development has, for many years, defied Moore's Law in delivering a sustained, doubling of performance every 6 months. The data of Table 3 suggest this trend will continue. This data, together with the observed linear improvement shown in Table 4, lead to a natural conjecture that, for scenes of comparable resolution and complexity, 32 of the 1.08GHz G200 GPUs could now deliver the 24 fps that is generally considered real-time performance.

The lattice-Boltzmann lighting model uses parameters derived from measurements of real plants to approximate global illumination. It solves a diffusion-like process for light scattering and absorption. Although not carried out as part of this study, distributing the LB model across multiple GPUs is reasonably straightforward. Boundary nodes in subgrids are replicated, and only these replicated boundary nodes need communicate with one another across GPUs.

There are several potential drawbacks to the proposed technique that yet need to be addressed. First, the LB model execution time shown is per plant instance. Thus the total pre-processing time for the scene of Figure 3(a), which contains 5 species, was 5 times that shown. If scenes with more than a hundred species (species not instances) are of interest, pre-processing time could approach an hour or more. Potential counter-measures include distributing across multiple GPUs, as noted, and reducing per-plant LB grid size. Note that reducing the grid edge dimension from 128 to 64 improves execution time by a factor of 16, 8 from the dimension reduction and 2 from the convergence time reduction. The extent to which such reduction would impact image quality remains to be tested. A second drawback is the memory constraint. All model data must be resident on each GPU. Although 5 plant instances do not consume the 1.5GB available on the Quadro FX 5600, hundreds of species could not be supported.

Finally, although adaptive transparency control was not used here due to its impact on final image quality, the performance enhancement available from this technique is substantial. It is likely that the ill-effects on image quality could be ameliorated by tying the adaption level to both scene view distance and gaze duration of the observer.

Acknowledgments

This work was supported in part by a Fellowship grant from NVIDIA Corp. and by the U.S. National Science Foundation under Award 0722313.

References

[1] J. Arvo. Transfer equations in global illumination. In *Global Illumination, SIGGRAPH '93 Course Notes*, August 1993.

[2] S. Behrendt, C. Colditz, O. Franzke, J. Kopf, and O. Deussen. Realistic real-time rendering of landscapes using billboard clouds. *Computer Graphics Forum*, 24(3):507–516, 2005.

[3] D. d'Humières, P. Lallemand, and U. Frisch. Lattice gas models for 3d hydrodynamics. *Europhysics Letters*, 2:291–297, 1986.

[4] A. Dietrich, C. Colditz, O. Deussen, and P. Slusallek. Realistic and Interactive Visualization of High-Density Plant Ecosystems . In *Eurographics Workshop on Natural Phenomena*, pages 73–81, Dublin, Ireland, 2005.

[5] T. Foley and J. Sugerman. Kd-tree acceleration structures for a gpu raytracer. In *HWWS '05: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 15–22, 2005.

[6] J. R. Frisvad, N. J. Christensen, and H. W. Jensen. Computing the scattering properties of participating media using lorenz-mie theory. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, pages 60–1 – 60–10, 2007.

[7] R. Geist, K. Rasche, J. Westall, and R. Schalkoff. Lattice-boltzmann lighting. In *Rendering Techniques 2004 (Proc. Eurographics Symposium on Rendering)*, pages 355 – 362,423, Norrkőping, Sweden, June 2004.

[8] R. Geist, J. Steele, and J. Westall. Convective clouds. In *Natural Phenomena 2007 (Proc. of the Eurographics Workshop on Natural Phenomena)*, pages 23 – 30, 83, and back cover, Prague, Czech Republic, September 2007.

[9] X. He, S. Chen, and G. Doolen. A novel thermal model for the lattice boltzmann method in incompressible limit. *Journal of Computational Physics*, 146:282–300, 1998.

[10] G. Henyey and J. Greenstein. Diffuse radiation in the galaxy. *Astrophysical Journal*, 88:70–73, 1940.

[11] D. R. Horn, J. Sugerman, M. Houston, and P. Hanrahan. Interactive k-d tree gpu raytracing. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 167–174, New York, NY, USA, 2007. ACM.

[12] H. Jensen, S. Marschner, M. Levoy, and P. Hanrahan. A practical model for subsurface light transport. In *Proceedings of SIGGRAPH 2001*, pages 511–518, August 2001.

[13] H. W. Jensen. *Realistic Image Synthesis Using Photon Mapping*. A.K. Peters, Natick, MA, 2001.

[14] A. Knapp and G. Carter. Variability in leaf optical properties among 26 species from a broad range of habitats. *American Journal of Botany*, 85(7):940–946, 1998.

(a) Test scene rendering

(b) Rendering with pine trees rather than beech trees

Figure 3: Sample renderings

[15] W. Lorensen and H. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proc. SIGGRAPH '87*, pages 163–169, 1987.

[16] T. Luft, M. Balzer, and O. Deussen. Expressive illumination of foliage based on implicit surfaces. In *Natural Phenomena 2007 (Proc. of the Eurographics Workshop on Natural Phenomena)*, pages 71 – 78, Prague, Czech Republic, September 2007.

[17] J. D. MacDonald and K. S. Booth. Heuristics for ray tracing using space subdivision. *The Visual Computer*, 6(3):153–166, 1990.

[18] T. Mller and B. Trumbore. Fast, minimum storage ray-triangle intersection. *journal of graphics tools*, 2(1):21–28, 1997.

[19] NVIDIA Corp. Nvidia cuda programming guide, version beta 2.0. www.nvidia.com/object/cuda_get.html#cuda2.0beta, April 2008.

[20] G. Organic-Software. Xfrogplants v 2.0. http://www.xfrogdownloads .com/greenwebNew/products/productStart.htm, 2008.

[21] T. J. Purcell, I. Buck, W. R. Mark, and P. Hanrahan. Ray tracing on programmable graphics hardware. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, pages 268–277, 2005.

[22] E. Reinhard, M. Stark, P. Shirley, and J. Ferwerda. Photographic tone reproduction for digital images. *ACM Trans. Graph.*, 21(3), 2002.

[23] P.-P. Sloan, J. Kautz, and J. Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 527–536, 2002.

[24] J. Stam. Multiple scattering as a diffusion process. In *Proc.* 6^{th} *Eurographics Workshop on Rendering*, pages 51–58, Dublin, Ireland, June 1995.

[25] I. Wald. Realtime Ray Tracing and Interactive Global Illumination. *PhD thesis, Saarland University*, 2004.

[26] L. Wang, W. Wang, J. Dorsey, X. Yang, B. Guo, and H.-Y. Shum. Real-time rendering of plant leaves. *ACM Trans. Graph.*, 24(3):712–719, 2005.