A Parallel Architecture for Interactive Rendering of Scattering and Refraction Effects

Daniele Bernabei †,   Ajit Hakke patil ‡   Francesco Banterle †   Marco Di Benedetto †

Fabio Ganovelli †   Sumanta Pattanail ‡   Roberto Scopigno †

Abstract—We present a new algorithm for the interactive rendering of complex lighting effects inside heterogeneous materials. Our approach combines accurate tracing of the light rays in heterogeneous refractive medium to compute high frequency phenomena, with a lattice-Boltzmann method to account for low-frequency multiple scattering effects. The presented technique is designed for parallel execution of these two algorithms on modern graphics hardware. In our solution, light marches from the sources into the medium, taking into account refraction, scattering and absorption. During the marching of the light rays inside the volumetric representation of the scene, irradiance is accumulated and it is diffused with the lattice-Boltzmann method to produce multiple scattering effects.

Index Terms—

I. Introduction

Photorealistic rendering of a virtual scene is a need in many application domains such as movie production, video games and virtual reality. In its most complete formulation, the problem is how to accurately simulate the behavior of light, knowing the optical characteristics of all the materials in the virtual scene.

The transportation of light through space is influenced by many factors. When light encounters the surface of an object, it may deviate from its path according to the change of refraction index, it may scatter as a result of collision of photons with particles of the material, it may be absorbed or emitted by the material. The problem is that although the laws governing light-matter interaction are known, their application in a simulated environment is a computationally demanding task, which always requires either simplifications of the laws themselves or assumptions on the nature of the materials, or both. Analyzing the vast literature we can find a large number of techniques for real-time simulation of light, each one modeling a subset of light effects. We propose a technique that exploits the high level of parallelism of modern GPUs to provide real-time rendering of scene considering reflection, refraction absorbtion and scattering phenomena. We use a voxelization of the scene, where each voxel stores refractive and absorbtion index. For each frame, the algorithm traces rays from the light sources through the scene deviating their path on the basis of the refractive index and depositing irradiance in the traversed voxels. While rays are still traversing the volume, a diffusion process starts spreading the irradiance deposited by the rays to neighboring voxels, simulating multiple scattering. When both the ray traversal and the diffusion process are

† Visual Computing Lab, CNR, Pisa (Italy)
‡ Graphics Lab, UCF, Orlando (USA)

complete, view rays are shot, traversing the scene and reading back the irradiance. For scenes where we have a tessellated boundary of the objects, this is used to compute the reflections on the surface and to avoid the jagged look of voxelized scenes.

Summarizing, this paper brings a twofold contribution to the state of the art:

- a real-time light transport computation engine that takes into account refraction, scattering and absorption in heterogeneous media;
- a novel parallel lighting computation framework specifically designed to harness the modern GPU architecture and to scale well for large scenes.

The rest of this paper proceeds as follows: we review the state of the art related to real-time volumetric light transport computation in Section II, explain our approach in detail in Section III and provide the results in Section IV.

II. Related Work

The interaction of the light with general media is modeled by the volume rendering equation [1]:

$$(\omega \cdot \nabla)L(x,\omega) + \sigma_t(x)L(x,\omega) = \varepsilon(x,\omega) +$$
$$\sigma_s(x)\int_{4\pi} p(x,\omega,\omega')L(x,\omega')d\omega' \quad (1)$$

where $L(x,\omega)$ is the radiance at point x and direction ω, ε is the emitted radiance, p is the phase function, σ_s and σ_t are respectively is the scattering and extinction coefficients. The volume rendering equation captures all lighting effects generated by the uncountable number of local interactions between the light and particle in the volume. Note that $\sigma_t = \sigma_s + \sigma_a$ where σ_s is the absorption coefficient. A lot of effort has been devoted to efficiently solving this equation (see Cerezo et al.'s survey [2] for an overview). Despite number of efforts, to our knowledge no solution strategy exists to solve this equation for fully dynamic scenes and heterogeneous materials (varying absorption, scattering, and index of refraction coefficients) at real-time rates.

We can classify the existing algorithms in three categories:

- Ray-based: The approaches that cast rays through the scene from the light sources and/or from the point of view and simulate their interaction with the matter. These approaches include classical ray tracing, path tracing, photon mapping etc.

- Space Partition-based: The approaches that partition the scene in finite elements and compute the energy exchange between them depending on light sources. These approaches include radiosity, Lattice Boltzmann Lighting (LBL) etc.
- Rasterization-based: The approaches using preeminently fixed the rasterization-based pipeline such as shadow mapping, screen space ambient occlusion, translucent shadow maps etc.

This classification is operated on the basis of what the processing unit (let it be the CPU or the GPU) is dedicated to: tracing rays, solving systems or rasterizing.

Ray-based Methods: these methods leverage on a ray-tracing core [3] and typically use Monte Carlo techniques to evaluate Equation 1 [4]. Some of the methods under this category are: pathtracing [1], bidirectional pathtracing [5], and Metropolis light transport [6], and photon mapping [7]. Amongst these, photon mapping is a very efficient method for fully heterogeneous materials (varying absorption, scattering, and index of refraction coefficients), for example see Guiterrez's work [8]. However, photon mapping requires to shoot many photons to achieve high quality results, and only recently modern GPUs allow interactive photon mapping for non-volumetric global illumination [3]. Recently, real-time methods for simulating complex refraction (varying in the volume) based on rays have become popular. This is especially due to modern GPUs, that can trace large number of rays simultaneously, which allows interactive/real-time performances. These methods are extremely relevant to our work, because they can be independently plugged on top of other integrators for other computations (e.g. multiple scattering). For example, Ihrke et al. [9] introduced a particle-based method derived from the Eikonal equation that performs at interactive rate. Sun et al. [10] instead presented a real-time renderer which constantly bends light rays at each voxel intersection in a fashion similar to volumetric photon mapping. More recently, this method was extended in order to handle a closed form analytical formulation of light ray trajectory on constant gradient refractive media [11]. Finally, Walter et al. [12] showed a real-time formulation to search for focal points on the boundary of an object, but they neglected internal refractions.

Space Partition-based Methods: these methods subdivide the space of the scene into sub-regions (e.g. voxels, tetrahedra, clusters, etc...) where the computations are performed per sub-region. This typically allows the computation of multiple scattering for homogeneous and/or heterogeneous materials efficiently. Moreover, these methods are straightforward to parallelize, thus they can be easily ported to run on GPUs. Even a classic finite element method (FEM), such as Radiosity, was applied to compute at interactive/real-time rates the dipole approximation with success [13]. However, general solutions to volume rendering equation are computationally intensive for accurate multiple scattering inside dense heterogeneous solids. These cases can be efficiently solved using the Discrete Ordinates Method [14] (DOM), coming from thermal engineering, where radiation transfer between small voxels is limited to a predefined set of discrete directions. This method has been re-visited and expanded over the years [15], [16], and it has benefited from the use of Spherical Harmonics [17]. Nevertheless, these methods suffer from the ray-effect, a banding artifact due to the discretization of directions. Recently, Fattal [18] introduced the *light propagation maps* which highly reduces this problem. A different approach was proposed by Stam [19], where multiple scattering was approximated as a diffusion process. Using this approximation Jensen et al. [20] introduced the use of dipoles to simulate the diffusion in an infinitely thick homogeneous material. The dipole approximation has been revisited and expanded by several authors and has been proved to have applications in real-time rendering [21], [22]. An efficient approximation to solve the diffusion equation is the Lattice-Boltzmann solver, as shown by Geist and Steele [23]. In their approach, the photon density is moved from adjacent nodes along 18 predefined directions. The system iteratively updates the values in the nodes and the links between them. Their mathematical formulation considers scattering and absorption, until the system reaches the equilibrium. The main advantage of lattice methods is that they are based on a sequence of local modifications. Therefore, they scale well with scene size and are straightforward to parallelize. For example, Peng et al. [24] proposed a parallel implementation in the context of fluid simulation on a cluster of PlayStation 3. Recently, Kaplanyan et al. [25] introduced the *Light Propagatino Volumes*, a voxel based local propagation scheme which only considers the 6 adjacent voxels along the principal axes. Their scheme allows to include occluders explicitly, but interactions between voxels are computationally expensive.

Rasterization: these methods exploit rasterization in graphics hardware to solve the participating media problem. In some cases, these leverage on precomputation for visibility or, in general, for the light transport matrix [26]–[28]. Such precomputations limit the solution to static scenes, where only scattering parameters or lighting can be modified in real-time. Dachsbacher et al. [29] proposed one of the first methods that can be applied to dynamic scenes. In their formulation, shadow maps are exploited to capture irradiance, normals, and depth for approximating the dipole evaluation as proposed in Jensen and Buhler [30]. More recently, texture-space methods have become popular. For example, Mertens et al. [31] introduced importance sampling of the BSSRDF (dipole method) for the multiple scattering. This is achieved in texture space using parametrization. More recently, the use of advance filtering in texture-space is widely employed for rendering translucent materials such as skin [32], [33].

All these methods present advantages and disadvantages in terms of computations and quality of the results. Ray-based methods produce very accurate results for the computation of the single scattering term, and refractions. However, they are slow for computing the multiple scattering term. On the contrary, space partition-based methods produce accurate multiple scattering at very high speed but they do not perform well with single scattering. Finally, rasterization methods work well for certain kind of material such as skin (including the evaluation of the BRDF part). Nevertheless, they have several limitations that must be typically addressed using precomputations.

Our method combines the best features of the presented

approaches. The single scattering term and refractions in heterogeneous materials are handled by ray-based methods for getting accurate results. The multiple scattering term is computed using space partition-based methods in order to speed computations up without scarifying quality. Finally, the BRDF evaluation and the first hit are based on OpenGL rasterization.

III. OUTLINE OF THE ALGORITHM

Our algorithm builds on a straightforward 4-pass approach:

1) *Init*: computes the voxelization of the scene and the gradient of the refractive index, and initializes the light rays.
2) *Marching*: traces the light rays from the sources through the volume, accounting for refractive index, scattering and absorption coefficients. Finally, it stores in the voxels the percentage of light that has to be scattered. The process ends when all rays have left the scene or there is no more light to carry.
3) *Diffusion*: computes the multiple scattering contribution of the light deposited in the Marching pass.
4) *View*: casts rays from the point of view and gathers the irradiance.

The parallel architecture of the algorithm is built around the observation that the diffusion pass may be run concurrently with the marching pass. More precisely, as soon as a portion of the volume has been traversed by the light rays, the irradiance left behind inside the volume elements can be diffused through all the volume without waiting for the end of ray traversal. During the diffusion pass, ray may continue to march through the volume. Figure 1.a shows a temporal diagram of the algorithm.

A. Init pass

The input to our algorithm is a scene described as a set of watertight meshes, and the role of this pass is to fill the *Material Volume* and to initialize the light rays.
Filling the Material Volume The *material volume* is a volumetric description of the scene where each voxel stores the *scattering coefficients* σ_{rgb}, the *refractive index* n, the *gradient of refractive index* ∇n and the *occupancy*, i.e. the percentage of non empty volume covered by the voxel. Homogeneous materials have a constant σ_{rgb} and n value, which will be assigned to all the internal voxels. For heterogeneous materials these values can be defined procedurally or with a mapping from the mesh to a volumetric texture. The gradient of the refractive index is computed by finite differences and the occupancy by supersampling the volume with respect to the size of a voxel. As in [9], we filter the gradient with a Gaussian kernel to avoid aliasing effects when refracting rays.

Initializing the light rays For each light source, we render the scene to build a depth map that will be used in the *view pass* for handling shadows. Furthermore, the depth values are converted in world space and used to set the starting point for the rays, so avoiding the marching of rays through the empty
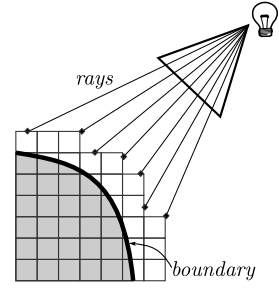

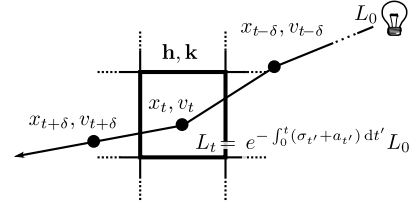Fig. 2. Setting the starting point of rays.

Fig. 3. Updating of particle motion and computation of irradiance to deposit.

volume. We set the starting slightly before the position found with depth value, more precisely by a distance equal to the number of voxels used for the Gaussian smoothing, as shown in Figure 2. This is an optimization that may be disabled if the light source is inside a participating media.

B. Marching Pass

This pass traces the rays through the volume until each ray traverses a predefined number of voxels k. At each step the ray direction is updated using explicit integration (see Figure 3):

$$\mathbf{x}_{t+\delta} = \mathbf{x}_t + \frac{\delta}{n}\mathbf{v}_t \qquad (2)$$

$$\mathbf{v}_{t+\delta} = \mathbf{v}_t + \delta\,\nabla n \qquad (3)$$

where δ is the integration time step, and x_t and v_t are respectively the position and the velocity of the particle along its path at distance t from the source, and n is the refractive index. The value ∇n in a point inside the volume is trilinearly interpolated from the values in the *material volume*. The irradiance left behind by the ray at position t is:

$$L_t = e^{-\int_0^t \sigma_s(t') + \sigma_a(t')\,\mathrm{d}t'}L_0$$

since we perform explicit integration, the integral is approximated as a sum, and σ_s and σ_a are considered constant along the path from t to $t+\delta$.

Implementation: Figure 4 shows a C-like description of the algorithm for a ray. Each ray is assigned to a dedicated OpenCL thread and executed in a Processing Element (PE). We start this pass by instancing a 2-dimensional computation grid with R threads, with R the number of rays. At the beginning, the rays are assigned to the same Streaming Multiprocessor (SM) in packets, so that, at least at the first marching steps, access to global memory tends to be coherent. Unlike in [9], we use a fixed *length* step, δ, because all threads in the same SM share the same instruction pointer.
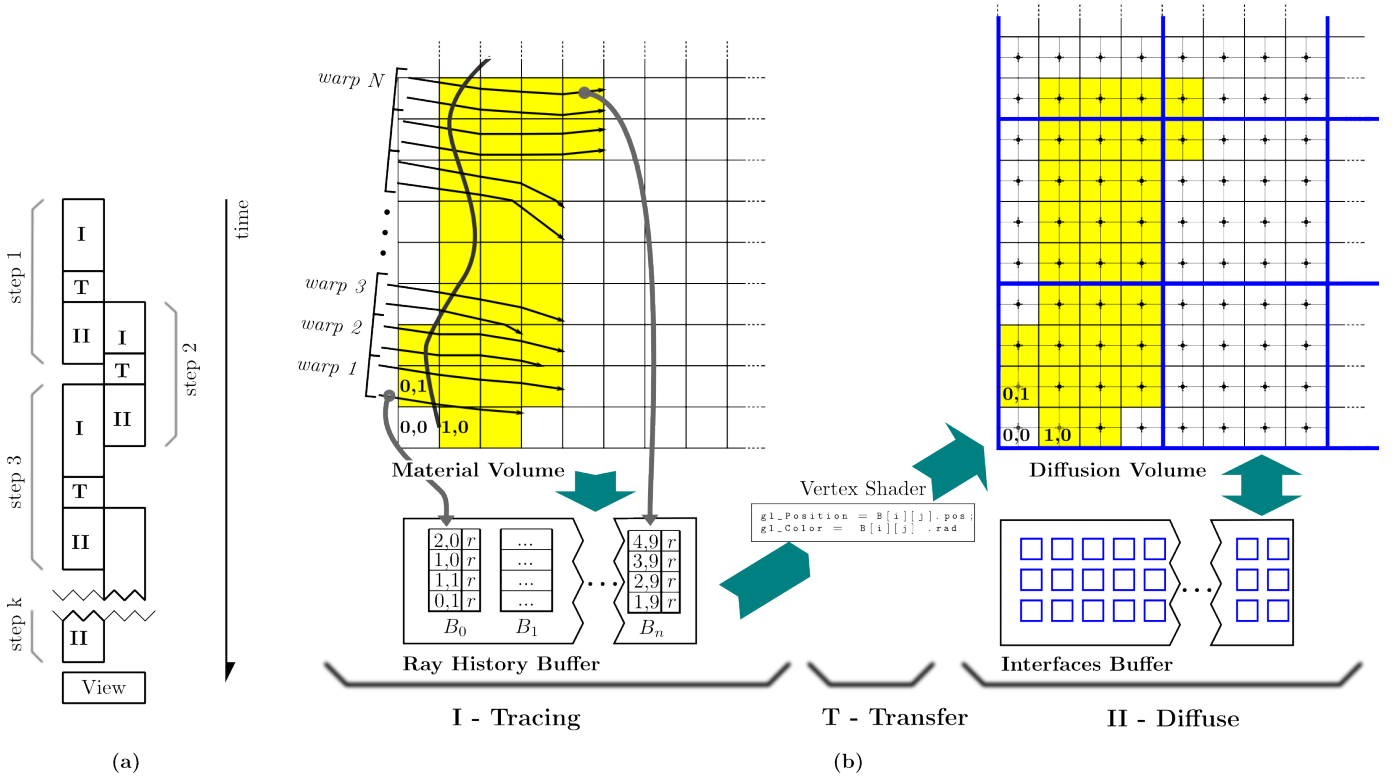
Fig. 1. A schematic description of our parallel rendering pipeline

```
RayMarching(x,v,L0){
  i = 0;
  do{
      h,k = voxel containing x;
      sigma_rgb,grad_n,occ = MaterialVolume[x,v]
      od += exp(— delta * (sigma_rgb+a));
      Lt =  od * L0;
      LocalRayHistoryBuffer[i] = [x,Lt];
      x=x+v/n *delta; v=v+ delta * grad_n/n
      ++i;
  }
  while( (i < k) || (occ==0));
  UploadToGlobalMemory();
}
```

Fig. 4.   The algorithm executed on each thread.

Furthermore, to obtain maximum processing throughput, all the instances of instructions which conditionally modify the control flow (e.g. conditional jumps) must generate the same execution path in every PEs of the SM have to complete their execution at the same time.

The *material volume* resides in global memory and it is read-only, while the *LocalRayHistoryBuffer* is a write-only vector of k positions and resides in the local memory of the SM. When the i-th voxel is traversed, the program writes to this vector at position i (the position of the voxel) the amount of irradiance to be deposited. The last instruction of the program, *UploadToGlobalMemory*, copies the *LocalRayHistoryBuffer* to global memory. When all threads terminate, we will have a buffer in global memory, the *Ray History Buffer*, which is filled with the index of all voxels traversed by all rays and the amount of irradiance left in each one (see Figure 1.b).

Transfer: This operation completes the marching pass by copying the content of the *Ray History Buffer* onto the *Diffusion Volume*. The *diffusion volume* is another grid in one-to-one correspondence with the *material volume*. A voxel of the *diffusion volume* stores an RGB irradiance value. We perform the transfer operation in OpenGL by binding the *diffusion volume* texture as render target and issuing the rendering of a batch of $R \times k$ points. In a vertex shader, we fetch the voxel coordinates and the irradiance from the *Ray History Buffer* and use them as output position and color, respectively. Since multiple rays may contribute to the same voxel, we enable blending to accumulate each contribution. The transfer is the only stage where we use OpenGL. Note that this operation cannot be done directly in OpenCL due to the lack of atomic floating point operation needed for irradiance accumulation. When the OpenGL accumulation step ends, we bind again the *Ray History Buffer* as output and restart the threads assigned to the marching pass and the threads assigned to Diffusion pass.

C. Diffusion pass

In this pass, we propagate the irradiance stored in the *diffusion volume*. As we did in the Marching step, we want to exploit the parallelism of the GPU by breaking up the problem into subproblems, and assigning each one to a dedicated SM. Therefore, we partition the *diffusion volume* in blocks of b^3 voxels (highlighted in blue in figure 1.b) and compute the diffusion of light within each block. We proceed iteratively: for each iteration we run the diffusion process in parallel for
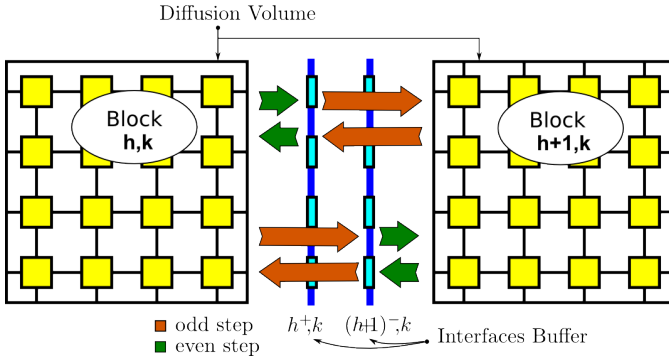
Fig. 5. Avoid conflicts in writing/reading to/from the Interface Buffer.

each block and store the leaving photon density in an ad hoc *interfaces buffer* of size $6 \times b \times b$ that will serve as input the neighbor blocks at the next iteration.

Diffusion inside a block: The transport of light inside a block is computed by means of the LBL approach [23]. LBL has proven to be a viable method to simulate the light diffusion process in a medium. The transport of light is discretized in space and time, by modeling the volume with a lattice where each node (i.e. each voxel) stores the photon density along a predefined set of directions and updated these densities at discrete time steps. In the original paper, each node is connected to its 6 neighbor nodes along the principal axis, and the 12 nodes along the diagonals on the 3 planes $X = 0$, $Y = 0$ and $Z = 0$. To comply with current memory limit of the SM, we use blocks of 4^3 voxels and only the 6 principal directions. Following the notation of the original paper, the photon density, $f_i(r,t)$, arriving at lattice site (i.e. the voxel) r at time t in the direction c_i is computed as:

$$f_i(r + \lambda c_i, t + \delta) = \Theta_i \ f(r,t) \qquad (4)$$

where:

$$\Theta_{0j} = \begin{cases} 0 & j = 0 \\ \sigma_a & j > 0 \end{cases}$$

$$\Theta_{ij} = \begin{cases} 1/6 & j = 0 \\ \sigma_s/6 & j > 0 \\ 1 - \sigma_t + \sigma_s/6 & j = i \end{cases} \qquad (5)$$

The initial boundary conditions consist of a presence of photon density in some nodes (usually at voxels on the boundary of the object, if the light comes from outside).

The memory consumption of the LBL data structure for $b = 4$ (i.e. the vector of photon density f) is 6144 bytes (6 directions, 4^3 voxels, 4 channels, and 4 bytes for storing the density). The *interface buffer* requires 1536 byte for a block (b^2 elements, 6 faces, 4 channels, 4 bytes for the density). In total, $6144 + 1536 = 7680$ bytes.
At the end of the Diffusion pass, the irradiance in the voxels is accumulated in a copy of the Diffusion Volume, that we call *Irradiance Volume*, which will be the one ultimately used in the View pass. Note that the Interfaces Buffer is accessed both for reading and writing by concurrent threads. Therefore, we must guarantee that there are no access conflicts. Figure 5

shows two adjacent blocks and the Interface Buffer among the two. Note that, within a block, we can use the same locations in the *interfaces buffer* both for reading and for writing, because each location is read and written only by the thread controlling the single voxel corresponding to it. Moreover, the irradiance written to the interface (h^+, k) has to be read in the subsequent step by the block $h+1, k$ (and vice-versa). Therefore, we simply alternate h^+, k and $(h+1)^-, k$ as *interfaces buffer* for the two blocks to avoid conflicts. This is shown in Figure 5.

Termination condition: Before performing the view pass and displaying the frame, we must guarantee that the system has reached an equilibrium state. This means that performing additional steps will not change the current state of irradiance on each voxel. The number of steps typically varies with the characteristic of the volume, so it cannot be fixed beforehand. We test for convergence by monitoring the amount of irradiance in a sparse set of points and stopping when the relative change is under a predefined threshold.

D. View pass

The View pass is the last step of our rendering algorithm. We instantiate a 2-dimensional computation grid of the size of the framebuffer such that the calculation of the color at each pixel will be assigned to a kernel instance. In this pass, the rays are shot from the observer's viewpoint toward the *irradiance volume* and marched through it, taking into account refraction. At each step of the marching, a view ray accumulates the irradiance read from the *irradiance volume*, calculating the color of the pixel. Finally, the direct lighting contribution is calculated evaluating the BRDF of the surface (i.e. Lambertian BRDF), where shadow maps computed in the Init pass are used for the visibility test.

IV. RESULTS AND DISCUSSION.

We performed several tests with different materials and lighting conditions on a Intel Core2 Duo 2.66 GHz, 2 GB RAM, equipped with a NVidia GeForce GTX 465.

The first test aims at evaluating the effect of the subdivision of the volumes in blocks operated in Section III-C. Applying LBL inside each block of voxels and transmitting the photon density through the *interfaces buffer* is not the same as performing a LBL globally over all the volume. Therefore, we run a simple test with a single light in a homogeneous material to evaluate the difference. Figure 6 shows a comparison for two values of albedo.

Figure 7 shows the comparison between our approach and a ground truth image of a sphere of homogeneous material lit by a spot light. The ground truth image was produced using classic volumetric photon mapping [7]. Particle tracing was modeled to match the same scattering model of our technique. Therefore, we employed the Eikonal equation to bend photons when moving into the medium. We adopted sampling techniques presented in Raab et al. [34] in order to have unbiased particle tracing, and to improve efficiency when integrating the final contributions.
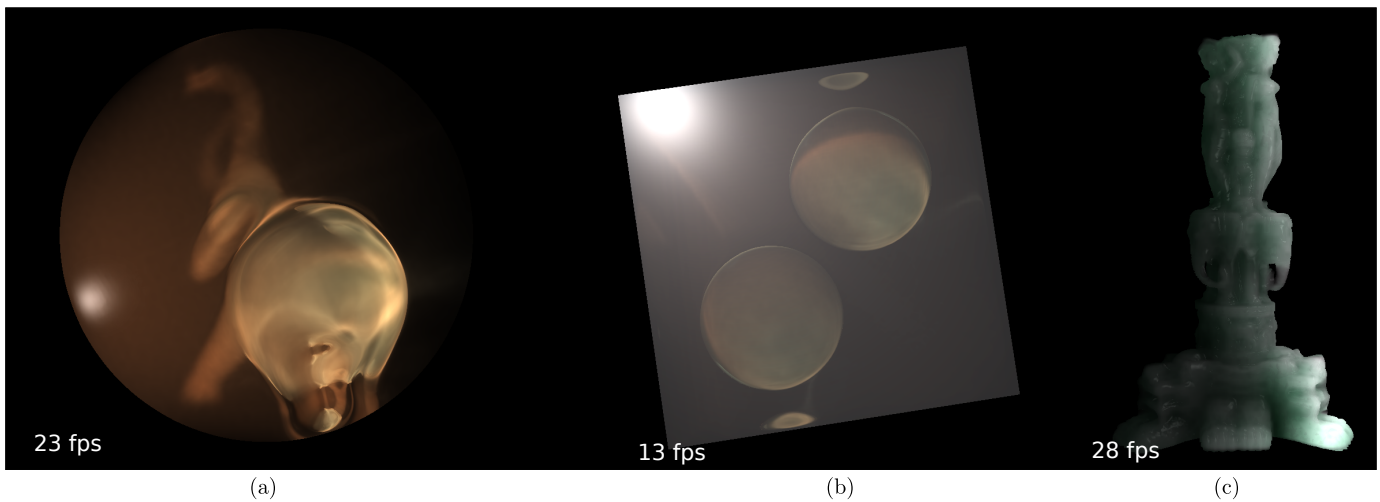
Fig. 8.   Renderings showing refraction and multiple scattering.
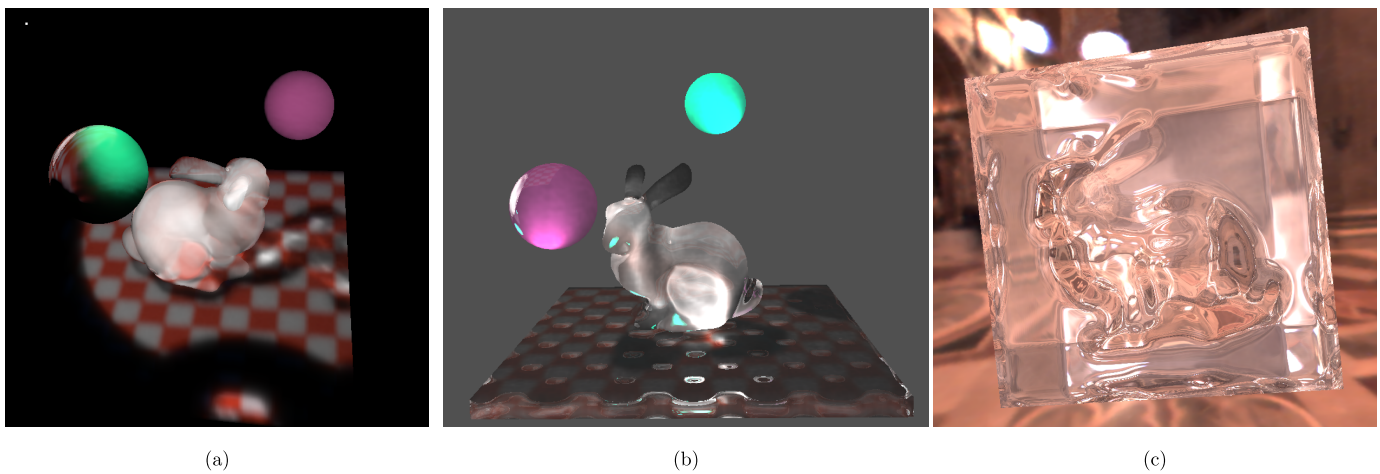

Fig. 9.   (a) Renderings showing refraction and multiple scattering in heterogeneous media. (b)-(c) Including an environment map.

Fig. 10.   A few frames from a real time animation (see accompanying video).

Figure 8.(a) shows a model of an elephant placed inside a sphere with different materials and a spot light placed opposite to the viewer. In this case, we used a material with a constant hue and scattering coefficients, only changing refractive index and extinction coefficient between the elephant and the sphere. We can observe how the sharpness of refraction effects varies with the depth inside the sphere. Figure 8.(b) shows a similar situation with two spheres inside a cube. Renderings in figure 9 are computed with high albedo materials and therefore refraction related effects are more prominent. For example, you can see the caustics on the checkboard and image of the checkboard on the surface of the green sphere. Figure 9.(c) shows bunny immersed in a cube and Figure 10 show 4 frames of the animation of a running elephant (see the video accompanying this submission). For these last two examples, we also used an environment map, that we are able to easily incorporate into our framework.

A. Performances

We conducted a number of experiments to analyze the efficiency of our algorithm when changing its parameters.

An important number is the size of the *Ray History Buffer*, because it determines the granularity of the parallelization
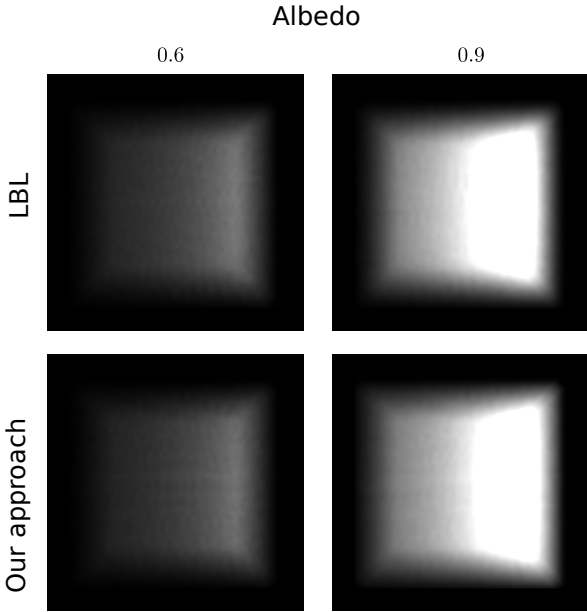
Albedo

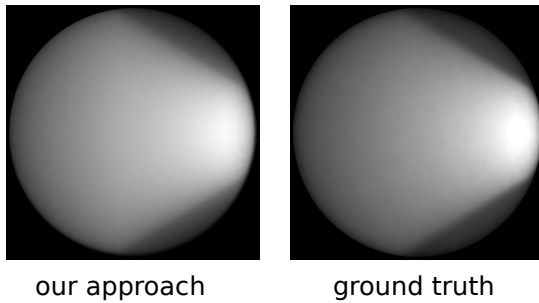Fig. 6. Comparison of Lattice Boltzmann Lighting with our approach for two albedo values.

our approach   ground truth

Fig. 7. Comparison of our approach against a ground truth image.

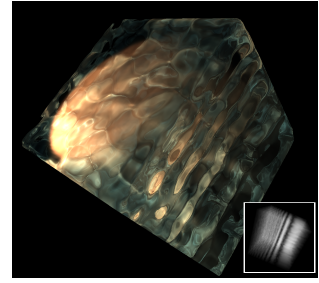| k	cube
2	86
4	81
8	73
16	**70**
32	79
64	82

TABLE I
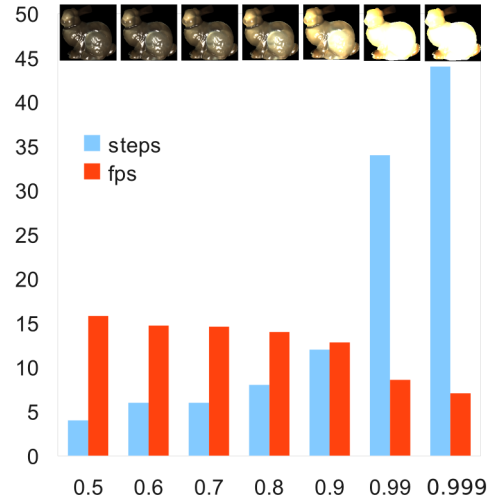RENDERING TIME (IN MS) FOR DIFFERENT SIZES OF *Ray History Buffer k* FOR THE CASE OF A 128^3 VOLUME ENTIRELY FILLED.

Fig. 11. Performances of our algorithm for different albedo values.

between *diffusion pass* and *marching pass* of consecutive steps and the total number of steps to convergence. We studied the relation between the size of the *Ray History Buffer* and the total number of steps to convergence. Small values of k correspond to a dense interleaving of marching and diffusion pass and a higher number of steps of the algorithm. Large values of k means less dense interleaving and small number of steps. In the extreme case of unlimited *Ray History Buffer*, we would have a single marching phase followed by a single diffusion phase. Table I shows the rendering time for different sizes of *Ray History Buffer* from 2 to 64 for the elephant dataset. Too small sizes of the *Ray History Buffer* lead to longer rendering times, because the overhead of the OpenGL transfer and because the work groups computing diffusion perform the same steps of LBL updates (8 in our experiments) independent of the value of irradiance in the voxels. For our hardware setting, *Ray History Buffer* with size $k = 16$ turned our to be an optimal value, while the rendering times were higher around this value. This is an indication that the proposed architecture is effective w.r.t. executing the two passes in a sequence, otherwise greater values of k should correspond to shortest times.

Figure 11 shows the dependence between the albedo of the material and the number of steps required to reach convergence for a 128^3 volume and produce a 1024^2 image. As expected, the algorithm takes more steps and hence is slower with materials with high values of albedo, but it never falls under 8 fps even for albedo approaching 1.

A test more directed to prove the effectiveness of the framework has been conducted by running a sequential execution of a single marching and diffusion phases against our algorithm, which shows that our algorithm is on the average 30% faster. Note that this gain is only due to the parallel execution of the diffuse phase of the i-th step with the marching phase of the $(i + 1)$-th step. We expect a further improvement in the out-of-core implementation of our framework, where our approach will also benefit the cache coherent access to memory, while in this implementation all the dataset is in video memory.

The Marching phase and the View pass of our algorithm is mostly done like in [10], but we also considering multiple scattering at little additional cost and preserving the possibility of changing light and material. The algorithm by Wang et al. [22] also implements multiple scattering of heterogeneous and does not suffer from the limitations due to voxelization in handling sharp features. However, it requires a complete

tetrahedralization of the model which takes several minutes of computation (10 minutes for the gargoyle model) and does not take into account refraction. The Ligh Propagation Maps [18] provide a more accurate solution for light transport in participating media, but it requires minutes to produce a single image. The mesh based approach by Walter et al. [12] produces more precise refraction effect than ours but it only handles single scattering and only for one boundary between two constant refractive-index materials.

Conclusions and Future Work

In this paper, we have presented a novel approach to real-time rendering of heterogeneous media, based on the idea of decoupling and parallelizing the computation of refraction and scattering phenomena. This is achieved by localizing the computation for lattice-Boltzmann lighting and thus overcoming the memory limitations of GPUs. To our knowledge, this is the first real-time approach for volume rendering that includes refraction and multiple scattering for heterogeneous meterials. This work sets the basis for an out-of-core version of the algorithm for large scenes, which will be addressed in future work.

REFERENCES

[1] J. T. Kajiya and B. P. V. Herzen, "Ray Tracing Volume Densities," in *Computer Graphics (ACM SIGGRAPH {'}84 Proceedings)*, vol. 18, Jul. 1984, pp. 165–174.

[2] E. Cerezo, F. Perez-Cazorla, X. Pueyo, F. Seron, and F. Sillion, "A Survey on Participating Media Rendering Techniques," *the Visual Computer*, 2005. [Online]. Available: http://artis.imag.fr/Publications/2005/CPPSS05

[3] S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, and M. Stich, "Optix: a general purpose ray tracing engine," *ACM Trans. Graph.*, vol. 29, pp. 66:1–66:13, July 2010. [Online]. Available: http://doi.acm.org/10.1145/1778765.1778803

[4] M. Pharr and G. Humphreys, *Physically Based Rendering: From Theory to Implementation*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

[5] E. Lafortune and Y. Willems, "Rendering Participating Media with Bidirectional Path Tracing," in *Proceedings of the Seventh Eurographics Workshop on Rendering, Porto, Portugal*, 1996, pp. 92–101.

[6] M. Pauly, T. Kollig, and A. Keller, "Metropolis light transport for participating media," in *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*. London, UK: Springer-Verlag, 2000, pp. 11–22. [Online]. Available: http://portal.acm.org/citation.cfm?id=647652.732117

[7] H. W. Jensen, *Realistic image synthesis using photon mapping*. Natick, MA, USA: A. K. Peters, Ltd., 2001.

[8] D. Gutierrez, A. Muñoz, O. Anson, and F. Seron, "Non-Linear Volume Photon Mapping," in *Eurographics Symposium on Rendering 2005*, 2005, pp. 291–300.

[9] I. Ihrke, G. Ziegler, A. Tevs, and C. Theobalt, "Eikonal rendering: efficient light transport in refractive objects," *ACM Trans. Graph.*, vol. 26, no. 3, 2007. [Online]. Available: http://portal.acm.org/citation.cfm?id=1275808.1276451

[10] X. Sun, K. Zhou, E. Stollnitz, J. Shi, and B. Guo, "Interactive relighting of dynamic refractive objects," in *ACM SIGGRAPH 2008 papers*. ACM, 2008, p. 35. [Online]. Available: http://portal.acm.org/citation.cfm?id=1360634

[11] C. Cao, Z. Ren, B. Guo, and K. Zhou, "Interactive Rendering of Non-Constant, Refractive Media Using the Ray Equations of Gradient-Index Optics," *Media*, 2010.

[12] B. Walter, S. Zhao, N. Holzschuch, and K. Bala, "Single scattering in refractive media with triangle mesh boundaries," 2009. [Online]. Available: http://portal.acm.org/citation.cfm?id=1576246.1531398

[13] N. Carr, J. Hall, and J. Hart, "GPU algorithms for radiosity and subsurface scattering," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association Aire-la-Ville, Switzerland, Switzerland, 2003, pp. 51–59. [Online]. Available: http://portal.acm.org/citation.cfm?id=844182

[14] S. Chandrasekhar, *Radiative transfer*. Clarendon Press, 1950.

[15] C. Patmore, "Simulated multiple scattering for cloud rendering," in *Proceedings of the IFIP TC5/WG5.2/WG5.10 CSI International Conference on Computer Graphics: Graphics, Design and Visualization*. Amsterdam, The Netherlands, The Netherlands: North-Holland Publishing Co., 1993, pp. 29–40. [Online]. Available: http://portal.acm.org/citation.cfm?id=645465.653826

[16] N. Max, "Efficient light propagation for multiple anisotropic volume scattering," in *In Proceedings of the 5th Eurographics Workshop on Rendering*, 1994, pp. 87–104.

[17] K. FRANKLIN EVANS, "The Spherical Harmonics Discrete Ordinate Method for Three-Dimensional Atmospheric Radiative Transfer," *American Meteorological Society*, pp. 429–446, 1998.

[18] R. Fattal, "Participating media illumination using light propagation maps," *ACM Transactions on Graphics*, vol. 28, no. 1, pp. 1–11, Jan. 2009. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1477926.1477933

[19] J. Stam, "Multiple scattering as a diffusion process," in *Eurographics Rendering Workshop 1995*. Citeseer, 1995, pp. 41–50. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.35.4213&rep=rep1&

[20] H. Jensen, S. Marschner, M. Levoy, and P. Hanrahan, "A practical model for subsurface light transport," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM New York, NY, USA, 2001, pp. 511–518. [Online]. Available: http://portal.acm.org/citation.cfm?id=383259.383319

[21] K. Zhou, Z. Ren, S. Lin, H. Bao, B. Guo, and H.-Y. Shum, "Real-time smoke rendering using compensated ray marching," *ACM Transactions on Graphics*, vol. 27, no. 3, p. 1, Aug. 2008. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1360612.1360635

[22] Y. Wang, J. Wang, N. Holzschuch, K. Subr, J. Yong, and B. Guo, "Real-time Rendering of Heterogeneous Translucent Objects with Arbitrary Shapes," in *Computer Graphics Forum*, vol. 29, no. 2. John Wiley & Sons, 2010, pp. 497–506. [Online]. Available: http://www3.interscience.wiley.com/journal/123500408/abstract

[23] R. Geist, K. Rasche, J. Westall, and R. Schalkoff, "Lattice-Boltzmann Lighting," in *Eurographics Symposium on Rendering*, A. Keller and H. W. Jensen, Eds. Norrkoping, Sweden: Eurographics Association, 2004, pp. 355–362. [Online]. Available: http://www.eg.org/EG/DL/WS/EGWR/EGSR04/355-362.pdf

[24] L. Peng, K.-i. Nomura, T. Oyakawa, R. K. Kalia, and A. Nakano, "Parallel Lattice Boltzmann Flow Simulation on Emerging," *Cell*, pp. 763–777, 2008.

[25] A. Kaplanyan and C. Dachsbacher, "Cascaded light propagation volumes for real-time indirect illumination," *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games - I3D '10*, p. 99, 2010. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1730804.1730821

[26] H. P. A. Lensch, M. Goesele, P. Bekaert, J. Kautz, M. A. Magnor, J. Lang, and H.-P. Seidel, "Interactive Rendering of Translucent Objects," *Computer Graphics and Applications, Pacific Conference on*, vol. 0, p. 214, 2002.

[27] X. Hao and A. Varshney, "Real-time rendering of translucent meshes," *ACM Trans. Graph.*, vol. 23, pp. 120–142, 2004.

[28] R. Wang, J. Tran, and D. Luebke, "All-frequency interactive relighting of translucent objects with single and multiple scattering," *ACM Transactions on Graphics (TOG)*, vol. 24, no. 3, p. 1207, 2005. [Online]. Available: http://portal.acm.org/citation.cfm?id=1073204.1073333

[29] C. Dachsbacher and M. Stamminger, "Translucent shadow maps," in *EGRW {'}03: Proceedings of the 14th Eurographics workshop on Rendering*. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2003, pp. 197–201.

[30] H. W. Jensen and J. Buhler, "A rapid hierarchical rendering technique for translucent materials," *ACM Trans. Graph.*, vol. 21, pp. 576–581, July 2002. [Online]. Available: http://doi.acm.org/10.1145/566654.566619

[31] T. Mertens, J. Kautz, P. Bekaert, F. V. Reeth, and H.-P. Seidel, "Efficient Rendering of Local Subsurface Scattering," *Computer Graphics Forum*, vol. 24, p. 41, 2005.

[32] E. d'Eon, D. Luebke, and E. Enderton, "Efficient rendering of human skin," in *Rendering Techniques 2007: 18th Eurographics Workshop on Rendering*, Jun. 2007, pp. 147–158.

[33] J. Jimenez, D. Whelan, V. Sundstedt, and D. Gutierrez, "Real-time realistic skin translucency," *IEEE Computer Graphics and Applications*, vol. 30, no. 4, pp. 32–41, 2010.

[34] D. S. Matthias Raab and A. Keller, "Unbiased global illumination with participating media," in *In Proceeding of Monte Carlo and Quasi-Monte Carlo Methods 2008.* Springer, 2008, pp. 591–606.