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 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 renderings of virtual scenes arer needed in many application domains such as movie production, video games and virtual reality. In its most complete formulation, the problem requires to accurately simulate the behavior of light, knowing the optical characteristics of all the materials in the virtual scene.

The transmission 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 refractive index, it may scatter as a result of a collision of photons with particles of the material, and finallly, it may be absorbed or emitted by the material. Even though the laws governing light-matter interaction are well 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 scenes accounting for reflection, refraction, absorbtion and scattering phenomena. We employ a voxelization of scene space, storing reractive and scattering coefficients in each voxel. At each frame, the algorithm traces rays from the light sources through the scene deviating their path according to 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 have terminated their computations, view rays are shot, traversing the scene and reading back the irradiance.

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 light with general media is modeled by the volume rendering equation:

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

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 the scattering and extinction coefficients. The volume rendering equation captures all lighting effects generated by the uncountable number of local interactions between light and volume particles. Note that $\sigma_t = \sigma_s + \sigma_a$ where σ_a is the absorption coefficient. A lot of effort has been devoted to efficiently solve this equation (see Cerezo et al.'s survey [1] for an overview). Despite the 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 camera position 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

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

between them depending on light sources. These approaches include radiosity, Lattice Boltzmann Lighting (LBL) etc.
- Rasterization-based: The approaches that rely on the rasterization-based hardware pipeline such as shadow mapping, screen space ambient occlusion, translucent shadow mapping, etc.

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

Ray-based Methods: these methods leverage a ray-tracing core and typically use Monte Carlo techniques to evaluate Equation 1, allowing fully heterogeneous materials (varying absorption, scattering, and index of refraction coefficients). These methods have now become popular thanks to modern GPUs, which can trace large numbers of rays simultaneously and allow interactive/real-time performances. For example, Ihrke et al. [2] introduced a particle-based method derived from the Eikonal equation that achieves interactive frame rates. Sun et al. [3] 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 has been extended in order to handle a closed form analytical formulation of light ray trajectory on constant gradient refractive media [4]. Finally, Walter et al. [5] showed a real-time formulation that determines the focal points on the boundary of an object.

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 choice 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. However, general solutions to the volume rendering equation are computationally intensive for accurate multiple scattering inside dense heterogeneous solids. To reduce complexity, Discrete Ordinates Methods (DOM) are used, where radiation transfer between small voxels is limited to a predefined set of discrete directions. Nevertheless, these methods suffer from the ray-effect, a banding artifact due to the discretization of directions. Recently, Fattal [6] introduced the *Light Propagation Maps* which highly reduces this problem.
A different approach was proposed by Stam [7], where multiple scattering was approximated as a diffusion process. Following this idea, Geist and Steele [8] adopted a Lattice-Boltzmann solver to compute the diffusion equation. 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. In a similar fashion Kaplanyan et al. [9] introduced *Light Propagation 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. One of the first methods that could be applied to dynamic scenes was proposed by Dachsbacher et al. [10] with the name of *Translucent Shadow Maps*. In their formulation, shadow maps are exploited to capture irradiance, normals, and depth to compute the BSSRDF of homogeneous media. More recently, texture-space methods have become popular. For example, Mertens et al. [11] introduced importance sampling of the BSSRDF (dipole method) for multiple scattering. This is achieved in texture space using parametrization. More recently, the use of advanced texture-space filtering is widely employed for rendering translucent materials such as skin [12].
All these methods present advantages and disadvantages in terms of computations and quality of the results. Ray-based methods efficiently produce very accurate results for the computation single scattering and refractive effects. However, they are slow in 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 kinds of materials 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 a ray-based algorithm for getting accurate results. The multiple scattering term is computed using space partition-based methods in order to speed computations up without sacrificing quality. Finally, the BRDF evaluation and the first hit are determined using 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, rays may continue to march through

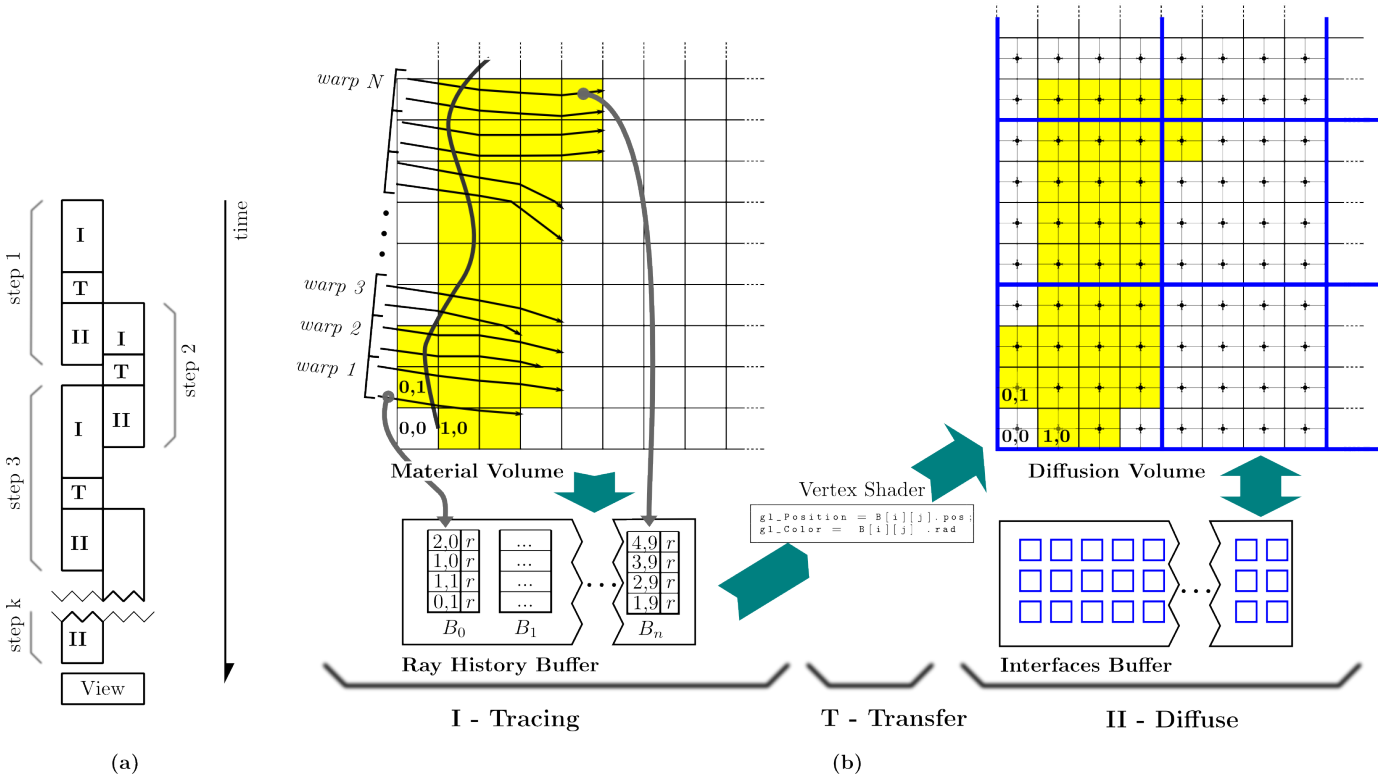


Fig. 1. A schematic description of our parallel rendering pipeline

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 [2], 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 volume. We set the starting point slightly before the position found with the depth value, more precisely by a distance equal

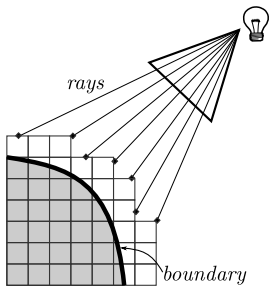


Fig. 2. Setting the starting point of rays.

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 the Eikonal equations (see Figure 3):

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

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

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

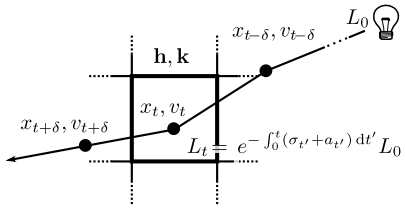


Fig. 3. Update of particle motion and computation of irradiance to deposit.

```
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.

interpolated from the values in the *material volume*. The irradiance left behind by the ray at position t is:

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

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 [2], we use a fixed *length* step, δ, because all threads in the same SM share the same instruction pointer. 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 a 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 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 [8]. 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 updates 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 limits 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 along direction c_i is computed as:

$$f_i(r+\lambda c_i, t+\delta) = \Theta_i \; f(r,t) \tag{4}$$

where:

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

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

The memory consumption of the LBL data structure for $b = 4$ (i.e. the vector of photon density f) is 6144 bytes (6

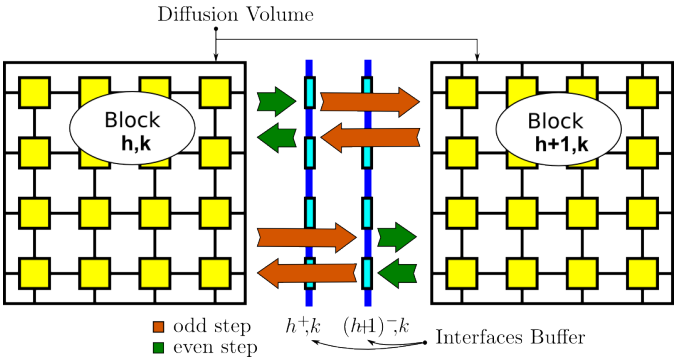


Fig. 5. Avoiding conflicts in writing/reading to/from the Interface Buffer.

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 an OpenCL thread. 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.

A. Comparison with Ground Thruth

Figure 6 shows a comparison between our approach and a ground truth image of a sphere of homogeneous material lit by a spot light. All ground truth images have been produced using classic volumetric photon mapping.

Particle tracing was adapted to match the same refraction model used in our technique; more precisely, we employed the Eikonal equation to bend photons when moving inside the medium. However, when moving from air into the medium, the photon tracer uses the normals of the mesh to compute Snell's equations, thus achieving higher quality renderings. The resulting difference can be noticed in figure 7 in details such as the Armadillo's hands or the Bunny's ears.

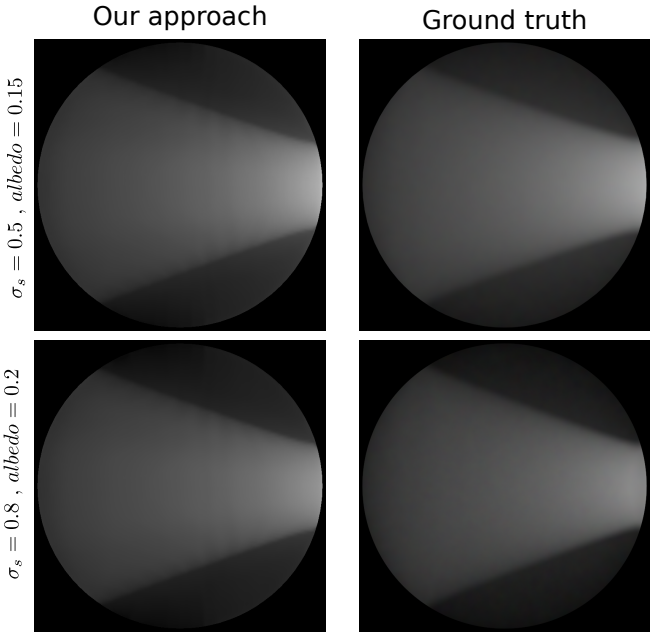


Fig. 6. Comparison of our approach against a ground truth image with different values of scattering and albedo.

V. Approximation of LBL

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 the entire volume. Therefore, we ran a simple test with a single light in a homogeneous material to evaluate the difference. Figure 9 shows a comparison for two different albedo values, showing how the two algorithms achieve comparable results.

Figure 10.(a) shows a model of a homogeneous elephant placed inside an otherwise homogeneous sphere. The difference in refrafractive indices between the two materials produces some caustics within the object.

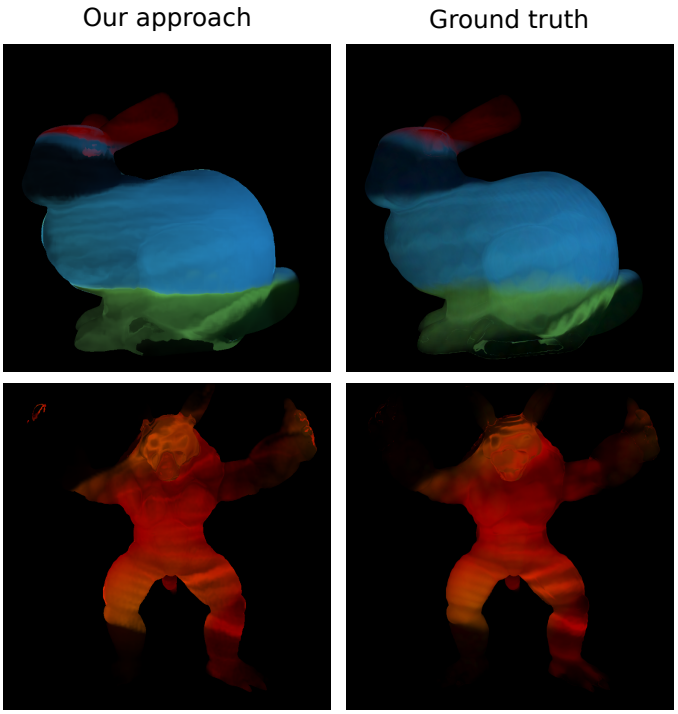


Fig. 7. On the left column, two renderings of the Bunny and Armadillo model at 12 fps made of heterogeneous materials. On the right column the scenes are rendered with photon mapping (228M and 63M of photons, respectively). The same spot light was placed on the right of the Bunny and on the left of the Armadillo.

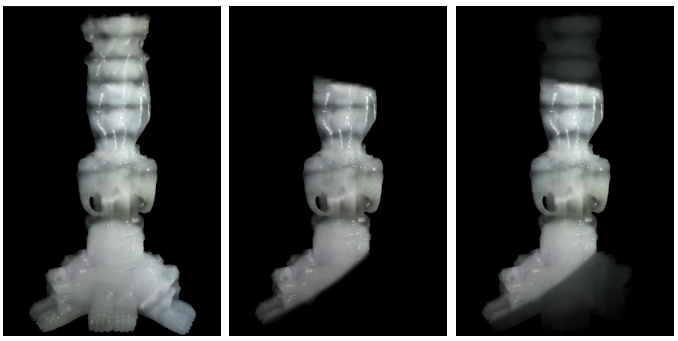


Fig. 8. Multiple and single scattering. Left) The Thai model lit by a point light placed at its right; Center) The same model lit by a spot light placed at the same position, **single** scattering only; Right) Same as center, with **multiple** scattering re-enabled.

Figure 10.(b) shows a similar situation with two spheres inside a cube. Renderings in figure 11 are computed from low scattering coefficient materials and therefore refraction related effects are more prominent. For example, caustics on the checkboard and image of the checkboard on the surface of the green sphere can be seen. Figure 11.(c) shows a bunny immersed in a cube and Figure 12 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.

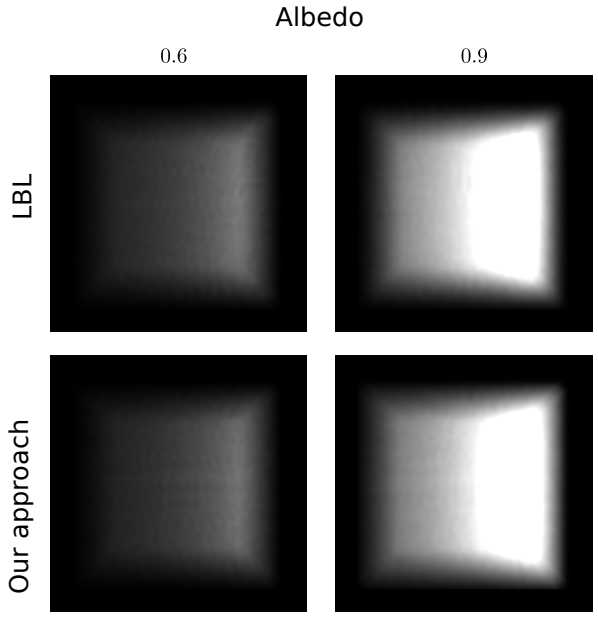


Fig. 9. Comparison of Lattice Boltzmann Lighting with our approach for two albedo values.

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 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 the *Ray History Buffer* from 2 to 64 for the bubbly cube dataset. Small sizes of the *Ray History Buffer* lead to longer rendering times, mainly due to the overhead of the OpenGL transfer. For our hardware setting, *Ray History Buffer* with size $k = 16$ turned out to be an optimal value. This suggests that the proposed architecture is effective w.r.t. executing the two passes in a sequence, otherwise greater values of k should correspond to shorter times.
Figure 13 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 phase against our algorithm, which shows that our algorithm is on the average 30% faster. Note that this gain is only due to the

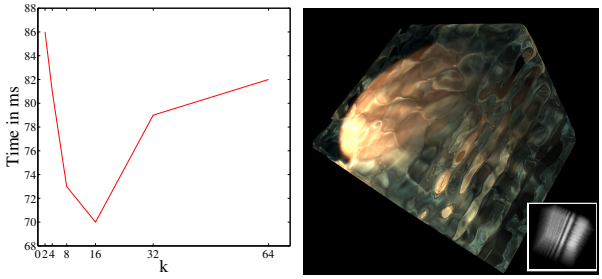


TABLE I
RENDERING TIME (IN MS) FOR DIFFERENT SIZES OF *Ray History Buffer k* FOR THE CASE OF A 128^3 VOLUME ENTIRELY FILLED.

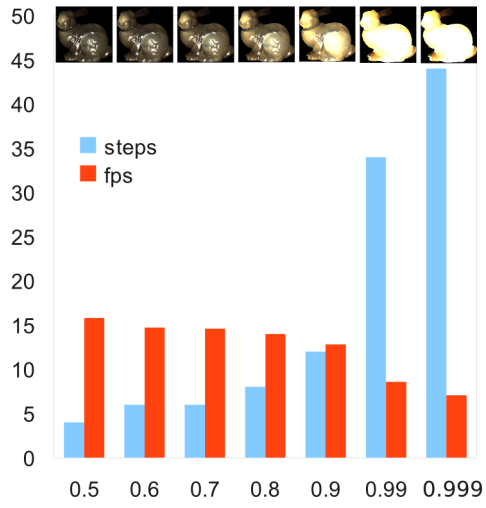


Fig. 13. Performances of our algorithm for different albedo values.

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 from the cache coherent access to memory. Note that in the present implementation the entire dataset resided in video memory.

While the Marching phase and the View pass of our algorithm is mostly performed like in [3], we are also able to compute multiple scattering at little additional cost, additionally preserving the possibility of changing light and material. The algorithm by Wang et al. [13] also implements multiple scattering of heterogeneous materials 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. Light Propagation Maps [6] provide a more accurate solution for light transport in participating media, but still require minutes to produce a single image. The mesh based approach by Walter et al. [5] produces more precise refraction effects 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 a future work.

REFERENCES

[1] E. Cerezo, F. Perez-Cazorla, X. Pueyo, F. Seron, and F. Sillion, "A Survey on Participating Media Rendering Techniques," *the Visual Computer*, 2005.

[2] 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.

[3] 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.

[4] 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.

[5] B. Walter, S. Zhao, N. Holzschuch, and K. Bala, "Single scattering in refractive media with triangle mesh boundaries," 2009.

[6] R. Fattal, "Participating media illumination using light propagation maps," *ACM Transactions on Graphics*, vol. 28, no. 1, pp. 1–11, Jan. 2009.

[7] J. Stam, "Multiple scattering as a diffusion process," in *Eurographics Rendering Workshop 1995*. Citeseer, 1995, pp. 41–50.

[8] 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.

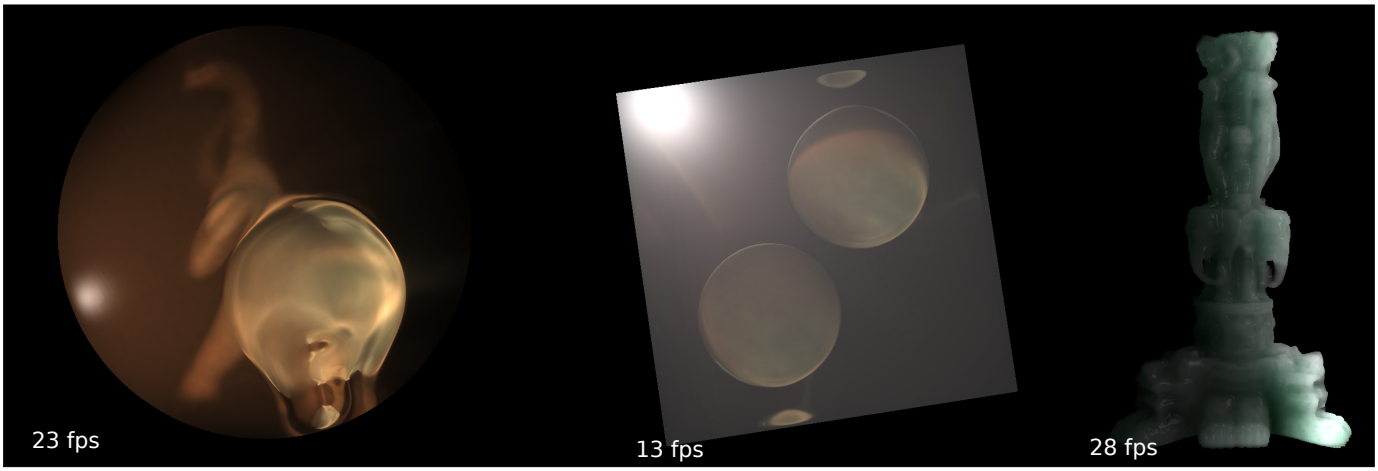
[9] A. Kaplanyan and C. Dachsbacher, "Cascaded light propagation volumes for real-time indirect illumination," *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games - I3D '10*, p. 99, 2010.

[10] 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.

[11] 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.

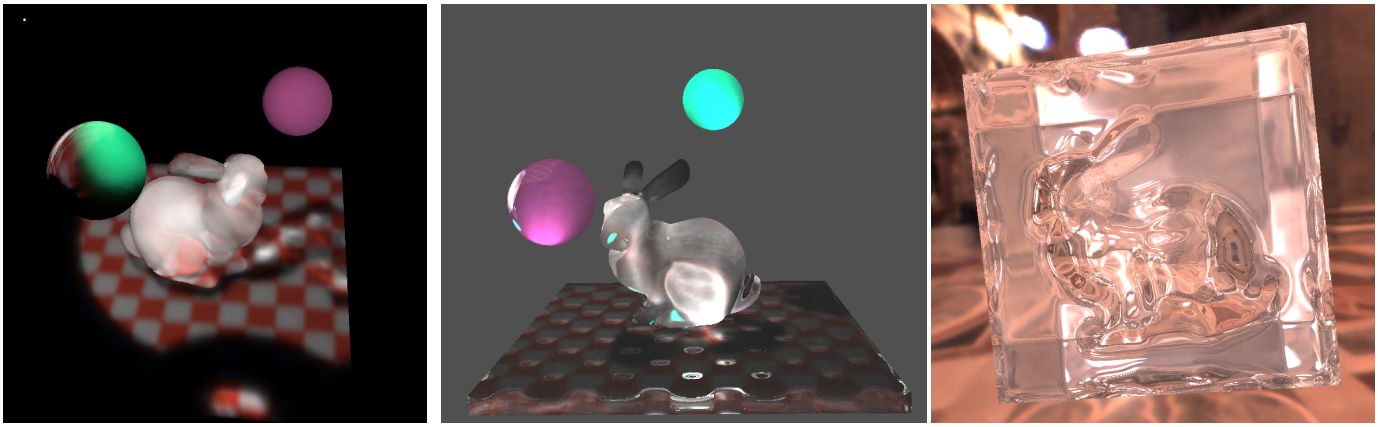
[12] J. Jimenez, D. Whelan, V. Sundstedt, and D. Gutierrez, "Real-time realistic skin translucency," *IEEE Computer Graphics and Applications*, vol. 30, no. 4, pp. 32–41, 2010.

[13] 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.



(a) (b) (c)

Fig. 10. Renderings showing refraction and multiple scattering.



(a) (b) (c)

Fig. 11. (a) Renderings showing refraction and multiple scattering in heterogeneous media. (b)-(c) Including an environment map.

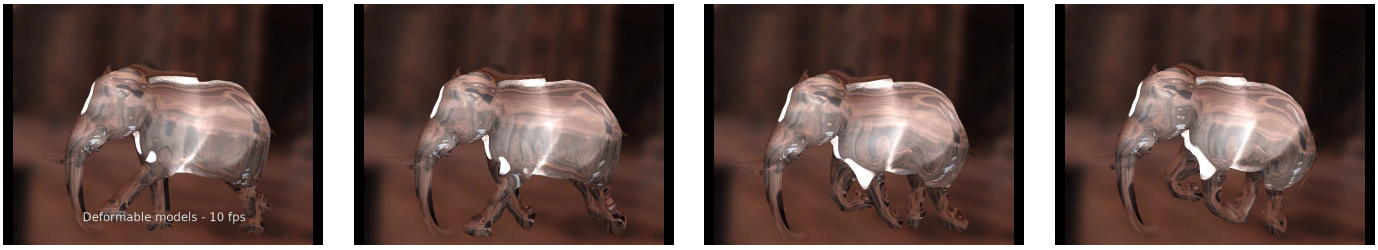


Fig. 12. A few frames from a real time animation (see accompanying video).