TransCut: Interactive Rendering of Translucent Cutouts

Dongping Li* Xin Sun† Zhong Ren* Stephen Lin† Yiying Tong‡ Baining Guo† Kun Zhou*

*Zhejiang University †Microsoft Research Asia ‡Michigan State University

Abstract—We present *TransCut*, a technique for interactive rendering of translucent objects undergoing fracturing and cutting operations. As the object is fractured or cut open, the user can directly examine and intuitively understand the complex translucent interior, as well as edit material properties through painting on cross sections and recombining the broken pieces – all with immediate and realistic visual feedback. This new mode of interaction with translucent volumes is made possible with two technical contributions. The first is a novel solver for the diffusion equation (DE) over a tetrahedral mesh that produces high-quality results comparable to the state-of-art finite element method of Arbree et al. [1] but at substantially higher speeds. This accuracy and efficiency is obtained by computing the discrete divergences of the diffusion equation and constructing the DE matrix using analytic formulas derived for linear finite elements. The second contribution is a multi-resolution algorithm to significantly accelerate our DE solver while adapting to the frequent changes in topological structure of dynamic objects. The entire multi-resolution DE solver is highly parallel and easily implemented on the GPU. We believe TransCut provides a novel visual effect for heterogeneous translucent objects undergoing fracturing and cutting operations.

Index Terms—subsurface scattering, heterogeneous material, diffusion equation, multi-resolution.

1 Introduction

The appearance of a heterogeneous translucent object arises from complex interactions between light and the material volume with spatially variant optical properties. Realistic rendering of such objects involves expensive light transport computations inside the object volume. As a result, it has not been possible to quickly and realistically render a translucent object when it undergoes fracturing [2], [3] and cutting operations [4], [5] despite the fact that fracturing and cutting are important in many applications including games and films [6]. The main challenge lies in the expensive light transport computations that need to be repeated for the fractured pieces and cutouts. In this work, we introduce *TransCut*, a technique for accurately and interactively rendering heterogeneous translucent objects undergoing fracturing and cutting operations.

While existing algorithms for heterogeneous translucent materials offer fast [1] or even real-time [7] rendering performance, none of them can efficiently handle dynamic object fracturing and cutting operations. The finite element method of Arbree et al. [1] produces excellent visual results but is unsuitable for interactive applications due to the high cost of numerical integration. In the real-time finite difference method of Wang et al. [7], the non-regular meshes that often result from cutting or fracturing operations lead to much degradation in rendering quality. Moreover, none of these methods provide an efficient way to support multi-resolution processing when the object topology changes, a frequent occurrence throughout the dynamic fracturing and cutting process.

To efficiently render dynamic translucent objects with high visual quality, *TransCut* incorporates two technical contributions. One is a new method to solve the diffusion equation (DE) to render subsurface scattering effects in translucent objects with heterogeneous scattering materials. The solver is derived from a direct discretization of the divergence operator based on finite elements/volumes. While producing results comparable to the state-of-art method of Arbree et al. [1], the discrete divergence based solver moreover takes advantage of analytic formulas for linear finite elements to efficiently construct the linear system matrix for solving the diffusion equation. In this way, it avoids the time-consuming numerical integration used in Arbree et al. [1] and obtains significantly faster performance.

To further accelerate our DE solver, we also present a multi-resolution algorithm that efficiently adapts to the changes in topological structure of dynamic objects. The complete multi-resolution DE solver is highly parallel and easily implemented on the GPU for interactive rendering. With its performance and visual quality, *TransCut* provides users with a new and intuitive form of interaction with translucent objects.

The most important contribution of this work is perhaps the introduction of a novel visual effect: realistic and interactive rendering of translucent objects undergoing fracturing and cutting operations. This is an important effect that cannot be achieved by existing rendering methods. As demonstrated in the accompanying video, our technique enables a natural way to understand, design and interact with heterogeneous translucent materials.

(a) (b) (c) (d) (e)

Fig. 1. TransCut allows users to cut out portions of a heterogeneous translucent object via strokes (a) to examine the interior of the volume (b) and to edit material properties through painting on cross sections (c), all at interactive frame rates. It also supports interactive rendering of a translucent object fracturing (d,e).

2 Related Work

Subsurface scattering in translucent objects can be directly simulated from physical principles by Monte Carlo path tracing [8], [9], [10] or photon mapping [11] with offline computation. Considerable speed-ups have been obtained for the case of homogeneous materials by employing the diffusion approximation in a bidirectional surface scattering reflectance distribution function (BSSRDF) [12]. This approximation has been extended to multi-layered translucent materials [13], [14] and modified to be accurate for highly absorbing materials and near the point of illumination [15]. Runtime rendering has also been accelerated with precomputed subsurface light transport [16], [17], but the use of precomputation effectively limits these techniques to fixed geometries or scattering properties, since the precomputed data are too expensive to update at runtime.

Multiple scattering effects have long been modeled in participating media by the diffusion equation [18]. The diffusion equation was introduced to computer graphics by Stam [19] for the rendering of smoke, and was later applied to subsurface scattering in homogeneous volumes [20].

The first use of the diffusion equation for rendering general heterogeneous materials was presented by Wang et al. [21]. They numerically solved the diffusion equation using the finite difference (FD) method formulated over a quasi-regular 6-connected structure that models the volume. While this method provides real-time rendering performance, it requires manual construction of the grid structure and is limited to objects with simple geometry. More recently, Wang et al. [7] extended the FD solver to tetrahedral meshes that can be automatically constructed and that well represent complex geometry. However, the quality of the FD solution highly depends on mesh uniformity. To reduce the approximation errors of their method, they need a mesh with nearly equilateral tetrahedra of regular size. This is very difficult to guarantee in a dynamic process like object cutting. Also, dynamic triangulation of the new surface nodes is too expensive for runtime processing.

Arbree et al. [1] solved the diffusion equation using the finite element method (FEM). They show the FEM approach to be more accurate than the FD solver of Wang et al. [21]and of comparable quality to path tracing. To assemble the linear system of the finite element solver, a numerical integration is performed for each pair of vertices in every tetrahedron. Though accurate, this FEM approach does not allow for interactive rendering due to the high cost of the numeric integrations. The finite element method was also used to solve the diffusion equation in optical tomography [22], but only for estimating the scattering properties of a 2D homogeneous disk. We solve the diffusion equation over a tetrahedral mesh by directly computing the discrete divergences and constructing the DE matrix using analytic formulas derived for linear finite elements. The resulting solver is able to produce high-quality results comparable to the solver of Arbree et al. [1] but at substantially higher speeds. We admit that our solver is based on standard results from the areas of FEM and discrete differential geometry [23], [24], [25]. However, combining these results and applying them to rendering, in particular solving the diffusion equation for rendering translucent materials, is novel and has never been tried before.

Like previous real-time techniques [21], [7], we accelerate our diffusion equation solver using multi-resolution methods. Unlike these techniques which require either a regular grid structure or a precomputed hierarchy of progressively coarser tetrahedral meshes, we need to construct a hierarchy at runtime because the mesh topology may change frequently in our application. Shi et al. [26] presented a fast multigrid algorithm for interactive mesh deformation, but is unsuitable for our application due to the different nature of the linear system for the diffusion equation (see the detailed analysis in Section 4).

Instead of solving the diffusion equation, an alternative way to simulate multiple scattering in participating media is to use Lattice Boltzmann methods to trace the photon transport on a discrete grid [27]. To correctly handle the light transport near detailed boundaries of translucent objects, high resolution

grids are needed, though makes the simulation difficult to fit into limited GPU memory even for relatively simple objects [28]. Recently, Bernabei et al. [29] proposed to overcome the GPU memory limitation by localizing the computation of Lattice Boltzmann lighting. However, this method is not efficient in modeling the diffusion process over relatively long distances. To correctly capture the smoothly varying multiple scattered radiance in regions with large changes of scattering properties, a large number of simulation steps are needed and interactive rendering performance may not be guaranteed.

3 Discrete Diffusion Equation Solver for Tetrahedral Meshes

Let the heterogeneous translucent volume be represented by a tetrahedral mesh defined by the vertex set $\{x_1, x_2, ...\}$. We assume the optical properties, namely the absorption coefficient μ and the diffusion coefficient κ, to be piecewise constant functions, i.e., constant within each individual tetrahedron.

The discrete diffusion equation is used to solve for the radiant flux $\phi(x_i)$ at each vertex. We employ a piecewise linear representation of radiant flux where $\phi(x)$ is a linear function of point x within each tetrahedron. In the following, we present a brief review of the diffusion equation and describe our discrete solver.

3.1 Diffusion Equation Overview

Given an object Ω with boundary $\partial\Omega$ and heterogeneous scattering properties given by the absorption coefficient $\mu(x)$ and reduced scattering coefficient $\sigma'_s(x)$ [12], the radiant flux $\phi(x)$ within this object is defined by the diffusion equation:

$$\nabla \cdot (\kappa(x)\nabla\phi(x)) - \mu(x)\phi(x) = 0, x \in \Omega, \quad (1)$$

where $\nabla\cdot$ is the divergence operator, ∇ is the gradient operator, and $\kappa(x) = [3(\mu(x)+\sigma'_s(x))]^{-1}$. The diffusive source boundary condition [1] is given by

$$\phi(x) + 2A\kappa(x)(\nabla\phi(x)\cdot n(x)) = \frac{4}{1-F_{dr}}q(x), x \in \partial\Omega, \quad (2)$$

where $q(x) = \int_{2\pi} L_i(x,\omega_i)(n(x)\cdot\omega_i)F_t(\omega_i)d\omega_i$ is the diffused incoming light at surface point x, and F_{dr} and F_t are the diffuse Fresnel reflectance and Fresnel transmittance, respectively [12]. $n(x)$ is the surface normal at point x, and $A = (1+F_{dr})/(1-F_{dr})$.

The outgoing radiance on the boundary can be computed by the query function derived by Arbree et al. [1]:

$$L_o(x,\omega_o) = \frac{F_t(\omega_o)}{4\pi}\left[\left(1+\frac{1}{A}\right)\phi(x) - \frac{4}{1+F_{dr}}q(x)\right], \quad (3)$$

where ω_o is the outgoing direction.

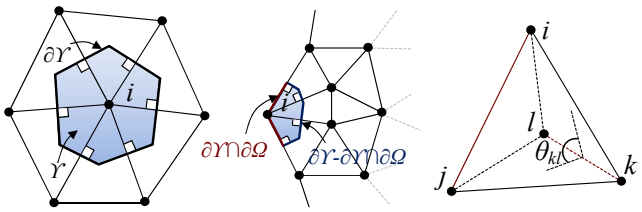


Fig. 2. Illustration of the Voronoi region and its boundary for an interior vertex (left) and a boundary vertex (middle), and the notation used in computing the weight of discrete divergence (right).

3.2 Our Discrete Solver

We solve the diffusion equation by directly discretizing the gradient and divergence operators in Eq. (1).

Since $\phi(x)$ is a piecewise linear function, $\nabla\phi(x)$ is constant within each tetrahedron t, and can easily be proven to have the value:

$$\nabla\phi_t = \sum_{j=1}^{4}\frac{S_j\phi(x_j)}{3V_t}n_j, \quad (4)$$

where V_t denotes the volume of t, and x_j are the vertices of t. S_j, n_j are respectively the area and normal of the face opposite to vertex x_j in the tetrahedron t, with n_j pointing toward x_j. Since $\kappa(x)$ is constant within each individual tetrahedron, $\kappa(x)\nabla\phi(x)$ is also a piecewise constant vector field defined over the tetrahedral mesh.

In the continuous case, the divergence of a vector field ξ at a point p is defined to be:

$$\nabla\cdot(\xi(p)) = \lim_{\Psi\to\{p\}}\int_{\partial\Psi}\frac{\xi(s)\cdot n(s)}{|\Psi|}ds, \quad (5)$$

where Ψ is a 3D region surrounding p, with boundary $\partial\Psi$. $|\Psi|$ is the volume of Ψ, and $n(s)$ is the outward unit normal of the boundary surface.

The discrete divergence (denoted as Div) of a piecewise constant vector field at a vertex i is defined to be the spatial average of the integral around x_i. By using finite element/volume methods [23], [24], it can be efficiently computed as the spatial average of the integral on the boundary faces of the Voronoi region surrounding x_i:

$$(\nabla\cdot(\kappa\nabla\phi))(x_i) \approx (\mathrm{Div}(\kappa\nabla\phi))(x_i) \quad (6)$$

$$\equiv \frac{1}{|\Upsilon|}\int_{\partial\Upsilon}\kappa(s)(\nabla\phi(s)\cdot n(s))ds, \quad (7)$$

where Υ is the Voronoi region surrounding x_i, with boundary $\partial\Upsilon$. Note that $\partial\Upsilon$ is composed of a set of Voronoi faces. $|\Upsilon|$ is the volume of Υ, which can be precisely calculated for each vertex. In practice $V_i/4$ provides a good estimation, where V_i is the sum of the volumes of all tetrahedra that share vertex i.

Interior Vertices: For each interior vertex i, the Voronoi region and Voronoi faces surrounding it lie completely within the tetrahedral mesh (see Fig. 2(left) for a 2D illustration). The integral in Eq. (7) can be easily computed by making use of the one-ring neighborhood information of i. Following [25], we have

$$(\mathrm{Div}(\kappa\nabla\phi))(x_i) = \frac{1}{|\Upsilon|}\sum_{t\in N(i)} -V_t\nabla\beta_{it}\cdot(\kappa_t\nabla\phi_t), \quad (8)$$

where $N(i)$ is the one-ring neighborhood of vertex i, and β_{it} is a piecewise-linear basis function with a value of 1 at vertex x_i and 0 at other vertices inside tetrahedron t. According to the proof in the Appendix, this discrete divergence can be analytically computed as

$$(\mathrm{Div}(\kappa\nabla\phi))(x_i) = \frac{1}{|\Upsilon|}\sum_{j\in N(i)} w_{ij}(\phi(x_i)-\phi(x_j)), \quad (9)$$

where w_{ij} is calculated as

$$w_{ij} = -\sum_{t=\{i,j,k,l\}} \frac{1}{6}\kappa_t|kl|\cot\theta_{kl}, \quad (10)$$

where $t=\{i,j,k,l\}$ denotes all tetrahedra containing edge $\{x_i,x_j\}$, with x_k, x_l as the other two vertices. θ_{kl} is the dihedral angle opposite the edge $\{x_i,x_j\}$ in t, and $|kl|$ is the edge length of $\{x_k,x_l\}$ (see Fig. 2(right)).

Boundary Vertices: For each boundary vertex i, some of the Voronoi faces lie on the boundary of the tetrahedral mesh (see Fig. 2(middle) for a 2D illustration). We divide the integral in Eq. (7) into two parts,

$$(\mathrm{Div}(\kappa\nabla\phi))(x_i) = I_1 + I_2,$$

where

$$I_1 = \frac{1}{|\Upsilon|}\int_{\partial\Upsilon-\partial\Upsilon\cap\partial\Omega}\kappa(s)(\nabla\phi(s)\cdot n(s))ds,$$

$$I_2 = \frac{1}{|\Upsilon|}\int_{\partial\Upsilon\cap\partial\Omega}\kappa(s)(\nabla\phi(s)\cdot n(s))ds.$$

I_1 corresponds to the integral on the Voronoi faces located inside the mesh, and can be analytically computed using Eq. (9) and Eq. (10).

I_2 corresponds to the integral on the Voronoi faces located on the mesh boundary and cannot be directly computed. By taking the integral on both sides of the boundary condition (Eq. (2)), it can be computed as:

$$I_2 = \frac{1}{2A|\Upsilon|}\int_{\partial\Upsilon\cap\partial\Omega}(-\phi(s)+\frac{4}{1-F_{dr}}q(s))ds.$$

Note that the above integral is actually computed on the Voronoi cell surrounding vertex x_i on the boundary mesh of the tetrahedral mesh. By taking the average value of the integrand as the function value at x_i, I_2 can be approximated as

$$I_2 \approx \frac{1}{2A|\Upsilon|}(-\phi(x_i)+\frac{4}{1-F_{dr}}q(x_i))|\partial\Upsilon\cap\partial\Omega|, \quad (11)$$

where $|\partial\Upsilon\cap\partial\Omega|$ is the area of the Voronoi cell surrounding vertex x_i, which can be approximated as $S_i/3$. S_i is the sum of the areas of all the boundary triangles sharing x_i.

Linear System: Let $\Phi=(\phi(x_1),\phi(x_2),...)^T$ be the vector of unknowns to be solved. We can construct a linear system $\mathbf{M}\Phi=\mathbf{b}$ by multiplying both sides of Eq. (9) and Eq. (11) by $-V_i$ (i.e., $-4|\Upsilon|$) and rearranging terms. The vector $\mathbf{b}$ has the following formula:

$$\mathbf{b}_i = \begin{cases} 0, & \text{if } i \text{ is an interior vertex,} \\ \frac{8S_i}{3A(1-F_{dr})}q(x_i), & \text{if } i \text{ is a boundary vertex.} \end{cases} \quad (12)$$

$\mathbf{M}$ is a sparse matrix. For each row i, only the diagonal element and the elements corresponding to x_i's one-ring neighboring vertices are non-zero. The diagonal element is computed as

$$\mathbf{M}_{ii} = \begin{cases} -4\sum_{j\in N(i)} w_{ij} + V_i\mu(x_i), & \text{if } i \text{ is interior,} \\ -4\sum_{j\in N(i)} w_{ij} + V_i\mu(x_i) + \frac{2S_i}{3A}, & \text{if } i \text{ is boundary,} \end{cases} \quad (13)$$

and the other non-zero elements have a simple form:

$$\mathbf{M}_{ij} = 4w_{ij}. \quad (14)$$

$\mathbf{M}$ is easily proved to be symmetric and positive definite. The linear system can thus be efficiently solved using conjugate gradient algorithms.

Discussion: Our linear system for the diffusion equation employs two approximations, namely the discrete divergence and the approximated integral in Eq. (11). These approximations are less significant than those required in the FDM of [7], particularly their need for nearly equilateral tetrahedra of regular size. The resulting differences in rendering quality are exemplified in Sec. 6. The algorithm of [1] obtains the FEM solution without approximation and produces highly accurate diffusion results. Our approximations are generally close to the FEM values except for tetrahedra with very poor aspect ratios, where the calculation of the weight in Eq. (10) may have numerical problems. For high quality, we need to avoid tetrahedra with very poor aspect ratios during user interaction, as described in Sec. 5. In Sec. 6 and the supplementary material, the results of our solver are shown to closely resemble the FEM solutions of [1].

Our discrete solver adopts the boundary condition derived by Arbree et al. [1], as we directly compare our results with those in [1]. Note that it is possible to use the boundary condition derived from the recent quantized-diffusion (QD) model [15], which will not affect the performance of our solver. The QD model is more accurate for high-frequency illumination and highly absorbing materials. In both cases, very dense tetrahedral meshes with tetrahedron sizes smaller than the mean free path of light are required to reflect the increased accuracy of the QD model. It is thus expensive for methods based on per-vertex shading, such as ours and those of [1], [7], to take

full advantage of the QD model and render high-frequency details of comparable size to the mean free path.

4 Parallel Multi-Resolution Solver

To represent complex shapes and inhomogeneous materials, we need dense tetrahedral meshes that may contain hundreds of thousands of vertices and tetrahedra. Directly solving the discrete diffusion equation as described in the last section is still slow for such meshes. In this section, we describe a GPU-based parallel multi-resolution algorithm to accelerate the discrete DE solver.

The algorithm constructs a hierarchy of progressively coarser representations for the tetrahedral mesh, solves a linear system at the coarsest level, and uses this solution as the starting point for iteratively solving the finer levels. Suppose $G^0 = (P^0, E^0)$ is a graph formed by the vertices and edges of the original tetrahedral mesh. To build the hierarchy $G^l = (P^l, E^l), l = 1, 2, \ldots$, we determine the vertices using a simple independent set procedure designed to take advantage of GPU parallelization, and add edges in a manner similar to [26]. Note that we only need a hierarchy of progressively coarser *graph* representations, not a hierarchy of high-quality *tetrahedral meshes* as in [7], which is computationally too expensive to construct on the fly.

4.1 Hierarchy Construction

Suppose $G^0 = (P^0, E^0)$ is a graph formed by the vertices and edges of the original tetrahedral mesh. We build a coarser graph G^{l+1} from G^l using a two-step approach. The first step computes an independent vertex set of P^l, in which no two vertices are adjacent. The second step enhances the obtained vertex set and constructs the edges connecting the vertices in the set.

Independent Vertex Set: Our independent set algorithm initializes P^{l+1} with NULL and copies P^l to an intermediate set Q^{l+1}, then iteratively removes vertices from Q^{l+1} and adds them to P^{l+1}. It first generates a random number for each vertex $i \in Q^{l+1}$ using the hashing algorithm described in [30], and computes a 32-bit code h_i by packing the random number and the index i with the random number as the higher-order bits. During each iteration, for each vertex $i \in Q^{l+1}$, let $\Gamma(i) = N^l(i) \cap Q^{l+1}$ be the set of 1-ring neighboring vertices of i at level l that are also in Q^{l+1}. If the code value h_i of i is smaller than the code values of all vertices in $\Gamma(i)$, we add i to P^{l+1}, and remove i and $\Gamma(i)$ from Q^{l+1}. The process is repeated until Q^{l+1} becomes empty. Note that the random number is introduced to break potential patterns in the original vertex indices, and the original indices are used to ensure a unique ordering of vertices.

It is easy to prove that for every vertex i in $P^l - P^{l+1}$, there is at least one vertex in $N^l(i)$ that is also in P^{l+1}.

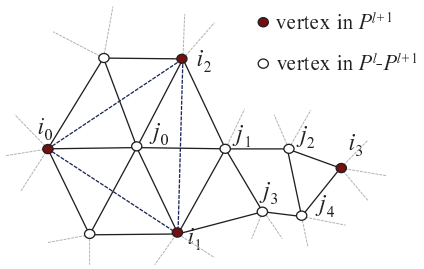


Fig. 3. Depiction of hierarchy construction.

This ensures that the interpolation from a coarser level, as will be described below, can be correctly performed. Each pair of vertices selected into P^{l+1} are not adjacent in G^l. This guarantees that this procedure can be easily parallelized and race conditions are avoided – concurrent threads are guaranteed to make consistent decisions on selecting or dropping a vertex.

Edge Set: As in Shi et al. [26], the edge set E^{l+1} at level $l+1$ can be constructed by adding an edge between two vertices in P^{l+1} if they are in the 2-ring neighborhood of each other in G^l. This can be easily performed in the following way. For each vertex $i \in P^l - P^{l+1}$, processed in parallel, if any two vertices in i's 1-ring neighborhood are retained in P^{l+1}, we add an edge E^{l+1} that connects them. Fig. 3 gives a simple example. The three vertices i_0, i_1, i_2 in P^{l+1} that are 1-ring neighbors of j_0 are connected by edges in G^{l+1}. Redundant edges are removed by first sorting all added edges and then removing the edges that are identical to the preceding one.

Enhancing the Graph: A potential problem with the above edge formulation is that vertices in the 3-ring neighborhood of each other in G^l may be retained in G^{l+1} but become unconnected. Fig. 3 shows such an example. Though the vertices i_1, i_3 are 3-ring neighbors at level l, their connectivity is lost in G^{l+1} because all their 1-ring neighbors j_1, j_2, j_3, j_4 are dropped at level $l+1$.

This problem can be solved by either directly adding an edge to connect i_1 and i_3, or inserting some of the vertices j_1, j_2, j_3, j_4 into P^{l+1} to enhance the graph. We choose to insert vertices since it ensures that only 2-ring neighbors need to be considered when interpolating solutions from a coarse level to a finer level.

For each edge in E^l whose two vertices i, j are in $P^l - P^{l+1}$, if $N^l(i) \cap N^l(j) \cap P^{l+1}$ is empty, we mark i and j since this edge connection will not be reflected at level $l+1$. All marked vertices then form a new candidate set Q^{l+1}, and the above independent set algorithm is used to select and add new vertices to P^{l+1}. This process is repeated until no more vertices are marked.

Note that by enhancing the graph, P^{l+1} is not necessarily an independent set of P^l anymore. Besides adding an edge to E^{l+1} to connect two vertices in

P^{l+1} that are in the 2-ring neighborhood of each other in G^l, if any two vertices in P^{l+1} are connected by an edge in G^l, the edge is also added to the edge set E^{l+1}.

4.2 Constructing and Solving the Linear System

We then construct the linear system at each level and interpolate solutions from coarser to finer levels. Suppose vertex i has the following linear equation at level l:

$$\mathbf{M}_{ii}^l \phi^l(x_i) + \sum_{j \in N^l(i)} \mathbf{M}_{ij}^l \phi^l(x_j) = \mathbf{b}_i^l. \quad (15)$$

Note that the linear system at level 0 is given by Eqs. (12)-(14), i.e., $\mathbf{M}^0 = \mathbf{M}$, $\mathbf{b}^0 = \mathbf{b}$.

We make use of the following formula to interpolate the solution from level $l+1$ to l:

$$\phi^l(x_i) = \begin{cases} \phi^{l+1}(x_i), & \text{if } i \in P^{l+1}, \\ \sum_{j \in R^l(i)} \frac{\lambda_{ij} \phi^{l+1}(x_j)}{\lambda_i}, & \text{if } i \in P^l - P^{l+1}, \end{cases} \quad (16)$$

where $R^l(i) = N^l(i) \cap P^{l+1}$ is the set of i's 1-ring neighbors at level l that are retained at level $l+1$. λ_{ij} are the interpolation weights, and $\lambda_i = \sum_{j \in R^l(i)} \lambda_{ij}$. Currently we use the reciprocal of edge length as weights, i.e., $\lambda_{ij} = 1/|x_i - x_j|$.

The implication of the above interpolation is as follows. If i is retained at level $l+1$, its value at level $l+1$ is directly used as its initial value at level l. Otherwise, i's initial value at level l is computed as the weighted average of its 1-ring neighbors at level l that are retained at level $l+1$. According to our graph construction process, if i is not in P^{l+1}, at least one of its 1-ring neighbors is in P^{l+1}, which ensures that $R^l(i)$ is not empty and the interpolation can be conducted.

By substituting Eq. (16) into Eq. (15), we obtain the equation for vertex i at level $l+1$:

$$\mathbf{M}_{ii}^l \phi^{l+1}(x_i) + \sum_{j \in R^l(i)} \mathbf{M}_{ij}^l \phi^{l+1}(x_j) + \sum_{j \in K^l(i)} \sum_{k \in R^l(j)} \frac{\mathbf{M}_{ij}^l \lambda_{jk} \phi^{l+1}(x_k)}{\lambda_j} = \mathbf{b}_i^l, \quad (17)$$

where $K^l(i) = N^l(i) - R^l(i)$ is the set of i's 1-ring neighbors at level l that are not retained at level $l+1$.

The above equation (Eq. (17)) can be reformulated into the same form as Eq. (15), with

$$\mathbf{b}_i^{l+1} = \mathbf{b}_i^l,$$

$$\mathbf{M}_{ii}^{l+1} = \mathbf{M}_{ii}^l + \sum_{j \in K^l(i)} \frac{\mathbf{M}_{ij}^l \lambda_{ji}}{\lambda_j},$$

$$\mathbf{M}_{ij}^{l+1} = \begin{cases} \sum_{k \in K^l(i) \cap N^l(j)} \frac{\mathbf{M}_{ik}^l \lambda_{kj}}{\lambda_k}, & \text{if } j \in N^{l+1}(i) - R^l(i), \\ \mathbf{M}_{ij}^l + \sum_{k \in K^l(i) \cap N^l(j)} \frac{\mathbf{M}_{ik}^l \lambda_{kj}}{\lambda_k}, & \text{if } j \in R^l(i). \end{cases}$$

The matrices constructed at coarse levels are no longer symmetric. We use biconjugate gradient methods to solve the linear systems at all levels except at level 0. Note that we cannot guarantee that matrices at coarse levels are invertible. In cases where the solution at a coarse level does not converge after several iterations, we could move to the next finer level. However, this problem was never encountered in any of our experiments.

We note that the multigrid solver proposed by Shi et al. [26] is unsuitable for our task. The reason is that the non-diagonal elements ($\mathbf{M}_{ij}$) in our matrix at the finest level (Eq. (10)) may take both negative and positive values. If we directly construct the interpolation (or prolongation) operator as in [26] (Eq. (14) in their paper), the denominator may be very close to zero or even equal to zero, resulting in extremely large elements in matrices at coarse levels. [26] does not have this problem because their elements (w_{ij}) are always equal to 1. While it is possible to use our current interpolation operator to develop a full multigrid solver, we found that our simple multi-resolution approach works well for all of our test data.

5 Implementation Details

Cutting Interaction: We currently allow users to cut translucent objects into pieces. The cutting operation is performed on the CPU using an approach similar to Nienhuys and van der Stappen [31]. After the user draws a stroke on the screen, all tetrahedra whose four vertices are located on the left side of the stroke are deleted. If the 1-ring tetrahedra around a vertex are all deleted, the vertex is also deleted. The remaining vertices and tetrahedra form the new tetrahedral mesh. To facilitate the painting interaction described below, we record the original tetrahedron ID for each tetrahedron of the new mesh. Then we move the remaining vertices on the left side of the stroke toward the cross-section as follows. Suppose x_i is a vertex to be moved. We compute x_i', the uniformly weighted average of x_i's 1-ring neighboring vertices. x_i is moved toward x_i' by a small step size, i.e., $x_i \leftarrow x_i + \rho(x_i' - x_i)$, where ρ is set to 0.1 in our implementation, unless it would move to the right side of the stroke or result in degenerated tetrahedra among x_i's 1-ring neighbors. The above process is performed for all movable vertices and is repeated for a user-specified number of iterations or until no vertices can be moved.

Painting Interaction: We allow users to directly paint on the surface of the cutout tetrahedral mesh to modify the material properties of tetrahedra in the original mesh. For each stroke painted by the user, we compute the 3D positions of its points from screen coordinates and the depth of their projections onto the surface. Then for each tetrahedron of the original mesh, the distance between its center and each 3D stroke point is evaluated. The painting operation is applied to the tetrahedron according to an influence weight determined by the distance of the closest point and a user-specified Gauss function. By adjusting the Gauss function parameter, the user can control the amount of stroke diffusion through the original

(a) ours (b) Arbree et al. [1]

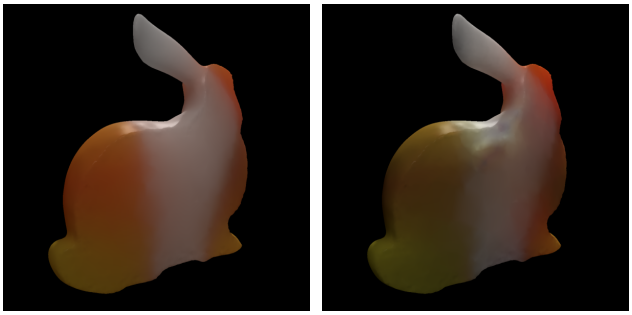
(c) reference (d) 4× diff.

Fig. 4. Comparison with [1], including a photon mapping reference and a plot of the relative radiant flux error of our method with respect to [1]. The model is lit by two directional light sources, one from the top-front and the other from the back. The inhomogeneous scattering material is generated according to the description in [1].

TABLE 1
Test data statistics.

	#Tet †	#Level	T_b(ms)	T_m(ms)	T_s(ms)
Bunny	511,019	3	72	26	49
Tweety	1,011,863	4	213	67	57
Horse	1,020,554	4	218	65	72
Feline	1,024,674	4	219	67	121
Gargoyle	687,724	3	166	47	49

† #Tet is the number of tetrahedra, #Level is the number of multi-resolution hierarchy levels, T_b is the time for building the vertex neighborhood and multi-resolution hierarchy, T_m is the time for constructing the DE matrix, and T_s is the time for solving the DE (initialized with zeros). All timings are in milliseconds.



(a) ours (b) Wang et al. [7]

Fig. 5. Comparison with [7], on a tetrahedral mesh cutout of the Stanford bunny. The model is lit mostly from behind, and also with some frontal lighting.

mesh for volumetric editing. Unlike cutting operations which recompute the entire linear system for the resulting mesh, painting operations often change material properties locally and require updating of only a few rows in the diffusion equation matrix. Specifically, we only update the rows corresponding to the vertices of tetrahedra that are affected by the painting operation.

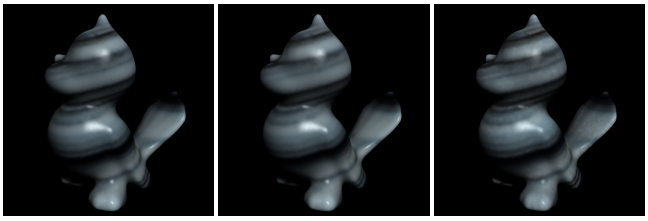
GPU Implementation: The entire algorithm except for the mesh cutting procedure is implemented on the GPU using NVIDIA CUDA 3.2. We use the newly released cusparse library in our solver for sparse matrix multiplication with vectors. The parallel primitives scan, sort and compact [32] are used in computing the 1-ring neighborhood information and the hierarchy representation required by the multi-resolution algorithm.

6 Experimental Results

We implemented our algorithm on a PC with a 2.3GHz Intel quad-core CPU and an NVIDIA GeForce GTX 480 graphics card. All images are rendered at 640×480 with $16\times$ anti-aliasing. Since the majority of computations are conducted on vertices/tetrahedra, anti-aliasing is obtained nearly for free. Surface shading is added to all results using the Cook-Torrance BRDF model.

Test Data Statistics: The statistics of the test data used in our paper are listed in Table 1. All tetrahedral meshes are generated using Tetgen [33]. Interactive rendering performance is achieved for all models, with tetrahedra counts ranging from 500K to 1M. Cutting is the most expensive operation since it is executed on the CPU and uses an iterative process to move vertices to cross sections. It also requires rebuilding of the vertex neighborhood, multi-resolution hierarchy and diffusion equation matrices. Nevertheless, it can be finished in less than three seconds for all examples in the paper and accompanying video. Painting and relighting can be done in real time on the GPU (over 10 fps) since they do not change the mesh and thus do not need geometric operations or matrix rebuilding. They can also initialize the current frame with the solution of the previous frame to expedite convergence.

Comparisons: Fig. 4 shows a comparison between our algorithm, implemented on the GPU, and that of [1], implemented on a 4-core CPU. The results are visually identical, with a maximum relative error of about 2%. A $4\times$ magnified difference image between our method and the photon mapping reference is also shown. Visible differences can be observed near the object silhouettes since the diffusion approxima-



(a) (b) (c)

Fig. 6. Comparing our algorithm (a) with Arbree et al.'s [1] (b) and Wang et al.'s [7] (c).

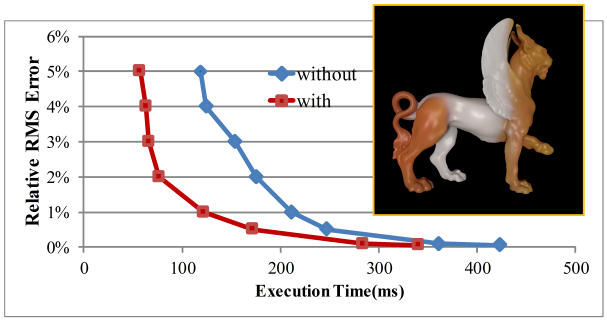


Fig. 7. Multi-resolution acceleration.

Fig. 8. Cutting a translucent object through gestures.

tion is less accurate in these optically thinner parts when lit from behind. Nonetheless, our result is still visually similar to the photon mapping reference. In Fig. 5, we also compare our method to the state-of-art interactive technique for rendering heterogenous translucent objects [7]. Their method is based on finite differences and requires a high quality tetrahedral mesh. Rendering the cut mesh with their method results in severe artifacts that are especially visible at the cut interface. Though artifacts could be reduced with careful optimization of mesh geometry, this is not practical for interactive applications. Even for the original mesh before cutting, their method still generates visible artifacts since the mesh generated by Tetgen is not well optimized. As shown in Fig. 6, the result of [7] (Fig. 6(c)) suffers from flux discontinuity, which is especially obvious in some regions where scattering is significant. By contrast, our solver always generates high-quality results comparable to those produced by Arbree et al.'s solver. Fig. 11 shows more comparisons with [1] using different tetrahedral meshes and materials.

According to our experiments, our solver achieves performance similar to Wang et al.'s method on the GPU. Their matrix construction is much simpler than ours, but the size of their matrix (determined by the number of tetrahedra plus the number of boundary triangles) is much larger than ours (determined by the number of vertices). In our implementation of Arbree et al.'s method, construction of the diffusion equation matrix is parallelized on our 4-core CPU and takes about 24 seconds to build the matrix. By contrast, the matrix construction in our method takes only about 1 second on the 4-core CPU. Note that once we construct the matrix using Arbree et al.'s method, it can be solved using our multi-resolution solver so no comparison of rendering performance is given.

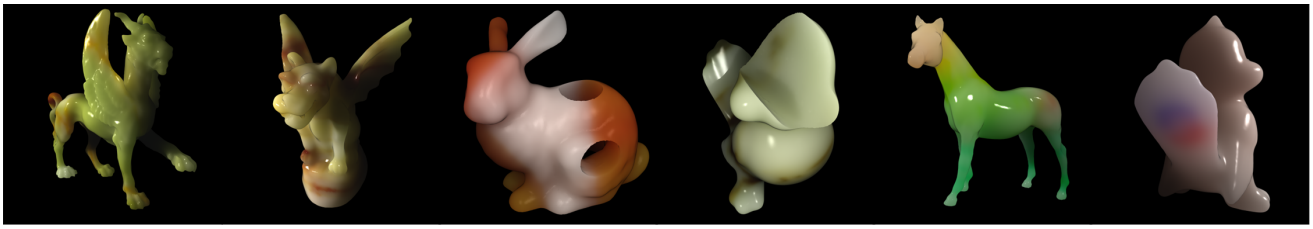
As reported in Table 1, our parallel hierarchy construction method typically runs at less than 220ms on the GPU for the examples in the paper. This performance is about 2× faster than the CPU method of Shi et al. [26]. Note that hierarchy construction is not the bottleneck of our multi-resolution algorithm, where the cutting operation requires approximately two seconds. To minimize time-consuming data transfer between CPU and GPU, we perform the hierarchy construction on the GPU where the other algorithm components (e.g., build mesh connectivity, construct matrix and direct rendering) are computed.

In Fig. 7, we show a plot of the relative errors versus execution time of our method with and without the multi-resolution method. With relative errors below 1% the rendering results are visually indistinguishable. At a relative error of 1%, the acceleration ratio achieved by the multi-resolution method is 73%. This proves that our parallel multi-resolution solver can achieve speedups comparable to Wang et al.'s multi-resolution acceleration [7]. Moreover, while Wang et al. need to precompute a hierarchy of high-quality tetrahedral meshes, which is unaffordable at runtime, our approach builds the multi-resolution hierarchy from scratch at interactive rates. This makes our approach very suitable for geometry interaction such as cutting.

Interaction: The results of various interactions such as cutting, relighting, and material editing are exhibited in Fig. 1 and Fig. 10. High frame rates and realistic appearance are obtained for these complex objects with many tetrahedra. We also integrated this technique with the Microsoft Kinect sensor system, to expand user interactivity with virtual translucent objects by enabling cutout through gestures. Fig. 8 shows two frames of a cutting interaction. For live action clips of the interaction system, please see the accompanying video.

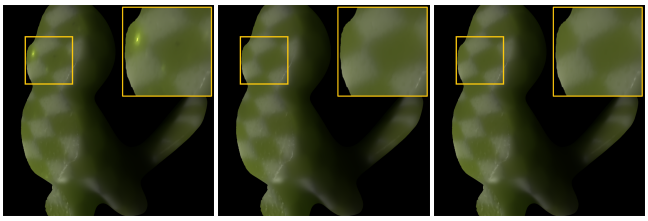
LIMITATIONS

Our technique has two main limitations. First, since cutout operations often expose the object interior, we need dense sampling throughout the object volume, resulting in large meshes. Also, we cannot make use of the adaptive mesh refinement employed in [1] due



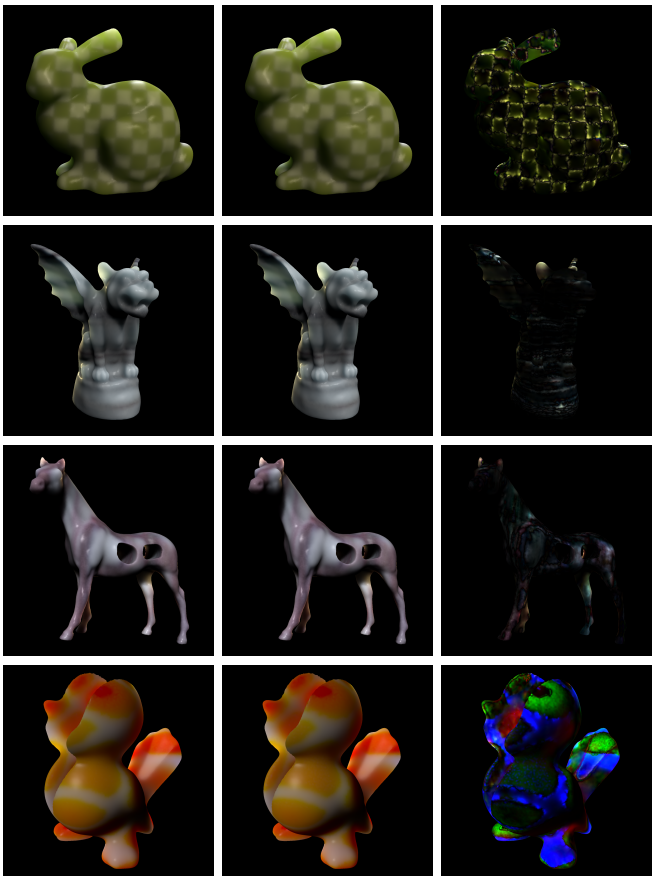
(a) (b) (c) (d) (e) (f)

Fig. 10. Additional interaction results. (a)(b) From relighting. (c)(d) From mesh cutting. (e)(f) From painting.



(a) (b) (c)

Fig. 9. (a) Illustration of numerical instability at tetrahedra with very poor aspect ratios. (b) Greater stability of numerical integration used in [1]. (c) Result by avoiding vertex moves that generate poorly-shaped tetrahedra.



(a) ours (b) Arbree et al. [1] (c) 32× diff.

Fig. 11. Additional comparisons with Arbree et al.'s method [1]. From top to bottom, the RMS errors are 3.5%, 2.4%, 4.9% and 8.4%, respectively.

to the high computational cost. Therefore, if the mean free path in a material approaches the size of tetrahedra, our algorithm cannot capture the subsurface details. Second, our discrete divergence computation needs to evaluate the cotangent of dihedral angles formed by the faces of tetrahedra (Eq. (10)). As we perform single precision computation on the GPU, tetrahedra with very poor aspect ratios may result in very large cotangent values and cause numerical problems, introducing artifacts in the final rendering. Fig. 9(a) shows such an example, where we force the vertices to move to the cross-sections without considering the shape quality of the resulting tetrahedra. The numerical integration method used in [1] is more stable in this case, and can generate artifact-free results even with single precision computation (Fig. 9(b)). We presently address this problem by forbidding any vertex to move if the move will result in poorly-shaped tetrahedra (Fig. 9(c)). In our current implementation, we define a tetrahedron whose minimal dihedral angle is less than 0.5 degree to be poorly-shaped. A side effect of this scheme is that in some situations the generated cross-sections may not smoothly conform to the user-specified cutting stroke.

One possible solution for adaptive mesh refinement is to use isosurface stuffing [34] for tetrahedra generation. As the tetrahedra can be created according to specific gradings, we would be able to distribute more tetrahedra in regions with more significant variation of scattering properties. Furthermore, as the graded tetrahedral background grid is created from an octree, local refinement should be easy with good runtime performance. On the other hand, cut interfaces need special treatment, since the original algorithm does not guarantee preservation of sharp edges or corners.

7 Conclusion

We presented a technique for realistic and interactive rendering of translucent objects undergoing fracturing and cutting operations, supported by a novel DE solver with multi-resolution acceleration. This technique offers users a natural way to interact with and edit translucent volumes. In future work, we plan to augment our user interface with more advanced

cutting and fracturing tools, including those that can be used to simulate a real-world carving process. We would also like to improve the scalability of our technique by investigating efficient and robust adaptive mesh refinement algorithms.

APPENDIX

Below, we derive the discrete divergence formula for each interior vertex i (Eq. (9) and Eq. (10)) from Eq. (8).

Let t be a tetrahedron in i's 1-ring neighborhood. Since β_{it} is a piecewise linear function, $\nabla\beta_{it}$ is constant and can be computed according to Eq. (4):

$$\nabla\beta_{it} = \frac{S_i}{3V_t}n_i.$$

Therefore, we have

$$\begin{aligned} V_t\nabla\beta_{it}\cdot(\kappa_t\nabla\phi_t) &= V_t\frac{S_i}{3V_t}n_i\cdot\sum_{j\in t}\kappa_t\frac{S_j\phi(x_j)}{3V_t}n_j \\ &= \sum_{j\in t}w_{ijt}\phi(x_j), \end{aligned}$$

where

$$w_{ijt} = \frac{\kappa_t S_i S_j}{9V_t}(n_i\cdot n_j).$$

In cases that $j\neq i$, with k,l denoting the other two vertices of t, we have

$$\begin{aligned} w_{ijt} &= \frac{-\kappa_t S_i S_j\cos\theta_{kl}}{9V_t} = \frac{-\kappa_t S_j\cos\theta_{kl}}{3h_i} \\ &= -\frac{1}{6}\kappa_t|kl|\cot\theta_{kl}, \end{aligned}$$

where h_i is height from vertex i to its opposite face.

In cases that $j=i$, we can easily prove that

$$w_{iit} = \frac{\kappa_t S_i S_i}{9V_t} = -(w_{ijt}+w_{ikt}+w_{ilt})$$

because $S_i = -(S_j(n_i\cdot n_j)+S_k(n_i\cdot n_k)+S_l(n_i\cdot n_l))$.

Based on the above derivations, we have

$$\begin{aligned} (\mathrm{Div}(\kappa\nabla\phi))(x_i) &= \frac{1}{|\Upsilon|}\sum_{t\in N(i)}-V_t\nabla\beta_{it}\cdot(\kappa_t\nabla\phi_t) \\ &= \frac{1}{|\Upsilon|}\sum_{j\in N(i)}w_{ij}(\phi(x_i)-\phi(x_j)), \end{aligned}$$

where

$$w_{ij} = \sum_{t=\{i,j,k,l\}}w_{ijt} = -\sum_{t=\{i,j,k,l\}}\frac{1}{6}\kappa_t|kl|\cot\theta_{kl}.$$

ACKNOWLEDGEMENTS

We wish to thank Tianyi Cui for his help on the Kinect demo. The work is partially supported by the NSF of China (No. 60825201 and No. 61003048).

REFERENCES

[1] A. Arbree, B. Walter, and K. Bala, "Heterogeneous subsurface scattering using the finite element method," *IEEE Comp. Graph. & Appl.*, vol. 17, no. 7, pp. 956–969, 2011.
[2] J. F. O'Brien and J. K. Hodgins, "Graphical modeling and animation of brittle fracture," in *ACM SIGGRAPH*, 1999, pp. 137–146.
[3] J. F. O'Brien, A. W. Bargteil, and J. K. Hodgins, "Graphical modeling and animation of ductile fracture," in *ACM SIGGRAPH*, 2002, pp. 291–294.
[4] S. Owada, F. Nielsen, M. Okabe, and T. Igarashi, "Volumetric illustration: Designing 3d models with internal textures," *ACM Trans. Graph.*, vol. 23, pp. 322–328, 2004.
[5] K. Takayama, O. Sorkine, A. Nealen, and T. Igarashi, "Volumetric modeling with diffusion surfaces," *ACM Trans. Graph.*, vol. 29, no. 6, pp. 180:1–180:8, 2010.
[6] E. G. Parker and J. F. O'Brien, "Real-time deformation and fracture in a game environment," in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Aug. 2009, pp. 156–166.
[7] Y. Wang, J. Wang, N. Holzschuch, K. Subr, J.-H. Yong, and B. Guo, "Real-time rendering of heterogeneous translucent objects with arbitrary shapes," *Computer Graphics Forum*, vol. 29, no. 2, pp. 497–506, 2010.
[8] J. Dorsey, A. Edelman, H. W. Jensen, J. Legakis, and H. K. Pedersen, "Modeling and rendering of weathered stone," in *ACM SIGGRAPH*, 1999, pp. 225–234.
[9] M. Pharr and P. Hanrahan, "Monte carlo evaluation of nonlinear scattering equations for subsurface reflection," in *ACM SIGGRAPH*, 2000, pp. 75–84.
[10] H. Li, F. Pellacini, and K. E. Torrance, "A hybrid monte carlo method for accurate and efficient subsurface scattering," in *Eurogr. Symposium on Rendering*, 2005, pp. 283–290.
[11] H. W. Jensen and P. H. Christensen, "Efficient simulation of light transport in scences with participating media using photon maps," in *ACM SIGGRAPH*, 1998, pp. 311–320.
[12] H. W. Jensen, S. R. Marschner, M. Levoy, and P. Hanrahan, "A practical model for subsurface light transport," in *ACM SIGGRAPH*, 2001, pp. 511–518.
[13] C. Donner and H. W. Jensen, "Light diffusion in multi-layered translucent materials," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 1032–1039, 2005.
[14] C. Donner, T. Weyrich, E. d'Eon, R. Ramamoorthi, and S. Rusinkiewicz, "A layered, heterogeneous reflectance model for acquiring and rendering human skin," *ACM Trans. Graph.*, vol. 27, no. 5, pp. 140:1–140:12, 2008.
[15] E. d'Eon and G. Irving, "A quantized-diffusion model for rendering translucent materials," *ACM Trans. Graph.*, vol. 30, no. 4, pp. 56:1–56:14, 2011.
[16] X. Hao and A. Varshney, "Real-time rendering of translucent meshes," *ACM Trans. Graph.*, vol. 23, no. 2, pp. 120–142, 2004.
[17] R. Wang, J. Tran, and D. Luebke, "All-frequency interactive relighting of translucent objects with single and multiple scattering," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 1202–1207, 2005.
[18] A. Ishimaru, *Wave Propagation and Scattering in Random Media*. Academic Press, 1978.
[19] J. Stam, "Multiple scattering as a diffusion process," in *Eurogr. Workshop on Rendering*, 1995, pp. 41–50.
[20] T. Haber, T. Mertens, P. Bekaert, and F. Van Reeth, "A computational approach to simulate light diffusion in arbitrarily shaped objects," in *Proc. Graphics Interface*, 2005, pp. 79–85.
[21] J. Wang, S. Zhao, X. Tong, S. Lin, Z. Lin, Y. Dong, B. Guo, and H.-Y. Shum, "Modeling and rendering of heterogeneous translucent materials using the diffusion equation," *ACM Trans. Graph.*, vol. 27, no. 1, pp. 1–18, 2008.
[22] A. P. Gibson, J. C. Hebden, and S. R. Arridge, "Recent advances in diffuse optical imaging," *Physics in Medicine and Biology*, vol. 50, no. 4, pp. R1–R43, 2005.
[23] K. Polthier and E. Preuss, "Identifying vector field singularities using a discrete hodge decomposition," in *Proc. VisMath*, 2002, pp. 112–134.
[24] M. Meyer, M. Desbrun, P. Schröder, and A. H. Barr, "Discrete differential-geometry operators for triangulated 2-manifolds," in *Proc. VisMath*, 2003, pp. 35–57.

[25] Y. Tong, S. Lombeyda, A. N. Hirani, and M. Desbrun, "Discrete multiscale vector field decomposition," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 445–452, 2003.
[26] L. Shi, Y. Yu, N. Bell, and W.-W. Feng, "A fast multigrid algorithm for mesh deformation," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 1108–1117, 2006.
[27] R. Geist, K. Rasche, J. Westall, and R. J. Schalkoff, "Lattice-boltzmann lighting," in *Eurogr. Symposium on Rendering*, 2004, pp. 355–362.
[28] R. Geist and J. Westall, *GPU Computing GEMS Emerald Edition*. Morgan Kaufmann, 2011, ch. 25.
[29] D. Bernabei, A. Hakke Patil, F. Banterle, M. Di Benedetto, F. Ganovelli, S. Pattanaik, and R. Scopigno, "A parallel architecture for interactively rendering scattering and refraction effects," *IEEE Trans. Vis. Comp. Graph.*, vol. 32, no. 2, pp. 34–43, 2012.
[30] S. Tzeng and L.-Y. Wei, "Parallel white noise generation on a GPU via cryptographic hash," in *ACM SIGGRAPH Symp. I3D*, 2008, pp. 79–87.
[31] H.-W. Nienhuys and A. F. van der Stappen, "A surgery simulation supporting cuts and finite element deformation," in *Proc. MICCAI*, 2001, pp. 145–152.
[32] S. Sengupta, M. Harris, Y. Zhang, and J. Owens, "CUDPPA project homepage," 2010, http://code.google.com/p/cudpp/.
[33] H. Si and K. Gaertner, "Meshing piecewise linear complexes by constrained delaunay tetrahedralizations," in *Proc. International Meshing Roundtable*, 2005, pp. 147–163.
[34] F. Labelle and J. R. Shewchuk, "Isosurface stuffing: fast tetrahedral meshes with good dihedral angles," *ACM Trans. Graph.*, vol. 26, July 2007. [Online]. Available: http://doi.acm.org/10.1145/1276377.1276448