Real-time Rendering of Heterogeneous Translucent Objects with Arbitrary Shapes
Yajun Wang, Jiaping Wang, Nicolas Holzschuch, Kartic Subr, Jun-Hai Yong, Baining Guo

To cite this version:

HAL Id: inria-00443634
https://inria.hal.science/inria-00443634
Submitted on 31 Dec 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Real-time Rendering of Heterogeneous Translucent Objects with Arbitrary Shapes

Yajun Wang†‡§¶ Jiaping Wang† Nicolas Holzschuch‡¶ Kartic Subr‡¶ Jun-Hai Yong§ Baining Guo†§

† Microsoft Research Asia  § Tsinghua University
‡ INRIA Grenoble Rhône-Alpes  ¶ CNRS and Université de Grenoble, Laboratoire Jean Kuntzman

Figure 1: Rendering results at 22 frames per-second of the Stanford Thai Statue (157 K triangles) with our system.

Abstract
We present a real-time algorithm for rendering translucent objects of arbitrary shapes. We approximate the scattering of light inside the objects using the diffusion equation, which we solve on-the-fly using the GPU. Our algorithm is general enough to handle arbitrary geometry, heterogeneous materials, deformable objects and modifications of lighting, all in real-time. In a pre-processing step, we discretize the object into a regular 4-connected structure (QuadGraph). Due to its regular connectivity, this structure is easily packed into a texture and stored on the GPU. At runtime, we use the QuadGraph stored on the GPU to solve the diffusion equation, in real-time, taking into account the varying input conditions: Incoming light, object material and geometry. We handle deformable objects, provided the deformation does not change the topological structure of the objects.

1. Introduction

Subsurface scattering of light is a complex phenomenon that occurs in many materials such as jade, marble and human skin. It plays an important role in the realism of rendered scenes, but unfortunately is also challenging to simulate. In translucent objects, the outgoing radiance at each point on the surface depends on three factors: Incoming radiance at all points on the surface, the path followed by the light inside the object as well as the optical properties along this path.

Accurately rendering a translucent object can take several hours with off-line physical simulation. To accelerate the rendering process, Jensen et al. [JMLH01, JB02] introduced the diffusion approximation: Assuming homogeneous materials and infinite planar boundaries, multiple scattering effects can be simulated efficiently using a dipole approximation. However, the constraints for this diffusion approximation are not fully satisfied by many real-world objects, that have complex shapes and are composed of heterogeneous scattering materials.

The diffusion equation [Ish78] fully describes subsurface multiple-scattering effects in translucent objects, including heterogeneous materials. In practice, solving the diffusion requires a discretized model of the interior of the translucent object. Wang et al. [WZT*08] used a regular grid, the polygrid; this grid had to be built manually, and is not suited for complex geometry (high genus or thin features).

In this paper, we present the first method to solve the diffusion equation, in real-time, on objects of arbitrary shapes with heterogeneous materials. Our method is based on a 4-connected structure, the QuadGraph. In a pre-processing
step, we build the QuadGraph automatically as the connectivity graph of a tetrahedralization of the object. At each node, we store the diffusion coefficients of the material. This QuadGraph is then stored on the GPU in a compact way, exploiting the regular connectivity. At run-time, we use the GPU to solve the diffusion equation at each node of the QuadGraph, in real-time, accounting for dynamic material properties and deformations to the geometry. Our algorithm is robust enough to handle objects with arbitrary genus and arbitrarily thin features (see figures 1 and 10).

We first review the literature on translucent materials and subsurface scattering (section 2). Next, we present the background on the diffusion equation (section 3). In section 4 we describe our datastructure, the QuadGraph, its construction and the linearization of the diffusion equation. Practical implementation details including compact storage of the QuadGraph on the GPU, solving the equation and rendering are explained in section 5. In section 6, we present experimental results and timings. Finally, in section 7, we propose directions for future work.

2. Related Work

Translucent Materials: Subsurface scattering effects can be simulated using Monte-Carlo methods [DEI’99, PH00, LPT05] or Photon Mapping [JC98]. These methods are physically accurate, and apply to any object or material, but require considerable rendering time for each frame (usually hours). Jensen et al. [JMLH01] introduced the diffusion approximation for homogeneous materials with an infinite planar boundary. With this approximation, the rendering time is lowered to minutes. Dachsacher and Stamminger [DS03] extended this work to achieve real-time rendering, using the same approximation. Several papers have extended this work to multi-layered translucent materials, e.g. [DJ05], and especially human skin [GHP’08, DWd’08]. Arbree et al. [AWB08] used the lightcuts approach [WFA’05, WABG06] to render objects of arbitrary shape, with homogeneous materials. Several papers have presented algorithms for fast rendering of participating media [MSM’04, ZRL’08, SKLU’09], but they are restricted to regular volumetric shape representation.

Acquisition: A different line of research targets the acquisition of material properties, for translucent objects: Tong et al. [TWL’05] for quasi-homogeneous translucent materials, Goesele et al. [GLL’04] for acquiring the properties of an entire object, Peers et al. [PvBM’06] for capturing the properties of a flat sample. These methods sample the complete light transport operator for the object or for a material sample. Consequently, they store very large data sets that restricts editability of material properties.

Precomputed Radiance Transfer: Recently Precomputed Radiance Transfer [SKS02] has been extended to render translucent objects in real-time, through pre-computation of the light transport due to subsurface scattering [HV04, WTL05]. Xu et al. [XGL’07] and Wang et al. [WCPW’08] precomputed the subsurface scattering of a series of 1D homogeneous basis scattering profiles and linearly combined them on-the-fly with varying weights to achieve runtime alteration of optical properties. These methods require precomputation of the light transport, with fixed material properties and/or geometrical shape of the object. They are not suited for applications that require dynamic editing of the material or the geometry.

Diffusion Equation: Multiple scattering effects can be modeled by the diffusion process [Ish78], described by a PDE– the diffusion equation– that can be numerically solved. The Diffusion Equation was first introduced in computer graphics [Sta95] for rendering participating media. Haber et al. [HMBV05] extended this work to objects of arbitrary shapes, using embedded boundary discretization. Wang et al. [WZT’08] achieved real-time rendering of translucent objects on the GPU, using a quasi-regular 6-connected structure, the polygrid. However, this method is restricted to objects with simple geometry (low genus, no sharp features).

Our algorithm is also based on the diffusion equation, allowing us to handle heterogeneous materials. However, the main difference with previous work is that we discretize translucent objects using tetrahedra instead of cubic grids. This allows us to handle arbitrary geometry, in real-time.

3. The Diffusion Equation

Light entering a translucent object interacts with the material of the object and is scattered several times before leaving the object. If we focus only on this multiply-scattered component, light within the object is related to the light entering the object by a PDE, the diffusion equation [Ish78, Sta95] with incoming radiance being the boundary condition (see figure 2). The solution to the diffusion equation expresses the distribution of light throughout the object and light leaving the object is obtained by computing the outgoing radiance at the surface.

More formally, consider an object \( \Omega \) with boundary \( \partial \Omega \), composed of a highly scattering, non-emissive, heterogeneous material, defined by its absorption coefficient \( \mu(x) \) and reduced scattering coefficient \( \sigma_s(x) \) [JMLH01]. The diffusion equation defines the radiant flux \( \phi(x) \) within this ob-

![Figure 2: Rendering with diffusion equation.](image-url)
\[ \kappa(x) = [3\mu(x) + \sigma_i(x)]^{-1}. \]

We use the diffusive source boundary condition as described by Arbree [AWB09] based on the Robin boundary condition [HST94, SAMD95], which relates the incoming radiance \( L_i \) on the surface of the object \( \partial \Omega \) to the radiative flux as

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

where

\[
q(x) = \int_{\partial \Omega} L_i(x, \omega_o) (\hat{n} \cdot \omega_o) F_1(\omega_o) d\omega_o, \quad (3)
\]

\[ A = (1 + F_{dr})/(1 - F_{dr}) \] and \( q(x) \) is the diffused incoming light at the surface point \( x \). \( F_1 \) and \( F_{dr} \) are the diffuse Fresnel transmittance and Fresnel reflectance coefficients, respectively [JMLH01]. Both are functions of the refraction index \( \eta \).

Once radiant flux \( \phi(x) \) inside the object is determined, we compute the outgoing radiance on the boundary \( L_o(x_o, \omega_o) \) as its derivative along the normal \( \vec{n} \) at point \( x_o \in \partial \Omega \) [AWB09]:

\[
L_o(x_o, \omega_o) = \frac{F_1(\omega_o)}{4\pi} \left[ \frac{1}{A} \phi(x) - \frac{4}{1 + F_{dr}} q(x) \right]. \quad (4)
\]

4. Solving the Diffusion Equation on QuadGraph

We represent the boundary condition and solve the diffusion equation in a discrete domain. A regular 4-connected volumetric graph called QuadGraph is proposed to serve as such a domain, which is automatically constructed given the surface geometry of the object. In this section, we describe how to construct the QuadGraph and solve the diffusion equation on it.

4.1. QuadGraph

The QuadGraph is a volumetric graph; nodes are connected to either 1 other node (surface nodes), or to 4 other nodes (inner nodes). Surface nodes (\( N_{\partial \Omega} \)) are positioned at the surface of the object, and they sample the incoming radiance and represent the boundary condition. Inner nodes (\( N_{\Omega} \)), distributed inside the volume, sample volumetric optical properties and represent the solution of the diffusion equation, radiant flux. The entire graph (\( N_{\Omega} \cup N_{\partial \Omega} \)) is the union of surface nodes and inner nodes, \( N_{\Omega} \cup N_{\partial \Omega} \).

As shown in figure 3, to build the QuadGraph, we start with an automatic tetrahedralization of the triangle mesh [ACSYD05, LS07]. After the tetrahedralization, a given tetrahedron can have from 0 to 3 of its faces on the boundary of the object (we assume that there are no isolated tetrahedra).

We split all tetrahedra with 2 or 3 faces on the boundary by adding a new vertex at their centroids as shown in figure 4, so that the resulting tetrahedral mesh only contains tetrahedra with 0 or 1 faces on the boundary.

We then build the QuadGraph as the connectivity graph of this tetrahedral mesh as illustrated in figure 5: each tetrahedron is converted into a graph node, located at its centroid. If two tetrahedra share a given face, then their centroids are connected in the graph. Tetrahedra with one face on the boundary of the object are converted into a set of two connected nodes: an inner node and a surface node. The geometric position of the surface node is determined as the closest intersection between the surface mesh and the ray linking it to the inner node. Figure 3 shows an example of the QuadGraph. Once we have built the initial QuadGraph, we refine

\[ C_0 \] to拒絕 a new vertex and four edges shown in blue. Splitting a \( C_2 \) tetrahedron results in two \( C_0 \) tetrahedra and two \( C_1 \) tetrahedra. Splitting a \( C_3 \) tetrahedron results in one \( C_0 \) tetrahedra and three \( C_1 \) tetrahedra.

\[ \text{inner nodes}. \] Surface nodes (\( N_{\partial \Omega} \)) are positioned at the surface of the object, and they sample the incoming radiance and represent the boundary condition. Inner nodes (\( N_{\Omega} \)), distributed inside the volume, sample volumetric optical properties and represent the solution of the diffusion equation, radiant flux. The entire graph (\( N_{\Omega} \cup N_{\partial \Omega} \)) is the union of surface nodes and inner nodes, \( N_{\Omega} \cup N_{\partial \Omega} \).

As shown in figure 3, to build the QuadGraph, we start with an automatic tetrahedralization of the triangle mesh [ACSYD05, LS07]. After the tetrahedralization, a given tetrahedron can have from 0 to 3 of its faces on the boundary of the object (we assume that there are no isolated tetrahedra).

We split all tetrahedra with 2 or 3 faces on the boundary by adding a new vertex at their centroids as shown in figure 4, so that the resulting tetrahedral mesh only contains tetrahedra with 0 or 1 faces on the boundary.

We then build the QuadGraph as the connectivity graph of this tetrahedral mesh as illustrated in figure 5: each tetrahedron is converted into a graph node, located at its centroid. If two tetrahedra share a given face, then their centroids are connected in the graph. Tetrahedra with one face on the boundary of the object are converted into a set of two connected nodes: an inner node and a surface node. The geometric position of the surface node is determined as the closest intersection between the surface mesh and the ray linking it to the inner node. Figure 3 shows an example of the QuadGraph. Once we have built the initial QuadGraph, we refine

\[ C_0 \] to拒绝 a new vertex and four edges shown in blue. Splitting a \( C_2 \) tetrahedron results in two \( C_0 \) tetrahedra and two \( C_1 \) tetrahedra. Splitting a \( C_3 \) tetrahedron results in one \( C_0 \) tetrahedra and three \( C_1 \) tetrahedra.

\[ \text{inner nodes}. \] Surface nodes (\( N_{\partial \Omega} \)) are positioned at the surface of the object, and they sample the incoming radiance and represent the boundary condition. Inner nodes (\( N_{\Omega} \)), distributed inside the volume, sample volumetric optical properties and represent the solution of the diffusion equation, radiant flux. The entire graph (\( N_{\Omega} \cup N_{\partial \Omega} \)) is the union of surface nodes and inner nodes, \( N_{\Omega} \cup N_{\partial \Omega} \).

As shown in figure 3, to build the QuadGraph, we start with an automatic tetrahedralization of the triangle mesh [ACSYD05, LS07]. After the tetrahedralization, a given tetrahedron can have from 0 to 3 of its faces on the boundary of the object (we assume that there are no isolated tetrahedra).

We split all tetrahedra with 2 or 3 faces on the boundary by adding a new vertex at their centroids as shown in figure 4, so that the resulting tetrahedral mesh only contains tetrahedra with 0 or 1 faces on the boundary.

We then build the QuadGraph as the connectivity graph of this tetrahedral mesh as illustrated in figure 5: each tetrahedron is converted into a graph node, located at its centroid. If two tetrahedra share a given face, then their centroids are connected in the graph. Tetrahedra with one face on the boundary of the object are converted into a set of two connected nodes: an inner node and a surface node. The geometric position of the surface node is determined as the closest intersection between the surface mesh and the ray linking it to the inner node. Figure 3 shows an example of the QuadGraph. Once we have built the initial QuadGraph, we refine

\[ C_0 \] to拒绝 a new vertex and four edges shown in blue. Splitting a \( C_2 \) tetrahedron results in two \( C_0 \) tetrahedra and two \( C_1 \) tetrahedra. Splitting a \( C_3 \) tetrahedron results in one \( C_0 \) tetrahedra and three \( C_1 \) tetrahedra.

\[ \text{inner nodes}. \] Surface nodes (\( N_{\partial \Omega} \)) are positioned at the surface of the object, and they sample the incoming radiance and represent the boundary condition. Inner nodes (\( N_{\Omega} \)), distributed inside the volume, sample volumetric optical properties and represent the solution of the diffusion equation, radiant flux. The entire graph (\( N_{\Omega} \cup N_{\partial \Omega} \)) is the union of surface nodes and inner nodes, \( N_{\Omega} \cup N_{\partial \Omega} \).
the position of the inner nodes for even distribution by applying reaction-diffusion [Tur91] to the 3D points.

Our method is based on the tetrahedralization method of Alliez et al. [ACSYD05], which produces nearly equilateral tetrahedra of regular size. It takes a single parameter, $K$, which controls how much larger tetrahedra inside the object are, compared to those near the surface.

4.2. Discretized Diffusion Equation

Once we have built the QuadGraph $\{N_{h}, N_{d_{h}}\}$, we discretize the Diffusion Equation (Equations 1 and 2) into linear equations using a Finite Difference Method (FDM), as in [Sta95]:

\[
\sum_{j=1}^{4} \frac{1}{d_{ij}^2} \chi(n_j) \phi(n_j) - (\omega \chi(n_i) + \mu(n_i)) \phi(n_i) = 0 \quad (5)
\]

\[
\phi(n_i) + 2\chi(n_i) \frac{\phi(n_i) - \phi(n_k)}{d_{ik}} = \frac{4}{1 - F_{d}r} q(n_i) \quad (6)
\]

The sum in Equation 5 is over the four nodes $n_j$ connected to the node $n_i$. In Equation 6, $n_k$ is the single inner node connected to the surface node $n_i$. In Equation 5, we use $w = \sum_{i=1}^{4} 1/d_{ij}^2$, and $d_{ik}$ denotes the distance between two nodes, or the length of the graph edge connecting them.

We sample optical properties ($\chi$ and $\mu$) at the graph nodes. $q(n_i)$ is the diffused incoming light for each surface node. We approximate the coefficients of Laplacian operator with the inverse distance ($\alpha = -2$) [Tur95]. Once we have solved the diffusion equation described in section 4.3, we find the outgoing radiance $L_w(n_i, \omega_k)$ according to the radiant flux of the corresponding surface node $\phi(n_i)$:

\[
L_w(n_i, \omega_k) = \frac{F_{\omega}(\omega_k) (\phi(n_i) - 2q(n_i))}{2\pi(1 + F_{d}r)}, \quad n_k \in N_{d_{h}} \quad (7)
\]

4.3. Solving the linearized equation

We solve the linearized equations (5, 6) on the QuadGraph using the relaxation scheme as in [Sta95, WZT*08]: we start by initializing the value of the radiant flux at each node to $\phi_0(n)$ (e.g. zero), and iterate the following steps until reach a user defined number of iterations.

\[
\phi_{i+1}(n_i) = \sum_{j=1}^{4} \frac{\chi(n_j) \phi(n_j)}{w \chi(n_i) + \mu(n_i)} \quad n_i \in N_{h} \quad (8)
\]

\[
\phi_{i+1}(n_k) = \frac{2\chi(n_k) \phi(n_k) + d_{ik} \frac{4q(n_k)}{1 + F_{d}r}}{2\chi(n_i) + d_{ik}} \quad n_k \in N_{d_{h}} \quad (9)
\]

To speed-up the convergence, we use a multi-resolution scheme, similar to [WZT*08]. We design multiple levels $h$ of the QuadGraph, $\{N^h_{h}, N^d_{h}\}$. Each level is built, independently, as a tetrahedralization of a simpler version of the surface mesh, with the number of surface points decreasing by a factor of four between two hierarchical levels. Note that the finest level uses the exact full QuadGraph, so no approximation is introduced by using the multi-resolution scheme if convergence is fulfilled.

After we have these multi-resolution representations of the object, we start by solving at the coarsest level, $h = 0$, use this solution as the starting point for solving at the next level, and iterate. The coarse level generates a blurred solution (figure 4b) with correct overall distribution and serves as the initialization for the finer level. Specifically, initial radiant flux at the level $h + 1$ is sampled from the solution of $b$ nearby nodes on level $h$, with interpolation weights proportional to the square of the inverse distance [She68]:

\[
\phi(n^h_{i+1}) = \sum_{j=1}^{b} w_{j} \phi(n^h_{j}) \quad (10)
\]

\[
w_j = \frac{w_{xy} \sum_{k=1}^{b} w_{xy}^{-2}}{w_{xy}^{-2}}
\]

We treat surface nodes and inner nodes independently in this hierarchical sampling process. We used $b = 4$ for surface nodes, and $b = 8$ for inner nodes, i.e. each surface node at a finer level is computed by interpolating the values from 4 surface nodes at the coarser level. The interpolation weights between the different levels of the QuadGraph ($w_{xy}$) are pre-computed as part of the pre-processing step. They are independent of the optical properties of the material, and invariant to smooth shape deformation.

5. GPU Implementation and Rendering

Our algorithm solves the diffusion equation inside the translucent object at each frame, without precomputation of the light transport, thus allowing for dynamic illumination, optical properties and shape deformations. We store the QuadGraph in textures on the GPU as a pre-processing step. At each frame, we first sample the incoming illumination using shadow mapping on the surface of the translucent object, then solve the Diffusion Equation on the GPU using it as a boundary condition.

5.1. Storage

We store the QuadGraph itself on the GPU using 2D textures (see figure 6). Each node of the QuadGraph corresponds to

![Figure 6: Textures for encoding parameters. Upper row (a): 16-bits float textures that encode variables. Bottom row (b): 32-bits integer textures that encode constants.](image)
use \( r = 32 \). Such a choice is based on the tetrahedralization density of the geometry models used in our experiments.

For packing together inner nodes, we use the connectivity inside the graph. For surface nodes, we use proximity on the surface. In our experiments, this localized storage of the QuadGraph in textures results in a 30% to 60% speedup over random storage.

5.2. Solving the Equation

We solve the diffusion equation at each frame, recursively on each hierarchical level of the QuadGraph, starting with the coarsest level \( h = 0 \) (see figure 9 for a pseudo-code of our algorithm).

For each level, we start by evaluating the incoming diffuse illumination on the surface, including shadows (using shadow mapping) and multiply with Fresnel transmission coefficient. Incoming light is stored in a 16-bits floating-point texture,  for each surface node of the QuadGraph.

We then solve the diffusion equation iteratively, using two auxiliary textures to store the radiant flux, before and after the current iteration (see figure 8 for the pseudo-code of a single iteration). After computing an iteration, we test for convergence. If we have converged at this hierarchical level, we move on to the next level, using the results at this level as a starting point, after re-sampling. Otherwise, we swap the two textures and compute the next iteration.

5.3. Displaying Results

We do not display the geometry of the original object; instead, we build a surface \( \Upsilon \) that maps directly to the QuadGraph, by triangulating all its surface nodes. Once we have computed the solution of the Diffusion Equation, we use it to compute the outgoing radiance \( L_{\omega} \) for all surface nodes \( n_s \) using Eq. 7, then we render \( \Upsilon \).

6. Experimental Results

We have implemented our algorithm on an Intel Core2Duo 2.13GHz CPU, with 2GB memory and an NVIDIA Geforce 8800GTX GPU with 768MB graphics memory. We used GLSL to implement the algorithm on the GPU.

![Figure 7: Texture packing. Topologically nearby nodes (red) are packed in the same texture block (blue).](image)

Figure 7: Texture packing. Topologically nearby nodes (red) are packed in the same texture block (blue).

![Figure 8: Pseudocode for a single iteration](image)

**Figure 8:** Pseudocode for a single iteration

At each new frame:

for each multi-resolution level \( h \)

compute incoming light \( q(n_i) \) on all surface nodes

initialize \( \phi(n_i) \) by sampling from level \( h = 1 \)

while not converged

update \( \phi(n_i) \) for all surface nodes (Eq. 9)

update \( \phi(n_i) \) for all inner nodes (Eq. 8)

end while

end for

compute radiance \( L_{\omega}(n_i, o_i) \) for all surface nodes (Eq. 7)

render the object surface using \( L_{\omega} \).

**Figure 9:** Pseudocode for our linear solver.
Table 1: Statistics of test scenes: $N_s$ and $N_i$ are the number of surface and inner nodes, respectively, of the QuadGraph at the finest level. $L$ is number of multi-resolution levels; Mem is the total size of textures over all levels; Prep is the preprocessing time including tetrahedralization; QuadGraph construction and texture packing; Iter is the number of iterations of the solver at each level.

<table>
<thead>
<tr>
<th>Scene</th>
<th>Fig.</th>
<th>QuadGraph</th>
<th>Mem (MB)</th>
<th>Prep (min)</th>
<th>Iter</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thai</td>
<td>1</td>
<td>157k 369k</td>
<td>4</td>
<td>13</td>
<td>35</td>
<td>10</td>
</tr>
<tr>
<td>Gargoyle</td>
<td>10a</td>
<td>62k 139k</td>
<td>0.5</td>
<td>3</td>
<td>18</td>
<td>40</td>
</tr>
<tr>
<td>Topmod</td>
<td>10b</td>
<td>121k 260k</td>
<td>0.2</td>
<td>3</td>
<td>36</td>
<td>15</td>
</tr>
<tr>
<td>Buddha</td>
<td>10c</td>
<td>66k 120k</td>
<td>0.5</td>
<td>3</td>
<td>17</td>
<td>30</td>
</tr>
<tr>
<td>Fertility</td>
<td>10d</td>
<td>82k 182k</td>
<td>0.3</td>
<td>3</td>
<td>25</td>
<td>12</td>
</tr>
<tr>
<td>Twirl</td>
<td>10e</td>
<td>82k 226k</td>
<td>0.1</td>
<td>3</td>
<td>27</td>
<td>16</td>
</tr>
<tr>
<td>Heptoroid</td>
<td>10f</td>
<td>82k 178k</td>
<td>0.2</td>
<td>2</td>
<td>24</td>
<td>11</td>
</tr>
<tr>
<td>Lucy</td>
<td>10g</td>
<td>150k 363k</td>
<td>0.2</td>
<td>4</td>
<td>33</td>
<td>20</td>
</tr>
</tbody>
</table>

6.1. Test Scenes

Figure 1 shows the rendering results of the Stanford Thai statue with the colorful agate material. Our system generates the subsurface effects due to the highly heterogeneous material. Complex surface details and small geometry features, such as the trunk in the middle and claw in the bottom, are well preserved and exhibit rich visual effects with different lighting conditions. Please refer to the accompanying video for animated sequences.

Figure 10 shows rendering results with various objects. Objects with high genus (b, c, i), high curvature (b, e, f) and thin features (a, c, e, g) are well handled. We can also handle any kind of material and heterogeneities, such as wax (a, c, e, f), jade (b), and marble (d, g). Our algorithm produces convincing results in all cases, which demonstrates various visual effect on translucent materials, such as color bleeding (a, d, e, g), back-lighting (b, c, d, f) and blurring of the geometry features (a, c, g).

6.2. Computation time

Figure 12 shows the computation time (in ms) for several test scenes, as a function of the number of tetrahedra. For three of our models, the Stanford Bunny, the double-hole donut (figure 3) and the Fertility statue (figure 10d), we have several tetrahedralization: they are plotted as lines. Other models, for which we have a single tetrahedralization, appear as points on the graph. For most of the models, the rendering time appears to vary linearly with respect to the number of tetrahedra. The two outliers are the Twirl and Heptoroid (figure 10f). These two models have thinner features, and require more iterations for convergence.

6.3. Comparison with Ground Truth

To study the effect of the parameter $K$ on the results of our algorithm, we compared resulting images for different values of $K$ against a ground truth reference image that was computed using photon mapping. Figure 11 tabulates these results for four different values of $K$ and shows difference images to the reference and scanline plots. A lower value for $K$ constrains the sizes of surface and internal tetrahedra to vary less. While reducing $K$ increases the number of tetrahedra and hence computation time, it results in rendered images that are closer to the ground truth. However, even the increased quality images render at interactive rates (8 fps). By increasing $K$, we can further increase the efficiency at the appropriate compromise of fidelity to the ground truth. The plots in the image show our result (blue) across two different scanlines compared against the ground truth (black). For $K = 0.1$, the blue curve is quite close to the reference black curve.

To obtain intuition about the “sweet spot”, or the value of $K$ at which we achieve a suitable trade-off between quality and efficiency, we plotted the relative root mean squared (RMS) error of our result (in comparison with the ground truth) against $K$ for multiple models (see figure 13). This plot reveals that even at $K = 1.0$ the RMS error is less than
Figure 10: We have tested our algorithm on several complex objects, including objects with high genus (b, c, f), high curvature (b, e, f) and with sharp geometric features (a, c, e, g).

<table>
<thead>
<tr>
<th>Ground Truth</th>
<th>$K = 1$, $N = 141k$</th>
<th>$K = 0.5$, $N = 246k$</th>
<th>$K = 0.3$, $N = 468k$</th>
<th>$K = 0.1$, $N = 973k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td><img src="image1" alt="Image" /></td>
<td><img src="image2" alt="Image" /></td>
<td><img src="image3" alt="Image" /></td>
<td><img src="image4" alt="Image" /></td>
</tr>
<tr>
<td>Difference</td>
<td><img src="image5" alt="Difference" /></td>
<td><img src="image6" alt="Difference" /></td>
<td><img src="image7" alt="Difference" /></td>
<td><img src="image8" alt="Difference" /></td>
</tr>
<tr>
<td>y=300</td>
<td><img src="image9" alt="Plot" /></td>
<td><img src="image10" alt="Plot" /></td>
<td><img src="image11" alt="Plot" /></td>
<td><img src="image12" alt="Plot" /></td>
</tr>
<tr>
<td>y=400</td>
<td><img src="image13" alt="Plot" /></td>
<td><img src="image14" alt="Plot" /></td>
<td><img src="image15" alt="Plot" /></td>
<td><img src="image16" alt="Plot" /></td>
</tr>
</tbody>
</table>

Figure 11: Comparison between ground truth and our method, for different values of $K$ [ACSYD05]. For this figure, we computed only the translucency effects, with no reflections on the surface, to ensure a fair comparison. First row: the actual pictures computed. Second row: pixel-by-pixel difference with reference image (top left). Third and fourth row: 1D plot of pixel values along a scanline. The positions of the two scanlines appear on the leftmost image of the second row. The values for the reference image are in black, our results are in blue. For $K = 0.1$, our results are quite close to the reference. Note that for all values of $K$, our algorithm finds the salient features of illumination, resulting in a good visual quality.
10\%. The plot also shows that we can achieve errors as low as 2\% at slight costs in efficiency.

Finally, we verified the contribution of the multi-resolution scheme to convergence rate by plotting relative errors against execution time, with and without the multi-resolution solver. Figure 14 shows that the use of a multi-resolution solver significantly increases convergence rate.

6.4. Shape Deformation and Material Editing

Our system also supports runtime shape deformation as shown in figure 15 and the accompanying video. Any deformation algorithm that is able to apply a deformation field to the object volume (e.g. [SP86, JSW05, LLCO08]), can be applied to drive the deformation of QuadGraph. Deformation described here is only changing on geometry shape, the material changing due to deformation (e.g. condensed by squeezing) is not considered. Runtime editing of the optical properties is demonstrated in figure 16 (and accompanying video).

We can only handle deformations that maintain the underlying tetrahedralization. Deformations are therefore limited in scope and amplitude: we cannot change the topology of the object.

Although we only show simple material optical editing operations, our system is flexible enough to work with complicated editing operations since the rendering takes the optical property volume as input directly.

6.5. Discussion

Real-time solver: The main advantage of our QuadGraph algorithm is its ability to solve the diffusion equation in real-time. Since there are no precomputations besides the tetrahedralization of the input geometry, our algorithm can handle dynamically changing the incoming lighting, material properties, and even limited deformations of the geometry, without the need for precomputing the light transport.
Quality vs. efficiency: The parameter $K$ can be used to trade-off accuracy for efficiency: large values of $K$ result in a smaller number of inside tetrahedra, thus making the algorithm run faster, at the expense of accuracy. Small values of $K$ give results that are more accurate, at the expense of rendering time. But even in its fast, low-quality version, our algorithm identifies the salient lighting features, resulting in a good visual quality, and it maintains RMS error under 10% (see figure 13).

Sampling: Just as any discretization scheme, our Quad-Graph domain requires adequate sampling for capturing high-frequency effects. That is, the tetrahedralization needs to be dense enough to capture variations in (1) incident illumination at the surface as well as (2) the optical properties of heterogeneous materials. The former is less of a problem since multiple scattering results in a decimation of high-frequencies in the incident surface radiance. The latter problem limits the range of editable materials that can be faithfully rendered without re-tetrahedralization.

Arbitrary shape: The dipole approximation [JMLH01] breaks down at points with high curvature, while the poly-grid method [WZT’08] is unable to handle thin features or geometries with high genus. Since our discretization is based on the input geometry, our algorithm is robust to variations in these features.

Discretization method: Our algorithm uses a Finite Difference Method for discretizing the diffusion equation. Other algorithms have used a Finite Element Methods instead [AWB08, AWB09]. The latter is more accurate, but the former maps well to the GPU, allowing us to better take advantage of the computing power of this architecture.

7. Conclusion and Future Work

In this paper, we introduced an algorithm for real-time rendering of translucent objects, of arbitrary shape, with heterogeneous materials. Our algorithm is robust enough to handle objects with arbitrary genus and fine sharp features. Since all illumination computations are conducted in real-time, we can handle deformable objects as well as dynamic changes in the optical properties and lighting conditions. Our algorithm is based on a simple 4-connected graph, the Quad-Graph, built automatically from a tetrahedralization of the triangle mesh defining the surface of the object.

As most existing algorithms for approximating subsurface scattering, our algorithm is restricted to the multiple scattering component of subsurface scattering. While this component is usually the most visible for optically dense materials, recent research [DLE’09, WZHB09] have shown that low-order scattering effects can be quite impressive in some circumstances. In future work, we will investigate computation of low-order scattering effects for increased realism.

Although our algorithm handles deformable objects, deformations are limited, in the sense that the underlying tetrahedralization must remain valid. We cannot change the topology of the object, or break it into pieces. Future work will include handling arbitrary deformations of the object.

Finally, we would like to extend our algorithm to decouple the local scattering and handle it by texture space filtering as in [dLE07], for fine detailed material variation.

Acknowledgements: We are grateful to Adam Arbree, Bruce Walter and Kavita Bala for sharing their latest work on the new diffusion equation formula and the helpful discussions on its accuracy and robustness. We thank Shuang Zhao and Yue Dong for their source code, as reference, for the forward diffusion equation and photo-tracing. We also thank Jean-Claude Paul for the helpful discussions.

The authors from Tsinghua University were supported by the National Science Foundation of China (60625202, 6091130368) and Chinese 863 Program (2007AA040401). This work was funded in part by ANR (ANR-07-BLAN-0331 “HFIBMR”). Yajun Wang is supported by an INRIA internship grant.

Geometry model credits: Gargoyle, fertility and twirl in figure 10(a,d,e) are downloaded from the AIM@SHAPE Shape Repository (http://shapes.aim-at-shape.net/). Thai Statue, Buddha and Lucy in figure 1 and figure 10(c,g) are downloaded from the Stanford 3D Scanning Repository (http://graphics.stanford.edu/data/3Dscanrep/). Heptagonal Toroid in figure 10(f) is generated with Carlo Séquin’s sculpture generator (http://www.eecs.berkeley.edu/~sequin/SFF/spec.heptoroid.html). Topmod Ball in figure 10(b) is obtained from David Morris and is originally created by Torolf Sauermann (http://www.evolution-of-genius.de/).

References


