Real-time Rendering of Translucent Materials with Directional Subsurface Scattering

Alessandro Dal Corso

M.Sc. in Digital Media Engineering
T.I.M.E. Double Degree Program

Supervisor: Jeppe Revall Frisvad

22nd August 2014

- Alessandro, Msc in DME
- Thesis title: RTR Translucent Materials with Dir SS
- DTU Compute
- Supervisor

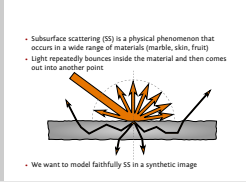
Real-time Rendering of Translucent Materials with Directional Subsurface Scattering

Alessandro Dal Corso

M.Sc. in Digital Media Engineering
T.I.M.E. Double Degree Program

Supervisor: Jeppe Revall Frisvad

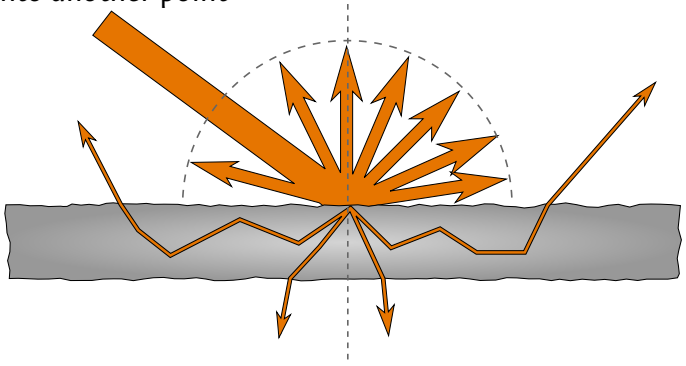
22nd August 2014



- Subsurface scattering: describe in words
- Light bouncing + comes out (show picture)
- State problem: physically render faithfully synthetic image

Introduction

- Subsurface scattering (SS) is a physical phenomenon that occurs in a wide range of materials (marble, skin, fruit)
- Light repeatedly bounces inside the material and then comes out into another point



- We want to model faithfully SS in a synthetic image

- Examples in nature
- Marble, leaves, wax
- Very present in nature

Introduction

- Approach
- Proposal: use an analytical model (BSSRDF) - opposed to numerical
- Close to offline - never implemented
- Low memory requirements (console)
- Real time - 100 ms - GPUs

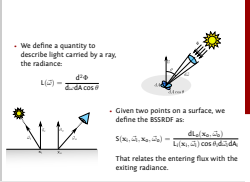
Our goal is to faithfully represent SS phenomena in a synthetically generated image:

- Using a analytical BSSRDF model [Frisvad et al., 2014]
- Visually close as possible to a offline-rendered solution
- With low memory requirements
- In real-time or at least at interactive frame rates (<100ms per frame) using a GPU-based solution

Problem statement

Our goal is to faithfully represent SS phenomena in a synthetically generated image:

- Using a analytical BSSRDF model [Frisvad et al., 2014]
- Visually close as possible to a offline-rendered solution
- With low memory requirements
- In real-time or at least at interactive frame rates (<100ms per frame) using a GPU-based solution

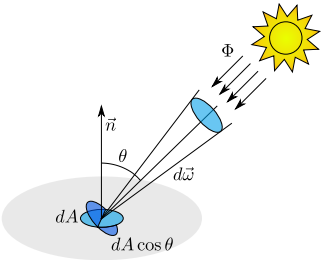


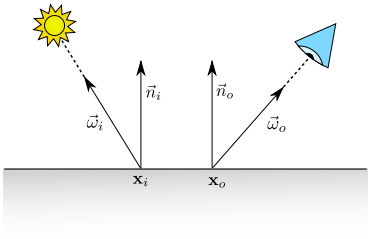
- small theoretical introduction
- define radiance and BSSRDF
- radiance flux over solid angle over projected area
- radiance and ray
- BSSRDF: bidir surface scattering distrib function
- Two points on a surface (figure)
- Out rad over in flux

Light transport

- We define a quantity to describe light carried by a ray, the radiance:

$$L(\vec{\omega}) = \frac{d^2\Phi}{d\omega dA \cos\theta}$$





- Given two points on a surface, we define the BSSRDF as:

$$S(\mathbf{x}_i, \vec{\omega}_i, \mathbf{x}_o, \vec{\omega}_o) = \frac{dL_o(\mathbf{x}_o, \vec{\omega}_o)}{L_i(\mathbf{x}_i, \vec{\omega}_i)\cos\theta_i d\vec{\omega}_i dA_i}$$

That relates the entering flux with the exiting radiance.

- Derivation from the previous
- Add emission and visibility terms
- Jensen, d'Eon, Jeppe (describe very briefly)

- From the definition of BSSRDF we obtain the area formulation of the rendering equation [Jensen et al., 2001]:
- Many BSSRDF functions have been proposed in literature [Jensen et al., 2001; D'Eon and Irving, 2011; Frisvad et al., 2014]

The rendering equation

- From the definition of BSSRDF we obtain the area formulation of the rendering equation [Jensen et al., 2001]:

$$L_o(\mathbf{x}_o, \vec{\omega}_o) = L_e(\mathbf{x}_o, \vec{\omega}_o) + \int_A \int_{2\pi} S(\mathbf{x}_i, \vec{\omega}_i, \mathbf{x}_o, \vec{\omega}_o) L_i(\mathbf{x}_i, \vec{\omega}_i)(\vec{n} \cdot \vec{\omega}_i) d\vec{\omega}_i dA_i$$

- Many BSSRDF functions have been proposed in literature [Jensen et al., 2001; D'Eon and Irving, 2011; Frisvad et al., 2014]

- Jensen dipole: diffusion approximation
- Ishimaru - light scatters isotropically after too many scattering events
- formula for point light infinite medium
- Exponential decay of fluence
- x is the position in space assuming the light is in the origin

- The standard dipole [Jensen et al., 2001] was one of the first BSSRDF functions proposed
- It uses the diffusion approximation [Ishimaru, 1997] of the radiative transport equation (RTE)
- The approximation describes how light propagates from a point source in an infinite scattering medium:

$$\phi(\mathbf{x}) = \frac{\Phi}{4\pi D} \frac{e^{\sigma_{tr} r}}{r}$$

Where

$$\phi(\mathbf{x}) = \int_{4\pi} L(\mathbf{x}, \vec{\omega}) d\vec{\omega}$$

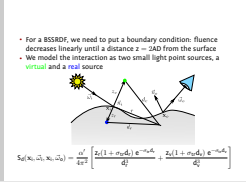
Standard dipole

- The standard dipole [Jensen et al., 2001] was one of the first BSSRDF functions proposed
- It uses the diffusion approximation [Ishimaru, 1997] of the radiative transport equation (RTE)
- The approximation describes how light propagates from a point source in an infinite scattering medium:

$$\phi(\mathbf{x}) = \frac{\Phi}{4\pi D} \frac{e^{\sigma_{tr} r}}{r}$$

Where

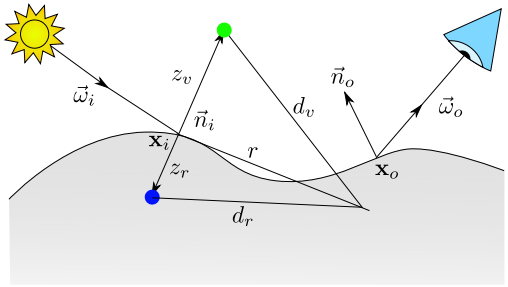
$$\phi(\mathbf{x}) = \int_{4\pi} L(\mathbf{x}, \vec{\omega}) d\vec{\omega}$$



- Boundary conditions (semi infinite medium)
- z = 2AD (mismatching indices of refraction)
- we can extend it to arbitrary geometry
- Solution in the form of a dipole
- Formula

Standard dipole

- For a BSSRDF, we need to put a boundary condition: fluence decreases linearly until a distance $z = 2AD$ from the surface
- We model the interaction as two small light point sources, a virtual and a real source



$$S_d(\mathbf{x}_i, \vec{\omega}_i, \mathbf{x}_i, \vec{\omega}_o) = \frac{\alpha'}{4\pi^2}\left[\frac{z_r(1+\sigma_{tr}d_r)\,e^{-\sigma_{tr}d_r}}{d_r^3} + \frac{z_v(1+\sigma_{tr}d_v)\,e^{-\sigma_{tr}d_v}}{d_v^3}\right]$$

- Different BSSRDF functions, more precise
- always use the diffusion approximation, but a second order one
- Directional term in the fluence (cosine) - same exponential part

- [Frisvad et al., 2014] defined a new BSSRDF function that keeps the directionality of the incoming light into account
- The models uses a more precise diffusion approximation, that yields the following fluence formula:

$$\phi(\mathbf{x},\theta) = \frac{\Phi}{4\pi D}\frac{e^{\sigma_{tr} r}}{r}\left(1 + 3D\frac{1+\sigma_{tr} r}{r}\cos\theta\right)$$

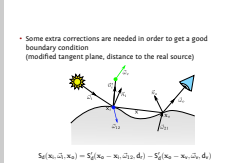
Where we have an extra term that accounts for directionality
- As in the previous model, a dipole is used, but with ray sources

Directional dipole

- [Frisvad et al., 2014] defined a new BSSRDF function that keeps the directionality of the incoming light into account
- The models uses a more precise diffusion approximation, that yields the following fluence formula:

$$\phi(\mathbf{x},\theta) = \frac{\Phi}{4\pi D}\frac{e^{\sigma_{tr} r}}{r}\left(1 + 3D\frac{1+\sigma_{tr} r}{r}\cos\theta\right)$$

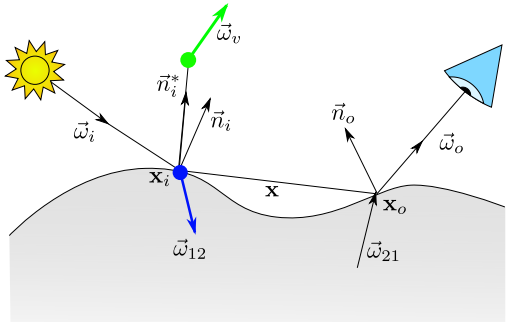
  Where we have an extra term that accounts for directionality
- As in the previous model, a dipole is used, but with ray sources



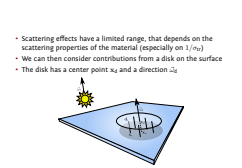
- Extra corrections and boundary conditions
- Modified tangent plane (show on picture normals)
- Distance to the real source (changed to avoid singularity - only on real)
- Formula

Directional dipole

- Some extra corrections are needed in order to get a good boundary condition
  (modified tangent plane, distance to the real source)



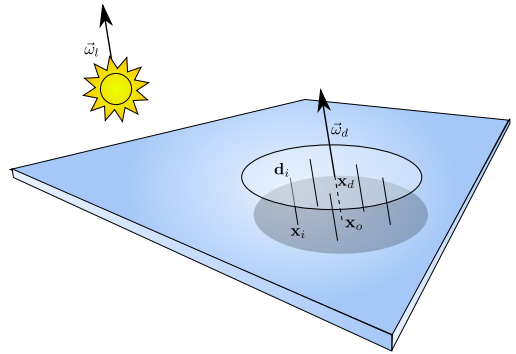
$$S_d(\mathbf{x}_i, \vec{\omega}_i, \mathbf{x}_o) = S'_d(\mathbf{x}_o - \mathbf{x}_i, \vec{\omega}_{12}, d_r) - S'_d(\mathbf{x}_o - \mathbf{x}_v, \vec{\omega}_v, d_v)$$

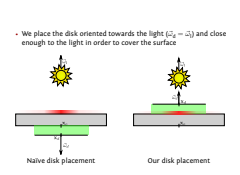


- Rationale behind our approach
- Limited range - effective transport coefficient
- We consider only point on a projected disk on a surface
- Show on image
- disk has xd and omegad

Method

- Scattering effects have a limited range, that depends on the scattering properties of the material (especially on $1/\sigma_{\text{tr}}$)
- We can then consider contributions from a disk on the surface
- The disk has a center point $\mathbf{x}_{\text{d}}$ and a direction $\vec{\omega}_{\text{d}}$

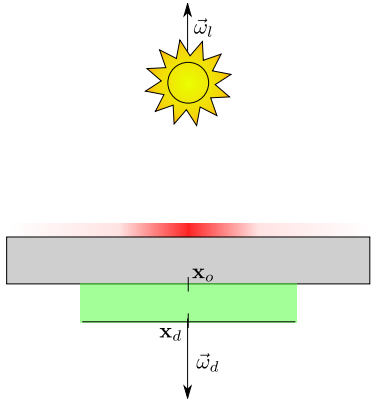




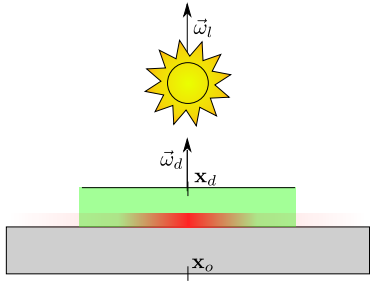
- Describe optimal placement on slab
- Naive: xo no
- Our: x weird, wl

Method

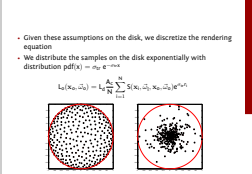
- We place the disk oriented towards the light ($\vec{\omega}_d = \vec{\omega}_l$) and close enough to the light in order to cover the surface



Naïve disk placement



Our disk placement

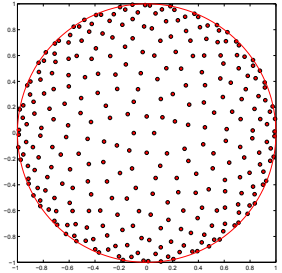


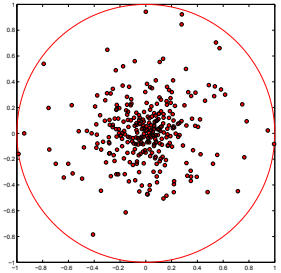
- We discretize the rendering eq - only a number of samples N.
- Sum BSSRDF over points (radiance)
- Exponential term and images

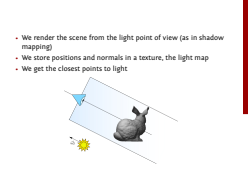
Method

- Given these assumptions on the disk, we discretize the rendering equation
- We distribute the samples on the disk exponentially with distribution pdf(x) = $\sigma_{tr}\, e^{-\sigma_{tr}x}$

$$L_o(\mathbf{x}_o, \vec{\omega}_o) = L_d \frac{A_c}{N} \sum_{i=1}^{N} S(\mathbf{x}_i, \vec{\omega}_l, \mathbf{x}_o, \vec{\omega}_o) e^{\sigma_{tr} r_i}$$



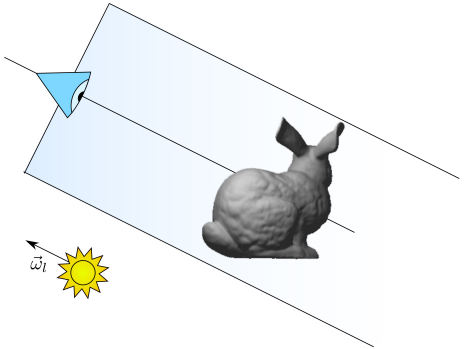


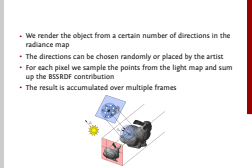


- Implementation how?
- Little details - three steps
- Render to light map
- Show on image
- closest point to the light

Implementation (step 1)

- We render the scene from the light point of view (as in shadow mapping)
- We store positions and normals in a texture, the light map
- We get the closest points to light

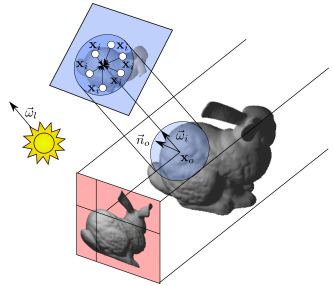


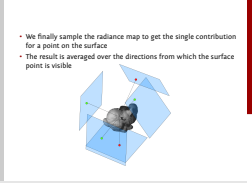


- Render the object from multiple directions
- directions random or placed, research on that
- radiance map
- fragment shader - sample light maps and accumulate bssrdf as in method
- Show on image

Implementation (step 2)

- We render the object from a certain number of directions in the radiance map
- The directions can be chosen randomly or placed by the artist
- For each pixel we sample the points from the light map and sum up the BSSRDF contribution
- The result is accumulated over multiple frames





- Combination - visibility
- Averaged over directions
- Depth map
- Show on image

Implementation (step 3)

- We finally sample the radiance map to get the single contribution for a point on the surface
- The result is averaged over the directions from which the surface point is visible

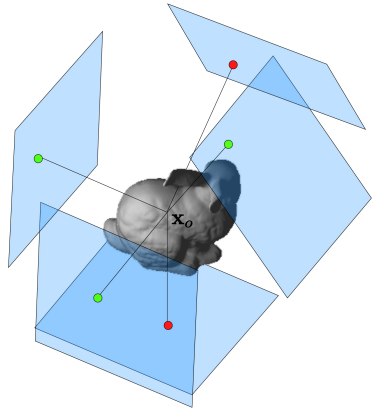


Figure: Stanford Bunny, potato material. Note the self shadowing automatically generated by the algorithm.

- Result at convergence - bunny potato - real time

Implementation

Figure: Stanford Bunny, potato material. Note the self shadowing automatically generated by the algorithm.

- self shadows
- occlusion - multiple object
- coupling shadow mapping comes in!
- compared to a full voxelization, low memory requirements
- forward and deferred paths - uncoupling

- Accounts for self shadowing and self occlusion
- Accounts for occlusion between different objects
- Directly coupled with an existing shadow mapping pipeline
- Low memory requirements compared to a full voxelization
- The final step can be adapted to forward and deferred shading pipelines

Advantages

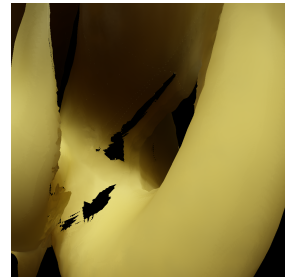
- Accounts for self shadowing and self occlusion
- Accounts for occlusion between different objects
- Directly coupled with an existing shadow mapping pipeline
- Low memory requirements compared to a full voxelization
- The final step can be adapted to forward and deferred shading pipelines

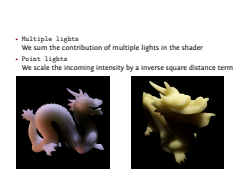
- Noise (see later in videos) - need blurring
- Tearing
- Constant shadow bias

- Noisy result, that need to be either accumulated or blurred in order to achieve a smooth result
- Cameras that cover the surface need to be placed manually (to avoid tearing)
- Inherited problems from shadow mapping (constant shadow bias)

Disadvantages

- Noisy result, that need to be either accumulated or blurred in order to achieve a smooth result
- Cameras that cover the surface need to be placed manually (to avoid tearing)
- Inherited problems from shadow mapping (constant shadow bias)





- Extensions to our base implementation, only one directional light
- Multiple: sum over (show image)
- Point lights: we change the term we multiply: no radiance, reduced intensity

Extensions

- `Multiple lights`
  We sum the contribution of multiple lights in the shader
- `Point lights`
  We scale the incoming intensity by a inverse square distance term

- Environment lighting
- 16 directional lights from a environment map
- Chosen on a distribution - more probable to have points on areas with higher luminosity
- show image

- We simulate it using 16 directional lights
- We choose "random" points on a environment map
- The distribution chosen accounts to make the points fall in areas with high luminosity

Environment lighting

- We simulate it using 16 directional lights
- We choose "random" points on a environment map
- The distribution chosen accounts to make the points fall in areas with high luminosity

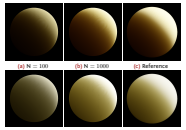
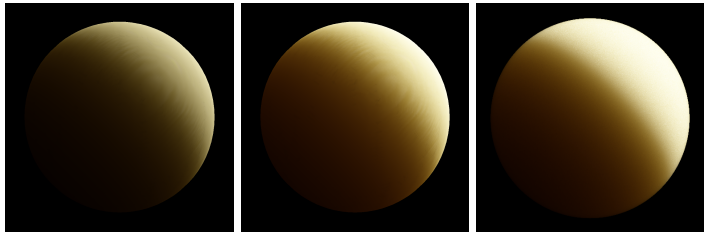


Figure: Comparison on spheres of potato and white grapefruit juice.

- Results - sphere first
- potato and grapefruit
- 100 is ok and realtime, 1000 is very ok but not realtime
- Very close to path traced solution

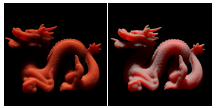
Results (quality)



(a) N = 100 (b) N = 1000 (c) Reference

(d) N = 100 (e) N = 1000 (f) Reference

Figure: Comparison on spheres of potato and white grapefruit juice.



(a) N = 50 (ca. 90 milliseconds) (b) Reference (6 hours, 16 millions samples)

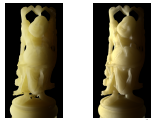
Figure: Result comparison for the Stanford Dragon model. Parameters for ketchup.

- Dragon - around 50000 vertices (simplified)
- Missing highlights in ketchup
- Improvements in the time of five orders of magnitude
- 16 millions samples prior to rejection

Results (quality)

(a) N $= 50$ (ca. 90 milliseconds) (b) Reference (6 hours, 16 millions samples)

Figure: Result comparison for the Stanford Dragon model. Parameters for ketchup.



(a) N = 100 (b) Reference (30 minutes, 10^6 samples)

Figure: Result comparison for the Happy Buddha model. Parameters for potato.

- Buddha - big model (millions of vertices)
- 300 ms
- potato
- Lose details
- Little greenish tone
- General appearance is captured - more work should be done in order to get the highlights

Results (quality)

(a) N $= 100$

(b) Reference (30 minutes, 10^6 samples)

Figure: Result comparison for the Happy Buddha model. Parameters for potato.

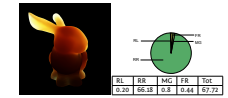
- Environment lighting
- Image from paper Frisvad
- Similar method to ours
- The result is similar, may have different lights

Results (quality)

(a) N = 90

(b) Reference [Frisvad et al., 2014]

Figure: Result comparison for the Happy Buddha model (environment lighting, Uffizi map). Parameters for potato.



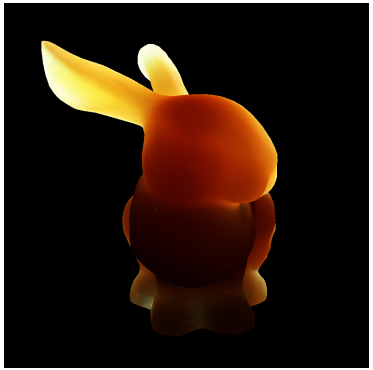
(a)Beer Bunny, 67.7ms. Point light. N = 180, M = 210, q = 2 (b)Detailed timings.

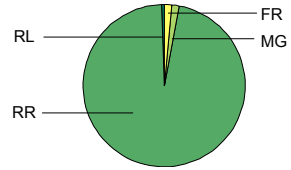
Figure: Graph that illustrates how the timings are split into the different phases of the algorithm. All tests use $W_l = W_s = 512$.

- Buddha - big model (millions of vertices)
- 300 ms
- potato
- Lose details
- General appearance is captured - more work should be done in order to get the highlights

- Performance analysis
- Nvidia GTX 780Ti
- Beer bunny split times
- Most of time lost on step 2
- The other steps are neglegible (multi light maybe)

Results (performance)



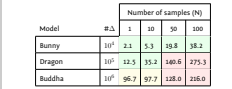


RL	RR	MG	FR	Tot
0.20	66.18	0.8	0.44	67.72

(a)Beer Bunny, 67.7ms. Point light. N = 180, M = 210, q = 2

(b)Detailed timings.

Figure: Graph that illustrates how the timings are split into the different phases of the algorithm. All tests use $W_l = W_s = 512$.



Model	#Δ	Number of samples (N) 1	10	50	100
Bunny	10^4	2.1	5.3	19.8	38.2
Dragon	10^5	12.5	35.2	140.6	275.3
Buddha	10^6	96.7	97.7	128.0	216.0

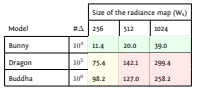
Table: Timings in milliseconds of our method for different models and number of samples N (potato material properties). One directional light, 16 directions for rendering and reconstructing.

- color code
- Performance increase with N
- 16 directions is conservative (random)
- explain why buddha rises less than dragon (area occupation)
- at low N rasterization wins

Results (performance)

Model	#Δ	Number of samples (N) 1	10	50	100
Bunny	10^4	2.1	5.3	19.8	38.2
Dragon	10^5	12.5	35.2	140.6	275.3
Buddha	10^6	96.7	97.7	128.0	216.0

Table: Timings in milliseconds of our method for different models and number of samples N (potato material properties). One directional light, 16 directions for rendering and reconstructing.



Model	#Δ	Size of the radiance map (W_s) 256	512	1024
Bunny	10^4	11.4	20.0	39.0
Dragon	10^5	75.4	142.1	299.4
Buddha	10^6	98.2	127.0	258.2

Table: Timings in milliseconds of our method for different models and size of the radiance map W_s (ketchup material properties). The other parameters were $N = 50$, $L = 1$, $W_l = 512$, $M = 1000$, $K = 16$.

- size of the radiance map
- 256 is a little bit too low, but depends on the complexity and the details
- Also here, fragment shader, so resolution dependent

Results (performance)

Model	#Δ	Size of the radiance map (W_s) 256	512	1024
Bunny	10^4	11.4	20.0	39.0
Dragon	10^5	75.4	142.1	299.4
Buddha	10^6	98.2	127.0	258.2

Table: Timings in milliseconds of our method for different models and size of the radiance map W_s (ketchup material properties). The other parameters were $N = 50$, $L = 1$, $W_l = 512$, $M = 1000$, $K = 16$.

Model	#Δ	Number of directions (K) 4	8	16	32
Bunny	10^4	6.6	10.0	20.1	42.1
Dragon	10^5	36.7	70.1	143.0	306.2
Buddha	10^6	32.5	55.8	128.3	363.5

Table: Timings in milliseconds of our method for different models and different number of directions K (ketchup material properties). The other parameters were N $= 50$, L $= 1$, $W_s = W_l = 512$, M $= 1000$.

- Number of directions
- Less directions, less time
- Many directions - filling of gpu memory - timings difficult to evaluate

Results (performance)

Model	#Δ	Number of directions (K) 4	8	16	32
Bunny	10^4	6.6	10.0	20.1	42.1
Dragon	10^5	36.7	70.1	143.0	306.2
Buddha	10^6	32.5	55.8	128.3	363.5

Table: Timings in milliseconds of our method for different models and different number of directions K (ketchup material properties). The other parameters were N $= 50$, L $= 1$, $W_s = W_l = 512$, M $= 1000$.

- Fast rendering
- Results comparables
- Five orders
- Flexible - in engines

- We implemented a solution for fast rendering of translucent materials using a state of the art directional BSSRDF
- We obtained results comparable to offline rendered solutions
- We obtained an improvement of five orders of magnitude over offline rendered solutions
- We obtained a flexible method that is applicable to current real-time graphics engines

Conclusions

- We implemented a solution for fast rendering of translucent materials using a state of the art directional BSSRDF
- We obtained results comparable to offline rendered solutions
- We obtained an improvement of five orders of magnitude over offline rendered solutions
- We obtained a flexible method that is applicable to current real-time graphics engines

Thank you!

Conclusions

Thank you!

Eugene D'Eon and Geoffrey Irving. A quantized-diffusion model for rendering translucent materials. In ACM SIGGRAPH 2011 Papers, SIGGRAPH '11, pages 56:1--56:14, New York, NY, USA, 2011. ACM.

J. R. Frisvad, T. Hachisuka, and T. K. Kjeldsen. Directional dipole for subsurface scattering in translucent materials. ACM Transactions on Graphics, 2014, -:--, 2014. To appear.

Akira Ishimaru. Wave propagation and scattering in random media. IEEE, 1997.

Henrik Wann Jensen, Stephen R. Marschner, Marc Levoy, and Pat Hanrahan. A practical model for subsurface light transport. In Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '01, pages 511--518, New York, NY, USA, 2001. ACM.

References

Eugene D'Eon and Geoffrey Irving. A quantized-diffusion model for rendering translucent materials. In ACM SIGGRAPH 2011 Papers, SIGGRAPH '11, pages 56:1--56:14, New York, NY, USA, 2011. ACM.

J. R. Frisvad, T. Hachisuka, and T. K. Kjeldsen. Directional dipole for subsurface scattering in translucent materials. ACM Transactions on Graphics, 2014, -:--, 2014. To appear.

Akira Ishimaru. Wave propagation and scattering in random media. IEEE, 1997.

Henrik Wann Jensen, Stephen R. Marschner, Marc Levoy, and Pat Hanrahan. A practical model for subsurface light transport. In Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '01, pages 511--518, New York, NY, USA, 2001. ACM.