Efficient interactive multi-resolution rendering for translucent materials under area lights

Koa, Ming Di

2017

Koa, M. D. (2017). Towards better prediction and content detection through online social media mining. Doctoral thesis, Nanyang Technological University, Singapore.

<http://hdl.handle.net/10356/72710>

<https://doi.org/10.32657/10356/72710>

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

**EFFICIENT INTERACTIVE MULTI-RESOLUTION
RENDERING FOR TRANSLUCENT MATERIALS
UNDER AREA LIGHTS**

**KOA MING DI
SCHOOL OF COMPUTER SCIENCE & ENGINEERING
2017**

**EFFICIENT INTERACTIVE MULTI-RESOLUTION
RENDERING FOR TRANSLUCENT MATERIALS
UNDER AREA LIGHTS**

KOA MING DI

SCHOOL OF COMPUTER SCIENCE & ENGINEERING

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Doctor of Philosophy

2017

Acknowledgements

In this opportunity I would like to express my gratitude especially to my supervisors Dr. Henry Johan and Prof. Alexei Sourin, for their numerous guidance and assistance they have given during my research.

I would like to thank my three fellow seniors, Dr Budianto, Dr Nicholas, Dr Chen Kan for their help and advice in the earlier years of my Ph.D. I would also like to thank the staff in MAGIC frontier: Prof. Seah Hock Soon, Dr. Chen Quan, Hong Ze and Elsie for their hospitality during the course of my Ph.D. I am also grateful to be able to work with my colleagues in MAGIC, Fraunhofer Singapore and ST Electronics (Training and Simulation Systems).

I also appreciate the help from my former colleagues in MAGIC: Dr Qiu Jie, Dr Quah Chee Kwang, Dr Xu Xiang, Dr Jigang, Anthony, Vincent, Hon Keat.

Last but not least, I also would like to thank my family, friends and my dearest girlfriend Xiaoqi, and all my friends in Singapore and in other places around the globe for the support, encouragement and sharing of ideas during my study.

Table of Contents

Acknowledgements	i
Table of Contents	ii
List of Figures	vii
List of Tables	xxii
List of Algorithms	xxiv
Abstract	xxv
1 Introduction	1
1.1 Interactive rendering and translucent materials	1
1.2 Challenges	5
1.3 Summary of our research and contributions	7
1.3.1 Efficient rendering of translucent materials	8
1.3.2 Multi-resolution rendering under area light source	9
1.3.3 Rendering translucent materials under area light source	10
1.4 Thesis organization	10
2 Literature Review	12
2.1 Fundamentals of radiometry and light transport	12
2.1.1 Radiometric quantities	13

2.1.2	Fresnel	15
2.1.3	Surface BRDF	16
2.1.4	Phase functions	17
2.1.5	Radiative transport equation	18
2.1.6	Global illumination formulation	19
2.2	Translucent materials rendering	20
2.2.1	Fundamentals of bi-directional surface scattering distribution func- tion (BSSRDF)	20
2.2.2	Single scattering	21
2.2.3	Dipole method	23
2.2.4	Mesh based approaches	27
2.2.5	Precomputed radiance transfer (PRT)	29
2.2.6	Texture/Screenspace methods	31
2.2.7	Voxel based methods	37
2.3	Efficient methods for area light illumination	43
2.3.1	Shadow mapping techniques	43
2.3.2	Image space sparse samples reconstruction	45
2.3.3	Multi-resolution techniques	45
2.4	Critical analysis	46
2.4.1	Translucent materials rendering - Precomputations	46
2.4.2	Translucent materials rendering - Storage	47
2.4.3	Translucent materials rendering - Visual comparisons	48
2.4.4	Area lights illumination - Computation efficiency	49
2.4.5	Area lights illumination - Accuracy	49
2.4.6	Comparisons with our work	50
2.4.7	Translucent material rendering	50
2.4.8	Illumination under area lighting	50

3	Interactive Rendering of Translucent Materials	51
3.1	Hypothesis	51
3.1.1	Translucent object rendering can be approximated efficiently	51
3.1.2	Optimizations in screenspace for area lights applies to translucent materials	52
3.1.3	Indirect Illumination from translucent materials can be rendered effectively using voxels	53
3.2	Research goals	54
3.3	Design and approach	55
3.4	Summary	58
4	ESLPV for Real-time Rendering of Translucent Materials	59
4.1	Our ESLPV pipeline	60
4.1.1	Voxelization	61
4.1.2	Single-scattering injection	62
4.1.3	Multiple scattering propagation	66
4.1.4	Rendering	70
4.2	Implementation	72
4.3	Results and discussion	73
4.4	Applications - Material editing	76
4.5	Summary	77
5	Multi-resolution Screenspace Optimization for Direct Illumination from Area Lights	82
5.1	Our rendering pipeline	83
5.1.1	Overview	84
5.1.2	Geometric discontinuity and light culling	85
5.1.3	Soft shadow refinement	86
5.1.4	Additional fragments generation	92

5.1.5	Screenspace irradiance computation	92
5.1.6	Screenspace single pass upsampling	94
5.2	Implementation	99
5.2.1	Depth and curvature discontinuity check	99
5.2.2	Ray intersection test	100
5.2.3	Discontinuity thresholding - Transform feedback shader	101
5.3	Results and discussion	102
5.4	Summary	108
6	Area Light Illumination Rendering for Subsurface Scattering	114
6.1	Our pipeline	116
6.1.1	Direct illumination for translucent objects from area lights	117
6.1.2	Indirect illumination from area lights	119
6.2	Implementation	122
6.2.1	Poisson samples generation	122
6.2.2	Gathering operations on Poisson samples	123
6.2.3	Indirect illumination voxels	124
6.3	Results and discussion	124
6.3.1	Limitations	126
6.4	Summary	127
7	Conclusions and Future Work	131
7.1	Conclusions	131
7.2	Future work	134
7.2.1	Global illumination for translucent materials	134
7.2.2	All frequency lighting for voxel based lighting	135
7.2.3	Multi-resolution light sampling	136
	My Research Publications	136

Bibliography	138
A Spherical Harmonics	154
A.1 Overview	154
A.2 Properties of spherical harmonics	155
A.2.1 Orthonormality	155
A.2.2 Double product integral	156
A.2.3 Rotational invariance	156
A.2.4 Rotation	156
A.3 Cartesian values for spherical harmonics basis functions	158
A.4 Rotation of clamped cosine lobe	158
A.5 Flux Propagation	160
A.6 Solid angle computation	163
A.7 Flux transfer between Voxels	164
A.8 Rendering incident radiance formulation	167
A.9 Analytic comparisons of Henyey Greenstein Phase Function with its 2nd order SH form	169

List of Figures

1.1	Common fields in photo-realistic rendering.	2
1.2	Photo-realistic image of a bedroom generated with 3DS Max. ¹	3
1.3	An actual photograph of a horse model made of jade material. The model is backlit to show the diffuse glowing look from the front. ¹	4
1.4	An actual photograph of a 3D printing material with strontium aluminate. The material is translucent in nature and has a property which stores and releases light. Light emits out of the material simulating a black lit glow. The surrounding surfaces are illuminated by it. ¹	5
1.5	Light scattering in atmospheric gases causes a red glow in the sky. ²	5
2.1	Flux describes the amount of light energy from all directions that hits a surface of finite area. Irradiance measures the amount of flux that hits a unit surface area around x. Radiance describes the amount of light moving in or out from x within a differential solid angle. The rotated cylinder in the above image represents the differential area that is perpendicular to the direction of the radiance	13
2.2	Reflection and Refraction of Light.	16
2.3	The parameter g in the Henyey Greenstein Phase function controls the angular distribution of light at any point x within the participating media. Image obtained from Jarosz [1].	18

2.4	Scattering of light in BRDF (left) and BSSRDF (right). In the BSSRDF, light can enter a point; undergo many scattering events, before exiting out at another point.	20
2.5	Blinn's single scattering model. The exitant radiance $\vec{\omega}_o$ can be seen if there are no particles blocking the light path (blue cylinder area).	22
2.6	Jensen et al.'s [19] single scattering model included sampling along the refracted light path as well as taking consideration of the two Fresnel term on the incident and exitant surface	22
2.7	The left figure indicates the searchlight problem [22], where light refracts through the medium and travels along the refracted direction until it ultimately undergoes scattering events and finally exits the material. The right figure shows the analytic solution using the dipole model Jensen et al. [19], where the multiple scattering events are represented using an isotropic point source. The mirrored negative source is used to account for the boundary conditions	23
2.8	An incoming ray is transformed into a dipole source and used to approximate the diffusion reflectance term. A real source is placed within the object at a distance z_r below the surface. Another virtual source is at placed $z_r + 2AD$ above the surface	24
2.9	The common pipeline for PRT based subsurface scattering techniques. (a) During the precomputation phase, a set of sample points (usually vertices) are chosen. (b) Their transport function T_d is evaluated. (c) Those transport functions are compressed using basis functions. (d) During run-time, lighting is evaluated at each point. (e) The lighting function is compressed into the same basis functions as the transport function. (f) The final radiance of a point is usually evaluated as a function of the dot product of the compressed transport vector and lighting vector.	32

2.10	Pipeline of texture space rendering for subsurface scattering. (a) Obtain the shadow maps from the scene. (b) A texture parameterization is performed on the mesh. (c) Incoming radiance from the shadow maps are injected into an offscreen texture obtained from (b). (d) The irradiance texture is blurred with a kernel, which can be separated into Gaussians [41] for faster rendering. (e) The blurred texture map is mapped back to the surface for the final render. (f) The final render is added with additional lighting effects and post processing to produce the final render.	33
2.11	Diffusion in Texture Space. The leftmost image shows the original irradiance map, and the next two images shows the irradiance map being blurred with kernels of different parameters. (Figure obtained from Jimenez et al. [46])	34
2.12	The image on the left shows a disconnected uv parameterization of the object on the right. Performing filtering on the image on the left will lead to incorrect results. (Image obtained from Jimenez et al. [45] presentation slides)	35
2.13	Overview of the screen-space subsurface scattering approach. The algorithm first retrieves three maps from the screen space (alpha matte, diffuse map, depth map). The diffuse map undergoes a filtering process based on multiple kernels defined by the depth map. The blurred diffuse maps are combined and added with specular highlights.	36
2.14	The overall pipeline of Bernabei et al. LBL technique [50]. (a) The system first initializes the voxel structure with a set of material properties. (b) Next, it performs shadow mapping to retrieve the positions of the surfaces visible to the light. (c) From those positions in the shadow map, they performed ray marching using Eikonal equations and start accumulating radiances into the voxels that are passed through by the rays. (d) Multiple diffusion iterations are performed until equilibrium is reached. (e) The final diffusion volume is used for the final viewing render. Images obtained from [50].	39

2.15	Overview of the SSLPV pipeline simulating multiple scattering (MS). The pipeline first renders a reflective shadow map (RSM) of the scene. Next, for each pixel in the RSM, energy is injected onto the surface based on a BTDF. A propagation process is then performed simulating a diffusion process, which distributes flux by an approximate scattering algorithm. Lastly, the final energy accumulated in each voxel is used in the final rendering	41
3.1	An observation by Ward et al. [73] noticed that indirect diffuse illumination varies in a very smooth fashion. Using this estimate, he reduced the number of times a Monte Carlo estimator would need to be evaluated by re-using previously calculated values in a "cache".	53
3.2	Multi-resolution [15] builds upon the same idea of re-using computations. Diffuse illumination is known to vary smoothly across flat geometry. Hence pixels can be clustered into fragments, in which large areas of pixels can be represented with simpler computations.	54
3.3	Evaluating the visibility function for each visible point and the area light source is unnecessary because some points can share similar visibility, such as regions in umbra or areas outside penumbra. Penumbra is the main region which most samples are needed. Illumination is known to be slowly varying in regions with consistent normals, depth and visibility.	55
3.4	(Left) A translucent bunny object (highlighted in red) is placed in an environment. It can act as a diffuse light source and distribute illumination to its surrounding voxels. Since the translucent object is already represented in its voxels (right), it has a readily accessible virtual point light intensity. . .	56
3.5	Light can escape from translucent materials and illuminate surrounding surfaces. These effects are usually represented as low-frequency illumination. Image from website [74].	58

4.1	Overview of the proposed ESLPV pipeline. The boxes highlighted in red indicate the modifications and extensions that have been added to the SSLPV.	61
4.2	Distance transform using cityblock distance is performed by a forward and backward sweep using the matrices in (a) and (b). The actual matrix is a 3D matrix, but for simplicity, only a 2D one is shown. For the forward scan, the matrix sweeps from top left to bottom right, and adds one to the distance above it, adds one to the left, and adds one to element in the next layer, then takes a minimum with itself and its 3 neighbours. For the backward pass, the matrix sweeps from bottom right towards the top left. It adds one to the distance on its right and below, and to the one in the previous layer. It then takes a minimum with itself and its 3 neighbours.	62
4.3	(a) Stanford Bunny rendered with single-scattering and an isotropic phase function. (b) Bunny rendered with a forward scattering phase function in the red color channel. The near plane of the spotlight is displayed as the square, and the green lines is its field of view.	63
4.4	Overview of single-scattering injection. (a) Trace the refracted light ray and compute locations of sample points using Eikonal equations. (b) Project the ray direction to the phase function. (c) Inject the SH coefficients into the nearest voxel center.	64
4.5	Single-scattering is the result of integrating along the refracted incoming ray $\vec{\omega}_i'$ and outgoing ray $\vec{\omega}_o'$.	66
4.6	The amount of propagation iterations needed for the initial set of VPLs (Propagation Iteration: 0) from the surface to reach the red voxel is three. Intermediate VPLs produced during each step are shaded in (red, green, purple) respectively.	68

4.7	We describe a single slice of voxel volume from the DTCT texture(right). The pixel intensity in the slice represents the distance away (in cityblock distance) from the nearest surface voxel. Optimally this information can be used to identify voxels that will never have their intensity propagated to the surface.	68
4.8	Images of a highly scattering dragon model viewed from the back with a spotlight illuminating it from the front. (a) Photon Beam Diffusion [77] with 4 samples per pixel rendered in 57 seconds by the Mitsuba Renderer [2]. (b) Our hierarchical approach uses two additional mipmap levels to enhance backlighting. (c) The SSLPV does not handle backlighting well for highly scattering models as too much energy is lost during the propagation phase. Scaling the flux does not result in similar backlighting quality as (a).	70
4.9	We show the single-scattering experiment setup of an incident beam of light shining on a semi-infinite medium with different materials. A perspective camera views the surface of the medium from the top.	72
4.10	We show the single-scattering falloff comparisons of an incident beam of light shining on a semi-infinite medium with different materials in each row. A perspective camera views the surface of the medium from the top. The column on the left shows the results of the exact solution. The column on the right shows our single-scattering technique. The first row represents a material with the following properties, $\sigma_s = (1, 1, 1), \sigma_a =$ $(0, 0, 0), \eta = 1.3, g = (0, 0, 0)$. The second row represents a material with $\sigma_s = (1, 1, 1), \sigma_a = (0.01, 0.1, 1), \eta = 1.3, g = (0, 0, 0)$ and the third row with $\sigma_s = (1, 1, 1), \sigma_a = (0, 0, 0), \eta = 1.3, g = (-1, 0.5, 0)$. Both results are computed from 32 points along the medium. This is done so that the size of the beam of the light can be reflected proportionally in the result because the width of the beam is limited by the ESLPV's voxel resolution. Both results are tonemapped with the same parameters.	73

4.11	Comparisons between the Hierarchical ESLPV and SSLPV for a material with albedo $> 95\%$. The model is illuminated with a spotlight, which its FOV displayed as green lines.	74
4.12	(a) Single-scattering Rendering of Stanford Thai statue (500k Triangles) with a simulated weak scattering material with refraction effects. $\sigma_a = 0.01$ and $\sigma_s=0.2$. (b) Same Stanford Thai model rendered with single and multiple scattering. (c) A rectangular block (12 triangles) with single-scattering illuminated by a checkered light (top left) which simulates multiple light beams. Tonemapping have been applied on the images.	74
4.13	Multiple frames of deformable models are rendered with their front and back view. (a) Rendering of Stanford Bunny model (80k Triangles) with single and multiple scattering. (b) Rendering of Stanford Buddha model (100k Triangles) with single and multiple scattering.	78
4.14	(a) A reference PBD render. (b) In our ESLPV, artifacts can be seen in some areas with thin geometry such as the dragon legs and horns. In areas such as the mouth of the dragon, the dimmer illumination is caused by the flux lost due to undefined boundary conditions.	79
4.15	(a)(b) Front and back view of Stanford Buddha model (100k triangles) in jade material. (c)(d) Front and back view of Stanford Armadillo model (345k triangles) in apple material. All images have single-scattering (SS) and multiple scattering (MS) components. Results have been tonemapped with gamma compression.	80
4.16	Material Editing User Interface. On the right is the user interface displaying the extinction spectrum. We display the 3D model and we select regions of the material we wished to modify. The output result is shown on the left. . .	81
5.1	Complete pipeline of our direct illumination with area lights. Boxes in red indicate new proposed stages in the multi-resolution framework.	84

5.2	Our shadow refinement pipeline. The pipeline receives fragments (except those of finest resolution) that have passed the depth-curvature tests and light culling check. It performs a sequence of process: Subdivision, Ray Generation, Ray Tracing, Bit Comparison, and finally Shadow Thresholding. The SFVT (green outlined boxes) performs the visibility testing. The number of new fragments generated is decided by its gradient in GASS (red outlined rounded box). Newly generated fragments undergo visibility tests again, while fragments that are stable are transferred out. The filled rounded boxes in gray indicate components of the transform feedback shader (TFS) stage that we use to receive and process fragments as well as stream out fragments.	88
5.3	Four sub-fragment gradients are stored into the first 20 bits of the integer max_rays and the maximum number of light rays, $Rays_M$, visible among the sub-fragments is stored in the remaining bits.	89
5.4	In the normal subdivision scheme, subdivision is performed whenever the total number of bit differences between the 4 sub-fragments exceeds a threshold. This high discontinuity is determined by the total bit difference within a fragment. (a) The original subdivision scheme only refines a fragment to its next level of mipmap, which is insufficient and requires additional refinement in the next pass. (b) In our work, we subdivide each sub-fragment with high gradients into 2x2 sub fragments. The refinement criteria is described in the <i>GASS</i> refinement scheme.	91
5.5	Three fragments (in red) are positioned in the 16x16 and 8x8 sized fragments. Their location enables us to use radial basis functions to compute fragment values in between. Texels in green indicate the regions with little visibility discontinuity during SFVT.	93

5.6	Image in the center shows a fragment map of the Sponza Scene after visibility discontinuities have been done. Image on the left shows a zoomed in version of the fragment map of the center image. The image on the right shows additional fragments, particularly more obvious in the yellow circle (seen as small cyan dots), that are generated to improve the accuracy of interpolating larger fragments.	93
5.7	Single pass upsampling receives fragments of different sizes and perform interpolation on them to produce a single irradiance map.	94
5.8	(a) Artifacts (dotted red ellipse) in the multi pass upsampling algorithm. Larger fragments are able to dominate pixel values and ignore smaller neighbouring fragments despite being less accurate. This artifacts appear as small holes or spikes near shadow boundaries. (b) Our single pass algorithm reduces these artifacts (not completely) by giving more weights to smaller fragments. (c) Both images in (a)(b) are rendered with the same fragment map based on Nichols et al. [9] refinement. As observed by the yellow dotted ellipse, sometimes a large fragment may fail to be refined.	96
5.9	Comparisons in per pixel square error with a Monte Carlo reference image of the Sponza scene (1280x960 pixels, 64 samples per fragment). Red color channel image on the right signifies the per pixel difference. The same fragment refinement and number of fragments are used for both renders. The errors have been scaled up to 9x for easier visualization. (a) Our adaptive variance scaling factor approach results in significantly smaller sum of square errors (1.112) than (7.482) our previous fixed variance scaling approach [3] in (b).	97

- 5.10 (a) Our upsampling scheme on a target blue fragment. The blue fragment is computed by using radial basis functions selected by samples closest to the two boundaries edge (in green) and its nearest fragment center. The cyan squares refer to the additional fragments that were generated for large fragments. (b) If the target texel falls within the top left of its parent fragment, the two edges in blue are traversed to look for neighbouring samples. Similarly if the fragment falls in the bottom right, the red edges are traversed. The similar can be said for the green and yellow fragments which falls in the top right and bottom left. The arrow indicates the direction to search for neighbouring samples. 100
- 5.11 In our transform feedback shader for shadow refinement, three streams of output are produced. The *TFS* receives fragments locations together with its ray tracing information. The first output stream (box in green outline) returns unstable fragments of mipmap level > 1 , which should be further refined in the next cycle. Stream 2 (box in red outline) stores corresponding screenspace 2D coordinates and visibility information as 2 output arrays. Stream 3 (box in blue outline) stores screenspace 2D coordinates of level 0 fragments. 102
- 5.12 Once our set of stable fragments is retrieved, fragments are converted to their 3D positions, x and normals N . Fragments larger than mipmap level 0 can reuse their visibility information from our SFVT to compute their irradiance based on the formulation for distributed ray tracing. Those at mipmap level 0 will need to perform a visibility test before their irradiance values can be computed. The fragments, with their irradiance values, will then be written to the appropriate mipmap level of textures via bindless image textures. 103

5.13	Rendering of the Sponza (McGuire Graphics Data) in 1280x960 pixels with direct illumination. In Figure 5.13d (center), we show the L2 error map (in red) in irradiance values (with scaling factor of 1x and 9x) between Figures 5.13a and 5.13c. Similarly Figure 5.13e (center) shows the L2 error map between Figures 5.13b and the reference image 5.13c. In Figure 5.13d,5.13e (right), we have an error color map indicating the percentage of pixel error. The errors are represented by five color zones from less than 5 % (blue) to more than 25 % (red). (f) shows the location and size of our planar light source.	105
5.14	Rendering of the Sibenik (McGuire Graphics Data) in 1280x960 pixels with direct illumination. In Figure 5.14d (center), we show the L2 error map (in red) in irradiance values (with scaling factor of 1x and 9x) between Figures 5.14a and 5.14c. Similarly Figure 5.14e (center) shows the L2 error map between Figures 5.14b and the reference image 5.14c. In Figure 5.14d,5.14e (right), we have an error color map indicating the percentage of pixel error. The errors are represented by five color zones from less than 5 % (blue) to more than 25 % (red). (f) shows the location and size of our planar light source.	106
5.15	Rendering of the hairball object (McGuire Graphics Data) in 1280x960 pixels with direct illumination. In Figure 5.15d (right), we show the L2 error map in irradiance values (with scaling factor of 1x and 9x) between Figures 5.15a and 5.15c. The error map is mapped to the red color channel. Similarly Figure 5.15e (right) shows the L2 error map between Figures 5.15b and the reference image 5.15c.	110

5.16	(a) The light configuration in the Sibenik scene. (b) A Monte Carlo reference is rendered for direct illumination from the area light source at 64 samples per fragment. (c) Rendering of our result at 64 samples per fragment. Visible random noise from under-sampling can be observed when the light is too large.	111
5.17	Experiments with varying area light sizes and their positions. Renderings are done in 1280x960 pixels. First row indicates 2 various sizes of lights that we used. Second row indicates the results of rendering for a light of size 2 for our result (c) and it's error map, at the top right, (scaled 20x) is computed by comparing against a Monte Carlo reference (d). Third row indicates the results of rendering for a light of size 3.5. Sum of square errors (NMSE) for the rendered images in (c) and (e) are 6.225 (4.87e-4) and 4.297 (3.59e-4). The bottom right image of (c),(e) shows the fragment map.	112
5.18	Plot of NMSE vs Light Size. The horizontal axis represent the one dimension length of the light, where the actual area of the lights used in the experiments are from 1^2 , 1.5^2 , to 3^2 . NMSE of our work is represented by the blue lines. Our error rate remains significantly lower than Nichols et al. [9]. Overall error in the image is unaffected by the light size.	113
6.1	Rendering of a water glass with a red scattering material illuminated with a single area light source inside the cup at 1024x768. We show our rendering in image (a) with both the direct and indirect illumination components, and a reference image generated by Mitsuba Path Tracer (64 samples per pixel) in image (b). The translucent material in the reference image is rendered with Photon Beam Diffusion method [77]. Image (a) is rendered in 146 ms. Image (b) is rendered in 54.96 seconds. Image (c) shows a diffuse water glass. No light is scattered to the exterior plane of the water glass.	115

6.2	The pipeline of our work with the indirect illumination combined with the direct illumination output.	117
6.3	Injecting transmitted and reflected light flux into ESLPV and LPV, respectively, with area lights. For diffuse materials, light is injected into the LPV voxel above the surface. For translucent materials, light is injected into the ESLPV voxel on the surface.	119
6.4	Injecting light intensities into a LPV with TPLs. Diffuse Poisson samples are represented in red circles while TPLs are represented in green circles. Each diffuse Poisson sample computes its reflected intensity by 'gathering' the illumination from TPLs.	122
6.5	Rendering of a Buddha object with a material property simulating an apple in images (a) and (b). We show our rendering in image (a) with both the direct and indirect illumination components, and the reference generated by Mitsuba Path Tracer (64 samples per pixel) in image (b) with the Photon Beam Diffusion method [77]. Image (b) is rendered in 28.97 seconds. Image (c) has been rendered with specular highlights.	127
6.6	Rendering of a Stanford bunny with material properties simulating jade in a colored Cornell box. We show our rendering in image (a) with both the direct and indirect illumination components, and the ground truth reference generated by Mitsuba Path Tracer (64 samples per pixel) and the Photon Beam Diffusion method [77] in image (b). Image (b) is rendered in 54.21 seconds. Image (c) is rendered with specular highlights.	128

6.7	Results of using the TPLs for injection produces a soft diffuse look from the green translucent Stanford dragon model. The scene is illuminated by a rectangular area light source from the top. Image (a) shows the indirect illumination. The walls and pillars of the lower level which are closer to the dragon appear much greener compared to the top level walls and pillars. The light source used in the scene is a small white rectangular area light. Images (b) and (c) show other rendered views with the direct and indirect illumination components. The soft shadow regions can be seen to exhibit indirect scattered illumination from the translucent material.	130
A.1	Two possible solid angles that can be subtended from the center of the source cell. Front facing solid angle (left) and side facing solid angle (right). The solid angle is the volume shaded in blue.	163
A.2	To compute the solid angle for the side facing case, we divide the solid angle to two tetrahedrons (green and blue) and use the solid angle computation for a tetrahedron.	164
A.3	Axis directions for calculating flux transfer for each face.	164
A.4	(a) A VPL is defined in the center of the voxel. (b) It is difficult to perform a ray trace from a refracted eye ray into the VPL. (c) The VPL is treated like an area light source on the surface of the material. (d) The area light source is assumed to be parallel to the surface orientation. Image obtained from [7]	168
A.5	Incident light is scattered in a particular distribution when it collides with a particle in the participating media. θ describes the angle which the scattering light direction makes with the incident light direction.	170

A.6	Plot of Henyey Greenstein Phase function (Magneta) against it's 2nd order SH reconstructed phase function (Blue) for $g = 0$. The function is plotted as pdf (y-axis) against scattering angle (in degrees) θ (x-axis). Both plots have almost identical values.	170
A.7	Plots of Henyey Greenstein Phase function (Magneta) against it's 2nd order SH reconstructed phase function (Blue) for $g < 0$. The function is plotted as pdf (y-axis) against scattering angle (in degrees) θ (x-axis).	171
A.8	Plots of Henyey Greenstein Phase function (Magneta) against it's 2nd order SH reconstructed phase function (Blue) for $g > 0$. The function is plotted as pdf (y-axis) against scattering angle (in degrees) θ (x-axis).	172

List of Tables

2.1	Definition and formulation of the diffusion profile, R_d	25
2.2	Comparisons regarding precomputations between different rendering techniques	46
2.3	Comparisons regarding storage requirements between different rendering techniques	47
2.4	Comparisons regarding visual accuracy between different rendering techniques	48
2.5	Comparisons regarding computation efficiency between different rendering techniques	49
2.6	Comparisons regarding visual accuracy between different rendering techniques	49
4.1	Storage requirements in megabytes (MB) for each phase of the ESLPV for a 32^3 voxel structure	75

4.2	Rendering time for some of the examples. We breakdown the respective time used for the stages in the ESLPV. Vox. refers to time required for voxelization. The column Inject (SS) refers to the time used for depositing virtual point lights in the ESLPV. Inject (MS) refers to the injection process identical to the SSLPV. Hier Prop refers to the Hierarchical Propagation in Section 4.1.3 that we used for all our rendering and DTCT has been used to accelerate this stage as well. SSLPV Prop refers to the time that the SSLPV uses for finishing the propagation stage. w/DTCT refers to the SSLPV Propagation accelerated with the DTCT. In renders where there are single-scattering, the rendering time is significantly larger due to ray marching. . . .	76
5.1	Rendering statistics for 1280x960 images. Figures 5.13a, 5.14a, 5.15a show our results, while Figures 5.13c, 5.14c, 5.15c are Monte Carlo references. Nichols et al.'s [9] result are shown in Figures 5.13b, 5.14b and 5.15b. L2 Error refers to the sum of square error when compared to the reference image. The normalized mean square error (NMSE) is also provided in the brackets after the L2 error. NMSE is represented using scientific notation. Each pixel is stored in normalized float format. A lower error indicates a better quality.	109
6.1	Rendering statistics for 1024x768 images. Direct Illum refers to the time required to render direct illumination (Chapter 5) only for diffuse surfaces. The remaining columns (Poisson Gathering, TPL Gathering, ESLPV, LPV) are timings for our work in this chapter.	129

List of Algorithms

- 4.1 Ray marching injection algorithm 67
- 4.2 Voxel discard algorithm 69

Abstract

Translucent materials exist all around us. They can be commonly found in materials such as marble, jade and fluids such as milk. Subsurface scattering describes one of the most complex yet beautiful effects of light in translucent materials. However, this light and media interaction is difficult to simulate. Although Monte Carlo approaches can simulate these effects, they are computationally expensive. Further complexities in simulation occur, when indirect illumination from translucent materials are considered, as they scatter light and illuminate other surfaces. In addition, when translucent materials are illuminated by area lights, fast techniques such as voxel based methods have difficulties in estimating direct visibility. In my research, I aim to develop efficient techniques for rendering homogeneous translucent materials under area light illumination at interactive speed.

In the first research of this thesis, I present a real-time approximation technique for subsurface scattering rendering that uses two separate voxel structures for rendering single scattering and multiple scattering effects. In my algorithm, a ray marching approach is used to traverse rays from the light source through the voxel structure, the Enhanced Subsurface Light Propagation Volumes (ESLPV), while leaving its radiance compressed into spherical harmonics in each marching step. During rendering, we use importance sampling to select important samples along the refracted ray. Some propagation optimization routines are used to remove unnecessary computations in the propagation iterations of Light Propagation Volumes (LPV) based algorithms and also accelerate the rendering time while maintaining the same visual quality. As the LPV is not a physically accurate algorithm, flux can be lost unnecessarily for each simulation step. I proposed a multi-resolution voxel structure to reduce the flux lost due to the numerous scattering events simulated in high resolution voxels and also preserve the radiance for backlighting at similar quality to the Photon Based Diffusion methods.

In the second research, I proposed a deferred multi-resolution approach for rendering direct illumination effects such as soft shadows when translucent and other opaque objects are illuminated by area lights. My approach subdivides the screenspace into multi-resolution 2D-fragments in which higher resolution fragments are created to represent geometric and depth discontinuities as well as shadow boundaries. I proposed a sub-fragment visibility test scheme that focuses on discontinuities within a fragment. A gradient aware subdivision scheme is developed to refine the discontinuities according to its gradients. My technique utilizes the stream-compaction feature of the transform feedback shader (TFS) in the graphics shading pipeline to filter out fragments for soft shadow refinement easily. A single pass screenspace irradiance upsampling scheme which uses radial basis functions (RBF) and an adaptive variance scaling factor is proposed for interpolating scattered fragments. This reduces artifacts caused by large fragments while also significantly reducing the number of visibility rays that we require.

In my final research, I combine the work of my first two researches and tackle two separate illumination problems for area lighting. First, the focus is on the appearance of translucent materials under illumination from area lights, which also casts soft shadows on underlying surfaces. Next, the translucent materials distribute this scattered lighting towards its surrounding surfaces. Next, we focus on the appearance of indirect illumination on the surrounding scene after receiving scattered light from translucent materials. To solve both direct and indirect illumination problems, we use a set of Poisson disk distributed samples on translucent objects and scene to gather illumination from area lights. The gathered illuminations are 'injected' as transmitted flux into the ESLPV to render the translucent objects. For simulating indirect illumination, the reflected flux are injected into the LPV and distributed.

The multi-resolution approach employed in this thesis provides accelerated rendering which can be used in various multimedia applications (e.g. games that require translucent object rendering or soft shadow rendering), and visual arts (e.g. pre-renders for rendering programs).

Keywords: translucent materials, subsurface scattering, photo-realistic, interactive rendering, area light illumination, direct illumination, indirect illumination.

Chapter 1

Introduction

1.1 Interactive rendering and translucent materials

Rendering is one of the most important concepts in 3D computer graphics. It is a process where an image is generated from a 2D/3D object or scene. It provides a final touch in visual appearance to a virtual object or scene by adding lighting, texture, shading and occlusion information. Rendering may be done as an off-line process, as seen in film animations, or can be implemented in real-time applications such as video games.

Interactive rendering is the art of using algorithms that allow a computer to produce images of a virtual environment or scene to an observer at interactive rates. The virtual environment or scene appears as a smooth stream of images on the computer, and the viewer can interact with these images providing feedback, manipulating the content of the next output image. Real-time rendering is often used in video games, but an improvement in computational power in GPU hardware have seen it used for a wide range of applications such as scientific visualization, real-time physical based renderers and particle based simulations.

Photo-realistic rendering is one of the key concepts in rendering. Its main purpose is to synthesize images that have its lighting effects and materials as close to real-life objects and environments. In simpler terms, we could say it is a process of synthesizing images that look

Figure 1.1: Common fields in photo-realistic rendering.

like photographs. The key to understanding how photo-realistic rendering algorithms work lies in how we analysed and approximate the physical behaviour of light with materials. Several fields of photo-realistic rendering, listed in Figure 1.1, are necessary in producing high quality renders.

Firstly, lighting affects the way we view the environment. There are basically two types of lighting, direct and indirect illumination. Direct illumination comes from light reflected from surfaces that are directly lit by the light source whereas indirect illumination is supplied from reflections from lit surfaces. The combination of direct illumination and indirect illumination is often known as global illumination. Both lighting types are affected by material properties as well as medium properties. For example, shiny opaque objects reflect more light than it absorbs. Light traveling through a fog tends to have its intensity attenuated more than light traveling through vacuum. Surfaces that receive direct and indirect illumination can exhibit hard and soft shadow depending on the size of the light source and its distance from occluders. Secondly, geometric details often contribute to the realism of

Figure 1.2: Photo-realistic image of a bedroom generated with 3DS Max.¹

the render. However, at times when geometric details are sacrificed for the need of performance, textures can be used to compensate for the lack of detailed geometry. Simple texture techniques such as bump mapping can be used to simulate a bumpy surface given a flat geometry. Lastly, for renders meant to simulate photographs, some additional rendering effects such as motion blur and lens flare exist to simulate a real camera.

Common applications of photo-realistic rendering can easily be found in the fields of animation, architectural design and education.

Photo-realistic rendering algorithms that are physically based are also the most computational intensive ones. Fortunately, with the enhancements and technology breakthrough in GPU technologies and rendering algorithms, many visually complex effects such as rendering area light illumination in Figure 1.2 [3] can now be generated exponentially faster compared to using a CPU path tracer. In this report, we are primarily focusing on interactive photo-realistic rendering of translucent materials, also known as subsurface scattering materials. We will also be further studying the illumination of translucent objects by area lights, and the distribution of indirect illumination caused by translucent materials.

Translucent materials are around us. They exist in solid materials such as marble, jade

¹URL: <http://themaisonette.net/interior-wallpaper-design-best-bedroom-designs-modern-bedroom-design>

Figure 1.3: An actual photograph of a horse model made of jade material. The model is backlit to show the diffuse glowing look from the front. ¹

(Figure 1.3 [4]) or wax, as well as fluids such as blood, milk or orange juice. They also exist in gases, such as clouds (Figure 1.5 [5]) or fogs. Scattering in air is similar to scattering in materials. Both of these phenomena can be explained with the radiative transport equation (described in Chapter 2). The presence of translucent materials can be easily seen in our environment, and their appearance is often characterized by a soft, diffuse appearance under direct illumination. For thin or highly translucent materials, there is often a backlighting effect that appears as a glow when the object is back-lit (Figure 1.4).

Rendering translucent materials is unfortunately a very sophisticated problem, since light can undergo scattering events at any point within the translucent materials. When simulating multiple scattering events, the complexity in rendering scales up. For this reason, many rendering algorithms make assumptions and reasonable approximations about the material properties. Complexity of the problem scales up when scattered light illuminates surrounding surfaces such as the one in Figure 1.4 [6].

The goal of this PhD research is to develop interactive light transport algorithms for rendering homogeneous translucent materials under area light illumination. The work in this report uses a combination of voxel based lighting methods [7, 8] and multi-resolution rendering [9] to produce a complete illumination pipeline for homogeneous translucent

¹URL: <https://www.flickr.com/photos/ironrodart/3904773382/>

Figure 1.4: An actual photograph of a 3D printing material with strontium aluminate. The material is translucent in nature and has a property which stores and releases light. Light emits out of the material simulating a black lit glow. The surrounding surfaces are illuminated by it. ¹

Figure 1.5: Light scattering in atmospheric gases causes a red glow in the sky. ²

materials. In addition, several other contributions to speed up on computation efficiency and also the rendering quality are explained in this thesis. We outline our major contributions in Section 1.3.

1.2 Challenges

As mentioned in the previous section, rendering translucent materials can be a sophisticated problem. This is because for the material to produce a soft diffuse glow like the 3d printing material in Figure 1.4, there might be millions of light interaction needed. The scientific

¹URL: <http://colorfabb.com/glowfill?acc=c81e728d9d4c2f636f067f89cc14862c>

²URL: <http://www.physicscentral.com/explore/pictures/images/sunset.jpg>

term used for this phenomenon is known as *subsurface scattering*. The most common standard ground truth technique used for subsurface scattering is the volumetric photon mapping solution [10]. This approach deposits multiple volume photons into the volume of the object. The final radiance is obtained by performing final gathering along sample points on the eye ray. The original implementation of the volumetric approach did not consider several variance reduction techniques and required a large amount of photons to eventually obtain a convergent solution. Several acceleration and variance reduction techniques such as Jarosz et al.'s beam radiance estimate [11], Photon Beams [12] and Novak et al.'s [13] virtual ray lights have been proposed for the volumetric photon approach. The new techniques accelerate the volumetric approach as well as reduce the number of volume photons needed to obtain the ground truth. However, as of now, none of them are able to produce results at interactive rates. The volumetric photon approach requires costly computations as well as large storage requirements. However, the rendering output undergoes a progressive refinement phase, which improves interactivity to a certain extent.

Despite numerous works in interactive translucent material rendering, subsurface scattering is still expensive to render especially when global illumination is involved. The following are four challenges in scaling up subsurface scattering for interactive rendering:

- **Precomputation Time** Several real-time techniques require a hefty amount of pre-computations to produce the final render. Some of the precomputations needed include geometry processing (Radiosity Techniques) and intensive sampling (Precomputed Radiance Transfer Techniques [14]) as well as compression. Precomputation based techniques are also limited to scenarios such as static lighting conditions and fixed material parameters. Any change of those limiting parameters would usually require another round of computations.
- **Interactive Material Updates** Material editing allows the user to change the material properties of the materials interactively. The user receives immediate feedback for any change of the material, and this will support an effective system for designing

rendering materials. Existing methods such as the dipole approximations, requires a large amount of sample points to be used, and these sample points have to be readjusted whenever the object's material properties changes. Most state-of-art techniques, such as precomputed radiance transfer or texture based methods, do not need to recompute sample points whenever there is a change of material properties. However, some other additional precomputations are usually required in existing techniques.

- **Interactive Area Lights Relighting** Although several methods are able to handle dynamic lights, several difficult light sources such as area light sources remain a challenge without precomputations.
- **Indirect Illumination** Each subsurface scattering material can be seen as an area light source as light gets scattered within the material and exits out in several directions. To produce realistic global illumination rendering, subsurface scattering objects that are illuminated have to be treated as light sources.

In this thesis, we propose methods to solve the problems mentioned above. We present the overview of our work in the following Section 1.3.

1.3 Summary of our research and contributions

The objective of our research is to develop efficient interactive subsurface scattering algorithms, such that our algorithms can handle material and lighting changes in interactive rates without any significant extra computations or precomputations. We also intend to use minimal storage requirements for rendering our translucent object so that the algorithm can be extended to larger scenes. When the translucent is placed in a large environment with area lighting, soft shadows are casted. A screenspace solution is proposed which tackles direct illumination from area lights for diffuse surfaces. A novel illumination approach is provided to illuminate the translucent object under area lights. We approximate the translucent object as an area light source after light gets scattered out of the translucent object and

illuminates surrounding objects. Our research will effectively handle all the challenges that have been mentioned.

1.3.1 Efficient rendering of translucent materials

We considered and solved the shortfall of voxel based methods by incorporating a single scattering injection procedure into the voxel structure using very little additional memory overhead and computation time. Together with the use of a compressed phase function, our voxel based method was able to incorporate the scattering effects created by phase functions in scattering materials. The single scattering framework is effectively designed as a two-pass technique.

We have developed an algorithm using distance transform on a conservative voxelized mesh for hastening the process of the Light Propagation Volumes (LPV) [8] in the propagation pass. This technique can be extended to every technique that is based on the LPV framework. This speedup allows us to perform additional rendering effects, such as single scattering, to enhance the realism of the translucent material. We also propose a hierarchical approach to improve the backlighting capability of the SSLPV [7]. This hierarchical approach improves rendering time significantly at the cost of very little additional memory.

We showed that it is possible to generate convincing results for translucent models using just 11MB of data. Using our technique, it is possible to render weakly scattering translucent materials interactively without needing large memory storage or precomputations.

Novelty Our voxelized solution is able to represent weakly scattering materials, where single scattering effects are more dominant, as well as highly scattering materials. Our method is designed to render single scattering effects with phase functions. It can also be used to render materials with strong scattering effects, which are more evident in backlighting. The new optimized algorithm also removes unnecessary computations that do not contribute to the final rendered image.

Applications In our work, we were able to simulate fast rendering of subsurface materials

without any precomputations. Our approach was able to render both the single scattering and multiple scattering components of translucent materials. Our work is suitable for real-time or interactive computer graphics applications.

1.3.2 Multi-resolution rendering under area light source

Direct illumination from area lights are known to vary smoothly across flat regions. Coarse sampling techniques, such as multi-resolution splatting by Nichols et al. [9, 15, 16, 17], were devised to take advantage of this property. Multi-resolution splatting proposes to dissect an image into patches known as fragments, where the fragment size depends on the depth, normal, and illumination variations within the patch. As illumination variation decreases, the illumination on a fragment can be represented using information from lower resolution fragments which reduces computation time. Our approach is extended from this.

Our final rendered image is reconstructed from a set of fragments. Our interpolation method gives greater weights to nearby higher resolution fragments such that texels that require interpolation can acquire more accurate values from higher resolution fragments near them. This also reduces artifacts related to the lack of refinement in visibility discontinuities. Though these artifacts are not present in Nichols et al. [9] due to their overly conservative diagonal refinement method, we show that those artifacts are reproducible when the number of fragments at these visibility discontinuities are reduced. Hence our approach is able to render high quality shadow boundaries with fewer fragments.

Novelty We devised a few methodologies that are able to approximate transitions in illumination with fewer fragments, which leads to faster rendering speed. Our scattered data interpolation technique significantly reduces errors caused by under-refinements while also having significantly lower error rates compared to former interpolation techniques. A new improved stream-compaction feature, which was not present in previous work, is designed to effectively speed-up the fragment generation process.

Applications Our screenspace technique is suitable for applications which requires big

scene rendering with area lights. It is capable of rendering and detecting areas with partial visibility from the light source and effectively re-lighting them.

1.3.3 Rendering translucent materials under area light source

We distribute a set of Poisson disk samples around the translucent objects. This set of Poisson disk samples collects lighting information (transmitted flux) as well as injects lighting information into our voxel structure (ESLPV) for rendering translucent materials. This set of Poisson disk samples is later downsampled and represented as translucent planar lights (TPL). This allows the translucent materials to be represented as area light sources with multiple planar lights. This set of downsampled planar lights allows for distributing illumination to diffuse surfaces.

Similarly, another set of Poisson disk samples is distributed around diffuse surfaces. This set of samples collects lighting information from area lights as well as translucent objects (secondary light sources) and injects illumination into the Light Propagation Volumes (LPV).

Novelty Our Poisson disk samples injection technique is a low cost solution for injecting energy into a voxel structure for indirect illumination. Computation time required from sparse set of Poisson disk samples is kept low to allow for interactive rendering. It also requires injecting fewer samples than shadow map based methods.

Applications Technique is suitable for rendering any scene containing translucent and diffuse objects illuminated by a generic diffuse area light.

1.4 Thesis organization

The organization of this thesis is as follows. In Chapter 2, we provide a literature review which explains some related work in computer graphics, i.e. general rendering definitions (radiance and global illumination definition), subsurface scattering and multi-resolution rendering. In Chapter 3, we devise our hypothesis and observations on achieving our re-

search goals. In Chapter 4, we present our research including the implementation and results of interactive subsurface scattering rendering under direct illumination from point lights and spotlights. In Chapter 5, we present our work in multi-resolution rendering for area lights. In Chapter 6, we extend the rendering of subsurface scattering materials to area lights using the framework provided in Chapter 4. Finally in Chapter 7, we provide conclusions and propose future work.

Chapter 2

Literature Review

2.1 Fundamentals of radiometry and light transport

Computer Graphics usually undertakes ray optics as its fundamental model for photo-realistic rendering. The assumptions made by ray optics generally describe the behaviour of light in a way that it can only be emitted, reflected, transmitted and scattered. Light is also assumed to travel in straight lines. When dealing with subsurface scattering, radiometric equations such as irradiance and flux control how much light enters the surface of the translucent material. Volumetric rendering equations on the other hand, control how light is scattered or absorbed in translucent materials. As subsurface scattering is a special case of volumetric rendering, the important visual effects in subsurface scattering can be reproduced by simplifying assumptions such as single and multiple scattering. Hence in this chapter, we explain the basic radiometric quantities leading to the rendering equations, followed by the volumetric equations for rendering light in participating media. In subsequent sections, we describe how the volumetric equations are simplified to approximate single and multiple scattering.

Figure 2.1: Flux describes the amount of light energy from all directions that hits a surface of finite area. Irradiance measures the amount of flux that hits a unit surface area around x . Radiance describes the amount of light moving in or out from x within a differential solid angle. The rotated cylinder in the above image represents the differential area that is perpendicular to the direction of the radiance

2.1.1 Radiometric quantities

Flux

Flux is a measurement for the total amount of energy that passes through a surface of finite area over a unit time. The quantity used for flux is Watts (W) or Joules per second (Js^{-1}). It is commonly denoted by the symbol Φ . (Refer to Figure 2.1 (left))

Irradiance

Irradiance (E) expresses the amount of energy that hits a unit surface area. Irradiance can also be seen as a measurement of flux arriving at a surface location. Radiant exitance refers to the amount of energy that leaves a unit surface. The measurement unit for irradiance and radiant exitance (M) is W/m^2 . The irradiance at any point inside an area A , given that the incoming light direction is parallel to the normal of the surface, is then

$$E = \Phi/A \quad (2.1)$$

The amount of flux reaching a surface is proportional to the cosine of the angle between the light direction and the surface normal. This is known as Lambert's law. Hence, if the

light direction is at an angle θ to the surface normal and the light source has a constant distribution of energy, irradiance at any point inside A , can be computed by:

$$E = (\Phi \cos \theta) / A \quad (2.2)$$

In cases where the flux distribution is not constant, we used the differential flux from the light over the differential area receiving the flux, hence the irradiance at any point x inside Area A is

$$E(x) = \frac{d\phi(x)}{dA(x)} \quad (2.3)$$

Radiance

Radiance (L) is the most important quantity in rendering. It refers to the flux arriving from a differential direction onto a surface with differential area perpendicular to the direction. (Refer to Figure 2.1 (right)).

$$L = \frac{d\phi}{(d\vec{\omega} dA^\perp)} \quad (2.4)$$

In practice, we should always use the radiance measured at the actual surface instead of a hypothetical surface perpendicular to the light flow. Hence $dA^\perp = dA(\vec{n} \cdot \vec{\omega})$ should be used to further evaluate the above equation.

$$L = \frac{d\phi}{(\vec{n} \cdot \vec{\omega}) d\vec{\omega} dA} \quad (2.5)$$

Radiance is the most important quantity used to render a pixel of a 3D environment. In simpler terms, when dA is small, radiance can be assumed as the amount of energy a single ray of light carries.

Incoming Radiance and Exitant Radiance

When discussing radiance at a point in an environment, it is important to differentiate between radiance arriving at a point or radiance leaving a point. In this paper, we use the function $L_i(x, \vec{\omega})$ to define radiance arriving at a point x in the direction $\vec{\omega}$, and $L_o(x, \vec{\omega})$ to define radiance exiting at a point x in the direction $\vec{\omega}$.

2.1.2 Fresnel

When a light moves from a medium of refractive index η_i to another medium of refractive index η_t , both reflection and refraction of the ray may occur. The Fresnel equation describes the amount of light energy that is reflected or transmitted through a medium of differing refractive indices. The Fresnel equations is defined using the incident angle of incoming light and the refracted angle of the light in the medium, which is computed using Snell's law as shown below.

$$\eta_i \sin \theta_i = \eta_t \sin \theta_t \quad (2.6)$$

The relative power of the light reflecting at the surface of the material is governed by:

$$R_{\parallel} = \frac{\tan(\theta_i - \theta_t)}{\tan(\theta_i + \theta_t)} \quad (2.7)$$

$$R_{\perp} = \frac{\sin(\theta_i - \theta_t)}{(\theta_i + \theta_t)} \quad (2.8)$$

where $R_{\parallel}$ indicates the light parallel to the surface, and $R_{\perp}$ indicates the light reflected that is perpendicular to the surface. In this report, we assume we are dealing with unpolarized light and hence, the expression for the Fresnel reflectance becomes:

$$R_{\theta} = 0.5 * (R_{\parallel} + R_{\perp}) \quad (2.9)$$

Figure 2.2: Reflection and Refraction of Light.

The Fresnel transmission term can be computed by taking the amount of energy that is not reflected.

$$T = 1 - R_\theta \quad (2.10)$$

With reference to Figure 2.2, the radiance of the refracted and reflected light are multiplied with the Fresnel Transmission term, T , and the Fresnel Reflectance term, R .

2.1.3 Surface BRDF

The bidirectional reflectance distribution function (BRDF) describes the ratio of reflected radiance to the irradiance incident of a point x on the surface. The BRDF can be described using a six-dimensional function.

$$f_r(x, \vec{\omega}, \vec{\omega}') = \frac{dL_o(x, \vec{\omega})}{(L_i(x, \vec{\omega}')(\vec{n} \cdot \vec{\omega}')d\vec{\omega}')} \quad (2.11)$$

where $dL_o(x, \vec{\omega})$ is the reflected radiance differential in the direction $\vec{\omega}$, $L_i(x, \vec{\omega}')$ is the incident radiance from direction $\vec{\omega}'$ and $\vec{n}$ is the normal of the surface at x . The term

$(\vec{n} \cdot \vec{\omega}')$ accounts for the attenuation of light at glancing angles. The BRDF can be further rearranged to form the rendering Equation 2.13.

$$dL_o(x, \vec{\omega}') = f_r(x, \vec{\omega}, \vec{\omega}') L_i(x, \vec{\omega}') (\vec{n} \cdot \vec{\omega}') d\vec{\omega}' \quad (2.12)$$

The total exitant radiance can then be computed as the integral of incident radiance contributions over the hemisphere of directions above x .

$$L_o(x, \vec{\omega}') = \int_{2\pi} f_r(\vec{\omega}, \vec{\omega}') L_i(x, \vec{\omega}') (\vec{n} \cdot \vec{\omega}') d\vec{\omega}' \quad (2.13)$$

The BRDF function also obeys Helmholtz reciprocity principle. The BRDF remains unchanged if the incident and outgoing light directions are swapped.

$$f_r(x, \vec{\omega}', \vec{\omega}) = f_r(x, \vec{\omega}, \vec{\omega}') \quad (2.14)$$

2.1.4 Phase functions

When dealing with participating media, the BRDF cannot account for the outgoing radiance if light is allowed to enter a medium. Light usually undergoes scattering in participating media, a process where a light ray is split into multiple rays which deviate from the original incident direction due to collision with particles in the medium. The phase function accounts for the amount of light from an incoming direction being scattered in a particular direction within a participating medium. For isotropic materials, the phase function is usually a constant term, indicating an equal amount of scattering in all directions. The most commonly used phase function is defined in Equation 2.15.

$$p_{isotropic}(\vec{\omega}, \vec{\omega}') = 1/4\pi \quad (2.15)$$

One of the more commonly used phase functions was developed by Henyey and Greenstein (Henyey and Greenstein 1941) to model materials containing random varying size

Figure 2.3: The parameter g in the Henyey Greenstein Phase function controls the angular distribution of light at any point x within the participating media. Image obtained from Jarosz [1].

particles.

$$p_{HG}(\vec{\omega}, \vec{\omega}') = \frac{1}{4\pi} \frac{(1 - g^2)}{(1 + g^2 - 2g\vec{\omega} \cdot \vec{\omega}')^{3/2}} \quad (2.16)$$

In this phase formulation, g determines the level of anisotropy. A positive value of g indicates forward scattering, while a negative value of g indicates backward scattering. When $g = 0$, the Henyey Greenstein Phase function is equivalent to the isotropic phase function (refer to Figure 2.3).

2.1.5 Radiative transport equation

Translucent materials are often made up small particles of varying size. When a translucent material is hit by light, two possible events: scattering and absorption may occur. Some light may get reflected off into another direction, while some suffer a reduced intensity due to light being absorbed by material. The absorption and scattering coefficient are denoted by the symbols σ_a and σ_s . The coefficients describe the amount of light being absorbed or scattered per unit volume.

As the beam of light moves through a small distance dz in the direction $\vec{\omega}$ through the medium, then the change of intensity due to scattering and absorption along that distance is

$$dL(x', \vec{\omega}) = -dz(\sigma_s + \sigma_a)L = -dz\sigma_t L \quad (2.17)$$

The term $\sigma_t = \sigma_s + \sigma_a$ is called the extinction coefficient, which acts as a total attenuation coefficient for the material. As a beam of light moves through a material, its intensity will decrease by an extinction amount. However, scattered light from other parts of the medium would also contribute to an intensity increase to a particular point in the medium. We can represent this in-scattered contribution by integrating the contributions over a sphere of incident directions and scaling the intensity by the phase function p .

$$L_{in}(x', \vec{\omega}) = \int_{4\pi} p(\vec{\omega}, \vec{\omega}') L(x, \vec{\omega}') d\vec{\omega}' \quad (2.18)$$

The in-scattered radiance Equation 2.18 needs to be scaled by a scattering factor in the direction dz .

$$L_{in} = \sigma_s dz \int_{4\pi} p(\vec{\omega}, \vec{\omega}') L(x', \vec{\omega}') d\vec{\omega}' \quad (2.19)$$

2.1.6 Global illumination formulation

Rendering global illumination typically involving solving Kajiya's rendering Equation [18];

$$L_o(\mathbf{x}, \tilde{\omega}_o) = L_e(\mathbf{x}, \tilde{\omega}_o) + \int_{\Omega} L_i(\mathbf{x}, \tilde{\omega}_i) f_r(\mathbf{x}, \tilde{\omega}_o, \tilde{\omega}_i) \cos(\mathbf{n}, \tilde{\omega}_i) d\tilde{\omega}_i, \quad (2.20)$$

where L_o is the outgoing radiance from a point $\mathbf{x}$ on a surface with direction $\tilde{\omega}_o$, L_e is the emitted radiance from point $\mathbf{x}$ to direction ω_o , L_i is the incoming radiance to point $\mathbf{x}$ from direction ω_i , f_r is the Bidirectional Reflectance Distribution Function (BRDF), and $\mathbf{n}$ is the normal at point $\mathbf{x}$ on the surface.

The outgoing radiance from a point on a surface is the sum of the emitted radiance of that point and the integration of the reflection from the incoming radiance from the upper hemisphere. Each incoming radiance may come from other point in the scene, thus Equation 2.20 is basically a recursive equation. The BRDF term f_r specifies the amount of light reflected from a point $\mathbf{x}$ to a direction given an incoming light ray.

2.2 Translucent materials rendering

In this chapter, we first explain the fundamentals of the Bi-directional surface scattering distribution function (BSSRDF) which is specifically used in translucent material rendering in Section 2.2.3 and its analytic solution using the dipole approximation in Section 3.2. In the remaining sections, real-time techniques developed for subsurface scattering will be described.

2.2.1 Fundamentals of bi-directional surface scattering distribution function (BSSRDF)

The BSSRDF, S , relates the outgoing radiance, $L_o(x_o, \vec{\omega}_o)$ at point x_o in direction $\vec{\omega}_o$ to the change of incident flux $d\phi_i(x_i, \vec{\omega}_i)$ at point x_i from direction $\vec{\omega}_i$.

$$dL_o(x_o, \vec{\omega}_o) = S(x_i, \vec{\omega}_i, x_o, \vec{\omega}_o) d\phi(x_i, \vec{\omega}_i) \quad (2.21)$$

The difference between the BRDF and BSSRDF (Figure 2.4) is that the computation for BRDF assumes that light enters and leaves the material at the same point while the BSSRDF describes that light enters at a point and undergoes multiple scattering events before exiting at another point. To compute the outgoing radiance for the BSSRDF, we would need to integrate the incident radiance over all incoming directions as well as the area, A , around the incident radiance as described by Equation 2.22.

Figure 2.4: Scattering of light in BRDF (left) and BSSRDF (right). In the BSSRDF, light can enter a point; undergo many scattering events, before exiting out at another point.

$$L_o(x_o, \vec{\omega}_o) = \int_A \int_{2\pi} S(x_i, \vec{\omega}_i, x_o, \vec{\omega}_o) L_i(x_i, \vec{\omega}_i) (\vec{n} \cdot \vec{\omega}_i) d\vec{\omega}_i dA(x_i) \quad (2.22)$$

Jensen et al. [19] modelled the BSSRDF model as a sum of the diffusion scattering S_d term and the single scattering term $S^{(1)}$ (explained in section 2.2.2) as shown in Equation 2.23. The diffusion scattering term is also known as the multiple scattering term.

$$S(x_i, \vec{\omega}_i, x_o, \vec{\omega}_o) = S_d(x_i, \vec{\omega}_i, x_o, \vec{\omega}_o) + S^{(1)}(x_i, \vec{\omega}_i, x_o, \vec{\omega}_o) \quad (2.23)$$

Stam [20] proposed using a multi grid scheme to solve the diffusion term numerically. However, their method suffers from high computational costs and storage requirements. Several other techniques have been derived to solve the diffusion equation in real time, the most common one being the dipole approximation. In the remaining of this chapter, we give a literature review on previous techniques used for real-time subsurface scattering.

2.2.2 Single scattering

Blinn scattering model

Blinn [21] was the first to introduce the single scattering model in computer graphics. His model describes that a ray of light will be visible in the model if there are no obstruction to the light path as it exits the material, shown in Figure 2.5.

The amount of incident light that reaches depth z at point x along the path can be computed as:

$$\int_0^z L(x, \vec{\omega}_i) e^{-\tau/(\cos\theta_i)}, \quad (2.24)$$

where τ is the optical depth and $\cos\theta_i = \vec{\omega}_i \cdot \vec{n}_i$ is the geometric term on the incident surface. The amount of light that will scatter back along the exitant path will be:

$$L(x, \vec{\omega}) = \sigma_s \int_0^z \rho(\vec{\omega}_i, \vec{\omega}_o) L(x, \vec{\omega}_i) e^{-\tau/\cos\theta_i} e^{-\tau/\cos\theta_o}, \quad (2.25)$$

Figure 2.5: Blinn's single scattering model. The exitant radiance $\vec{\omega}_o$ can be seen if there are no particles blocking the light path (blue cylinder area).

where $\cos\theta_o = \vec{\omega}_o \cdot \vec{n}_o$.

Jensen et al.[19] devised a single scattering model similar to Blinn's, with the exception that the Fresnel term was encompassed into the equation. The single scattering light path was also changed to a refracted light path, shown in Figure 2.6. The radiance on the viewing ray is obtained by performing ray marching along the refracted incoming and outgoing light in the material.

Figure 2.6: Jensen et al.'s [19] single scattering model included sampling along the refracted light path as well as taking consideration of the two Fresnel term on the incident and exitant surface

Figure 2.7: The left figure indicates the searchlight problem [22], where light refracts through the medium and travels along the refracted direction until it ultimately undergoes scattering events and finally exits the material. The right figure shows the analytic solution using the dipole model Jensen et al. [19], where the multiple scattering events are represented using an isotropic point source. The mirrored negative source is used to account for the boundary conditions

2.2.3 Dipole method

The problem of determining the exitant radiance at all positions and directions on an infinitely flat surface due to illumination from a single directional focused beam light is known as the searchlight problem [22] in Figure 2.7. Monte Carlo methods and photon mapping can produce accurate solutions to the searchlight problems, but they take extremely long to converge. It was later discovered by Stam [20] that diffusion approximation solutions proved to be a good approximation for the searchlight problem. They observed that the light distribution in highly scattering materials tend to be isotropic.

Jensen et al. [19] introduced a diffusion dipole as an approximation to the diffusion equation. The dipole model approximates the light distribution and scattering events of the beamlight in the medium by placing an isotropic point light source at a distance of one mean free path from the surface of the medium. To properly account for the boundary conditions, another virtual point light source is mirrored a distance away on top of the surface. Figure 2.7 shows the searchlight problem and the dipole model.

The diffusion term S_d in Equation 2.23 was reformulated as:

$$S_d(x_i, \vec{\omega}_i, x_o, \vec{\omega}_o) = \frac{1}{\pi} F_t(x_i, \vec{\omega}_i) R_d(r) F_t(x_o, \vec{\omega}_o) \quad (2.26)$$

where F_t is the Fresnel transmittance at the entry and exit locations x_i and x_o , and r is the Euclidean distance between x_i and x_o . The term R_d is the diffuse reflectance term which is approximated by the diffusion dipole. R_d can be seen as a diffusion term that represents the falloff of light intensity at r distance away from x_i . A full derivation of the diffusion approximation is not within the scope of this chapter, but it can be found in Chapter 5 of Craig Donner's Thesis [23]. We instead, only explain the main ideas of their formulation. The dipole approximation suggest that using two sources (as shown in Figure 2.8) above and below the target position x_i , and summing their illumination contribution to the surface, we are able to get the diffusion profile in terms of a function represented by the distance $r = \|x_i - x_o\|$. The full equation of the diffusion dipole R_d is shown in Table 2.1.

Figure 2.8: An incoming ray is transformed into a dipole source and used to approximate the diffusion reflectance term. A real source is placed within the object at a distance z_r below the surface. Another virtual source is at placed $z_r + 2AD$ above the surface

We substitute R_d back into Equation 2.26 to solve for the BSSRDF. The outgoing ra-

Table 2.1: Definition and formulation of the diffusion profile, R_d

$$\begin{aligned}
R_d(r) &= \frac{\alpha'}{4\pi} \left[\frac{z_r(1+\sigma_{tr}d_r)e^{-\sigma_{tr}d_r}}{d_r^3} + \frac{z_v(1+\sigma_{tr}d_v)e^{-\sigma_{tr}d_v}}{d_v^3} \right] \\
z_r &= 1/(\sigma'_t), z_v = z_r + 4AD \\
d_r &= \sqrt{(r^2 + z_r^2)}, d_v = \sqrt{(r^2 + z_v^2)} \\
A &= \frac{(1+F_{dr}(\eta))}{(1-F_{dr}(\eta))} \\
F_{dr} &= -1.440/\eta^2 + 0.710/\eta + 0.668 + 0.0636\eta \\
D &= \frac{1}{(3\sigma'_t)}, \sigma_{tr} = \sqrt{3}\sigma_a\sigma'_t \\
\sigma'_t &= \sigma_a + \sigma'_s \\
\alpha' &= \sigma'_s/\sigma'_t
\end{aligned}$$

diancance function in Equation 2.22 can be simplified by changing area integrals to patch integrals. We first substitute Equation 2.26 into 2.22 and we have:

$$L_o(x_o, \vec{\omega}_o) = \int_A \int_{2\pi} \left(\frac{1}{\pi} F_t(\eta_o, \vec{\omega}_o) R_d(\|x_i - x_o\|) F_t(\eta_i \vec{\omega}_i) L_i(x_i, \vec{\omega}_i) (\vec{n} \cdot \vec{\omega}_i) d\vec{\omega}_i dA(x_i) \right) \quad (2.27)$$

The outgoing Fresnel term $F_t(\eta_o, \vec{\omega}_o)$ can be moved out of the integral, and the R_d term into just the area integral.

$$L_o(x_o, \vec{\omega}_o) = \frac{1}{\pi} F_t(\eta_o, \vec{\omega}_o) \int_A R_d(\|x_i - x_o\|) \int_{2\pi} F_t(\eta_i, \vec{\omega}_i) L_i(x_i, \vec{\omega}_i) (\vec{n} \cdot \vec{\omega}_i) d\vec{\omega}_i dA(x_i) \quad (2.28)$$

The inner integral in 2.28 can be approximated by the product of the irradiance E and diffuse Fresnel transmittance $F_{dt}(\eta_i)$.

$$\int_{2\pi} F_t(\eta_i, \vec{\omega}_i) L_i(x_i, \vec{\omega}_i) (\vec{n} \cdot \vec{\omega}_i) d\vec{\omega}_i \approx F_{dt}(\eta_i) E(x_i, \vec{n}) \quad (2.29)$$

However, in some literature, the $F_{dt}(\eta_i)$ term is usually ignored leaving only the irradiance term. Equation 2.28 can now be approximated further by moving the $F_{dt}(\eta_i)$ term out of the area integral.

$$L_o(x_o, \vec{\omega}_o) = \frac{1}{\pi} F_t(\eta, \vec{\omega}_o) F_{dt}(\eta) \int_A R_d(\|x_i - x_o\|) E(x_i, \vec{n}) dA \quad (2.30)$$

The remaining terms in the integral over the area gives the radiant exitance B_o at the point x_o due to subsurface scattering from irradiance. We can consider approximating the radiant exitance B_o with a sum of j area elements centered at x_j with Area A_j and average irradiance E_j giving :

$$B_o(x_o) = \int_A R_d(\|x_i - x_o\|) E(x_i, \vec{n}) dA \approx \sum_j R_d(\|x_j - x_o\|) E_j A_j \quad (2.31)$$

With this simplification, the final outgoing radiance can be expressed as

$$L_o(x_o, \vec{\omega}_o) \approx \frac{1}{\pi} F_t(\vec{\omega}_o) F_{dt}(x_o) B_o(p_o) \quad (2.32)$$

Jensen et al. [24] developed an efficient octree structure to evaluate $B_o(x_o)$. Hao et al. [25] suggested modifying all light sources to be directional lights. This removes the incident lighting direction integral term therefore allowing Equation 2.28 to be re-arranged into:

$$F_t(\eta_o, \vec{\omega}_o) \vec{\omega}_i \cdot \int_A \frac{1}{\pi} R_d(\|x_i - x_o\|) F_t(\eta_i, \vec{\omega}_i) L_i(x_i, \vec{\omega}_i)(\vec{n}) dA(x_i) \quad (2.33)$$

The integral in Equation 2.33 can be precomputed in a pre-processing stage for each vertex. In cases where there are multiple lights, the integral have to be computed for every light source, and their contribution to each vertex is accumulated.

2.2.4 Mesh based approaches

Radiosity

Radiosity [26] is a geometrical based approach originally used for global illumination. Rushmeier et al. [27] later proposed a radiosity approach for calculating intensities in volumes of participating media, and subsequently for subsurface scattering materials [28]. In recent years, several techniques for subsurface scattering have based their algorithm on radiosity. Lensch et al. [29] formulated the subsurface scattering as a combination of a global and local response. The global response referred to radiance contributed from large distances (lots of scattering) while the local response was formulated as radiance contributed from short distances (little scattering). They formulated the global response by using a vertex to vertex throughput factor and the local response with a texel to texel throughput factor with information from a texture atlas. However, their result requires a large number of form factors which may in turn require a large amount of computation as well as storage.

Carr et al. [30] improved Lensch et al.'s [29] method by using a multi-resolution mesh atlas which divides the surface into multiple patches. They reformulated the radiant exitant B_o from Equation 2.31 as a sum of inter-patch contribution, known as radiosity, as shown in Equation 2.34.

$$B_i = \sum_{j=1}^N S_{ij} E_j \quad (2.34)$$

where S_{ij} is the form factor defining the throughput between patch i and patch j and E_j is the irradiance on patch j . The form factor for a patch can be computed as a function of the dipole function R_d (Table 2.1) and the area of the patch A_i :

$$S_{ij} = \frac{1}{A_i} \int_{x_i \in A_i} \int_{x_j \in A_j} R_d(\|x_j - x_i\|) dx_j dx_i \quad (2.35)$$

Mertens et al.[31] suggested using a hierarchy of clustered triangles similar to hierarchical radiosity. Carr et al. [30] used a hierarchical mipmap to store the form factors enabling

faster integration of radiosities depending on the Level of Detail (LOD) information needed. Similar to Carr et al. [30], Mertens et al. [31] also computed the form factor between pair of patches instead of pair of vertices.

Yu et al. [32] extended the radiosity method for subsurface scattering to support multiple bounce global illumination. They devised two separate form factors, S_{ij} for internal subsurface scattering (contribution between patches within the material) and F_{ij} for free space propagation (contribution between patches outside the material). The light emitted by patch j on the first lighting hit is computed using $\sum_i S_{ji}E_i$, which describes the contribution from every other patch i . For the second light bounce, the light received by patch k from every other patch j in free space is $\sum_j F_{kj} \sum_i S_{ji}E_i$. In matrix form, the radiosity, $\mathbf{B}$, of every patch, can be represented as

$$\mathbf{B} = \mathbf{FSE}, \quad (2.36)$$

where $\mathbf{F}$ and $\mathbf{S}$ are $n \times n$ matrices and n is the total number of patches. In general, more light bounces can be added to the scene by modifying Equation 2.36 to:

$$\mathbf{B} = (\mathbf{SF})^{N-1}\mathbf{SE}, \quad (2.37)$$

where N is the number of bounces. The entire radiosity of the scene can then be represented as the sum of all bounces of light, which has to be solved by iterative methods such as Jacobi Iteration or Gauss-Seidel:

$$\mathbf{B} = \sum_{N=1}^{\infty} (\mathbf{SF})^{N-1}\mathbf{SE} = (\mathbf{I} - \mathbf{SF})^{-1}\mathbf{SE} \quad (2.38)$$

Radiosity methods are very mesh dependent and the time needed to generate links and creating form factors can be considerably large for highly triangulated models. Storage requirements may also be large depending on the number of form factors used. Yu et al. [32] used approximately 2.7GB of memory for a 72k polygon model. This amount of

memory requirement is exceptionally large and can only fit into high-end GPUs.

Polygon based systems

Wang et al.[33] represented the object volume using a polygrid structure of nodes, with six connections among each node inside the volume. They then used the polygrid representation to efficiently discretize the light diffusion equation. Each node will basically propagate its intensity to its neighbouring nodes until a convergent solution is reached. However, their method was meant for simple geometry. Although not specifically mentioned in their paper, their time for mesh construction was not an automated process and can take hours to be completed.

Wang et al. [34] and Adam et al. [35] devised a tetrahedralization technique for discretizing the mesh into a 4-connected mesh structure. Their technique is able to handle much more complex geometries, especially those with thin features. However, the time needed to process the mesh, though automated, may still take minutes.

2.2.5 Precomputed radiance transfer (PRT)

PRT was originally developed by Sloan et al. [36] for real-time rendering of diffuse and glossy objects in low-frequency lighting environment. The technique precomputes a light transport matrix/vector (vector for diffuse surfaces, matrix for arbitrary BRDF) for each point on the object and they are compressed with low-order spherical harmonics (SH) basis. During render time, the evaluation of radiance for each sample point (Equation 2.39), reduces to simple dot product between the SH coefficients of the lighting environment and the SH coefficients of the compressed light transport matrix/vector.

PRT reformulates the exitant radiance Equation 2.13 into Equation 2.39 by adding a visibility term $V(x, \vec{\omega})$, which indicates the direct light visibility between the light source and the point x :

$$L_o(x, \vec{\omega}_o) = \int_{2\pi} L(x, \vec{\omega}) V(x, \vec{\omega}) f_r(x, \vec{\omega}, \vec{\omega}_o) H_N(x, \vec{\omega}) d\vec{\omega} \quad (2.39)$$

$$H_N(x, \vec{\omega}) = \max(\vec{n} \cdot \vec{\omega}, 0) \quad (2.40)$$

$$L(x, \vec{\omega}) = \sum_i l_i y_i(\vec{\omega}) \quad (2.41)$$

$$\begin{aligned} L_o(x, \vec{\omega}_o) &\simeq \int_{2\pi} \sum_i l_i y_i(\vec{\omega}) V(x, \vec{\omega}_i) f_r(x, \vec{\omega}, \vec{\omega}_o) H_N(x, \vec{\omega}) d\vec{\omega} \\ &\simeq \sum_i l_i \int y_i(\vec{\omega}) V(x, \vec{\omega}_i) f_r(x, \vec{\omega}, \vec{\omega}_o) H_N(x, \vec{\omega}) d\vec{\omega} \end{aligned} \quad (2.42)$$

The inner integral in Equation 2.42 can be evaluated as a projection of the product of the visibility, BRDF f_r and cosine term H into a set of combined SH coefficients M_i , where y_i is the corresponding SH basis functions. The SH coefficients M_i can also be considered as a light transport vector which represents a linear transformation of the input lighting vector to an output radiance function. The entire spherical radiance can then be evaluated as the dot product of the light transport vector coefficients and the incident lighting as shown in Equation 2.43.

$$(L_o(x))_i = \sum_{i=1}^{n^2} (M(x))_i l_i \quad (2.43)$$

Wang et al. [37] and Jiawan et al. [38] extended the PRT technique to handle subsurface scattering materials. The multiple scattering equation in 2.28 is reformulated to incorporate a visibility term as shown by Equation 2.46:

$$L_o(x_o, \vec{\omega}_o) = \frac{1}{\pi} F_t(\eta, \vec{\omega}_o) \int_{2\pi} L(\vec{\omega}_i) T_d(x_o, \vec{\omega}_i) d\vec{\omega}_i \quad (2.44)$$

$$T_d(x_o, \vec{\omega}_o) = \int_A R_d(\|x_i - x_o\|) E(x_i, \vec{\omega}_i) dA(x_i) \quad (2.45)$$

$$E(x_i, \vec{\omega}_i) = F_t(\eta, \vec{\omega}_i) V(x_i, \vec{\omega}_i) (\vec{n} \cdot \vec{\omega}_i) \quad (2.46)$$

The irradiance transport function $E(x_i, \vec{\omega}_i)$ is first computed for all directions at each vertex of the object. The transport functions are then compressed using wavelets and stored as a vector of coefficients into a K-d tree similar to Jensen et al.'s [24] hierarchical BSS-RDF technique. The multiple scattering transport vector T_d is evaluated by accumulating the compressed irradiance transport vectors from the lower nodes in the K-d tree. Further compression can be done by only keeping a certain number of wavelet coefficients. Re-lighting then reduces to a dot product between the lighting vector and the transport vector:

$$L_d(x_o, \vec{\omega}_o) = \frac{1}{\pi} F_t(\eta, \vec{\omega}_o) \sum_j T_d^w(x_o, \vec{\omega}_j) L^w(\vec{\omega}_j), \quad (2.47)$$

where T_d^w and L^w are the wavelet coefficients vectors of the multiple scattering transport function and incident lighting function.

Several extensions have been done to PRT techniques to allow material editing. Wang et al. [39] suggested using a precomputed material property set R_d and compressing them using Principal Component Analysis (PCA) followed by another compression using wavelets. Several other basis functions have been proposed for R_d , such as linear piecewise polynomials [40] and Gaussian basis functions [41]. We show a summary of standard PRT technique in Figure 2.9.

2.2.6 Texture/Screenspace methods

Texture/Screenspace blurring

A common insight of subsurface scattering is that it is similar to blurring high frequency details on a surface, and several techniques such as Borshukov et al.'s Matrix Reloaded's face rendering [42], Isidoro's et al. skin rendering [43] and other sum of Gaussian kernel smoothing [41], [44] were realized on texture space. Texture space diffusion was intro-

Figure 2.9: The common pipeline for PRT based subsurface scattering techniques. (a) During the precomputation phase, a set of sample points (usually vertices) are chosen. (b) Their transport function T_d is evaluated. (c) Those transport functions are compressed using basis functions. (d) During run-time, lighting is evaluated at each point. (e) The lighting function is compressed into the same basis functions as the transport function. (f) The final radiance of a point is usually evaluated as a function of the dot product of the compressed transport vector and lighting vector.

duced by Borshukov et al.[42] particularly for skin rendering. They suggested unwrapping the 3D mesh of the object into a 2D texture while storing the irradiance information onto this texture. The 2D texture, known as irradiance map, then undergoes a convolution process with a diffusion profile as the filtering kernel. Figure 2.10 describes the texture space blurring pipeline proposed by d'Eon et al. [41]. Figure 2.11 shows the result of the face texture being blurred by different diffusion profile.

d'Eon et al. [41] further extended the technique by approximating the profile $R_d(r)$ as a weighted sum of Gaussians $G(v, r)$:

Figure 2.10: Pipeline of texture space rendering for subsurface scattering. (a) Obtain the shadow maps from the scene. (b) A texture parameterization is performed on the mesh. (c) Incoming radiance from the shadow maps are injected into an offscreen texture obtained from (b). (d) The irradiance texture is blurred with a kernel, which can be separated into Gaussians [41] for faster rendering. (e) The blurred texture map is mapped back to the surface for the final render. (f) The final render is added with additional lighting effects and post processing to produce the final render.

$$R_d(r) \approx \sum_{i=1}^k w_i G(v_i, r). \quad (2.48)$$

The diffusion profile is fitted using the minimization equation as below:

$$\int_0^\infty (r(R_d(r) - \sum_{i=1}^k (w_i G(v_i, r)))^2 dr \quad (2.49)$$

where the weights w_i and the variances v_i for k Gaussians are allowed to vary, and the Gaussian of variance v is defined as:

$$G(v, r) = \frac{1}{2\pi v} e^{-r^2/2v}. \quad (2.50)$$

The advantages of such a technique were to do away the lengthy process of multiplications, divisions, FFTs and inverse FFTs involving convolutions. Gaussians are also

Figure 2.11: Diffusion in Texture Space. The leftmost image shows the original irradiance map, and the next two images shows the irradiance map being blurred with kernels of different parameters. (Figure obtained from Jimenez et al. [46])

symmetrical separable functions. A 2D Gaussian function can be separated into two 1D Gaussian functions to reduce computations. Another important property of the convolution process is that the convolution of an image by a kernel that is a weighted sum of functions is the same as a weighted sum of images, each being the original image convolved by each function:

$$I * \left(\sum_{i=1}^k (w_i G(v_i, r)) \right) = \sum_{i=1}^k (w_i I * G(v_i, r)), \quad (2.51)$$

where I is the irradiance texture obtained from unwrapping the object mesh. The main problem with texture space approaches lies in their inability to handle texture discontinuities. Adjacent points in 3D world space may not be adjacent in texture space, this will naturally lead to unnecessary irradiance leaking to unwanted areas into the texture (Figure 2.12). There are also extra geometric computations needed to account for the stretch correction for the irradiance textures. This is because for curved surfaces, distances between texels do not correspond to the actual distances in the world space, thus we need to ensure that the distances between the texels are the same as the Gaussian kernels used in the convolution. Lastly, they also involve an overcomputation of irradiance, as not all computed irradiance contributes to the final render.

Isidoro et al. [43] suggested reducing the cost of image space blurs by using an Early-Z

Figure 2.12: The image on the left shows a disconnected uv parameterization of the object on the right. Performing filtering on the image on the left will lead to incorrect results. (Image obtained from Jimenez et al. [45] presentation slides)

to cull pixels which are in the back faces, as well as culling pixels that are not visible in the lighting pass. Jimenez et al. [44] provided an enhanced culling method that performs better than the depth buffer. They also provided an additional clipping procedure on the irradiance maps with visibility information from the screen space which ensures that unnecessary computation will not be done on pixels that do not contribute to the final render. Their method produces a speedup of up to 2.77 on low-end GPUs compared to d'Eon's [41] implementation.

Jimenez et al. [45] proposed overcoming the limitations of texture space rendering by using a screen space approach. The screen space approach does away with the need of irradiance maps and is unaffected by texture parameterizations.

Their method first retrieves an alpha matte of the screen, limiting the pixels to be processed. Then they follow by retrieving a depth map, diffuse map and specular map. The depth map determines the size of the kernel used for the convolution, as pixels further away from the camera will need a smaller kernel width. The diffuse map is blurred by multiple 1D Gaussian kernels as described in d'Eon et al.[41] to produce the subsurface scattering effects. The specular map is added to the final result of blurred diffuse map. The pipeline of the Screen Space approach in Jimenez et al.'s [45] approach is displayed in Figure 2.13.

The downside of the Screen Space approach when compared to the Texture Space approach is that it lacks irradiance information on the back of the object. Jimenez et al.[46]

Figure 2.13: Overview of the screen-space subsurface scattering approach. The algorithm first retrieves three maps from the screen space (alpha matte, diffuse map, depth map). The diffuse map undergoes a filtering process based on multiple kernels defined by the depth map. The blurred diffuse maps are combined and added with specular highlights.

proposed an algorithm to model the transmittance phenomenon, such that lighting from the back of the object can be viewed from the front. Their technique uses a shadow map to store depth information but does not require the uv coordinate of the irradiance maps as there is none in the screen-space. The final transmittance function is approximated as a function of depth using a sum of weighted Gaussians and stored as a look-up texture during precomputation. Their approach makes several assumptions such as assuming the irradiance on the back of the object is constant, transmittance is a low frequency phenomenon and assumes that the normal of the front and back surfaces are parallel. Due to these assumptions, their technique is only meant suitably for highly scattering materials such as skin.

Echevarria et al. [47] and Munoz et al. [48] proposed a hybrid method involving the Screen Space and Texture Space Approach. They enclose the object with a set of equally spaced planes perpendicular to the viewing direction. Next, irradiance information of the object is projected orthogonally onto the nearest two planes. Next, they computed the sum of inter-plane radiance contributions of each plane by performing convolutions with a distance based varying size diffusion kernel, which is to be used as the final render. Their res-

ults produce convincing translucent effects similar to Jensen's [19] Monte Carlo approach, but suffer from temporal artifacts from changing viewpoints due to inconsistent irradiance information from the changing geometry of the irradiance planes.

2.2.7 Voxel based methods

Voxelization techniques are common in rendering participating media. The object or scene is partitioned into voxels and each voxel have their corresponding information (flux, radiance, material etc) stored. The voxel structure supports efficient computation of multiple scattering by limiting flux transfer to a set of pre-defined directions. We describe three common forms of voxelization techniques in the sections below.

Lattice boltzmann lighting

Geist et al. [49] proposed a grid based propagation scheme using Lattice Boltzmann Lighting (LBL) which is highly parallel and easy to implement. They discretized their scattering volume into a set of 3-Dimensional voxel structure. Each voxel stores a representation of the flux density in the form of 19 discretized directions. They then use a 19×19 update matrix, which is defined by absorption σ_a and scattering properties σ_s of the medium, to define the amount of flux density transferred from the source voxel to its neighbouring voxel in each time step. However, the original LBL technique required a large amount of storage for the directional densities (19 floats), and the amount of computation for the flux transfer in each voxel can be expensive due to the large matrix multiplication involved. Convergence to the final result generally requires iterations of twice the count of the largest voxel dimension.

The LBL based method was later extended to subsurface scattering materials by Bernabei et al. [50]. The voxel structure is initialized with material properties such as gradients for index of refraction, index of refraction, absorption and scattering properties. Bernabei et al. [50] first compute a shadow map which contains the depth of the visible surfaces from

the light source. The data from the shadow map is used to define the initial injection of flux into the voxel structure. A ray marching step is performed with the initial positions of the ray obtained from the shadow map. Ray marching is done using Eikonal equations [51]. For every voxel that the ray step passes through, its radiance value is stored into the corresponding voxel using a set of directional densities. Once the ray marching is completed, the voxel structure undergoes a diffusion process, which is similar to the updating step in the LBL. The diffusion process uses two sets of device buffers, defined as diffusion volumes, which alternates their functions as input and output buffers during odd and even stages of the diffusion iteration. The diffusion iteration terminates when the system reaches equilibrium or convergence. This means that performing additional iterations will not produce significant changes to the voxels.

The final render then consists of marching through the viewing rays and accumulating the radiance from the final diffusion volume. Additional effects such as direct lighting and shadowing can be added to the surface. This technique is highly suitable for generating refractive, single scattering and multiple scattering effects in real time.

Bernabei et al. also proposed to reduce the storage requirements of 19 directional densities to only 6 principal axis directions. As such, the destination block only receives radiance from the 6 neighbouring blocks on its principal axis. This also reduces the update matrix to a 6x6 matrix, reducing computation time of the original LBL method and also making it suitable for real-time applications.

Light propagation maps

Fattal [52] developed a technique to solve the RTE Equation 2.19. He proposed using six 2D Light Propagation Maps (LPM) oriented to the six principal axes. Each LPM is an array of point elements, each emitting light in n directions. None of the LPM's set of directions overlapped each other. Each light ray of the LPM is used to march through the physical space which is discretized into 3D voxels. As each light ray passes through a voxel, it updates its own ray radiance, as well as the scattered radiance and unscattered radiance

Figure 2.14: The overall pipeline of Bernabei et al. LBL technique [50]. (a) The system first initializes the voxel structure with a set of material properties. (b) Next, it performs shadow mapping to retrieve the positions of the surfaces visible to the light. (c) From those positions in the shadow map, they performed ray marching using Eikonal equations and start accumulating radiances into the voxels that are passed through by the rays. (d) Multiple diffusion iterations are performed until equilibrium is reached. (e) The final diffusion volume is used for the final viewing render. Images obtained from [50].

based on a recommended 9x9 directional bin in the intersected voxel. The final render is computed by integrating the unscattered radiance in the voxels during a ray casting step.

Gruson et al. [53] developed a GPU efficient approach that groups rays based on its propagation direction. They develop a scheme to ensure that each voxel is only traversed by one ray at a time. This removes the time delay caused by race conditions when multiple rays are updating the same voxel. Their technique produces a speedup of 20x compared to Fattal's method. However, light propagation maps do not consider light refraction and Fresnel effects, which might lead to a loss of accuracy in rendering.

Light propagation methods

Light propagation volumes (LPV) was first developed by Kaplanyan et al. [8] as an efficient solution for single bounce indirect diffuse illumination. This technique does not require any precomputation. Kaplanyan et al. partitioned the entire physical space into a regular voxel structure known as the LPV, and used 4- Spherical Harmonics coefficients for each color channel to represent the flux distribution in each voxel. The initial flux distribution is defined using a Reflective Shadow Map (RSM) ([54]). The position, normal and flux density of each texel obtained in the RSM is injected into the corresponding voxel in the LPV. The LPV then undergoes a propagation process, where each step of the propagation process allows the flux distribution in each voxel to be distributed to its neighbouring voxel. After the flux distribution in the LPV has reached equilibrium, the LPV can then be used for the final render.

Kaplanyan et al. [8] proposed that performing a ray marching step through the LPV would allow the rendering of participating media. Billeter et al. [55] suggested that two separate LPV (standard LPV and a single scatter LPV) should be stored. Billeter et al [55] also developed an algorithm that facilitates the transfer of flux in LPV for participating media. During the final render, ray marching can be used on the single scatter LPV to generate single scattering effects.

Borlum et al. [7] reformulated the LPV technique, and changed the LPV into a Sub-surface Light Propagation Volumes (SSLPV). Unlike the LPV, the SSLPV is a voxelized structure of an object and not a scene. The flux is also injected into the material instead of outside the material. The authors also demonstrated that SSLPVs using the same propagation algorithm as the LPV was very suitable for rendering highly scattering materials. However, their technique is not known for rendering highly translucent materials. In addition, shadow mapping techniques from the LPV are also not known to work for environment lighting or area lighting.

As our method is an extension of the SSLPV [7], we provide a summary of the method

Figure 2.15: Overview of the SSLPV pipeline simulating multiple scattering (MS). The pipeline first renders a reflective shadow map (RSM) of the scene. Next, for each pixel in the RSM, energy is injected onto the surface based on a BTDF. A propagation process is then performed simulating a diffusion process, which distributes flux by an approximate scattering algorithm. Lastly, the final energy accumulated in each voxel is used in the final rendering

here as shown in Figure 2.15. The SSLPV operates on a regular voxel structure. Each voxel stores a virtual point light (VPL), with its intensity I in direction $\vec{\omega}$ in four SH coefficients per color channel as defined by:

$$I(\vec{\omega}) = \sum_0^{l_{max}} \sum_{m=-l}^l c_{lm} y_{lm}, \quad (2.52)$$

where l refers to the SH band and c_{lm} are the SH coefficients corresponding to the SH basis y_{lm} .

Intensity is injected based on a bi-directional transmission distribution function (BTDF) in the form of a cosine lobe and compressed using four SH coefficients per color channel through information supplied by a reflective shadow map (RSM)[54]. The SSLPV then undergoes several propagation iterations where radiance from a voxel s is distributed to its neighbouring cell d . The new intensity in cell d can be computed by $c_{d \leftarrow s} = \mathbf{M}_{d \leftarrow s} \cdot c_s$, where $c_{d \leftarrow s}$ refers to the contribution of flux, in SH coefficients, from the source voxel s to neighbouring voxel d . c_s contains the SH coefficients of the source voxel and $\mathbf{M}_{d \leftarrow s}$ is the transfer function that maps the source voxel contribution to a neighbouring destination voxel. We refer to [7] for a better clarification on how the transfer function is computed. We also show better clarification on the transfer function in our appendix A.6.

Rendering the multiple scattering component is straightforward. For every viewing ray that hits the translucent object, the hit point coordinate is converted into voxel coordinates and the relevant texel from the accumulation grid texture based on the normalized voxel coordinate is retrieved. In general, the outgoing radiance $L_o(x_o, \vec{\omega}_o)$ can be obtained by

multiplying an outgoing Fresnel term $T(\vec{\omega}_i, \vec{\omega}_o)$ with the radiance provided by the VPLs, $L_i(x_o, \vec{\omega}_i)$, in the SSLPV.

$$L_o(x_o, \vec{\omega}_o) = T(\vec{\omega}_i, \vec{\omega}_o) L_i(x_o, \vec{\omega}_i) \quad (2.53)$$

However, the voxel values cannot be used directly as it is stored as a distribution function. As of now, the accumulation buffer stores SH coefficients, $c^{(z)}$, that were defined with respect to the world space z-axis. In general, the world-space z-axis does not coincide with the normal of the surface. A rotation would be needed to rotate the SH coefficients to the surface normal before we can perform an evaluation. The transformation of SH coefficients from the normal-axis coordinate system to the world space z-axis coordinate system is performed using a passive spherical harmonics rotation matrix.

$$c^{(\vec{n})} = [R_{SH}(\alpha, \beta, \gamma)]^{-1} \cdot c^{(z)}, \quad (2.54)$$

where $R_{SH}(\alpha, \beta, \gamma)^{-1}$ is a passive rotation matrix form by the three zyz Euler angles (α, β, γ) .

The SH coefficients are multiplied with the integral of their basis functions to obtain the final incident radiance stored on the surface voxel in Equation 2.55, where A is the area of a voxel face in the SSLPV. An in-depth explanation for the choice of approximation for the VPL radiance can be found in the original SSLPV paper (Borlum et al. [7]) and will be elaborated in Appendix A.8.

$$\begin{aligned} L_i &= \frac{1}{A\pi} \sum_{l=0}^{l_{max}} \sum_{m=-l}^l c_{lm} \int_0^{2\pi} \int_0^{\pi/2} y_{lm}(\theta, \phi) \sin\theta d\theta d\phi \\ &= \frac{2}{A} \sum_{l=0}^{l_{max}} c_{l0}^{(\vec{n})} \int_0^{\pi/2} y_{l0}(\theta) \sin\theta d\theta \end{aligned} \quad (2.55)$$

The final outgoing radiance evaluated at point x_o in direction $\vec{\omega}_o$ can be computed by

performing a passive rotation of the SH coefficients to the surface normal, $\vec{n}$:

$$L_o(x_o, \vec{\omega}_o) = F(\vec{\omega}_i, \vec{\omega}_o) \frac{2}{A} \left(\sqrt{\frac{1}{4\pi}} c_{00}^{(\vec{n})} + \frac{1}{2} \sqrt{\frac{3}{4\pi}} c_{10}^{(\vec{n})} \right), \quad (2.56)$$

where $c_{lm}^{(\vec{n})}$ are the SH coefficients retrieved from the *accumulation grid* and rotated to $\vec{n}$. F is the Fresnel outgoing term for light transmitting out of the medium, and A refers to the 2D area of a voxel face.

2.3 Efficient methods for area light illumination

Soft shadows are an important visual element of area light illumination. This effect usually appears when objects are illuminated with partial illumination from area lights. As most real world shadows rarely exhibits hard edges, soft shadows adds an degree of realism to our rendering. A shadow is rendered by evaluating the value of a shadow ray that is casted from a point of interest towards the light. If the shadow ray hits an object before reaching the point on the light, it is considered occluded. This evaluation is simple for point light sources; however, for area lights, the light must be sampled as its occlusion value can only be evaluated across the surface of the light.

2.3.1 Shadow mapping techniques

Standard shadow map techniques [56] can approximate visibility fast and are commonly used in real-time applications. They operate by back-projecting a visible point onto the light viewing plane. Each texel on the shadow map describes the distance between a point light source and a corresponding 3D point visible from the point light source. The shadow map can be efficiently computed using a rasterizer. During visibility tests, comparisons are made between the projected visible points' depths and the depths stored on the shadow map. If the depth stored on the shadow map is lesser than the projected point depth, the point is occluded from the light. This technique although fast, are neither able to estimate

penumbra regions nor capable of generating soft edges.

Fernando [57] proposed percentage-closer soft shadows (PCSS). PCSS gives an approximation for the penumbra size and a filter corresponding to the penumbra size is used to take samples from a specific region on the shadow map. Schwärzler et al. [58] extended the PCSS by re-using visibility values across frames. Annen et al.'s [59] exponential shadow maps (ESM) replaced the binary output in visibility test to one with an exponential function. The visibility function is smoothed with an exponential function, where the exponential function used appears to overly smooth regions even with sharp visibility discontinuities. (VSM) [60] method creates an upperbound for visibility probability, which usually is an exact result when the receiver surface is parallel to the light plane. However, in scenes with high depth complexity, such as having multiple overlapping receivers, high frequency light leaking artifacts can be observed. This is known as the 'non-planar' condition, where the Chebyshev's inequality gives a poor upper bound approximation due to high variance from samples in the filter. Variance soft shadow maps (VSSM) [61] on the other hand, made use of a kernel subdivision scheme, that identifies particular regions in the shadow map that are 'normal' (regions with low variance) and 'non-planar'. VSM can be used for regions that are 'normal', while PCSS works well for regions that are 'non-planar'.

Annen et al.'s [62] convolution shadow maps represent the visibility function in Fourier basis functions which allows for filtering to be applied. These shadow maps are able to handle shadows in high depth complexity environments but they only approximate visibilities by blurring areas near the penumbra. They only produce a rough approximation to the pixel visibility and do not take into consideration the nature of the light (e.g. shape of the light, normal of the light). He et al.'s [63] multi-rate shading algorithm detects shadow edges using depth derivatives on shadow map and forces their pipeline to perform shadow calculations for these identified locations at higher resolutions. Their method is similar to our work as it uses a multi-resolution approach in finding regions that require sampling at higher resolutions. However, using a single shadow map only limits the sampling on the area light source to a single point due to the perspective projection used. Although we can

create multiple shadow maps to represent multi point sampling on area lights, performance issues such as a rise in textured memory and drop in rendering speed are expected.

2.3.2 Image space sparse samples reconstruction

Image space methods perform per-pixel error estimates and allocate more samples to difficult regions using various sampling techniques. A fixed set of samples per pixel is initially used to obtain an error estimate and variance. Rouselle et al. [64] and Li et al.[65] aimed to focus on using bilateral filters to reduce the variance in filtered pixels. Every pixel has a variance associated to it, and it is blurred respectively with a kernel of varying size depending on its variance.

Similarly, Mehta et al.[66, 67] and Yan et al.[68] described about analyzing light field based on its frequency domain. The image is first rendered using a set of sparse samples. From the sparse samples, the sampling noise is analyzed in its frequency domain to decide on the number of samples required per pixel as well as the width of the kernel required for filtering. The created filter is an axis aligned filter or a shear filter designed to specifically share samples among pixels to reduce computations. These methods focus on reducing samples per pixels.

Skala [69, 70] reconstructed images with sparsely distributed samples by radial basis functions, however these samples are reconstructed from a uniformly distributed set of samples in a stratified grid pattern. Their results have proven that it is possible to reconstruct an image with sparse samples especially without geometric information. However, their work deals entirely on 2D pixel values and are not capable of rendering or approximating illumination transitions.

2.3.3 Multi-resolution techniques

Direct illumination from area lights are known to vary smoothly across flat regions. Coarse sampling techniques, such as multi-resolution splatting by Nichols et al. [9, 15, 16, 17],

were devised previously to take advantage of this property. Multi-resolution splatting proposes to partition an image into patches known as fragments, where the fragment size depends on the depth, normal and illumination variations within the patch. As illumination variation decreases, the illumination on a fragment can be represented using information from lower resolution fragments which reduces computation time. Fragments of various sizes are interpolated using a multi-pass linear upsampling manner. Though this upsampling process produces reasonable results, the rendered image is overly blurred due to errors propagating from the lower resolution levels. The fragments generated are also in excessive numbers due to their strict neighbourhood discontinuity refinement criteria.

2.4 Critical analysis

In this section, we describe certain elements in rendering that we aim to achieve, and analyze the comparisons against previous works.

2.4.1 Translucent materials rendering - Precomputations

Table 2.2: Comparisons regarding precomputations between different rendering techniques

Techniques	Description
Voxelization	These techniques do not require precomputations.
Texture/Image Space	For direct processing of textures/images in screen-space or light space, no precomputation is needed. However, to compensate for geometric distortions in texture space, certain precomputations such as texture parameterizations may be needed. A pre-computed list of dipole values may need to be pre-computed in the case of image splatting.

Geometric Based Techniques	Radiosity requires heavy computation because of form factors. Mesh based approaches also requires large amount of mesh processing to discretize the mesh.
Precomputed Radiance Transfer	Requires heavy computation as a large amount of samples needs to be computed per vertex to evaluate information such as visibility and irradiance.

2.4.2 Translucent materials rendering - Storage

Table 2.3: Comparisons regarding storage requirements between different rendering techniques

Techniques	Description
Voxelization	Storage is dependent on the size of the voxel structure. However, voxelization techniques do not require large storage per voxel to produce good renders.
Texture/Image Space	These techniques usually do not require any storage, except for some precomputed texture parameterizations.
Geometric Based Techniques	Radiosity form factors computed during the off-line process may take considerable large amount of memory as the storage exponentially increases based on the number of triangles.
Precomputed Radiance Transfer	May require moderate amount of storage depending on the compression used for the precomputed data.

2.4.3 Translucent materials rendering - Visual comparisons

Table 2.4: Comparisons regarding visual accuracy between different rendering techniques

Techniques	Description
Voxelization	Algorithm mainly based on diffusion model. Physically accurate, but accuracy may be inconsistent across complex geometry as the voxel representation may limit the accuracy. Large voxel size may be needed to generate high frequency lighting. If boundary conditions are not well-defined, may also lead to energy lost along the boundaries of the voxels structure. Previous techniques may support single scattering to a certain extent.
Texture/Image Space	An approximation method mainly for optically thick materials. It assumes that the scattering is strong, such that a large amount of blurring occurs.
Geometric Based Techniques	Radiosity approaches are usually closed to the ground truth since they consider light interaction for every patch. Mesh based approaches are physically accurate models based on the diffusion equation. However, does not support single scattering since the diffusion equation does not work for single scattering models.
Precomputed Radiance Transfer	Proven to be physically accurate. However, the single scattering were mainly targeted at optically thick materials, where single scattering effects were weak.

2.4.4 Area lights illumination - Computation efficiency

Table 2.5: Comparisons regarding computation efficiency between different rendering techniques

Techniques	Description
Shadow Map based techniques	Fast algorithm that uses the rasterizer for backprojection. Always known to be real-time.
Image Space Reconstruction	Computation limited by blurring final image. Even if computed in Fourier domain, can be expensive. Some are known to produce results at interactive rates. Image construction by radial basis functions cannot be rendered on-the-fly.
Multi-resolution techniques	Known to produce results at real-time. Computation power limited by visibility tests, which is the same as image space reconstruction techniques. However, uses less computation for ray-tracing compared to image space reconstruction techniques.

2.4.5 Area lights illumination - Accuracy

Table 2.6: Comparisons regarding visual accuracy between different rendering techniques

Techniques	Description
Shadow Map based techniques	Relies mainly on blurring and smoothing. Results might be inaccurate for complex geometry.
Image Space Reconstruction	Known to be very accurate since a noise analysis has been performed based on a sparse set of samples on every pixel.

Multi-resolution techniques	Accuracy based on the assumption that illumination only varies in regions with light-to-point visibility discontinuities. This assumption is usually true for diffuse surfaces.
-----------------------------	---

2.4.6 Comparisons with our work

2.4.7 Translucent material rendering

Our proposed technique performs real-time translucent material rendering without precomputations, which ensures that it requires no additional time compared to geometric based techniques or precomputed radiance transfer techniques. We employ a voxel-based technique that is able to efficiently keep storage costs low, while still able to represent the high-frequency single scattering component which most voxel techniques are not able to represent. Our method supports material editing interactively.

2.4.8 Illumination under area lighting

Our proposed multi-resolution technique preserves the fragment-level optimizations in multi-resolution techniques and also achieves good quality results at interactive speed and has low errors compared to previous multi-resolution work. A much faster stream-compaction framework is proposed which utilizes the more recent graphics pipeline. We chose not to use image-space reconstruction techniques as they are guaranteed to be more computationally expensive than our work. Soft shadow map techniques are not chosen as they are blurring techniques that might have high errors in areas with complex geometry.

For illumination on translucent materials by area lights, our work does not require precomputations and supports material editing. Our low number of sample points required for illumination (direct and indirect) ensures that the algorithm has low memory costs.

Chapter 3

Interactive Rendering of Translucent Materials

In the previous chapter, we have surveyed methods which can be used for subsurface scattering and indirect illumination and have concluded that several techniques can be improved to render good quality results at interactive rates.

3.1 Hypothesis

3.1.1 Translucent object rendering can be approximated efficiently

Human perception can rarely understand the sophisticated light complexity involved in translucency. As mentioned by Fleming et al. [71]:

“the physics of translucency is simply too complex for the visual system to run the generative equations in reverse and estimate intrinsic physical parameters via inverse optics.”

In their paper [71], he ties down realistic translucency material rendering to a few factors e.g. mainly glossiness, specular highlights and color saturation. Secondly, effects such as light bleeding in shadowed regions, blurred fringes around sharp corners increase the realism in translucent materials by simulating the backlighting in scattering events. Thirdly, translucent materials appear more translucent when they are small, as light requires fewer

scattering events before exiting the object. Lastly, translucent objects are much more visible when illuminated from the back and viewed from the front. Gkioulekas et al. [72] on the hand, mentions the fact that phase functions can play a big part in the visual role of how translucent a material is. As phase functions typically controls how much light is back scattered or forward scattered, it features as an important component in rendering translucent materials.

Based on the human perception of realistic translucent rendering, we can deduce that the above mentioned factors are the main components that we need in rendering translucent materials. This means that illumination information can be compressed and reduced to a much more efficient format without a loss in realism in translucency material rendering. In our work, we decide that a low resolution voxel based approach would be the most suitable design. We represent the 2 main components (multiple scattering and single scattering) of translucent materials to be rendered with voxel based methods, utilizing the efficiency of such methods in precomputationless rendering.

3.1.2 Optimizations in screenspace for area lights applies to translucent materials

When a translucent object is placed into a scene, basic rules of screenspace optimizations can be applied. The translucent object casts shadows on the surfaces around it. Surrounding surfaces are still required to be rendered with direct illumination. Direct Illumination in simple scenes with consistent geometry differs by an unnoticeable amount between pixels across smooth geometry. In Ward et al.'s irradiance caching [73], he observed that indirect illumination tend to vary smoothly across flat regions. Using this observation, he devised an algorithm that allocates more samples in regions of high geometric discontinuity. For pixels in regions with fairly consistent geometry, samples can be re-used easily for interpolating the irradiance of the pixel (Figure 3.1). Screen space optimization routines can be applied such that we only focus rendering on important regions. In tiled rendering methods such as

Irradiance Caching

Figure 3.1: An observation by Ward et al. [73] noticed that indirect diffuse illumination varies in a very smooth fashion. Using this estimate, he reduced the number of times a Monte Carlo estimator would need to be evaluated by re-using previously calculated values in a "cache".

multi-resolution rendering (Figure 3.2), pixels with similar values are clustered together as fragments and information across all pixels within the fragment are re-used. For e.g. normals, depth and visibility information should be consistent within a fragment. This allows easier sharing of information within all pixels in a fragment hence reducing computational overheads by a significant amount.

We chose to target area lighting mainly because of two phenomenon visible from this type of light source: area lights produces soft shadows and previous voxel based methods do not handle area lighting on translucent objects. We only need to allocate more samples in regions with varying visibility discontinuities from the area light source, as regions inside the umbra and outside the penumbra are fairly consistent in illumination (Figure 3.3).

3.1.3 Indirect Illumination from translucent materials can be rendered effectively using voxels

When a subsurface scattering object is illuminated, it becomes a virtual light source by itself. The SSS object can be represented as a collection of Virtual Point Lights (VPLS)

Multi Resolution Rendering

Figure 3.2: Multi-resolution [15] builds upon the same idea of re-using computations. Diffuse illumination is known to vary smoothly across flat geometry. Hence pixels can be clustered into fragments, in which large areas of pixels can be represented with simpler computations.

based on its voxel representation (Figure 3.4). Since light from multiple scattering events are usually approximated to be diffuse, the distribution of this light can easily be represented by the fast low-frequency rendering algorithms such as the Light Propagation Volumes.

3.2 Research goals

The research goal of this thesis is to develop an efficient rendering algorithm for rendering translucent materials such that the following are incorporated:

1. Translucent materials can be rendered efficiently in terms of memory costs and without precomputations, while retaining visual elements (single scattering and multiple scattering) necessary for them. This is explained in Chapter 4.
2. Illumination from area light sources can be rendered efficiently and interactively by focusing on important regions in an image. This is explained in Chapter 5.
3. Translucent materials can be illuminated by area light sources and rendered interactively. Indirect illumination from translucent materials must also be rendered interactively.

Figure 3.3: Evaluating the visibility function for each visible point and the area light source is unnecessary because some points can share similar visibility, such as regions in umbra or areas outside penumbra. Penumbra is the main region which most samples are needed. Illumination is known to be slowly varying in regions with consistent normals, depth and visibility.

ively. We explain this in Chapter 6.

3.3 Design and approach

To fulfill the research goals, we propose the following solutions:

ESLPV: Enhanced Subsurface Light Propagation Volumes

In our work to tackle efficient translucent material rendering, we proposed the ESLPV as a customized voxel representation for translucent objects. Light is first deposited into the voxels containing surfaces through an "injection" process for multiple scattering simulations. For single scattering, light rays are traversed into the voxel structure, and at each step, intensities are deposited into the corresponding voxels. Light intensities from multiple directions are stored in their spherical harmonics basis, which provides a reasonable amount of directional representation. This data representation approach allows for rendering of two scattering components of subsurface scattering in real-time.

Figure 3.4: (Left) A translucent bunny object (highlighted in red) is placed in an environment. It can act as a diffuse light source and distribute illumination to its surrounding voxels. Since the translucent object is already represented in its voxels (right), it has a readily accessible virtual point light intensity.

For multiple scattering, light is distributed using a propagation process which distributes energy to nearby voxels. We observe that several propagations appear unnecessary as light rarely reach those voxels from the surface, or the light intensities in those voxel rarely get propagated to the surface. We propose using an efficient propagation system that only focuses on "important" voxels. We first perform solid conservative voxelization on the translucent object. Next, we compute a distance transform map for every voxel, defining a distance away from the nearest surface. Light propagation within voxels is only required if the propagation directly affects a voxel that contributes to a surface voxel, where illumination is usually retrieved. A hierarchical system of voxel structure is also used which allows better control of "back lighting" in translucent materials.

For the single scattering during the rendering phase, we perform ray marching on the voxel structure from the refracted camera ray and accumulate the flux from every voxel that has been traversed. We use an importance sampling function to place sample points efficiently on the refracted light ray. Eventually, the accumulated single scattered flux (in SH) is uncompressed into its radiance and added to the multiple scattering component during rendering.

Direct Illumination from Area Lights on diffuse surfaces

As earlier identified in Section 3.1.2, illumination is hard to approximate in regions with varying discontinuities, particularly in regions within the penumbra. For any other areas,

diffuse reflectance from area light can be easily approximated using fragments. Although existing work on multi-resolution rendering are capable of reaching interactive performance, we target to further improve them by reducing the number of fragments required for rendering.

Direct Illumination from Area Lights On Translucent Objects

The standard way for depositing light intensities into voxels requires a shadow map. Each texel in the shadow map is used to represent a single light ray, and each light ray carries part of the light energy. When using area lights, we note that shadow maps are poor representations as they restrict each point sample on the light to a single direction due to the back-projection used. We instead suggest a light "gathering" operation which computes the amount of light that each voxel can potentially receive. A set of Poisson disk distributed samples is placed on the translucent object. Each Poisson disk distributed sample performs a gathering operation by creating shadow rays (in a stratified sampling manner) towards the area light. Transmitted flux is injected into the surface voxel for every visible light ray to the area light source. The intensities in the voxel is normalized by the number of Poisson disk samples in the voxel. This implementation removes the necessity of a shadow map for light intensity injection.

Indirect Illumination from Area Lights for Diffuse Materials

Similarly, indirect illumination for the entire scene can be represented in Light Propagation Volumes (LPV) [8]. Similar from previous predecessors, this method requires a shadow map based representation as well for injecting light intensities into voxel. To avoid using shadow maps, we can use a set of Poisson disk distributed samples for gathering reflected flux from the area light, which is similar to our above method for translucent objects.

Indirect Illumination from Translucent Objects As our translucent object has its light intensities represented in a voxel structure, each voxel can act as a translucent point light (TPL). A hierarchical representation can be used, considering that it is easy to mipmap a regular voxel structure such that it can represent fewer virtual point lights. Indirect illumination from translucent surfaces are also usually represented as low-frequency lighting

in real-time applications shown in Figure 3.5 [74]. The LPV can be used for depositing the intensity of TPLs and propagating them.

Figure 3.5: Light can escape from translucent materials and illuminate surrounding surfaces. These effects are usually represented as low-frequency illumination. Image from website [74].

3.4 Summary

In this chapter, we defined our goals for rendering translucent materials in an efficient manner. We presented an overview of the algorithms and framework we intend to use to achieve our goals. Each research goal is achieved by tackling several individual problems related to the goal.

Chapter 4

ESLPV for Real-time Rendering of Translucent Materials

In this chapter, we present an Enhanced Subsurface Light Propagation Volumes (ESLPV) [5](My Research Publications section) method for real-time rendering of translucent materials. Our method is an extension of the Subsurface Light Propagation Volumes (SSLPV) technique. We improve the SSLPV by incorporating a single-scattering framework that uses the same spherical harmonics (SH) storage structure as the SSLPV. The new single-scattering technique deposits radiance as SH coefficients during a ray marching procedure. The final result is rendered using a ray tracer with importance sampling along the camera ray. This framework also enables the ESLPV to render refractive objects. In addition, we also propose a distance transform optimization that can remove the unnecessary computations during the propagation cycle of LPV [8] based methods. A hierarchical propagation process is also proposed to render highly translucent materials. Similar to the SSLPV, our ESLPV method contains no precomputations, and has low storage requirements that are independent of the mesh size.

The need for no precomputation and low storage cost add a favourable dimension in real-time rendering. In practical cases such as material or lighting changes, it is essential for algorithms to be robust enough to perform the necessary updates in rendering in real-

time. Precomputed Radiance Transfer (PRT) techniques rely on an excessive amount of uncompressed information and precomputations before it is able to represent the lighting information in a compressed and memory efficient manner. Having a large precomputed dataset (such as in PRT) to prepare for such changes may be impractical as it indirectly affects the scalability of the algorithm to larger scenes. With regard to subsurface scattering, Børllum et al. [7] presented the Subsurface Light Propagation Volumes (SSLPV) method that can render the effects at little memory costs and does not require precomputations. The SSLPV framework was chosen because it has no precomputations, and only requires small storage to render a translucent object. Its pipeline is also independent of the geometry size. In addition, the Light Propagation Volumes (LPV) uses a simplified lighting model of using four SH coefficients per color channel to represent light intensity which also texture access efficient on the GPU. This allows for very fast GPU computation as well as removing any required precomputations. Contrary to the SSLPV, our technique is able to incorporate single-scattering and handle refractive objects as well as highly translucent materials.

4.1 Our ESLPV pipeline

The overall pipeline of our ESLPV is shown in Figure 4.1. We perform solid and conservative voxelization on the input mesh to support single-scattering and also to optimize the propagation phase. A single-scattering (SS) injection process is added to allow the SSLPV to support single-scattering as well as refractive objects. In addition, ray marching is performed on the single-scattered radiance information during the render phase. A modified propagation phase is proposed where two hierarchical mipmap buffers are used for improving backlighting, as well as an optimization procedure which makes use of the computed 3D distance transformed information to discard unwanted computations in the propagation phase.

Figure 4.1: Overview of the proposed ESLPV pipeline. The boxes highlighted in red indicate the modifications and extensions that have been added to the SSLPV.

4.1.1 Voxelization

Although our target is to achieve zero precomputations, we can either consider the voxelization procedures as a one-time precomputed process or it can be processed as a per-frame basis for deformable objects. The total time needed for these voxelizations is just mere milliseconds.

We perform GPU conservative [75] and solid voxelization [76] on the input mesh based on a 32^3 voxel structure and store the results as 3D textures. Conservative voxelization ensures that every voxel containing any mesh is flagged. Solid voxelization ensures that the interior voxels of a watertight conservative voxel structure is flagged. The solid voxelization information is important for indicating the occupancy of the voxel during the single-scattering ray marching process. Next, we perform 3D distance transform on the conservative-voxelized information using the two cityblock matrix described in Figure 4.2 (also known as Manhattan distance) matrix.

The distance transformed output will now contain distance information from the nearest surface voxel. We will refer the distance transform 3D output as the *DTCT*. This information will be needed to optimize the propagation phase. Details will be explained in Sect.4.1.3.

0	1
1	S

S	1
1	0

(a) Cityblock forward scan matrix (b) Cityblock backward scan matrix

Figure 4.2: Distance transform using cityblock distance is performed by a forward and backward sweep using the matrices in (a) and (b). The actual matrix is a 3D matrix, but for simplicity, only a 2D one is shown. For the forward scan, the matrix sweeps from top left to bottom right, and adds one to the distance above it, adds one to the left, and adds one to element in the next layer, then takes a minimum with itself and its 3 neighbours. For the backward pass, the matrix sweeps from bottom right towards the top left. It adds one to the distance on its right and below, and to the one in the previous layer. It then takes a minimum with itself and its 3 neighbours.

4.1.2 Single-scattering injection

The single-scattering injection process is described in Figure 4.4. Our technique receives the same input from the RSM as the SSLPV’s injection process. It traces a refracted light ray into the translucent medium and injects light intensity into its corresponding LPV voxel at certain steps along the ray. Our method follows Bernabei et al. [50] technique of using a modified Eikonal ray tracing [51] of uniform step size δ . The ray is advanced based on the index of refraction η and gradient of index of refraction $\nabla\eta$. Similar to their method, we use a two-pass approach, with the first pass storing VPLs into the voxels, and the second pass traces the camera ray through the volume during rendering.

Unlike Bernabei et al.’s [50] method, we compress our intensities to four SH coefficients instead of six directional intensities. This reduces the number of texture reads. The second difference is that their method stores the current energy of the ray at fixed steps, we note that this will end up injecting more energy into the voxels than the ray’s original energy. Our method, on the other hand, stores the scattered energy of the ray. This is especially important for rendering transparent objects where little energy is lost in the material.

Figure 4.3: (a) Stanford Bunny rendered with single-scattering and an isotropic phase function. (b) Bunny rendered with a forward scattering phase function in the red color channel. The near plane of the spotlight is displayed as the square, and the green lines is its field of view.

The last difference is that their directional intensities did not incorporate the use of a phase function while we did incorporate the Henyey-Greenstein (HG) phase function. The significance of the phase function for translucent materials is described in [72]. Figure 4.3 shows the difference in results between an isotropic phase function and one with forward scattering. Single-scattering contributes significantly for weak scattering and transparent materials such as glass. We provide a video link in the results section to highlight the significance of the phase function. The results in the thin areas around the bunny ears appear darker due to refraction effects from the dark background. There is a lack of RSM texels that cover the ears, which results in lesser single scattering rays being refracted through those voxels.

We choose to use the single-scattering equation defined in [77] for more accurate rendering. We do note that their equation is meant for objects with diffuse boundary and needs to be remodified for smooth Fresnel boundary. The single-scattering equation is defined as

Figure 4.4: Overview of single-scattering injection. (a) Trace the refracted light ray and compute locations of sample points using Eikonal equations. (b) Project the ray direction to the phase function. (c) Inject the SH coefficients into the nearest voxel center.

follows for a single sample along the incoming refracted ray $\vec{\omega}'_i$:

$$L_o^{(1)}(x_o, \vec{\omega}_o) = \sigma_s(x_o) Fp(\vec{\omega}'_i, \vec{\omega}'_o) e^{-s'_i \sigma_t(x_i)} * e^{-s'_o \sigma_t(x_o)} L_i(x_i, \vec{\omega}'_i) / s_o'^2, \quad (4.1)$$

where F is the product of the Fresnel transmittance and outgoing term for light entering and leaving the medium. $\vec{\omega}'_i$ and $\vec{\omega}'_o$ refer to the refracted incoming and outgoing paths of the light. The scattering path lengths of samples taken along direction $\vec{\omega}'_i$ and $\vec{\omega}'_o$ are defined by s'_i and s'_o . The extinction coefficient and scattering coefficient are referred by σ_t and σ_s . The extinction coefficient is equal to $\sigma_a + \sigma_s$, where σ_a is the absorption coefficient. p refers to the phase function. As an exact solution might be expensive to compute, Jensen et al. [19] suggested taking multiple samples to compute the single-scattered radiance numerically as shown in Figure 4.5.

We modify the single-scattering equation in Equation (4.1) to a two-pass evaluation stage in Equation (4.3), β and γ . β is the amount of scattered intensity computed during a specific ray marching step. The algorithm for injecting β into a voxel is described in Algorithm 4.1. We use Beer's Law to compute the energy lost due to extinction on an input ray with energy, L_i , at a specific ray step of distance δ as $\nabla L_i = (1 - e^{-\sigma_t \delta}) L_i$. The energy lost to scattering can be computed as $(1 - e^{-\sigma_s \delta}) L_i$. The albedo $\alpha = \sigma_s / \sigma_t$ in β retrieves

this scattered energy from ∇L_i . In Line 15 of Algorithm 4.1, ∇L_i is projected into its SH coefficients by multiplying its magnitude against a rotated HG function in direction $\vec{v}$, defined by $\rho_{lm}(\vec{v})$ in its SH coefficients. The SH coefficients of a Henyey-Greenstein phase function can be found in [78], which we have listed down in Equation 4.2.

$$\begin{aligned}
 c_{l=0}^{m=0} &= 2\sqrt{2\pi} \\
 c_1^{-1} &= 0 \\
 c_1^0 &= 2g\sqrt{4\pi} \\
 c_1^1 &= 0
 \end{aligned} \tag{4.2}$$

γ on the other hand is evaluated during the rendering stage by performing ray marching on the refracted camera ray. $s_o'^2$ can be neglected since the camera ray would be only considering the VPLs along its path. Note that the single-scattering injection uses a separate voxel structure of the same dimensions compared to the multiple scattering one.

$$L_o^{(1)}(x_o, \vec{\omega}_o) = \beta\gamma, \tag{4.3}$$

where

$$\beta = \alpha p(\vec{\omega}_i', \vec{\omega}_o') F_t(\eta, \vec{\omega}_i) \nabla L_i(x_i, \vec{\omega}_i) (\vec{n}_i \cdot \vec{\omega}_i), \tag{4.4}$$

$$\gamma = \sigma_s(x_o) e^{-s_o' \sigma_t(x_o)} F_t(\eta, \vec{\omega}_o) (\vec{n}_o \cdot \vec{\omega}_o). \tag{4.5}$$

Once single-scattering injection is done, we perform a 3D gaussian smoothing with a 5^3 kernel. This is to remove unwanted noise from undersampling. Note that we do not perform any propagation iteration for the single-scattering injection data.

Figure 4.5: Single-scattering is the result of integrating along the refracted incoming ray $\vec{\omega}'_i$ and outgoing ray $\vec{\omega}'_o$.

4.1.3 Multiple scattering propagation

Our propagation process is identical to the SSLPV. However, we propose two additional modifications to the propagation process meant for simulating multiple scattering. The first is an optimization process that removes redundant computations, and the second is a hierarchical extension that is meant for handling highly scattering materials.

Propagation optimization

We provide an additional optimization routine to reduce the number of unnecessary voxel computations in the GPU. The SSLPV technique only makes use of the accumulation texture for its final render. Despite doing a full calculation for every texel in the 32^3 *accumulation grid*, only the values from voxels containing surfaces, as well as its neighbouring voxels (used for trilinear interpolation for final render) of one voxel distance away, will be used for the final render. Hence, our *DTCT* is meant for identifying specific voxel computations that will not contribute to the final render.

We describe our optimization in Algorithm 4.2 for each propagation cycle. We identify two scenarios where voxel computations are unnecessary. The first being the interior voxels (red voxel in Figure 4.6) not receiving any significant contribution due to the lack

Algorithm 4.1: Ray marching injection algorithm

```

1 for each valid texel from RSM do
2    $x_i \leftarrow \text{RSM}(\text{position}, \text{texel\_coord});$ 
3    $L_i \leftarrow \text{RSM}(\text{flux}, \text{texel\_coord});$ 
4    $\vec{v} \leftarrow \text{refract}(L);$  //refracted light vector at  $x_i$ 
5    $L_i \leftarrow F_t * L_i;$  //attenuate by Fresnel
6    $L_{\text{new}} \leftarrow L_i;$ 
7   while  $x_i \in \text{AABB}(\text{ESLPV})$  AND  $L_i > 0$  do
8      $v\_pos \leftarrow \text{world\_to\_voxel}(x_i);$  //voxel position(i,j,k)
9     Obtain material properties  $\nabla\eta, \sigma_s, \sigma_a$  in  $v\_pos;$ 
10     $\sigma_t = \sigma_a + \sigma_s;$ 
11     $\alpha = \sigma_s / \sigma_t;$ 
12    if  $\text{occupied}(v\_pos)$  then
13       $\nabla L_i \leftarrow (L_i - L_{\text{new}});$ 
14      //Project  $\nabla L_i$  into Phase Function
15       $L_{i,lm} \leftarrow \nabla L_i * \rho_{lm}(\vec{v}) / 4\pi;$ 
16      Inject  $\alpha L_{i,lm}$  into ESLPV; // Inject  $\beta$ 
17    end
18    //Advance step size based on Eikonal Eqns
19     $x_i \leftarrow x_i + \delta \|\vec{v} / \eta\|;$   $\vec{v} \leftarrow \vec{v} + \delta \nabla \eta;$ 
20     $L_i \leftarrow L_{\text{new}};$ 
21     $L_{\text{new}} \leftarrow e^{-\delta \sigma_t} L_i;$  //update ray energy
22  end
23 end

```

of propagation iterations, and the second identifying interior voxel computations that are unlikely to contribute to the surface voxels. The variable *max_iteration* in Algorithm 4.2 is set to the 1D resolution of the voxel resolution. A slice of the DTCT texture is shown in Figure 4.7.

Multiple scattering - Hierarchical propagation

We propose a hierarchical ESLPV approach which offers a quick solution for rendering highly translucent objects. As the SSLPV follows a voxel-based diffusion approach, radiance intensity is lost during the first few propagation steps. This is because each propagation step is equivalent to simulating a scattering process, even though the voxel size has no relationship with the actual mean free path of the material. As noted from the transfer function described in Equation 4.6 from [7] (as well as appendix A.7), in cases where there is no ex-

Figure 4.6: The amount of propagation iterations needed for the initial set of VPLs (Propagation Iteration: 0) from the surface to reach the red voxel is three. Intermediate VPLs produced during each step are shaded in (red, green, purple) respectively.

Figure 4.7: We describe a single slice of voxel volume from the DTCT texture(right). The pixel intensity in the slice represents the distance away (in cityblock distance) from the nearest surface voxel. Optimally this information can be used to identify voxels that will never have their intensity propagated to the surface.

tion at all, the maximum energy that is scattered to a neighbouring voxel is only around one sixth of the original voxel's radiance. g^l refers to the multipole moments, represented in SH, of the phase function g as described in Equation 4.2. By discretizing the scattering events along multiple voxels from the front to the back of the object, the original energy remaining from the surface would be very low. Hence, the SSLPV would usually underestimate back-scattered light. It can be seen that the back-lit glow from the SSLPV model (Figure 4.8c) is dimmer compared to the reference image generated by the photon beam diffusion (PBD) method [77] (Figure 4.8a). Although the SSLPV provides the flexibility of fine-tuning the number of propagation iterations, higher number of propagations does not

Algorithm 4.2: Voxel discard algorithm

```

1 for each  $voxel_i \in ESLPV$  do
2   //Obtain distance to the nearest surface voxel
3    $dist \leftarrow texture(DTCT, voxel_i);$ 
4   //Scenario 1: surface flux does not reach voxel
5   if  $current\_iteration < max\_iteration / 2$  then
6     if  $dist - current\_iteration > 1$  then
7       |   discard voxel computation;
8     end
9   else
10    //Scenario 2: voxel flux does not reach surface
11    if  $dist - (max\_iteration - current\_iteration - 1) > 1$  then
12      |   discard voxel computation;
13    end
14  end
15 end

```

improve the backlighting.

$$I(\vec{\omega}) = \sum_{l=0}^{l_{max}} \sum_{m=-l}^l [(1 - \sigma'_s) + \sigma'_s g^l] c_{lm} y_{lm}(\vec{\omega}) \quad (4.6)$$

Our hierarchical ESLPV approach reduces the radiance lost, by creating two additional coarser mipmap levels of the ESLPV and combining their effects. Each mipmap level will have a smaller voxel resolution but larger voxel size. We deem the lowest mipmap resolution of 8^3 to be the minimum resolution that can preserve low frequency illumination. By reducing the amount of propagation steps that one region takes to transfer to a neighbouring region, we can reduce the amount of radiance lost in the process. This is also equivalent to adding much more variation in scattering length in the scattering process. Our hierarchical ESLPV approach differs from previous cascaded approaches such as [8] and [79]. Their approach classifies specific irradiance samples to only one specific level of the LPV buffer or splatting buffer, while in our case, we assign the flux from every RSM sample to all levels of buffers. Olovsson et al. [80] proposed an octree-based LPVs but each voxel in the coarser level of their octree receives its intensity as an average of its eight finer neighbours while in our case, we receive the total accumulated intensity of its eight finer neighbours.

The hierarchical propagation approach is done by performing eight (not a strict limitation) propagations on each mipmap level and accumulating the radiance of each layer. Next, the inverse cell area, $1/A$, in Equation (2.56), would be $1/4$ the size of its finer mipmap level. We show our rendered results with the hierarchical propagation method in Figs. 4.8 and 4.11 and its comparison to the SSLPV.

Figure 4.8: Images of a highly scattering dragon model viewed from the back with a spot-light illuminating it from the front. (a) Photon Beam Diffusion [77] with 4 samples per pixel rendered in 57 seconds by the Mitsuba Renderer [2]. (b) Our hierarchical approach uses two additional mipmap levels to enhance backlighting. (c) The SSLPV does not handle backlighting well for highly scattering models as too much energy is lost during the propagation phase. Scaling the flux does not result in similar backlighting quality as (a).

4.1.4 Rendering

Our rendering process for the multiple scattering component uses Equation 2.56. The hierarchical propagation is technically an additive blend of the results from the two other lower mipmap levels. Rendering the single-scattering effects requires ray marching with a ray tracer to trace a refracted camera ray into the translucent medium.

Single-scattering ray marching

We take sample points along the refracted camera ray based on the sampling heuristic in [37]. N sample points are chosen based on $s(\xi_i) = -\ln(1-\lambda\xi_i)/|\sigma_t|$, where ξ_i , a uniformly

distributed number is defined by $(i - 0.5)/N$ and $|\sigma_t|$ is the luminance of the extinction coefficient. The probability distribution function for this importance sampling method is $\text{pdf}(s) = |\sigma_t| e^{-|\sigma_t|s} / \lambda$ where $\lambda = 1 - e^{-|\sigma_t|d_m}$ is a normalization factor, where d_m is the maximum distance of the refracted ray obtained using a ray tracer. The SH coefficients, which are equivalent to β in Equation 4.3, are retrieved from the single-scattering injection buffer. Next, the SH coefficients are attenuated with γ from Equation 4.5 and then divided by their pdf, for choosing that particular sample point. They are then accumulated for all samples along the ray before averaging them by the number of samples.

For our experiments, we used 16-32 samples for each camera ray. The accumulated SH coefficients for each ray is rotated to the surface normal, a process similar to how the multiple scattering is evaluated in Equation (2.56). In Figure 4.9, we designed an experiment where we position an incident beam of light straight into a translucent medium and we visualise the light scattering out of the object from a camera located vertically on top of the medium. In Figure 4.10, we show the rendered results of the setup in Figure 4.9 using several semi-infinitely thick mediums of various material properties illuminated by an incident beam of light. We compare the exact single-scattering solution [19] and our ESLPV. We do note that our results are similar enough.

Concavities and refraction effects

In our rendering, we only handle up to four object depths in the single-scattering case due to performance issues. This is based on the assumption that the object is not geometrically complex such that refraction effects can be seen easily by tracing up to four depths. All accumulated SH coefficients must be rotated to the outgoing refracted ray direction and also attenuated by a Fresnel transmission term each time they pass through a medium-to-air boundary. We set the intensity of environment refraction to $\kappa * (d_m - (1/\sigma_t))$, where κ is a user-defined constant to adjust the intensity of the environment map with respect to the light power.

Figure 4.9: We show the single-scattering experiment setup of an incident beam of light shining on a semi-infinite medium with different materials. A perspective camera views the surface of the medium from the top.

4.2 Implementation

The RSM is generated using OpenGL Shading Language (GLSL) as the hardware rasterizer provides a faster visible surface determination than a ray tracer. The RSM data are next shared by the multiple scattering injection program (in GLSL) and a CUDA kernel which performs the single-scattering injection.

CUDA is chosen for the single-scattering injection process because it offers atomic float operations such that we can perform synchronous read-write operations into the same memory location from different threads. This handles situations where multiple light rays write values into the same voxel.

We use the GLSL shader for the propagation stage, while supplying it with the DTCT information such that insignificant fragments can be discarded. For our final render, we use the OptiX ray tracer [81] since we may need to obtain several ray depths for the ray marching procedure to handle concavities as well as refraction.

Figure 4.10: We show the single-scattering falloff comparisons of an incident beam of light shining on a semi-infinite medium with different materials in each row. A perspective camera views the surface of the medium from the top. The column on the left shows the results of the exact solution. The column on the right shows our single-scattering technique. The first row represents a material with the following properties, $\sigma_s = (1, 1, 1)$, $\sigma_a = (0, 0, 0)$, $\eta = 1.3$, $g = (0, 0, 0)$. The second row represents a material with $\sigma_s = (1, 1, 1)$, $\sigma_a = (0.01, 0.1, 1)$, $\eta = 1.3$, $g = (0, 0, 0)$ and the third row with $\sigma_s = (1, 1, 1)$, $\sigma_a = (0, 0, 0)$, $\eta = 1.3$, $g = (-1, 0.5, 0)$. Both results are computed from 32 points along the medium. This is done so that the size of the beam of the light can be reflected proportionally in the result because the width of the beam is limited by the ESLPV’s voxel resolution. Both results are tonemapped with the same parameters.

4.3 Results and discussion

We have implemented our method on an Intel i7 2.40GHz CPU with 8GB of memory utilising a NVIDIA GeForce GTX 680M laptop GPU. We tabulate the storage costs accordingly for the recommended SSLPV resolution of 32^3 and an RSM of 256^2 resolution, respectively, in Table 4.1. In practice, we do not recommend using a 64^3 voxel structure because it requires more computations for the propagation stage and will approximately take 15x longer to render a frame. The time required in each of the ESLPV processes is shown in Table 4.2. From the results in Table 4.2, we observed that our hierarchical propagation, with DTCT, is four times faster than the SSLPV propagation. The DTCT alone provides an average speedup of 1.35-1.61x for the propagation phase.

Figure 4.12a and 4.12b shows the rendering of a Stanford Thai statue with single and multiple scattering components. In most highly translucent materials, the contribution of single-scattering is usually significantly less dominant than multiple scattering. In materials which are more transparent and exhibit weak scattering, single-scattering effects are much more evident as shown in Figure 4.12c. Figure 4.12c is illuminated with a textured lighting simulating 8 light beams. The single-scattering effect can be seen from the falloff in the

Figure 4.11: Comparisons between the Hierarchical ESLPV and SSLPV for a material with albedo $> 95\%$. The model is illuminated with a spotlight, which its FOV displayed as green lines.

intensities of the light beams. Additional rendering of jade and apple materials are shown in Figure 4.15. Videos of real-time renderings of our results can be found in the link ¹.

Figure 4.12: (a) Single-scattering Rendering of Stanford Thai statue (500k Triangles) with a simulated weak scattering material with refraction effects. $\sigma_a = 0.01$ and $\sigma_s=0.2$. (b) Same Stanford Thai model rendered with single and multiple scattering. (c) A rectangular block (12 triangles) with single-scattering illuminated by a checkered light (top left) which simulates multiple light beams. Tonemapping have been applied on the images.

In addition to the previous rendering, we also show that our technique is able to render deformable objects as all necessary computations can be computed within each frame. We show our results for deformable models (Stanford Bunny and Buddha) in Figure 4.13, where

¹https://www.dropbox.com/s/uu4as0687zevzp7/ESLPV_2.MPG?dl=0

Table 4.1: Storage requirements in megabytes (MB) for each phase of the ESLPV for a 32^3 voxel structure

Storage Costs (MB) of a 32^3 ESLPV	
Conservative, Solid Voxelization	0.062
3D Distance Transform Image	0.031
RSM (<i>positions, normals, flux</i>)	3.000
Injection buffers	1.500
Injection Single-scattering buffers	1.500
Propagation and Accumulation buffers	3.000
Material ($\sigma_a, \sigma_s, \nabla\eta, \eta$)	1.625
Hierarchical Mipmap buffers	0.418
Total memory cost:	11.14

each image corresponds to a particular frame during a deformation. The top row of images show the front lighting, while the lower row shows the backlighting. An animated video is provided in the link below ².

There are several limitations to our method. Similar to the SSLPV, our technique does not handle the illumination on geometry that is near to the boundary of the ESLPV. This is because there are no proper boundary conditions defined when flux is lost to the external boundaries of the ESLPV. The RSM injection process also provides poor coverage to thin geometries, and often leads to a lack of injected energy in some voxels.

In addition, our single-scattering algorithm is only recommended for objects with smooth Fresnel surfaces. We have further provided analytical comparisons of using 2nd order SH basis to reconstruct the Henyey Greenstein phase function. There exist certain range of values of g , where $|g| > 0.3$, in which the pdf of certain angular scattering distribution cannot be well represented. These are provided in Appendix A.9. Although we made physical comparisons of our single-scattering with physically based methods, we need to note that the ESLPV's multiple scattering, similar to its predecessor, uses a dimensionless σ_s , which has no physical basis. The hierarchical propagation method provides a quick solution to backlighting, but it is not a physically accurate method and hence the number of iterations in each layer should be fine-tuned.

²<https://www.dropbox.com/s/xdkng7uf0lw17lq/BuddhaBunny.mp4?dl=0>

Table 4.2: Rendering time for some of the examples. We breakdown the respective time used for the stages in the ESLPV. Vox. refers to time required for voxelization. The column Inject (SS) refers to the time used for depositing virtual point lights in the ESLPV. Inject (MS) refers to the injection process identical to the SSLPV. Hier Prop refers to the Hierarchical Propagation in Section 4.1.3 that we used for all our rendering and DTCT has been used to accelerate this stage as well. SSLPV Prop refers to the time that the SSLPV uses for finishing the propagation stage. w/DTCT refers to the SSLPV Propagation accelerated with the DTCT. In renders where there are single-scattering, the rendering time is significantly larger due to ray marching.

Figure	Resolution	Vox.	RSM	Time (ms)		Render	Fps		
				Inject (SS/MS)	Hier Prop			SSLPV w Prop	/DTCT
4.12a	768x768	7.39	0.48	1.91/ —	-	68.9	14.0	-	-
4.12b	768x768	7.39	0.48	1.91/0.24	1.64	69.5	13.5	7.97	4.94
4.12c	768x768	24.7	0.05	6.71/4.43	1.64	12.6	38.8	7.97	6.71
4.11	512x512	7.39	0.48	— /0.23	1.64	8.24	94.4	7.97	4.94
4.15a	768x768	2.64	0.25	2.39/0.28	1.63	62.7	14.9	7.97	5.02
4.15d	768x768	8.39	0.99	2.11/0.25	1.84	31.3	27.4	7.97	5.87
4.3	512x512	1.66	0.44	2.42/ —	-	20.2	43.4	-	-

4.4 Applications - Material editing

The number of processes that are needed for updating a material when it undergoes material changes are numbered as below:

1. Computing Finite Differences for xyz direction for the IOR volume (only if there are any changes)
2. Time for transferring 1.625MB of material information to GPU (almost negligible time)
3. Recompute Injection Process
4. Recompute Propagation Process
5. Recompute Rendering

We created a material editing user interface to create simple changes on the 3D model as shown in Figure 4.16. Users can modify the material properties by selecting a set of voxels within a voxel plane slice.

4.5 Summary

We have presented the ESLPV as an efficient method for real-time subsurface scattering rendering. The ESLPV is able to handle refractive and transparent objects through the use of a separate single-scattering injection buffer. It is also able to render highly translucent materials with two additional mipmap levels at lower memory costs. The distance transform optimization allows the unnecessary propagation methods to be removed at little additional memory costs. Our algorithm effectively achieves our research goal of interactive rendering with no precomputation and high storage costs. It is also able to generate convincing translucent material rendering.

Some geometry may receive fewer sample points from the texels of the RSM due to them having a lesser projection area on it. This, in turn, generates lesser irradiance in the voxel which is eventually reflected as poorly illuminated regions in Figure 4.14. We show an accurate PBD [77] render in the same figure for comparison. We proposed using alternatives forms of sampling such as Poisson disk sampling which will create a consistent distribution of sample points per voxel. Further extensions are described in Chapter 6.

(a) Deformable Bunny

(b) Deformable Buddha

Figure 4.13: Multiple frames of deformable models are rendered with their front and back view. (a) Rendering of Stanford Bunny model (80k Triangles) with single and multiple scattering. (b) Rendering of Stanford Buddha model (100k Triangles) with single and multiple scattering.

Figure 4.14: (a) A reference PBD render. (b) In our ESLPV, artifacts can be seen in some areas with thin geometry such as the dragon legs and horns. In areas such as the mouth of the dragon, the dimmer illumination is caused by the flux lost due to undefined boundary conditions.

(a) SS + MS front-lit

(b) SS + MS back-lit

(c) SS + MS front-lit

(d) SS + MS back-lit

Figure 4.15: (a)(b) Front and back view of Stanford Buddha model (100k triangles) in jade material. (c)(d) Front and back view of Stanford Armadillo model (345k triangles) in apple material. All images have single-scattering (SS) and multiple scattering (MS) components. Results have been tonemapped with gamma compression.

(a) Material editing on Dragon model

(b) Material editing on Buddha model

Figure 4.16: Material Editing User Interface. On the right is the user interface displaying the extinction spectrum. We display the 3D model and we select regions of the material we wished to modify. The output result is shown on the left.

Chapter 5

Multi-resolution Screenspace

Optimization for Direct Illumination from Area Lights

Since our research deals with area light illumination for translucent materials, it is also important for us to render its direct illumination effects. Interactive rendering of direct illumination from area lights has often been constraint by the integration of the visibility function and radiance over the light surfaces. Direct illumination from area lights produces varying illuminated regions. These effects are usually visible as soft shadows. A complex scene with multiple objects of complex geometry usually requires a large amount of visibility samples to produce a noise-free image if an area light is present. These illumination effects are essential for realism in virtual worlds. Since our research deals with area light illumination for translucent materials, it is also important for us to render its direct illumination effects.

Several methods have been developed to render direct illumination from area lights. Monte Carlo approaches with distributed ray tracing can be used by taking numerous shadow rays per pixel, restricting the rays to the solid angle extended by the lights. There exist popular real-time methods such as variance shadow maps (VSM) [60] and convo-

lution shadow maps (CSM) [62], that avoid the computation overheads of Monte Carlo methods. Nevertheless, these methods approximate visibility by blurring edges in shadow maps, which only produces a rough approximation of visibility. This rough approximation would not be sufficient for realism as they tend to produce overly smooth shadows edges. In our work, we rely on point sampling on the light source for more accurate results.

Multi-resolution rendering [15] is an effective adaptive sampling method for reducing samples in screenspace. While standard Monte Carlo approaches [82] and distributed ray tracing techniques [83] emphasize on concentrating more samples and rays on difficult regions to allow an estimated value to converge per pixel, multi-resolution rendering instead focuses on finding ways to share and re-use information between large areas of pixels. Its concept is similar to irradiance caching [73], in which samples in low-varying illuminated regions can be re-used for computing illumination information for areas without samples. However, existing multi-resolution screenspace techniques are overly conservative which cause them to generate excessive fragments and visibility tests.

In this chapter, we aim to handle dynamic lights and viewpoints while interactively rendering direct illumination from area lights for diffuse materials. Our technique [3],[2](My Research Publications section) draws inspiration from the multi-resolution approach [15]. This multi-resolution approach will be used to render direct illumination on shadows generated by translucent objects.

5.1 Our rendering pipeline

Figure 5.1 shows an overview of our deferred shading method for diffuse materials. Our pipeline receives input textures (depth, normal and albedo) from the screenspace deferred shading. A center stage converts these input textures into fragments based on discontinuities in the normals, depth and visibility. The final rendered image is an overlaid result of the deferred shading using direct illumination multiplied by the albedo of visible objects. The red boxes indicate new methods added to the multi-resolution pipeline.

Figure 5.1: Complete pipeline of our direct illumination with area lights. Boxes in red indicate new proposed stages in the multi-resolution framework.

5.1.1 Overview

The direct illumination stage starts by generating a multi-resolution depth-curvature discontinuity mipmap. This depth-curvature discontinuity mipmap undergoes a thresholding process using a *TFS*, in which fragments that have geometric discontinuities are identified and generated in its relevant mipmap resolution. In our work, we refer to a fragment as a texel unit belonging to a mipmap level. In Figure 5.1, these fragments are represented as square patches, where the cyan texels represent fragments at the finest resolution. These fragments are transferred into our light culling and shadow refinement processes where they are processed again to detect visibility changes. After the process is completed, we have a set of 'stable' fragments. The irradiance of all 'stable' fragments are computed for different mipmap levels and stored in a multi-resolution texture known as the *illumination* texture. This texture is used in a single pass *RBF* interpolation process to approximate texels with no samples in the finest resolution irradiance texture. The irradiance texture is multiplied by the albedo to obtain direct illumination.

5.1.2 Geometric discontinuity and light culling

We give a brief description of the initial fragment refinement stages, Geometric Discontinuity and Light Culling stages which are also described in Nichols et al.'s paper [9].

Geometric discontinuity

Our first stage of multi-resolution refinement receives the depth and normal curvature discontinuity mipmaps similar to Nichols et al. [15, 16, 17, 9]. This depth and curvature discontinuity maps are obtained by first rendering a scene as seen from the camera, as well as storing depth, normal and albedo into textures. Next, a depth derivative and normal curvature max-mipmap can be generated by downsampling the depth and normal maps. This is generated from the maximum depth derivative from each of the four finer resolution texels. The normal curvature is computed by $\kappa_x = 2 * \sin(\arccos(\vec{N} \cdot \vec{N}_x)/2)$, where $\vec{N}$ is the normal of the current texel and $\vec{N}_x$ is the normal of the neighbouring texel in the x-direction. The same is done for κ_y in the y-direction. The magnitude of both curvature derivatives are computed by $\sqrt{\kappa_x^2 + \kappa_y^2}$. Fortunately, computing the derivative using neighbouring fragment information is a highly parallel process in the graphics shader pipeline. We currently restrict our depth discontinuity to only refine fragments up to 2x2 pixel size. This will avoid over-refinement caused by glancing camera angles on points far away from the camera.

Light culling

Fragments in screenspace can be culled off easily using the information of the location and orientation of the light. First, we can ignore any fragments on the light surface, since we do not render surface illumination on the light source. Secondly, we can detect geometry that are facing away from the light by testing $\vec{N} \cdot \vec{L}_j$, where $\vec{L}_j$ is the vector from the fragment center to a corner on the light and $\vec{N}$ is the normal of the fragment center. We can discard the fragments if $\vec{N} \cdot \vec{L}_j < 0$ for all j on the light. The light culling step can be performed by

ensuring that all fragments produced in the Geometric Discontinuity stage fulfill the front facing light condition.

5.1.3 Soft shadow refinement

We illustrate our shadow refinement technique named sub-fragment visibility test (SFVT) in Figure 5.2. The shadow refinement pipeline retrieves fragments that have passed the depth, curvature and light culling tests. It further performs ray tracing tests to check whether these fragments receive consistent illumination from the area light. We describe the stages of our refinement method in this section.

Sub-fragment visibility test (SFVT)

It is important to locate fragments where shadow boundaries are likely to appear. These fragments need further refinement to represent soft shadows. Shadow refinement is performed in Nichols et al. [9] by employing ray tracing to 256 samples (Virtual Point Lights (VPLs)) on the light surface. The visibility to VPLs are stored in a 256 binary bit array, where each bit represents the visibility to a light sample. The ray traced results are compared against their 8 neighbouring fragments in a 3x3 neighbourhood to check for discontinued visibility to the light samples. This requires 9 binary bit arrays to be computed for comparisons. If the visibility bit arrays differ in the neighbourhood, the fragment is further subdivided. For this fragment refinement metric, comparisons are made against neighbouring information outside the fragment instead of information purely within the fragment. This misalignment potentially causes unnecessary subdivisions as well as having potential misses for discontinuities within the fragment.

In our work, instead of comparing the visibility bit arrays with neighbouring fragments, we compare the visibility bit arrays computed from the 4 sub-fragments. This is because our refinement metric should be based on information located on the fragment of interest rather than information located outside of fragment. We refer to sub-fragments as evenly divided

points within a fragment that are used for visibility testing. We check if the 4 sub-fragments' visibility arrays differ from each other by a certain threshold. We use a threshold of 2 for small fragments of pixel size 1, 2^2 and 4^2 . For larger fragments of 8^2 pixels and above, we use a threshold of 1. This threshold indicates that we flag a discontinuity for approximately 8.25% difference in visibility bits. We use a low threshold, compared to Nichols et al.'s work [9], for a few reasons. Firstly, their work was measuring visibility bit differences across larger distances, while we are measuring across smaller distances. We are expected to have smaller changes in visibility differences compared to theirs. Secondly, we do not have a conservative diagonal refinement criterion like theirs which helps to generate extra fragments on the diagonals. We have to rely on a low threshold to generate these fragments instead. Lastly, we made observations that the chance of a refinement being flagged is low if we only have a small number of VPLs and are using a high threshold. This applies to Nichols et al. [9] work as well. Our proposed method resolves the potential issues caused by misalignments in Nichols et al.'s [9] work.

The number of bit arrays that we need to compute per fragment in Nichols et al.'s work [9] varies from 1 to 9, while it is 4 in our case. Checking for discontinuity within itself also generates lesser fragments as discontinuities tend to be smaller when comparisons are done across smaller distances compared to those of larger distances in neighbouring fragments. We note that it is redundant to further subdivide any fragments at the finest resolution, and hence these fragments should be ignored from the shadow refinement.

This implementation is still too generic if applied to all fragments as larger fragments might require more sample points rather than four. The largest fragment size in our case is at 128x128 pixel resolution. Subdividing the fragment to four sub-fragments of size 64x64 would still be too coarse to detect any visibility change. We instead decide that fragments at mipmap level, m , which are larger or equal to a certain mipmap level N , have to be subdivided into 16 sub-fragments of 2^{m-2} pixel width instead of 4 sub-fragments of 2^{m-1} pixel width. We use $N=5$, hence only splitting fragments that are 32^2 and above to 16 sub-fragments for visibility testing.

Figure 5.2: Our shadow refinement pipeline. The pipeline receives fragments (except those of finest resolution) that have passed the depth-curvature tests and light culling check. It performs a sequence of process: Subdivision, Ray Generation, Ray Tracing, Bit Comparison, and finally Shadow Thresholding. The SFVT (green outlined boxes) performs the visibility testing. The number of new fragments generated is decided by its gradient in GASS (red outlined rounded box). Newly generated fragments undergo visibility tests again, while fragments that are stable are transferred out. The filled rounded boxes in gray indicate components of the transform feedback shader (TFS) stage that we use to receive and process fragments as well as stream out fragments.

Ray generation and ray tracing

We generate K rays from each sub-fragment to random stratified positions on the light source. In our implementation, we use $K=16$ due to CUDA’s efficiency in dealing with threads of warp sizes. Hence, each fragment generates 64 rays from its sub-fragments. In cases where there are 16 sub-fragments, we trace 4 rays from each sub-fragment. This keeps the total number of rays fired to 64 rays per fragment.

Bit array computation

Our ray tracing produces visibility results between each of the 4 sub-fragments and points on the light. We use the visibility bit array similar to Nichols et al.’s work [9] for each sub-fragment. We label each of our sub-fragments in this section as A, B, C, D (refer to Figure 5.2). Our main fragment thread counts the bit difference, ray_diff , in visible rays

between each of its subdivided fragment. ray_diff can be computed by several OR ($\vee$) operations of all XOR ($\oplus$) operation of all pair combinations of the sub-fragments's binary visibility array, A_v to D_v , as seen in Equation 5.1.

$$\begin{aligned} ray_diff = & (A_v \oplus B_v)|(B_v \oplus C_v)|(C_v \oplus D_v)| \\ & (D_v \oplus A_v)|(A_v \oplus D_v)|(B_v \oplus D_v) \end{aligned} \quad (5.1)$$

$$Rays_M = \max(Count(A_v), Count(B_v), Count(C_v), Count(D_v)) \quad (5.2)$$

Figure 5.3: Four sub-fragment gradients are stored into the first 20 bits of the integer *max_rays* and the maximum number of light rays, $Rays_M$, visible among the sub-fragments is stored in the remaining bits.

In addition, we store three additional integers. One integer variable (32 bits), max_rays , stores the maximum number of rays ($Rays_M$) (Equation 5.2) that reach a fragment and gradient information of sub-fragments ($A_v \oplus D_v, B_v \oplus A_v, C_v \oplus B_v, D_v \oplus C_v$). The gradient information is used in Section 5.1.3. $Rays_M$ can be computed by counting the bits of the sub-fragment with the most binary '1' bits (refer to Equation 5.2). The function, $Count$, returns the number of '1' bits in a bit array. The 4 gradients are stored inside the first 20 bits of max_rays . Figure 5.3 shows how the gradient information are stored together with $Rays_M$ into the integer, max_rays . Another 2 integer variables, F_{v1}, F_{v2} , store the visibility array (64 rays into 64 bits) of the 4 sub-fragments. Figure 5.2 (gray box with green outline) shows the output from the SFVT. In cases where 16 sub-fragments are used, the ray information (4 rays per sub-fragment) in each of the 16 sub-fragments are accumulated to

4 lower resolution sub-fragments (16 rays each). We use nearest neighbour downsampling for this work. SFVT is performed on the 4 sub-fragments after downsampling. Similarly, the ray information from the 4 lower resolution fragments are stored in integers F_{v1}, F_{v2} .

Discontinuity thresholding for soft shadows

In our work, we use a GPU ray tracer which is meant for processing a point array rather than fragments from a 2D image. We use the standard graphics pipeline's *TFS*, which could generate a compact array of fragments' data which needs ray tracing. We supply a fragment list containing visibility information of its sub-fragments into the *TFS*. The total bit difference, ray_diff , is threshold against a user-defined value (we use 1-2 bit difference). If the bit difference is higher than the threshold value, it indicates that the fragment has varying visibility and should be subdivided into four or more fragments and output into a transform feedback stream (see Section 5.2.3). We note that our shadow refinement process could identify regions near shadow boundaries as visibility discontinuities tend to appear within fragments corresponding to these boundaries.

Gradient aware soft shadow refinement (GASS)

Once the above discontinuity thresholding is done, we can perform an additional refinement process that is able to refine a fragment to a maximum of 16 fragments of higher resolution instead of 4. We note that in some obvious scenarios, such as Figure 5.4a, a refinement to 4 higher resolution fragments is not sufficient to simulate soft shadow transitions and additional refinement passes are required in the next Transform Feedback pass. These additional refinement passes will generate additional visibility ray queries during each pass. The key idea to reducing unnecessary refinement passes is to identify regions where a single refinement is not sufficient. These can be identified from the areas with high gradients. We first compute the gradients based on the absolute number of visibility bit differences in each sub-fragment along the directions indicated by the red arrows in Figure 5.4b. The numbers inside the red arrows refer to the absolute gradient between the neighbouring sub-fragments.

Figure 5.4: In the normal subdivision scheme, subdivision is performed whenever the total number of bit differences between the 4 sub-fragments exceeds a threshold. This high discontinuity is determined by the total bit difference within a fragment. (a) The original subdivision scheme only refines a fragment to its next level of mipmap, which is insufficient and requires additional refinement in the next pass. (b) In our work, we subdivide each sub-fragment with high gradients into 2x2 sub fragments. The refinement criteria is described in the *GASS* refinement scheme.

If the absolute gradient is higher than the threshold value (we use 7 bit differences), the 2 sub-fragments used in the gradient calculation are refined into 4 higher resolution fragments each. This is equivalent to being refined 2 levels finer than the original fragment. For small gradients (below 7 bit differences), the sub-fragment only produces a single fragment. While it is arguable that we should use information within the fragment to decide for the second level of refinement, the *GASS* basically skips the need for this extra information by predicting fragments' configurations ahead by one level. This enables us to refine down two mipmap levels instead of one, which in turn reduces unnecessary visibility tests that are usually required in-between. These four gradient information of the four sub-fragments can be easily stored into the first 20 bits of the integer (*max_rays*), where each gradient data uses 5 bits for its magnitude.

Dark region culling

Fragments that are completely occluded from the light, based on $Rays_M$, can be removed from further refinement because they do not contain illumination. We can directly write the zero color value with an alpha bit set into the illumination texture at the same mipmap level as the fragment. This indicates that the fragment in the illumination texture still has valid information for upsampling and interpolation.

5.1.4 Additional fragments generation

The single sample location in a large stable fragment center may still miss regions with thin shadows. Hence, additional fragments are added to the refinement. Placing additional fragments along the edges of each fragment also resolves interpolation/extrapolation issues. We generate three additional fragments with a specific pattern to maximize coverage using a minimum set of samples. These three fragments of mipmap level 0 are positioned at the top-center, top-left and left-center of the fragment. Additional fragments are only generated on fragments of mipmap level greater than 2. Figure 5.5 shows the placement of these new fragments (highlighted in cyan). The green region describes the stable region that is defined by the soft shadow discontinuity thresholding process. For a scene shown in Figure 5.6, only additional 2100 fragments were added on top of the previous 50k fragments, in which their additional computation time in the final render are negligible.

5.1.5 Screenspace irradiance computation

The ray intersection visibility information obtained from ray tracing can be re-used directly to compute irradiance for fragments between mipmap level 1 to N_{max} , where N_{max} is the lowest resolution fragment mipmap level. For fragments in mipmap level 0, rays need to be generated from the point to a stratified sampled position on the light to compute visibility. The irradiance at point x_i can be computed with the ray information from K rays as follows:

Figure 5.5: Three fragments (in red) are positioned in the 16x16 and 8x8 sized fragments. Their location enables us to use radial basis functions to compute fragment values in between. Texels in green indicate the regions with little visibility discontinuity during SFVT.

Figure 5.6: Image in the center shows a fragment map of the Sponza Scene after visibility discontinuities have been done. Image on the left shows a zoomed in version of the fragment map of the center image. The image on the right shows additional fragments, particularly more obvious in the yellow circle (seen as small cyan dots), that are generated to improve the accuracy of interpolating larger fragments.

$$L(x_i, \vec{\omega}) = \frac{1}{\pi} \sum_1^K \frac{(\vec{N} \cdot \vec{L}_k) * (\vec{N}_{light} \cdot -\vec{L}_k) * A * I_{intensity}}{|x_i - x_{k_light}|^2}, \quad (5.3)$$

where $\vec{N}$ refers to the normal of point x_i . $\vec{L}_k$ refers to the vector from x_i to a point x_{k_light} on the light. $\vec{N}_{light}$ refers to the normal direction of the light. $I_{intensity}$ refers to the intensity at x_{k_light} with area A . This irradiance computation is similar to that in distributed ray tracing. We do not include the bi-directional reflectance distribution function (BRDF) as intensity might not change smoothly when there are BRDF differences between neighbouring fragments. We instead multiply the final irradiance texture with an albedo term when

Figure 5.7: Single pass upsampling receives fragments of different sizes and perform interpolation on them to produce a single irradiance map.

we render the final image.

5.1.6 Screenspace single pass upsampling

After generating fragments and computing their irradiance information in an illumination texture, the irradiance values of various fragment sizes are combined into a full image at its finest resolution, known as the irradiance texture. In the previous work by Nichols et al. [9], a multi pass upsampling algorithm is used, which performs bilinear interpolation upsampling and addition of illumination information for each mipmap level starting from the coarsest resolution. The multi pass algorithm gives too much influence to fragments from lower resolution. This is mainly because in each pyramid upsampling step, only information from lower resolutions can be obtained. This usually leads to artifacts seen in Figure 5.8a, as fragments from lower resolution may have errors propagating to the higher resolution fragments. These errors can be reduced by using their conservative diagonal refinement criteria [17] which generates excessive fragments near such visibility discontinuities. However, if these refinements were to be missed in their refinement stage, as seen

in the fragment map in Figure 5.8c, these artifacts are expected to be observed. In our proposed single pass upsampling, we were able to properly reduce the impact of those errors (Figure 5.8b) while using the same set of fragments as Nichols et al. [9]. This is done by using radial basis functions (RBF) to interpolate fragments' value.

Our single pass algorithm works by processing a full resolution texture, generating a fragment thread for each texel. In this section, we refer to the texel of the final irradiance texture as a *target texel*. If the target texel has illumination from mipmap level 0 to 2, we perform a direct copying of texel value from the illumination texture into the target texel. Subsequently, if the target fragment is from mipmap level 3 and above, we perform a boundary search as defined by the nearest two edges based on the quadrant that the target fragment falls in (Figure 5.10b). We only record down information of the nearest neighbouring fragment for each colored edge. We always use the additional fragments that were generated previously if they are closer to the target fragment than the neighbour texel's center. The chosen neighbouring sample must be also within a distance of less than two times the texel size of the target fragment's mipmap level.

Since our sample data are scattered, we use scattered data interpolation techniques [84]. We employ Gaussian RBF as they provide a naturally smoothing function for interpolating scattered samples. Any two neighbouring texels or internal additional samples with the 2D center position of the target texel form a group of scattered samples, texel $\mathbf{x}_i$, needed for our basis functions Φ_i in Equation 5.4. To obtain an estimated irradiance value $\hat{I}(\mathbf{x})$ at target texel $\mathbf{x}$, we first need to evaluate weight w_i for each basis function Φ_i . This can be done by solving the linear Equation 5.5a, where $\mathbf{w}$ is a vector of weights w_i and Φ is a correlation/distance matrix consisting of i rows and j columns of Φ . $\mathbf{I}$ is a vector consisting of irradiance values from the chosen samples.

$$\hat{I}(\mathbf{x}) = \sum_i^3 w_i \Phi_i(\|\mathbf{x} - \mathbf{x}_i\|), \quad (5.4)$$

(a) Nichols et al. [9] multi pass upsampling

(b) Our single pass upsampling

(c) Fragment map

Figure 5.8: (a) Artifacts (dotted red ellipse) in the multi pass upsampling algorithm. Larger fragments are able to dominate pixel values and ignore smaller neighbouring fragments despite being less accurate. This artifacts appear as small holes or spikes near shadow boundaries. (b) Our single pass algorithm reduces these artifacts (not completely) by giving more weights to smaller fragments. (c) Both images in (a)(b) are rendered with the same fragment map based on Nichols et al. [9] refinement. As observed by the yellow dotted ellipse, sometimes a large fragment may fail to be refined.

(a) Adaptive Variance Scaling Factor

(b) Fixed Variance Scaling Factor [3]

Figure 5.9: Comparisons in per pixel square error with a Monte Carlo reference image of the Sponza scene (1280x960 pixels, 64 samples per fragment). Red color channel image on the right signifies the per pixel difference. The same fragment refinement and number of fragments are used for both renders. The errors have been scaled up to 9x for easier visualization. (a) Our adaptive variance scaling factor approach results in significantly smaller sum of square errors (1.112) than (7.482) our previous fixed variance scaling approach [3] in (b).

where

$$\mathbf{w} = \Phi^{-1} * \mathbf{I}, \quad (5.5a)$$

$$\Phi_{ij} = \exp(-d^2/C) \quad (5.5b)$$

In Equation 5.5b, d is the L2 distance in texels between the chosen sample location, $\mathbf{x}_i$, and the target fragment $\mathbf{x}$. We note that d was divided by the mipmap width of the target texel in our earlier work [3]. This scaling factor enabled samples from coarser resolution to have higher variance, and those of finer resolution to have lower variance. Although, this approach has solved some artifacts related to the previous multi pass upsampling method, it led to fixed pattern artifacts in some parts of our results as seen in the error map in Figure 5.9b (right). For large fragments, this inversely large scaling factor will cause nearby high resolution samples to have no significant difference in weights. In the worst case, the correlation matrix, Φ , will be singular. We introduce a varying scaling factor, C , for fine tuning the variance in Equation 5.5b based on the samples chosen. We describe the computation for C in Equation 5.6 and also in the next paragraph.

$$C = 2 * (\hat{M} + 1) \quad (5.6)$$

$$\alpha_i = \exp(-\hat{d}^2 / (2 * t_{min})) \quad (5.7)$$

$$\hat{M} = \frac{\sum \alpha_i * M_i}{\sum \alpha_i} \quad (5.8)$$

Firstly, for a chosen set of samples for a target texel, we find the smallest fragment width, t_{min} (in texels), among the samples and use them as a variance scaling factor for our Gaussian weights in Equation 5.7. The value $\hat{d}$ is the normalized L2 distance between a sample i and the target texel. Next, we compute a Gaussian weight, α_i , and retrieve the mipmap level, M_i , for each sample and compute a normalized weighted sum to get a

distance weighted mipmap level $\hat{M}$. This term is used to compute our variance scaling factor, C . An additional of one added to $\hat{M}$ prevents the division by zero error. Intuitively, Equation 5.8 conveys that a high resolution sample among the chosen set will reduce the variance. Hence, if a target texel is located in a large fragment but has several higher resolution samples chosen, it will have a low variance. These higher resolution samples will reduce the impact of the lower resolution fragment.

We use only 3 to 4 RBFs such that the inverse could be easily calculated using the inverse function in the standard shader pipeline. In Figure 5.10a, we show the region interpolated by our RBFs in the yellow triangle for computing the value of a target fragment (in blue). In target texels with less than 3 RBFs, we only need to do weighted interpolation using the function in Equation 5.5b between 2 samples, $\mathbf{x}_1$ and $\mathbf{x}_2$, which produces the following interpolated value:

$$\hat{I}(\mathbf{x}) = \frac{(\Phi(\|\mathbf{x} - \mathbf{x}_1\|) * I(\mathbf{x}_1) + \Phi(\|\mathbf{x} - \mathbf{x}_2\|) * I(\mathbf{x}_2))}{\Phi(\|\mathbf{x} - \mathbf{x}_1\|) + \Phi(\|\mathbf{x} - \mathbf{x}_2\|)} \quad (5.9)$$

We show comparisons with our earlier work [3] which uses a fixed variance scaling factor based on fragment size in Figure 5.9b and our current work which uses an adaptive variance scaling factor based on weighted fragment size in Figure 5.9a. We exhibit 5x lower sum of square pixel error for this particular image. Further comparisons with Nichols et al. and Monte Carlo reference images are presented in Section 5.3.

5.2 Implementation

5.2.1 Depth and curvature discontinuity check

We utilize the stream-compaction feature for the transform feedback shader (*TFS*) in OpenGL 4.0 (also available in DirectX 11). This pipeline allows us to produce four separate arrays for our results. We are able to generate a separate list of 2D fragments that recursively require to be checked for discontinuity in its finer mipmap levels. The input fragments for

Figure 5.10: (a) Our upsampling scheme on a target blue fragment. The blue fragment is computed by using radial basis functions selected by samples closest to the two boundaries edge (in green) and its nearest fragment center. The cyan squares refer to the additional fragments that were generated for large fragments. (b) If the target texel falls within the top left of its parent fragment, the two edges in blue are traversed to look for neighbouring samples. Similarly if the fragment falls in the bottom right, the red edges are traversed. The similar can be said for the green and yellow fragments which falls in the top right and bottom left. The arrow indicates the direction to search for neighbouring samples.

the shadow refinement stage are also accompanied by their normals and positions. We use the transform feedback shader since it is the fastest parallel processing pipeline to generate a filtered compact stream from an unordered list of inputs.

5.2.2 Ray intersection test

As our input to the shadow refinement stage is in a tightly packed array, we can easily make use of CUDA's GPGPU advantage to subdivide these input sample points and create ray information which are suitable for OptiX Prime ray tracer [81]. CUDA's *shuffle* operations or *SHFL* also makes it easier for us to perform reduction operations such as ray counting or ray summation when generating the visibility array.

5.2.3 Discontinuity thresholding - Transform feedback shader

Similar to the depth-curvature discontinuity *TFS*, we make use of OpenGL’s stream-compaction feature to branch our results in the shadow refinement process. The first stream stores 4 to 16 sub-fragments coordinates depending on how GASS decides. This stream is for transferring fragments that need further visibility testing. The second stream stores the 2D normalized screenspace positions of fragments that are greater than level 0 for those stable fragments. Tagged together with the second stream stores the visibility information of 64 rays using two 32-bit integers, this data can be re-used for computing irradiance. The last stream stores the 2D normalized screenspace positions of fragments that belong to mipmap level 0, the finest resolution. Once the refinement process is completed, we can read the 3D positions and normals from the fragments’ normalized 2D screenspace coordinates since screenspace positions and normals are provided in the initial screenspace render. Fragments of level 0 are separated into a different stream because they do not have any visibility information that can be re-used for irradiance computation. They should be stored separately and appended to the remaining fragments. We refer the reader to Figure 5.11 for the streaming process.

The irradiance of each fragment in stream 2 is computed in CUDA using the visibility information that is also present from the stream. They are then rendered into an illumination texture. We note that there is an implementation difference compared to our previous work [3]. In our earlier work [3], we have to sort the fragments into their respective mipmap level such that each fragments can be rendered into their appropriate mipmap level in the framebuffer via multiple passes of rendering. In this work, we use bindless image textures (introduced in OpenGL 4.2 `GL_ARB_bindless_texture` together with `ARB_shader_image_load_store`) to write into every mipmap level of the irradiance texture concurrently without attaching any textures to the framebuffer. This approach removes any computation overheads involved in sorting fragments. In Nichols et al. [9]’s method, a flattened texture, which consists of all mipmap levels being appended to a single

layer, is used instead. We avoid using their method as we have to recopy the texture to its un-flattened version for more efficient texture reading. We describe our bindless rendering process in Figure 5.12.

Figure 5.11: In our transform feedback shader for shadow refinement, three streams of output are produced. The *TFS* receives fragments locations together with its ray tracing information. The first output stream (box in green outline) returns unstable fragments of mipmap level > 1 , which should be further refined in the next cycle. Stream 2 (box in red outline) stores corresponding screenspace 2D coordinates and visibility information as 2 output arrays. Stream 3 (box in blue outline) stores screenspace 2D coordinates of level 0 fragments.

5.3 Results and discussion

We show our rendering results (Sponza, Sibenik and hairball scenes) in Figures 5.13a, 5.14a and 5.15a. The images are rendered with 64 samples per fragment in 1280x960 resolution with a large majority of samples being re-used from the shadow refinement stage. Table 5.1 shows the performance and data of the 3D models that we rendered. The rendering was performed on an Intel i5 3.40GHz CPU with a NVIDIA GeForce GTX 980 GPU. The time needed in milliseconds (ms) for the visibility tests and total rendering time are provided in the table. Upsampling takes a fairly little amount of time in all scenarios (1ms). This is mainly due to the fact that it is a screenspace algorithm. We show the Monte Carlo references with 64 samples per pixel in Figures 5.13c, 5.14c, 5.15c while Figures 5.13d, 5.14d, 5.15d show visual representation of the fragments we used, with cyan being the

Figure 5.12: Once our set of stable fragments is retrieved, fragments are converted to their 3D positions, x and normals N . Fragments larger than mipmap level 0 can reuse their visibility information from our SFVT to compute their irradiance based on the formulation for distributed ray tracing. Those at mipmap level 0 will need to perform a visibility test before their irradiance values can be computed. The fragments, with their irradiance values, will then be written to the appropriate mipmap level of textures via bindless image textures.

color of the highest resolution fragment and white being the lowest resolution fragment. Our results, particularly shadow boundary regions such as the shadows caused by a large area light in Figure 5.13a behind the pillars, are similar to our Monte Carlo references.

As seen in Table 5.1, our timings are 24% to 45% faster and generates 9% to 37% fewer fragments than Nichols et al. [9]. Our single pass upsampling stage also significantly reduces errors in magnitudes lower than Nichols et al.’s multi pass technique (33x to 81x). The improvement in per pixel sum of square error is more evident in scenes with complex geometry. In Figure 5.13a, our technique directly reduces the number of visibility samples compared to Nichols et al. [9] by 2.8 times. Figures 5.13d (center), 5.14d(center), 5.15d (right) show that the difference in our results (for direct illumination) compared to the Monte Carlo references is barely visible unless it is scaled. The absolute difference in the pixel values to the reference images is very small unless magnified. Nevertheless, it should be understood that error metrics such as sum of square error scales up with the overall brightness in the scene. Hence we have also provided the normalized sum of square error (NMSE), which is equivalent to the sum of square error normalized by the

sum of pixel intensity in the reference image. Similarly, the results from Nichols et al.'s work [9] are shown in Figures 5.13b, 5.14b, 5.15b with their error maps visualized in Figures 5.13e(center), 5.14e(center), 5.15e(right). Additional error color maps are provided in Figures 5.13e(right), 5.14e(right), where blue pixels indicate low errors and red pixels indicate the high errors.

The hairball object in Figure 5.15 is a much more complex object and we were able to well approximate its fine details and the shadows it generates. For this particular scene, we were able to achieve errors of 80x lesser than that in Nichols et al.'s work [9], mainly because we avoid blurring the geometric details on the surface of the hairball. However in terms of fragments generated, our scheme only reduces it by 9%. This is because the soft shadow regions only take up a small proportion of pixels compared to the entire image. The rendering time for this model is higher compared to that in the Sponza and Sibenik scene despite fewer fragments. This is mainly due to the computational overheads of ray tracing through a complicated mesh of 2.8 million triangles. Additional videos of our renderings can be found in the link¹.

Our technique can handle dynamic lighting, dynamic viewpoints and deformable or moving geometry. However, rendering time for moving geometry can be constrained by the number of triangles in the geometry. This is due to the time needed for reconstructing the acceleration structure in a ray tracer. A rasterizer ray tracer, which uses a voxel acceleration structure, will perform better in this aspect, but we will need to consider the amount of ray marching in rasterizer ray tracers which will make ray tracing slower. As the research is not emphasized on the type of ray-intersection tests used, both voxel-based visibility methods and ray tracing methods are viable for visibility testing. Nichols et al. [9] was able to render their results much faster with a voxel based ray intersection tests, but their refinement criteria and interpolation technique resulted in higher errors and over-smoothing in some unwanted scenarios.

The single pass upsampling algorithm produces certain fixed pattern artifacts. This is

¹https://www.dropbox.com/s/hkthjr2mfviuh25/CAG_Supplemental%20Video.MP4?dl=0

Figure 5.13: Rendering of the Sponza (McGuire Graphics Data) in 1280x960 pixels with direct illumination. In Figure 5.13d (center), we show the L2 error map (in red) in irradiance values (with scaling factor of 1x and 9x) between Figures 5.13a and 5.13c. Similarly Figure 5.13e (center) shows the L2 error map between Figures 5.13b and the reference image 5.13c. In Figure 5.13d, 5.13e (right), we have an error color map indicating the percentage of pixel error. The errors are represented by five color zones from less than 5 % (blue) to more than 25 % (red). (f) shows the location and size of our planar light source.

Figure 5.14: Rendering of the Sibenik (McGuire Graphics Data) in 1280x960 pixels with direct illumination. In Figure 5.14d (center), we show the L2 error map (in red) in irradiance values (with scaling factor of 1x and 9x) between Figures 5.14a and 5.14c. Similarly Figure 5.14e (center) shows the L2 error map between Figures 5.14b and the reference image 5.14c. In Figure 5.14d, 5.14e (right), we have an error color map indicating the percentage of pixel error. The errors are represented by five color zones from less than 5 % (blue) to more than 25 % (red). (f) shows the location and size of our planar light source.

mainly due to the artifacts caused by extrapolation in RBF interpolation, as well as the lack of correlation between samples used between neighbouring fragments. Nevertheless, this issue can be solved by adding more samples to the RBF but with performance in consideration, we only used up to 4 samples. Skala [69] proposed using an incremental block matrix method to compute the inverse of the correlation matrix. His method could be used when dealing with more than 4 samples, which we leave for future work. In our single pass upsampling method, we do gain some trade-offs in using lesser texture memory due to the removal of intermediate textures that were formerly needed in the multi pass upsampling approach. Further optimizations, such as approximating depth discontinuity based on the size of the light, can be considered. Artifacts may also be present due to undersampling in the presence of large area lights. For these cases, it is advisable to use more visibility rays. We show further results with planar lights of varying sizes. Although casting 16 rays per sub-fragment is sufficient for these light sizes based on our experiments in Figure 5.17, noise from undersampling would be expected for larger lights. This noise can be observed in Figure 5.16a where the light source is large and 64 samples per fragment is insufficient in removing the noise. However, this noise is unlikely to contribute towards higher error rates. This is because for larger light sources, higher number of fragments are expected to be produced, which in turn, reduces overall error as observed in Figure 5.17e. We show further results in our 'NMSE vs Light Size' chart in Figure 5.18 for the Sibenik scene configuration in Figure 5.14a. The NMSE error rate does not fluctuate much and remains significantly lower than Nichols et al.'s [9] approach. Although we have not demonstrated using textured lighting in our work, it can be done by generating a set of VPLs on the textured light for visibility testing in a similar way to Nichols et al.'s [85]. We can set a limit on the maximum number of VPLs allowed to ensure interactive performance. Similar to Nichols et al.'s paper [9], we do not refine a fragment based on differences in light energy entering it but on visibility differences, hence the textured appearance of the light has no effect on the rendering performance.

Our current experiments were designed for diffuse materials, mainly due to the effi-

ciency of overlaying the irradiance texture over a screenspace albedo texture. It should be preferred over Nichols et al.'s [9] if we are rendering illumination from small area lights under slowly-varying geometry. If specular reflections are required, we would need to perform importance sampling (by taking visibility samples to a specular cone) similar to Nichols et al.'s [9]. The reflected lighting has to be multiplied by a BRDF function rather than a constant albedo term.

Nevertheless, there are three situations where a multi-resolution approach might perform poorly due to the large amount of refinements needed. The first situation occurs when large area lights are present. The second occurs when complex geometry dominates the visible scene. The third occurs when complex BRDF are present in the scene. Under these scenarios, Monte Carlo approaches can produce better rendering results.

5.4 Summary

We have presented a multi-et al. approach that is able to render direct illumination efficiently by culling off large portion of unnecessary fragments using our sub-fragment visibility test (SFVT) and gradient aware soft shadow refinement (GASS) techniques. The SFVT scheme performs visibility discontinuity check across a smaller distance and area, and tends to generate less fragments compared to previous conservative methods. Our GASS technique then decides on the number of fragments for refinement. A single pass Gaussian RBF interpolation upsampling approach was proposed to reduce the impacts of shadow artifacts that were visible in the previous multi pass upsampling approach. In addition, our shadow refinement approach was able to fully utilize the streaming architecture of the transform feedback shader as well as the bindless texture extension. Both of these new graphics extension effectively optimizes the multi-resolution framework. From our results, we were able to achieve our research goal of interactive area light re-lighting. Our designed algorithm continues to effectively requires no precomputations.

Table 5.1: Rendering statistics for 1280x960 images. Figures 5.13a, 5.14a, 5.15a show our results, while Figures 5.13c, 5.14c, 5.15c are Monte Carlo references. Nichols et al.’s [9] result are shown in Figures 5.13b, 5.14b and 5.15b. L2 Error refers to the sum of square error when compared to the reference image. The normalized mean square error (NMSE) is also provided in the brackets after the L2 error. NMSE is represented using scientific notation. Each pixel is stored in normalized float format. A lower error indicates a better quality.

Figure	Triangles	Fragments	Visibility Rays	Visibility Test (ms)	Up-sampling (ms)	Time (ms)	L2 Error (NMSE)
5.13a - Ours	66450	327,775	7,817k	44	1	105	12.14 (5.42e-4)
5.13b - Nichols et. al. [9]	66450	517,778	23,071k	91	4	194	204.1 (7.28e-3)
5.13c - Monte Carlo	66450	1,228,800	-	-	-	219	-
5.14a - Ours	75284	380,889	4,143k	28	1	89	7.108 (2.82e-4)
5.14b - Nichols et. al. [9]	75284	431,637	12,032k	58	4	131	134.3 (5.32e-3)
5.14c - Monte Carlo	75284	1,228,800	-	-	-	173	-
5.15a - Ours	2,850,000	223,231	1,820k	57	1	258	18.38 (1.07e-3)
5.15b - Nichols et. al. [9]	2,850,000	245,864	5,450k	118	4	340	1473 (7.10e-2)
5.15c - Monte Carlo	2,850,000	1,228,800	-	-	-	484	-

(a) Our result

(b) Nichols et al.'s [9] result

(c) Monte Carlo reference

(d) Our fragment map and error map

(e) Nichols et al.'s [9] fragment map and error map

Figure 5.15: Rendering of the hairball object (McGuire Graphics Data) in 1280x960 pixels with direct illumination. In Figure 5.15d (right), we show the L2 error map in irradiance values (with scaling factor of 1x and 9x) between Figures 5.15a and 5.15c. The error map is mapped to the red color channel. Similarly Figure 5.15e (right) shows the L2 error map between Figures 5.15b and the reference image 5.15c.

Figure 5.16: (a) The light configuration in the Sibenik scene. (b) A Monte Carlo reference is rendered for direct illumination from the area light source at 64 samples per fragment. (c) Rendering of our result at 64 samples per fragment. Visible random noise from under-sampling can be observed when the light is too large.

Figure 5.17: Experiments with varying area light sizes and their positions. Renderings are done in 1280x960 pixels. First row indicates 2 various sizes of lights that we used. Second row indicates the results of rendering for a light of size 2 for our result (c) and its error map, at the top right, (scaled 20x) is computed by comparing against a Monte Carlo reference (d). Third row indicates the results of rendering for a light of size 3.5. Sum of square errors (NMSE) for the rendered images in (c) and (e) are 6.225 (4.87e-4) and 4.297 (3.59e-4). The bottom right image of (c),(e) shows the fragment map.

Figure 5.18: Plot of NMSE vs Light Size. The horizontal axis represent the one dimension length of the light, where the actual area of the lights used in the experiments are from 1^2 , 1.5^2 , to 3^2 . NMSE of our work is represented by the blue lines. Our error rate remains significantly lower than Nichols et al. [9]. Overall error in the image is unaffected by the light size.

Chapter 6

Area Light Illumination Rendering for Subsurface Scattering

Interactive rendering of scenes with highly scattering translucent objects is difficult to compute efficiently. Although approximation techniques can model the local illumination of translucent materials, techniques that model their indirect illumination interactively on surrounding surfaces are rarely researched. Translucent objects can act as area lights when they release scattered light to their surroundings to produce visual effects in the appearance as soft diffuse color bleeding, e.g. Figures 6.1a, 6.1b. Although it is easier for rendering algorithms to work with translucent materials using point light sources, area lights often produce soft shadows which exhibit coloration by scattered lighting within the translucent materials. This effect contributes greatly to the visual realism in interactive rendering. This effect is not present on diffuse surfaces as they are usually optically thick (e.g. Figure 6.1c). Additionally no indirect illumination will reach the external objects as it is physically impossible.

Monte Carlo techniques [86, 87] can render interreflections from translucent materials but they are too computationally expensive. Improvements in other methods, such as photon mappings, allow translucent materials to produce indirect illumination effects [88, 89], but they are impractical for generating images at interactive rates. Current radiosity [32] ap-

Figure 6.1: Rendering of a water glass with a red scattering material illuminated with a single area light source inside the cup at 1024x768. We show our rendering in image (a) with both the direct and indirect illumination components, and a reference image generated by Mitsuba Path Tracer (64 samples per pixel) in image (b). The translucent material in the reference image is rendered with Photon Beam Diffusion method [77]. Image (a) is rendered in 146 ms. Image (b) is rendered in 54.96 seconds. Image (c) shows a diffuse water glass. No light is scattered to the exterior plane of the water glass.

proaches allow for real-time computations, but at the expense of long precomputation time and large storage data for precomputing form-factors.

In this chapter, we present a voxel based illumination approach [4], [1](My Research Publications section) that renders translucent objects under direct area light illumination and renders indirect illumination from translucent and diffuse surfaces interactively without precomputations.

6.1 Our pipeline

We combine the ESLPV, in Chapter 4 and LPV [8], for our illumination pipeline. The ESLPV renders local subsurface scattering effects while the LPV distributes the indirect illumination from diffuse surfaces as well as translucent objects. The 3D objects in our scene and the translucent objects are voxelized for the LPV. The translucent objects can be voxelized according to the solid voxelization algorithm described in Schwarz et al. [75]. Voxelization allows us to store geometric information and material properties into these voxels, which are used for propagation. In our work, we use the LPV structure from Doghramachi's work [90] because of the way geometric blockers are represented in the voxels. Their method of storing normals of blockers into a tetrahedron face allows us to extract the closest normals relevant to the surface we are rendering. This allows more accurate illumination computations.

As previous work for the ESLPV in Chapter 4 and LPV [8] did not deal with area light illumination, we design a complete framework for rendering translucent materials under area lights as shown in Figure 6.2. Our system starts off with area light and scene information (e.g. mesh information). We generate Poisson disk samples [91] for both our translucent and diffuse objects. We use the notation diffuse Poisson disk samples (DPDS) and translucent Poisson disk samples (TPDS) to describe the two sets of Poisson disk samples generated on diffuse and translucent surfaces respectively. Every TPDS will have its light intensity computed and transferred into the ESLPV for injection. The ESLPV renders the translucent object with the given set of TPDS. These set of TPDS are downsampled to a reduced set of translucent planar lights (TPLs). This set of TPLs is used to represent the translucent object as an area light source. The intensities of the TPLs can be obtained by sampling the ESLPV voxels with a rendering equation after the propagation stage has been completed.

Next, we compute the reflected flux at the location of each DPDS. The reflected flux is computed with both the area light information and the TPLs, indicating that each DPDS

Figure 6.2: The pipeline of our work with the indirect illumination combined with the direct illumination output.

now contains the reflected flux of direct illumination from the area light and the TPLs. The DPDS are injected by accumulating their intensities with previous TPLs into the LPV and rendered. By propagating the light intensity in the LPV designed by Doghramachi [90], we are able to simulate the indirect illumination of translucent materials as well as diffuse materials in the scene. The final rendered image is combined with our direct illumination (for diffuse surfaces), which handles area light direct illumination. We implemented the previously proposed method of direct illumination [2]) using multi-resolution rendering under area lighting (Chapter 5). This method is able to produce high quality soft shadows and is suitable for overlaying our indirect illumination component. The right side of Figure 6.2 shows the various output of each injection stage.

6.1.1 Direct illumination for translucent objects from area lights

We first distribute a set of TPDS on the surface of a translucent object. For each Poisson disk sample, we perform a ‘gathering’ operation in which the transmitted flux from each refracted light ray from the area lights is computed. We create a ESLPV voxel structure of 32^3 resolution for the translucent objects. The light intensities are stored according to the refracted light directions. The transmitted flux entering the translucent medium from each refracted light ray is converted to its SH representation of a clamped cosine lobe and is

accumulated into the voxels corresponding to their locations. We describe the transmitted flux at point x_i from a light ray emitted at a virtual point light (VPL) (from uniformly divided patch from the area light) location x_{light} from direction $\vec{\omega}$ as $L_t(x_i, \vec{\omega})$ in Equation 6.1a. We use x_{light} as the representative point for each uniformly divided patch on the area light from uniform sampling. $T(\vec{\omega}, \vec{\omega}')$ refers to the Fresnel term for describing the fraction of light energy transmitted into the translucent material after entering from direction ω_k and refracted to direction $\vec{\omega}'$. A represents the area of a uniform patch on the area light defined by the VPL from uniform point sampling. A is used as part of the form factor computation between area to point contribution. $\vec{N}$ refers to the normal of the TPDS, and $\vec{L}$ refers to the normalized ray direction from the TPDS to the light. $I_{intensity}$ refers to the light intensity from the VPL. Equation 6.1a can be converted to its Spherical Harmonics (SH) representation in Equation 6.1b where the injected flux for each Poisson disk sample is represented in the SH coefficients, $c_{lm}^{(\vec{\omega}')}$, of a clamped cosine-lobe oriented at the refraction vector, $\vec{\omega}'$. y_{lm} refers to the SH basis functions. Figure 6.3 describes how the refracted light energy is injected into the ESLPV voxels. The SH coefficients in each ESLPV voxel are accumulated by summation and then inversely scaled by the total number of TPDS located in them. Once the light intensities are injected into the ESLPV voxels, we can perform the light propagation as usual, which creates the local information required for rendering the translucent appearance.

$$L_t(x_i, \vec{\omega}) = \frac{T(\vec{\omega}, \vec{\omega}')(\vec{N} \cdot \vec{L}) * (\vec{N}_{light} \cdot -\vec{L}) * A * I_{intensity}}{|x_i - x_{light}|^2}, \quad (6.1a)$$

$$L_{t,lm}(x_i, \vec{\omega}) = \frac{[\sum_{l=0}^{\infty} \sum_{m=-l}^l c_{lm}^{(\vec{\omega}')} y_{lm}]}{\pi} * L_t(x_i, \vec{\omega}) \quad (6.1b)$$

Figure 6.3: Injecting transmitted and reflected light flux into ESLPV and LPV, respectively, with area lights. For diffuse materials, light is injected into the LPV voxel above the surface. For translucent materials, light is injected into the ESLPV voxel on the surface.

6.1.2 Indirect illumination from area lights

Indirect illumination for diffuse surfaces from area lights

As the usual LPV scheme does not handle area lights, our proposed solution generates a set of Poisson disk distributed samples [91] on the 3D scene we are rendering. The illumination for each Poisson disk sample is computed by performing a ‘gathering’ operation to the light source. We only gather contribution from rays that pass the visibility test. We compute the reflected flux, L_{VPL} , for each diffuse Poisson disk sample, x_i , which is a summation of the contribution of each individual light ray to the sample point (refer to Equation 6.2b):

$$F_s = (\vec{N} \cdot \vec{L}_k) * (\vec{N}_{light} \cdot -\vec{L}_k), \quad (6.2a)$$

$$L_{VPL}(x_i, \vec{\omega}) = \frac{1}{\pi} \sum_1^K \frac{F_s * A * I_{intensity} * \rho}{|x_i - x_{k_light}|^2} \quad (6.2b)$$

where $\vec{N}$ refers to the normal of Poisson disk sample x_i . $\vec{L}_k$ refers to the vector from x_i to a VPL, x_{k_light} on the light. $\vec{N}_{light}$ refers to the normal direction of the light. F_s refers to the foreshortening factor. $I_{intensity}$ refers to the intensity at x_{k_light} with area A from

uniform sampling. A is used for area to point form factor computation. ρ refers to the diffuse coefficients of the Poisson disk sample.

The reflected flux in each Poisson disk sample is then deposited into the respective voxel at one unit normal distance away from the Poisson disk sample in their Spherical Harmonics (SH) representation. Following Kaplanyan et al. [8], we can first use a clamped cosine lobe oriented in the z-axis, represented by zonal harmonics, rotated to the direction $\vec{N}$. The flux distribution when converted to SH Coefficients is represented in Equation 6.3. A normalization factor of $\frac{1}{\pi}$ is required to conserve the energy in the reflected flux.

$$I(x_i, \vec{\omega}) = \frac{L_{VPL}(x_i, \vec{\omega})}{\pi} * \sum_{l=0}^{\infty} \sum_{m=-l}^l c_{lm}^{(\vec{N})} y_{lm} \quad (6.3)$$

In the LPV implementation, we are only required to store the 4 SH coefficients (2 bands) multiplied by $\frac{L_{VPL}(x_i, \vec{\omega})}{\pi}$ into each voxel. Each color channel (R,G,B) is stored separately as 4 floats of SH coefficients each. Appendix A.4 shows further explanation for the SH rotations. Once we have obtained the SH coefficients by a cosine lobe orientated in direction $\vec{N}$, we can scale these coefficients by the irradiance of the Poisson sample. For Poisson samples that lie in the same voxels, the intensities can be accumulated in the same voxel.

The remaining process follows the standard LPV [8] pipeline. Propagation is performed to distribute the light intensity throughout the LPV. The final result is an approximation to the indirect illumination distributed by diffuse surfaces. We refer to Figure 6.3 for the light injection process from area lights. The contents in this section is mentioned in our paper [4].

Indirect illumination from translucent materials

Scattered light from the translucent material can be represented by a sparse set of planar lights, with a point location allocated to its center. We downsample the set of TPDS that was generated earlier in Section 6.1.1 to easily get a reduced set of samples known as TPLs. The main intention for downsampling is to ensure that we are able to reduce the total

number of samples for representing the translucent object. This reduces computation time significantly. For our downsampling algorithm, we basically reduced the 32^3 ESLPV to a 8^3 representation and temporarily deposit each TPDS into their corresponding locations in the 8^3 voxel volume. We then iterate each voxel of the 8^3 voxel volume and find the TPDS that is nearest to the center of each of the voxels in the 8^3 volume. Since the original Poisson disk sampling approach by Bowers et al. [91] distributes samples according to some voxel resolution, our downsampling approach would still allow us to reasonably obtain samples that are evenly distributed. The nearest TPDS in each voxel is chosen as the TPL. This chosen sample maintains its location on the surface of the object, unlike interpolation methods which might provide a non-surface sample. TPLs are represented as virtual planar light sources with an area allocated to them. This representation corresponds to how voxels are represented as planar area lights in the SSLPV [7] (Equation 2.56). The output radiance of each TPL can be computed by Equation 6.4. A_{voxel} is the area of a voxel face of the ESLPV. There is no outgoing Fresnel term as it is incorporated when DPDS ‘gather’ illumination in the irradiance term from the TPL (Equation 6.5b).

$$I_{TPL}(x_{TPL}, \vec{\omega}_o = \vec{n}) = \frac{2}{A_{voxel}} \left(\sqrt{\frac{1}{4\pi}} c_{00}^{(\vec{n})} + \frac{1}{2} \sqrt{\frac{3}{4\pi}} c_{10}^{(\vec{n})} \right) \quad (6.4)$$

Once a set of TPLs is computed, each of the DPDS will perform a ‘gathering’ operation on the TPLs (refer to Figure 6.4). DPDS are represented in red circles while TPLs are represented in green circles. The reflected flux contribution from each TPLs is then added to the original illumination contribution $L_{VPL}(x_i, \vec{\omega})$ from area lights in Section 6.1.2. This contribution is then stored into the respective voxel location corresponding to each of the DPDS’ location. The reflected flux contributed by K number of TPLs is computed in Equation 6.5b, where K is the number of TPLs visible from x_i . $I_{TPL}(x_{TPL}, \vec{\omega}_o = \vec{n})$ is the intensity of the TPL computed by Equation 6.4. $T(\vec{\omega}, \vec{\omega}')$ is the Fresnel transmission term describing the percentage of light energy leaving the translucent medium from direction $\vec{\omega}'$ to $\vec{\omega}$. F_s is the foreshortening factor as described in Equation 6.2a. ρ is the albedo of the

material at x_i . A_{TPL} refers to the surface area represented by each TPL. This is computed by dividing the total surface area of the translucent object with the total number of TPLs on the object.

$$L_{TPL_Single}(x_i, \vec{\omega}) = \frac{F_s A_{TPL} T(\vec{\omega}, \vec{\omega}') I_{TPL}(x_{TPL}, \vec{\omega}' = \vec{n})}{|x_i - x_{TPL}|^2} \quad (6.5a)$$

$$L_{TPL}(x_i, \vec{\omega}) = \frac{\rho}{\pi} \sum_1^K L_{K_TPL_Single} \quad (6.5b)$$

Figure 6.4: Injecting light intensities into a LPV with TPLs. Diffuse Poisson samples are represented in red circles while TPLs are represented in green circles. Each diffuse Poisson sample computes its reflected intensity by 'gathering' the illumination from TPLs.

6.2 Implementation

6.2.1 Poisson samples generation

We implemented Bowers et al. [91] Poisson disk sampling as they are properly distributed across a regular grid structure. Although a GPU implementation of their algorithm has been created [92], we found that it is unnecessary to re-generate samples regularly. A new set of Poisson disk samples would only be required when a new diffuse surface or translucent

object is added to the scene, or if any object in the scene is deformed.

6.2.2 Gathering operations on Poisson samples

As we need to generate multiple visibility rays from our Poisson disk samples to VPLs/TPLs, we utilize NVIDIA's CUDA [93] and OptiX Prime [81] for our ray creation and ray tracing process. Firstly, a ray is created from every (diffuse and translucent) Poisson disk sample to a VPL on the area light source. The visibility tests are performed using OptiX Prime.

For every visible ray from a VPL to the TPDS, the transmitted flux is computed and compressed into its SH basis based on its refraction direction. $L_t(x_i, \vec{\omega})$ is projected into the SH coefficients of a clamped cosine lobe oriented at refracted direction $\vec{\omega}'$ in Equation 6.1b (refer to appendix A.4 for clarification). $c_{lm}^{(\vec{\omega}')}$ refers to the SH coefficients of the clamped cosine lobe based on the refracted ray direction $\vec{\omega}'$. Using CUDA's SHFL [94] functions, the illumination can be gathered quickly if the samples are in multiples of warp sizes (16,32). The gathered transmitted flux for each TPDS at position, x_i , is computed by accumulating the contribution of every visible ray in Equation 6.6. We only need to inject the values of 4 SH coefficients for each color channel, as described by $L_{t,lm}^v(x_i, \vec{\omega})$ in Equation 6.7, into the relevant voxel location of the ESLPV.

$$L(x_i, \vec{\omega}) = \sum_1^K L_{t,lm}^k(x_i, \vec{\omega}), \quad (6.6)$$

$$L_{t,lm}^v(x_i, \vec{\omega}) = \text{float4}\left(\frac{c_{0,0} * L_t(x_i, \vec{\omega})}{\pi}, \frac{c_{1,-1}v * L_t(x_i, \vec{\omega})}{\pi}, \frac{c_{1,0} * L_t(x_i, \vec{\omega})}{\pi}, \frac{c_{1,1} * L_t(x_i, \vec{\omega})}{\pi}\right), \quad (6.7)$$

Next, the TPDS is downsampled to TPLs, which are usually less than 256. All DPDS would then gather illumination (Equation 6.5b) from TPLs after the TPLs have been computed with the results from the ESLPV using Equation 6.4. Each TPL now represents an area light source with radiance and normals.

6.2.3 Indirect illumination voxels

Reflected flux from all diffuse Poisson disk samples, after ‘gathering’ from VPLs and TPLs, are injected into the LPV. The SH coefficients of the reflected flux are computed by multiplying the reflected flux of the Poisson disk sample and the clamped cosine SH lobe of the normal vector of the Poisson disk sample. The SH coefficients of the reflected flux from the Poisson disk samples are injected into the LPV at an offset of one unit voxel away in the normal direction. We keep a counter using a 3D texture and GLSL’s `imageAtomicAdd` function to keep track of the number of samples inside each voxel. Ultimately, the intensities stored in the voxels should be normalized by the number of samples in them. This normalization is not required in the original ESLPV and LPV as the normalization has already been done by dividing the total light energy by the number of texels in the RSM. However, in our case we are unable to use a RSM to represent an area light.

6.3 Results and discussion

We show the rendering results of four scenes: a translucent water glass, a translucent Buddha model and a plane, a translucent bunny in a colored Cornell box, a translucent dragon in the Sponza scene in Figures 6.1a, 6.5a, 6.6a, 6.7, respectively, in 1024x768 pixel resolution. We tabulate the computation timings in Table 6.1. The rendering was performed on an Intel i5 3.40GHz CPU with an NVIDIA GeForce GTX 980 GPU. The direct illumination for diffuse surfaces in our results are rendered with our screenspace multi-resolution technique [3] in Chapter 5, with 64 samples per fragment. The direct illumination provides the soft shadows effects. The translucent materials in our reference images (generated by Mitsuba [2]) are rendered with Photon Beam Diffusion [77]. Due to the sparse distribution of the Poisson disk samples, light ‘gathering’ operations are considered cheap operations that can be computed in tens of milliseconds. Furthermore, these ‘gathering’ operations only need to be performed when there is a change in object positions, material properties or light positions.

In Figure 6.1a, we placed an area light inside a water glass so that it would be easier to only see the indirect illumination produced by the translucent materials as no direct illumination would be able to penetrate the water glass. Our result produced a reddish color bleed as observed on the floor outside the water glass. The color is also observed in the reference image in Figure 6.1b. In Figure 6.5, we illuminate a Buddha model floating on top of a planar surface. Figure 6.5a shows a soft shadow produced by the Buddha model. The shadow is not completely dark and has a yellowish faint partially illuminated by light from the translucent material. These effects are more evident in the soft shadow region of the shadows. A reference image is provided in Figure 6.5b. Figure 6.5c shows our result with specular highlights added onto the surface for a more convincing look. In Figure 6.6a, we can see that there is a faint coloration of reddish and greenish in the shadows of the bunny. This coloration is created by the indirect illumination from both the colored walls of the Cornell box and the remaining scattered light from the bunny. We show that the rendered reference image with the Mitsuba renderer’s path tracer [2] in Figure 6.6b and the coloration in their shadow are similar to ours. In Figure 6.6c, we show our results rendered with specular highlights. In Figure 6.7a, we specifically render the indirect illumination only. It can be seen that scattered green light is present around the diffuse surfaces near the dragon. The scattered green light is also present in the soft shadows areas formed by the pillars or the dragon in Figures 6.7b, 6.7c.

With reference to Table 6.1, ‘Direct Illum.’ refers to the time used for rendering direct illumination (excluding translucent material rendering) using a multi-resolution screenspace approach [2]. ‘Poisson Gathering’ refers to the time used in ‘gathering’ illumination from Poisson disk samples generated on diffuse and translucent surfaces after being lit by VPLs. ‘TPL Gathering’ refers to the time needed for all diffuse Poisson disk sample to gather light from TPLs. ‘ESLPV’ represents the time needed to render the illumination on the translucent object using our ESLPV method. ‘LPV’ represents the time required for the LPV to render indirect illumination on the entire scene. We can observe that direct illumination of translucent materials using area lights can be quickly computed and rendered in less than

10 ms. Indirect illumination for the entire scene can be rendered in less than 20 ms for most of our results. 'Total time' refers to the time required for rendering a single frame with both direct and indirect illumination. The columns 'DPDS', 'TPDS' and 'TPL' describe the number of samples required for diffuse Poisson disk samples, translucent Poisson disk samples and Translucent Planar Lights, respectively. One advantage of voxel-based techniques is that the timings for ESLPV and LPV do not scale up with the scene complexity, which allows our indirect illumination for translucent materials to be rendered at fast speed.

We do note that there are some differences compared to the reference images in other areas of the images. In Figure 6.6, we note that our result (Figure 6.6a) has a different ceiling color compared to the reference image. Our method is not a physically-based algorithm and it uses a very abstract representation of geometry and reflectance property to distribute indirect illumination. Hence, we cannot expect the same quality of results as that in radiosity, photon mapping or path-tracing methods when rendering multi-bounce illumination. However, we can still expect color-bleeding effects to be rendered. We are unable to simulate high frequency effects for translucent materials with weak scattering properties due to the highly compressed nature of our spherical harmonics representation. However, this is compensated through its low-memory consumption and fast rendering speed. Further results are provided in the video link below ¹.

6.3.1 Limitations

Using Bowers et al.'s [91] Poisson disk samples to allocate samples based on a minimum 3D Euclidean separation distance criteria is fast and efficient but it will create problems for thin geometry. For example, if we are allocating Poisson disk samples on a thin planar geometry, it is highly possible for one side of the plane geometry to receive more samples than the other side of the plane geometry. Optimally, what we should do is to allocate samples with a minimum geodesic distance such as Fu et al.'s [95].

¹<https://www.dropbox.com/s/lysescn3c7evc1j/Chapter6Video.mp4?dl=0>

Figure 6.5: Rendering of a Buddha object with a material property simulating an apple in images (a) and (b). We show our rendering in image (a) with both the direct and indirect illumination components, and the reference generated by Mitsuba Path Tracer (64 samples per pixel) in image (b) with the Photon Beam Diffusion method [77]. Image (b) is rendered in 28.97 seconds. Image (c) has been rendered with specular highlights.

6.4 Summary

We have presented an efficient and low cost solution for illumination of translucent materials from area lights. This is first done by generating a set of Poisson disk samples over the translucent materials and performing an illumination ‘gathering’ operation over them. The Poisson disk samples are then injected into the ESLPV [5], which distributes the light intensities through the translucent material. Similarly, Poisson disk samples are distributed around diffuse surfaces in the LPV for gathering direct illumination from the area light.

Figure 6.6: Rendering of a Stanford bunny with material properties simulating jade in a colored Cornell box. We show our rendering in image (a) with both the direct and indirect illumination components, and the ground truth reference generated by Mitsuba Path Tracer (64 samples per pixel) and the Photon Beam Diffusion method [77] in image (b). Image (b) is rendered in 54.21 seconds. Image (c) is rendered with specular highlights.

Table 6.1: Rendering statistics for 1024x768 images. Direct Illum refers to the time required to render direct illumination (Chapter 5) only for diffuse surfaces. The remaining columns (Poisson Gathering, TPL Gathering, ESLPV, LPV) are timings for our work in this chapter.

Figure	Triangles	Direct Illum. [2](ms)	Poisson Gathering (ms)	TPL Gathering (ms)	ES-LPV (ms)	LPV (ms)	Total time (ms)	DPDS	TPDS	TPL
6.1a - Water glass	9,280	35	3	9	5	9	61	3,711	17,077	179
6.5a - Buddha	110,346	77	3	5	2	10	97	3,680	4,913	236
6.6a - Bunny	80,020	90	4	9	5	9	117	2,603	8,438	177
6.7b - Dragon	238,076	80	6	24	5	9	124	21,366	8,495	84

In order to further illuminate the scene using the translucent object, each of the Poisson disk samples in the LPV will gather illumination from a downsampled set of translucent planar lights (TPLs) on the translucent object. These samples are then injected into the LPV. The LPV distributes the light, simulating indirect illumination with contributions from the diffuse and translucent objects. Our method efficiently distributes indirect illumination with very little computation overheads.

Using a Poisson disk sampling approach, we ensure that translucent materials and their indirect illumination can be rendered with good quality under area lighting. This method achieves interactive rendering while remaining precomputationless and maintaining low storage costs.

The diffuse Poisson disk samples (DPDS) in the scene increase with scene complexity, hence leading to a overall increase in time for ‘gathering’ illumination from TPLs. A multi-resolution approach should be designed to select an appropriate set of samples. Other optimizations such as Debevec’s light probe sampling [103] could be more efficient in choosing our TPLs from our TPDS. However, it was designed for sampling a 2D envir-

Figure 6.7: Results of using the TPLs for injection produces a soft diffuse look from the green translucent Stanford dragon model. The scene is illuminated by a rectangular area light source from the top. Image (a) shows the indirect illumination. The walls and pillars of the lower level which are closer to the dragon appear much greener compared to the top level walls and pillars. The light source used in the scene is a small white rectangular area light. Images (b) and (c) show other rendered views with the direct and indirect illumination components. The soft shadow regions can be seen to exhibit indirect scattered illumination from the translucent material.

onment map but it can well be extended to simulate importance sampling for clustering of TPLs. In addition, Debevec's [103] method can be used to sample environment lighting in our work. This can easily replace the uniform stratified sampling that we have used for our area light sources.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this thesis, I have presented some challenges in interactive rendering of translucent materials and our new approaches in solving these challenges. I first presented a survey of the relevant literature related to real-time subsurface scattering. Next, I presented several literature related to area light illumination and highlighted various optimization methods that can be used. I described several challenges in rendering translucent materials, rendering under area light illumination, as well as rendering translucent materials under area light illumination. In order to solve these challenges, I designed algorithms and methods for a complete framework that is able to render translucent materials with global illumination interactively. I describe a summary of my research novelties which have solved the original research goals:

1. The first problem raised was rendering translucent materials without precomputations and with a memory efficient framework. Additionally, material editing should be supported. I decided on the use of a voxel representation for representing light distribution within the translucent materials. This is because voxel based methods are mainly precomputation-less and memory efficient. My main idea of improving computation time and rendering quality for voxel based methods was to separate the

multiple scattering component and single scattering component. Though, this has been presented in literature such as Billeter et al. [55] and Bernabei et al.[50], their model fail to take care of the anisotropic nature of some translucent materials.

Research Novelty: My designed voxel structure, ESLPV, is able to represent weakly and highly scattering materials. A phase function is incorporated into the single scattering framework, which greatly enhances the realism of the rendering for voxel-based techniques. My hierarchical framework, combined with a distance transform optimization texture, enables our rendering to be done at a faster speed and also to represent highly scattering materials. Further rendering results were produced and compared against previous works. The ESLPV continues to benefit from the existing low-storage costs and interactive material-editing features of previous voxel implementations. This work has been published in a journal [5] (My Research Publications section).

2. Our challenge in the second work was dealing with interactive re-lighting and rendering the effects (e.g. soft shadows) produced by area light sources. I presented a multi-resolution rendering approach, which performs rendering on a fragment level rather than pixel level. Computation time can be saved, as fragments can represent large number of pixels. However, fragment generation based on existing refinement metric are excessive due to comparisons made against information from neighbouring fragments. This refinement metric has misalignments which may cause issues such as under/over-refinement. The existing fragment interpolation and upsampling algorithm also produces high errors due to their reliance on over-smoothing.

Research Novelty: My multi-resolution framework uses an efficient subdivision scheme based on sub-fragment values and gradients. The fragments generated are significantly lower than previous methods and regions with soft shadows can be easily rendered. A radial basis function (RBF) interpolation method together with an adaptive variance scaling factor is able to correct possible erroneous regions caused by

under-refined fragments. The RBF interpolation scheme generates significantly lower error rates, compared to previous work. The error rates have been validated against reference images. I also described the use of a new extension multi-stream compaction feature in the transform feedback shader which efficiently generates and streams fragments. My newly designed multi-stream feature speeds up multi-resolution rendering and allows for significant ease of generating fragments. My work achieved the relevant goal of interactive area light re-lighting, without precomputations. This work has also been published in the following conferences and journals [4, 3, 2](My Research Publications section).

3. My challenge in the third work lies in rendering translucent material under area lights. Previous LPV approaches relied on shadow map to generate significant samples for injection. Approximating area lights using multiple shadow maps may poorly distribute samples due to the perspective nature of shadow maps. Computing indirect illumination can often be expensive especially when multiple light bounces are involved. Translucent materials scatter light to their surroundings creating additional illumination. I proposed extending the Light Propagation Volumes [8] (LPV) and ESLPV framework to allow for direct and indirect illumination from area lights.

Research Novelty: A novel improvement to the LPV framework, which allows the LPV structure to render illumination from area lights with a low number of Poisson disk samples was created. This allows us to render translucent materials under area lights, and also indirect illumination. I have validated the rendering results against offline path tracing algorithms, and similar indirect illumination could be achieved. This novel LPV improvement is published in [4], [1](My Research Publications section). This research has produced a complete interactive framework for rendering direct and indirect illumination from diffuse and translucent surfaces, as well as achieving the intended goals of the Ph.D research.

I have produced new algorithms for rendering translucent materials under point, spot and area lights. I was able to achieve interactive rendering rates while maintaining features of convincing translucent material rendering, low storage costs, no precomputations and supporting material editing and re-lighting interactively.

I hope that my work could benefit the computer graphics communities, particularly real-time rendering. For instance, low memory and low computation overhead algorithms are often sought after in mobile development. Multi-resolution rendering have gained recent publicity [96, 97], as they are able to accelerate rendering while maintaining visual quality. These are important requirements for running graphics applications on mobile devices with limited computation resource.

7.2 Future work

7.2.1 Global illumination for translucent materials

Global illumination is an important concept in rendering. My illumination model tackles direct illumination using a multi-resolution approach and indirect illumination using the light propagation approach. While light propagation volumes (LPV) may be memory efficient and fast, they are not good approximations for high frequency lighting. Computing high frequency lighting requires a gathering operation on a pixel level, in which our voxel structures are unable to represent.

Translucent objects often contribute a soft diffuse lighting to its nearby objects and it also receives indirect illumination from nearby surfaces. In the work of Sheng et al. [32], their radiosity approach provides an unified solution for computing surface to surface contribution and it is able to simulate multiple bounces in a rather fast manner at the expense of large precomputation time. In recent years, Crassin et al.'s work on multi-resolution voxels [98] has been extended to several game engines due to its fast computation of illumination. They present a multi-resolution voxel approach that is able to store and retrieve

geometric information in a fast and efficient manner. Their voxel cone tracing [99] method combined with multiple pre-filtered voxel mipmaps approximate occlusion information as well as lighting information. Their cone tracing solution efficiently performs a light gathering operation over the target pixel. However, their approach sacrifices a large amount of CPU and GPU memory which are used to pre-allocate memory for voxel information. This is larger than the GPU acceleration structure used for ray tracing. Sugihara et al. [100] proposed using a layered shadow map to store illumination information in the reflective shadow map instead while voxel structures continue to hold geometric information. This approach significantly reduced the memory consumption. We intend to look towards these methods that are able to approximate this gathering operation using a limited amount of memory.

7.2.2 All frequency lighting for voxel based lighting

Precomputation-less all frequency lighting remains a huge challenge in rendering translucent materials. Current precomputed radiance techniques such as Wang et al. [37], [39] and Yan et al. [101] use spherical Gaussians and wavelet basis to approximate high frequency illumination. Nevertheless, these techniques require precomputations for compressing high frequency lighting and cannot be computed in real-time. They are also storage intensive methods. Some other additional effects such as beam lights which produce high frequency details cannot be displayed on our object because the smallest scattering effect that can be rendered is proportional to the width of a SSLPV voxel. Further work need to be investigated if we could devise a more efficient basis function instead of spherical harmonics to store radiances in the voxels more accurately. It would be also necessary to explore if intensity propagation can be done on voxels containing high frequency information, as well as how these high frequency representations interpolate across voxels. Recent work by Heitz et al. [102] suggests that clamped cosine distributions can provide a good approximation to physically based BRDFs and that it can be analytically integrated over spherical surfaces.

This representation may be more suited to intensity distribution in the voxel structure, considering that intensity distribution in the LPV is based on point to voxel surface integration.

7.2.3 Multi-resolution light sampling

My proposed method over-samples the TPLs. If we intend to bring the work into real-time rates, a more efficient sampling method would need to be employed for choosing a useful set of TPLs. Simpler 2D approaches for clustering light sources can be useful for rendering from textured area lights, such as Debevac's [103] and Nichols et al.'s [85]. Techniques such as Light Cuts by Walter et al. [104] and the latest real-time solution for light cuts by Laurent et al. [105] provides a good tree system for clustering light sources. This tree system certainly helps to select the appropriate set of cut for different regions in the screenspace. Their light-cut algorithm will allow our work to be more scalable to larger scenes with more translucent objects.

My Research Publications

- [1] Ming Di Koa, Henry Johan, and Alexei Sourin. Voxel-based interactive rendering of translucent materials under area lights using sparse samples. In *Proceedings of Cyberworlds 2017*, International Conference on CyberWorlds '2017. IEEE CPS, 2017. [Contributed to Chapter 6], In Print, Best Full Paper.
- [2] Ming Di Koa, Henry Johan, and Alexei Sourin. Interactive screenspace fragment rendering for direct illumination from area lights using gradient aware subdivision and radial basis function interpolation. *Computers And Graphics*, 64:37 – 50, 2017. [Contributed to Chapter 5].
- [3] Ming Di Koa, Henry Johan, and Alexei Sourin. Interactive screenspace stream-compaction fragment rendering of direct illumination from area lights. In *Proceedings of Cyberworlds 2016*, International Conference on CyberWorlds '2016, pages 57–64. IEEE CPS, 2016. [Contributed to Chapter 5].
- [4] Ming Di Koa and Henry Johan. Efficient screenspace rendering for area lights. In *Proceedings of the 33rd Computer Graphics International*, CGI '16, pages 29–32, New York, NY, USA, 2016. ACM. [Contributed to Chapters 5, 6].
- [5] Ming Di Koa and Henry Johan. ESLPV: Enhanced subsurface light propagation volumes. *The Visual Computer*, 30(6-8):821–831, June 2014. Springer. [Contributed to Chapter 4].

Bibliography

- [1] Wojciech Jarosz. *Efficient Monte Carlo Methods for Light Transport in Scattering Media*. PhD thesis, UC San Diego, La Jolla, CA, USA, September 2008.
- [2] Wenzel Jakob. Mitsuba renderer, 2010. <http://www.mitsuba-renderer.org>.
- [3] The Maisonette. Interior wallpaper design, December 2012.
- [4] Royce Blair. Jade horse, March 2008.
- [5] Physics Central. Scattering sky, March 2008.
- [6] colorFabb. Glowfill, March 2016.
- [7] Jesper Børlum, Brian Bunch Christensen, Thomas Kim Kjeldsen, Peter Trier Mikkelsen, Karsten Ostergaard Noe, Jens Rimestad, and Jesper Mosegaard. Sslpv: Subsurface light propagation volumes. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, HPG '11, pages 7–14. ACM, 2011.
- [8] Anton Kaplanyan and Carsten Dachsbacher. Cascaded light propagation volumes for real-time indirect illumination. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '10, pages 99–107. ACM, 2010.
- [9] Greg Nichols, Rajeev Penmatsa, and Chris Wyman. Interactive, multiresolution image-space rendering for dynamic area lighting. In *Proceedings of the 21st Euro-*

- graphics Conference on Rendering*, EGSR'10, pages 1279–1288. Eurographics Association, 2010.
- [10] Henrik Wann Jensen and Per H. Christensen. Efficient simulation of light transport in scenes with participating media using photon maps. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 311–320, New York, NY, USA, 1998. ACM.
- [11] Wojciech Jarosz, Matthias Zwicker, and Henrik Wann Jensen. The beam radiance estimate for volumetric photon mapping. In *ACM SIGGRAPH 2008 Classes*, SIGGRAPH '08, pages 3:1–3:112, New York, NY, USA, 2008. ACM.
- [12] Wojciech Jarosz, Derek Nowrouzezahrai, Robert Thomas, Peter-Pike Sloan, and Matthias Zwicker. Progressive photon beams. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, SIGGRAPH ASIA '11, page 181:1–181:12, New York, NY, USA, 2011. ACM.
- [13] Jan Novák, Derek Nowrouzezahrai, Carsten Dachsbacher, and Wojciech Jarosz. Virtual ray lights for rendering scenes with participating media. *ACM Trans. Graph.*, 31(4):60:1–60:11, July 2012.
- [14] Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 527–536, New York, NY, USA, July 2002. ACM.
- [15] Greg Nichols, Jeremy Shopf, and Chris Wyman. Hierarchical image-space radiosity for interactive global illumination. In *Proceedings of the Twentieth Eurographics Conference on Rendering*, EGSR'09, pages 1141–1149. Eurographics Association, 2009.

- [16] Greg Nichols and Chris Wyman. Multiresolution splatting for indirect illumination. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games, I3D '09*, pages 83–90. ACM, 2009.
- [17] Greg Nichols and Chris Wyman. Interactive indirect illumination using adaptive multiresolution splatting. *IEEE Transactions on Visualization and Computer Graphics*, 16(5):729–741, Sep 2010.
- [18] James T. Kajiya. The rendering equation. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143–150. ACM, 1986.
- [19] 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. ACM, 2001.
- [20] Jos Stam. Multiple scattering as a diffusion process. In PatrickM. Hanrahan and Werner Purgathofer, editors, *Rendering Techniques '95*, Eurographics, pages 41–50. Springer Vienna, 1995.
- [21] James F. Blinn. Light reflection functions for simulation of clouds and dusty surfaces. *SIGGRAPH Comput. Graph.*, 16(3):21–29, July 1982.
- [22] S. Chandrasekhar. On the diffuse reflection of a pencil of radiation by a plane-parallel atmosphere. *Proceedings of the National Academy of Sciences of the United States of America*, 44(9):933–940, 1958.
- [23] Craig Steven Donner. *Towards Realistic Image Synthesis of Scattering Materials*. PhD thesis, University of California at San Diego, La Jolla, CA, USA, 2006. AAI3226771.

- [24] Henrik Wann Jensen and Juan Buhler. A rapid hierarchical rendering technique for translucent materials. *ACM Trans. Graph.*, 21(3):576–581, July 2002.
- [25] Xuejun Hao and Amitabh Varshney. Real-time rendering of translucent meshes. *ACM Trans. Graph.*, 23(2):120–142, April 2004.
- [26] Michael F. Cohen, John Wallace, and Pat Hanrahan. *Radiosity and Realistic Image Synthesis*. Academic Press Professional, Inc., San Diego, CA, USA, 1993.
- [27] Holly E. Rushmeier and Kenneth E. Torrance. The zonal method for calculating light intensities in the presence of a participating medium. *SIGGRAPH Comput. Graph.*, 21(4):293–302, August 1987.
- [28] Holly E. Rushmeier and Kenneth E. Torrance. Extending the radiosity method to include specularly reflecting and translucent materials. *ACM Trans. Graph.*, 9(1):1–27, January 1990.
- [29] Hendrik P. A. Lensch, Michael Goesele, Philippe Bekaert, Jan Kautz, Marcus A. Magnor, Jochen Lang, and Hans-Peter Seidel. Interactive rendering of translucent objects. In *Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*, PG '02, pages 214–. IEEE Computer Society, 2002.
- [30] Nathan A. Carr, Jesse D. Hall, and John C. Hart. Gpu algorithms for radiosity and subsurface scattering. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, HWWS '03, pages 51–59. Eurographics Association, 2003.
- [31] Tom Mertens, Jan Kautz, Philippe Bekaert, Frank Van Reeth, and Hans-Peter Seidel. Efficient rendering of local subsurface scattering. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, PG '03, pages 51–58. IEEE Computer Society, 2003.

- [32] Yu Sheng, Yulong Shi, Lili Wang, and Srinivasa G. Narasimhan. A practical analytic model for the radiosity of translucent scenes. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '13, pages 63–70. ACM, 2013.
- [33] Jiaping Wang, Shuang Zhao, Xin Tong, Stephen Lin, Zhouchen Lin, Yue Dong, Baining Guo, and Heung-Yeung Shum. Modeling and rendering of heterogeneous translucent materials using the diffusion equation. *ACM Trans. Graph.*, 27(1):9:1–9:18, March 2008.
- [34] Yajun Wang, Jiaping Wang, Nicolas Holzschuch, Kartic Subr, Jun-Hai Yong, and Baining Guo. Real-time Rendering of Heterogeneous Translucent Objects with Arbitrary Shapes. *Computer Graphics Forum*, 29(2):497–506, May 2010. EURO-GRAPHICS 2010 (full paper) - Session Rendering I.
- [35] Adam Arbree, Bruce Walter, and Kavita Bala. Heterogeneous subsurface scattering using the finite element method. *IEEE Transactions on Visualization and Computer Graphics*, 17(7):956–969, July 2011.
- [36] Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Trans. Graph.*, 21(3):527–536, July 2002.
- [37] Rui Wang, John Tran, and David Luebke. All-frequency interactive relighting of translucent objects with single and multiple scattering. *ACM Trans. Graph.*, 24(3):1202–1207, July 2005.
- [38] Zhang Jiawan, Gao Yang, Sun Jizhou, and Jin Zhou. Rendering of translucent objects based upon prt techniques. In *Proceedings of the 2007 International Conference on Computational Science and Its Applications - Volume Part III*, ICCSA'07, pages 1056–1065, Berlin, Heidelberg, 2007. Springer-Verlag.

- [39] Rui Wang, Ewen Cheslack-Postava, Rui Wang, David Luebke, Qianrong Chen, Wei Hua, Qunsheng Peng, and Hujun Bao. Real-time editing and relighting of homogeneous translucent materials. *The Visual Computer*, 24(7-9):565–575, 2008.
- [40] Kun Xu, Yue Gao, Yong Li, Tao Ju, and Shi-Min Hu. Real-time homogenous translucent material editing. *Computer Graphics Forum*, 26(3):545–552, 2007.
- [41] Eugene d'Eon, David Luebke, and Eric Enderton. Efficient rendering of human skin. In *Proceedings of the 18th Eurographics conference on Rendering Techniques*, EGSR'07, pages 147–157. Eurographics Association, 2007.
- [42] George Borshukov and J. P. Lewis. Realistic human face rendering for "the matrix reloaded". In *ACM SIGGRAPH 2003 Sketches & Applications*, SIGGRAPH '03, pages 1–1, New York, NY, USA, 2003. ACM.
- [43] John Isidoro, Chris Oat, and Jason Mitchell. Next generation skin rendering. In *Game Tech Conference*, 2004.
- [44] J. Jimenez and D. Gutierrez. Faster rendering of human skin. In *Proceedings of the CEIG*, pages 21–28, 2008.
- [45] Jorge Jimenez, Veronica Sundstedt, and Diego Gutierrez. Screen-space perceptual rendering of human skin. *ACM Trans. Appl. Percept.*, 6(4):23:1–23:15, October 2009.
- [46] J. Jimenez, D. Whelan, V. Sundstedt, and D. Gutierrez. Real-time realistic skin translucency. *Computer Graphics and Applications, IEEE*, 30(4):32–41, 2010.
- [47] Jose I. Echevarria, Adolfo Munoz, Francisco J. Seron, and Diego Gutierrez. Screen-space rendering of translucent objects. In *XX Congreso Español de Informatica Grafica (CEIG 2010)*, pages 127–133. Eurographics, 2010.

- [48] Adolfo Munoz, Jose I. Echevarria, Francisco J. Seron, and Diego Gutierrez. Convolution-based simulation of homogeneous subsurface scattering. *Computer Graphics Forum*, 30(8):2279–2287, 2011.
- [49] Robert Geist, Karl Rasche, James Westall, and Robert Schalkoff. Lattice-boltzmann lighting. In *Proceedings of the Fifteenth Eurographics conference on Rendering Techniques*, EGSR’04, pages 355–362. Eurographics Association, 2004.
- [50] D. Bernabei, A.H. Hakke-Patil, F. Banterle, M. Di Benedetto, F. Ganovelli, S. Pattanaik, and R. Scopigno. A parallel architecture for interactively rendering scattering and refraction effects. *Computer Graphics and Applications, IEEE*, 32(2):34–43, 2012.
- [51] Ivo Ihrke, Gernot Ziegler, Art Tevs, Christian Theobalt, Marcus Magnor, and Hans-Peter Seidel. Eikonal rendering: efficient light transport in refractive objects. *ACM Trans. Graph.*, 26(3), July 2007.
- [52] Raanan Fattal. Participating media illumination using light propagation maps. *ACM Trans. Graph.*, 28(1):7:1–7:11, February 2009.
- [53] Adrien Gruson, Ajit Hakke Patil, Remi Cozot, Kadi Bouatouch, and Sumanta Pattanaik. Light Propagation Maps on Parallel Graphics Architectures. In Hank Childs, Torsten Kuhlen, and Fabio Marton, editors, *Eurographics Symposium on Parallel Graphics and Visualization*. The Eurographics Association, 2012.
- [54] Carsten Dachsbacher and Marc Stamminger. Reflective shadow maps. In *Proceedings of the 2005 Symposium on Interactive 3D graphics and games*, I3D ’05, pages 203–231. ACM, 2005.
- [55] Markus Billeter, Erik Sintorn, and Ulf Assarsson. Real-time multiple scattering using light propagation volumes. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D ’12, pages 119–126. ACM, 2012.

- [56] Lance Williams. Casting curved shadows on curved surfaces. *SIGGRAPH Comput. Graph.*, 12(3):270–274, August 1978.
- [57] Randima Fernando. Percentage-closer soft shadows. In *ACM SIGGRAPH 2005 Sketches*, SIGGRAPH '05. ACM, 2005.
- [58] Michael Schwärzler, Christian Luksch, Daniel Scherzer, and Michael Wimmer. Fast percentage closer soft shadows using temporal coherence. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '13, pages 79–86. ACM, 2013.
- [59] Thomas Annen, Tom Mertens, Hans-Peter Seidel, Eddy Flerackers, and Jan Kautz. Exponential shadow maps. In *Proceedings of Graphics Interface 2008*, GI '08, pages 155–161, Toronto, Ont., Canada, Canada, 2008. Canadian Information Processing Society.
- [60] William Donnelly and Andrew Lauritzen. Variance shadow maps. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, I3D '06, pages 161–165. ACM, 2006.
- [61] Baoguang Yang, Zhao Dong, Jieqing Feng, Hans-Peter Seidel, and Jan Kautz. Variance soft shadow mapping. *Computer Graphics Forum*, 29(7):2127–2134, 2010.
- [62] Thomas Annen, Tom Mertens, Philippe Bekaert, Hans-Peter Seidel, and Jan Kautz. Convolution shadow maps. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques*, EGSR'07, pages 51–60. Eurographics Association, 2007.
- [63] Yong He, Yan Gu, and Kayvon Fatahalian. Extending the graphics pipeline with adaptive, multi-rate shading. *ACM Trans. Graph.*, 33(4):142:1–142:12, July 2014.
- [64] Fabrice Rousselle, Claude Knaus, and Matthias Zwicker. Adaptive rendering with non-local means filtering. *ACM Trans. Graph.*, 31(6):195:1–195:11, November 2012.

- [65] Tzu-Mao Li, Yu-Ting Wu, and Yung-Yu Chuang. Sure-based optimization for adaptive sampling and reconstruction. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH Asia 2012)*, 31(6):186:1–186:9, Nov 2012.
- [66] Soham Uday Mehta, Brandon Wang, and Ravi Ramamoorthi. Axis-aligned filtering for interactive sampled soft shadows. *ACM Trans. Graph.*, 31(6):163:1–163:10, November 2012.
- [67] Soham Uday Mehta, Brandon Wang, Ravi Ramamoorthi, and Fredo Durand. Axis-aligned filtering for interactive physically-based diffuse indirect lighting. *ACM Trans. Graph.*, 32(4):96:1–96:12, Jul 2013.
- [68] Ling-Qi Yan, Soham Uday Mehta, Ravi Ramamoorthi, and Fredo Durand. Fast 4d sheared filtering for interactive rendering of distribution effects. *ACM Trans. Graph.*, 35(1):7:1–7:13, December 2015.
- [69] Vaclav Skala. A practical use of radial basis functions interpolation and approximation. In *Proceedings of Revista Investigacion Operacional*, volume 37 of *IWOR '15*, pages 137–145, 2016.
- [70] Vaclav Skala. Fast interpolation and approximation of scattered multidimensional and dynamic data using radial basis functions. *WSEAS Trans.on Mathematics*, 12(5):501–511, 2013.
- [71] Roland W. Fleming and Heinrich H. Bühlhoff. Low-level image cues in the perception of translucent materials. *ACM Trans. Appl. Percept.*, 2(3):346–382, July 2005.
- [72] Ioannis Gkioulekas, Bei Xiao, Shuang Zhao, Edward H. Adelson, Todd Zickler, and Kavita Bala. Understanding the role of phase function in translucent appearance. *ACM Trans. Graph.*, 32(5):147:1–147:19, October 2013.
- [73] Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. A ray tracing solution for diffuse interreflection. In *Proceedings of the 15th Annual Conference on*

Computer Graphics and Interactive Techniques, SIGGRAPH '88, pages 85–92, New York, NY, USA, 1988. ACM.

- [74] Colin Barre-Brisebois and Marc Bouchard. Approximating translucency for a fast, cheap and convincing subsurface scattering look, 2011. Date accessed : 2011-03-07.
- [75] Michael Schwarz and Hans-Peter Seidel. Fast parallel surface and solid voxelization on gpus. *ACM Trans. Graph.*, 29(6):179:1–179:10, December 2010.
- [76] Elmar Eisemann and Xavier Décoret. Single-pass gpu solid voxelization for real-time applications. In *Proceedings of graphics interface 2008*, GI '08, pages 73–80. Canadian Information Processing Society, 2008.
- [77] Ralf Habel, Per H. Christensen, and Wojciech Jarosz. Photon beam diffusion: A hybrid monte carlo method for subsurface scattering. *Computer Graphics Forum (Proceedings of EGSR 2013)*, 32(4), June 2013.
- [78] Matt Pharr and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., 2004.
- [79] Guojun Chen, Pieter Peers, Jiawan Zhang, and Xin Tong. Real-time rendering of deformable heterogeneous translucent objects using multiresolution splatting. *The Visual Computer*, 28:701–711, 2012.
- [80] John David Olovsson and Michael Doggett. Octree light propagation volumes. In *Proceedings of SIGRAD 2013*, pages 27–33, 2013.
- [81] Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. Optix: a general purpose ray tracing engine. *ACM Trans. Graph.*, 29(4):66:1–66:13, July 2010.
- [82] Peter Shirley, Changyaw Wang, and Kurt Zimmerman. Monte carlo techniques for direct lighting calculations. *ACM Trans. Graph.*, 15(1):1–36, January 1996.

- [83] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. In *Seminal Graphics*, pages 77–85. ACM, New York, NY, USA, 1998.
- [84] Ken Anjyo, J. P. Lewis, and Frédéric Pighin. Scattered data interpolation for computer graphics. In *ACM SIGGRAPH 2014 Courses*, SIGGRAPH '14, pages 27:1–27:69, New York, NY, USA, 2014. ACM.
- [85] Greg Nichols and Chris Wyman. Direct illumination from dynamic area lights. In *SIGGRAPH '09: Posters*, SIGGRAPH '09, pages 82:1–82:1, New York, NY, USA, 2009. ACM.
- [86] Matt Pharr and Pat Hanrahan. Monte carlo evaluation of non-linear scattering equations for subsurface reflection. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 75–84, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.
- [87] Henrik Wann Jensen, Justin Legakis, and Julie Dorsey. Rendering of wet materials. In *Proceedings of the 10th Eurographics Conference on Rendering*, EGWR'99, pages 273–282, Aire-la-Ville, Switzerland, Switzerland, 1999. Eurographics Association.
- [88] Craig Donner and Henrik Wann Jensen. Rendering translucent materials using photon diffusion. In *ACM SIGGRAPH 2008 Classes*, SIGGRAPH '08, pages 4:1–4:9, New York, NY, USA, 2008. ACM.
- [89] Wojciech Jarosz, Henrik Wann Jensen, and Craig Donner. Advanced global illumination using photon mapping. In *ACM SIGGRAPH 2008 Classes*, SIGGRAPH '08, pages 2:1–2:112, New York, NY, USA, 2008. ACM.
- [90] Hawar Doghramachi. *GPU Pro 4: Advanced Rendering Techniques - Rasterized Voxel-Based Dynamic Global Illumination*. A. K. Peters/CRC Press, Natick, MA, USA, 2013.

- [91] John Bowers, Rui Wang, Li-Yi Wei, and David Maletz. Parallel poisson disk sampling with spectrum analysis on surfaces. *ACM Trans. Graph.*, 29(6):166:1–166:10, December 2010.
- [92] Xiang Ying, Shi-Qing Xin, Qian Sun, and Ying He. An intrinsic algorithm for parallel poisson disk sampling on arbitrary surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 19(9):1425–1437, September 2013.
- [93] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda. *Queue*, 6(2):40–53, March 2008.
- [94] Justin Luitjens. Faster parallel reductions on kepler. Accessed: 2014-02-13.
- [95] Yan Fu and Bingfeng Zhou. Direct sampling on surfaces for high quality remeshing. In *Proceedings of the 2008 ACM Symposium on Solid and Physical Modeling, SPM '08*, pages 115–124, New York, NY, USA, 2008. ACM.
- [96] Youngwook Kim, Woong Seo, Yongho Kim, Yeongkyu Lim, Jae-Ho Nah, and Insung Ihm. Adaptive undersampling for efficient mobile ray tracing. *Vis. Comput.*, 32(6-8):801–811, June 2016.
- [97] Won-Jong Lee, Seok Joong Hwang, Youngsam Shin, Soojung Ryu, and Insung Ihm. Adaptive multi-rate ray sampling on mobile ray tracing gpu. In *SIGGRAPH ASIA 2016 Mobile Graphics and Interactive Applications, SA '16*, pages 3:1–3:6, New York, NY, USA, 2016. ACM.
- [98] Cyril Crassin, Fabrice Neyret, Sylvain Lefebvre, and Elmar Eisemann. Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games, I3D '09*, pages 15–22, New York, NY, USA, 2009. ACM.

- [99] Cyril Crassin, Fabrice Neyret, Miguel Sainz, Simon Green, and Elmar Eisemann. Interactive indirect illumination using voxel cone tracing. *Computer Graphics Forum (Proceedings of Pacific Graphics 2011)*, 30(7), Sep 2011.
- [100] Masamichi Sugihara, Randall Rauwendaal, and Marco Salvi. Layered Reflective Shadow Maps for Voxel-based Indirect Illumination. In Ingo Wald and Jonathan Ragan-Kelley, editors, *Eurographics/ ACM SIGGRAPH Symposium on High Performance Graphics*. The Eurographics Association, 2014.
- [101] Ling-Qi Yan, Yahan Zhou, Kun Xu, and Rui Wang. Accurate translucent material rendering under spherical gaussian lights. *Comput. Graph. Forum*, 31(7pt2):2267–2276, September 2012.
- [102] Eric Heitz, Jonathan Dupuy, Stephen Hill, and David Neubelt. Real-time polygonal-light shading with linearly transformed cosines. *ACM Trans. Graph.*, 35(4):41:1–41:8, July 2016.
- [103] Paul Debevec. A median cut algorithm for light probe sampling. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06, New York, NY, USA, 2006. ACM.
- [104] Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and Donald P. Greenberg. Lightcuts: A scalable approach to illumination. *ACM Trans. Graph.*, 24(3):1098–1107, July 2005.
- [105] Gilles LAURENT, Cyril DELALANDRE, Gregoire De LA RIVIERE, and Tamy BOUBEKEUR. Forward light cuts: A scalable approach to real-time global illumination. *Computer Graphics Forum (Proc. EGSR 2016)*, 35(4):79–88, 2016.
- [106] Peter-Pike Sloan. Stupid spherical harmonics (sh) tricks introduction, 2008.

Appendix A

Spherical Harmonics

Spherical harmonics is a frequency-space basis for representing functions defined over a sphere. They are the spherical analogue to the Fourier basis on the real line. They are used in several areas of computer graphics especially rendering [36]. They can be used for representing BRDF, phase functions, and lighting distributions. In this Appendix, some fundamental properties of SH that were used in this paper will be described.

A.1 Overview

Given a spherical function $f(\vec{\omega})$ with direction $\vec{\omega} = (x, y, z)$ represented as its standard spherical parameterization $\vec{\omega} = (\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta)$, the real Spherical Harmonic basis functions can be defined as:

$$y_m^l(\theta, \phi) = \begin{cases} K_l^m \cos(m\phi) P_m^l \cos(\theta) & \text{if } m \geq 0 \\ K_l^{|m|} \sin(|m|\phi) P_{|m|}^l \cos(\theta) & \text{if } m < 0 \end{cases}$$

where m indexes the $2l+1$ band- l basis functions and K_l^m is a normalization term and P_m^l is the associated Legendre polynomials. Each of the $2l+1$ band- l basis functions is an order- l polynomial in (x, y, z) .

$$K_l^m = \sqrt{\frac{2l+1}{4\pi} \frac{(l-|m|)!}{(l+|m|)!}} \quad (\text{A.1})$$

Any real-valued spherical function f may be expanded as a linear combination of basis functions

$$f(\vec{\omega}) = \sum_{l=0}^{\infty} \sum_{m=-l}^l f_l^m y_l^m(\vec{\omega}) \quad (\text{A.2})$$

where the coefficients f_l^m are computed by projecting f onto each basis function y_l^m

$$f_l^m = \int_{\Omega} f(\vec{\omega}) y_l^m(\vec{\omega}) d\vec{\omega} \quad (\text{A.3})$$

The spherical harmonics expansion is exact if the number of bands l is infinite. However, in practise, we only take a limited number of bands to get an approximation $f(\vec{\omega})$ of the original function $f(\vec{\omega})$.

The y_l^0 terms are symmetric in the z-axis. They are also known as Zonal Harmonics can be used for fast rotation for spherical symmetric functions.

In some literature, it is seen as more convenient to reformulate the indexing scheme under a single index $i = l(l+1) + m$. An n -order approximation would have n^2 coefficients:

$$f(\vec{\omega}) = \sum_{i=0}^{n^2-1} y_i(\vec{\omega}) f_i \quad (\text{A.4})$$

A.2 Properties of spherical harmonics

A.2.1 Orthonormality

The spherical harmonics bases are orthogonal for different l and different m . The inner product of any two spherical harmonics basis functions is the Kronecker delta function.

$$\int_{\Omega} y_i(\vec{\omega}) y_j(\vec{\omega}) d\vec{\omega} = \delta_{ij} \quad (\text{A.5})$$

A.2.2 Double product integral

The orthonormality property provides a simple evaluation for the integrand of two functions represented in SH basis. The integral product of two SH approximated functions $a(\vec{\omega})$ and $b(\vec{\omega})$ can be expanded as a dot product of the coefficient vectors of the two SH functions.

$$\begin{aligned}
 \int_{\Omega} a(\vec{\omega})b(\vec{\omega})d(\vec{\omega}) &= \int_{\Omega} \left(\sum_i a_i y_i(\vec{\omega})\right) \left(\sum_j b_j y_j(\vec{\omega})\right) d\vec{\omega} \\
 &= \sum_i \sum_j a_i b_j \int_{\Omega} y_i(\vec{\omega}) y_j(\vec{\omega}) d\vec{\omega} \\
 &= \sum_i \sum_j a_i b_j \delta_{ij} \\
 &= \sum_i a_i b_i
 \end{aligned} \tag{A.6}$$

A.2.3 Rotational invariance

The SH basis functions are rotationally invariant, which means that rotating the SH coefficients, $R_{SH} f_l^m$ is similar to a rotated lookup into the unrotated SH function $\tilde{f}(R \cdot \vec{\omega})$.

A.2.4 Rotation

Rotation of a SH function can be performed by using a linear transform (R_{SH}) on the coefficients f_l^m .

$$\tilde{f}(R \cdot \vec{\omega}) = R_{SH} f_l^m \tag{A.7}$$

R_{SH} is a high dimensional rotation matrix, where the i -th SH coefficient of the rotated function $\tilde{f}(R \cdot \vec{\omega})$ is simply:

$$\tilde{f} = \sum_j R_{ij} f_j \tag{A.8}$$

This means that the coefficients for one band of f would only affect the same band of

coefficients in $\tilde{f}$, which leads to a sparse block diagonal matrix. For simplicity, we only show the rotation matrix for two bands.

$$R = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & X & X & X \\ 0 & X & X & X \\ 0 & X & X & X \end{pmatrix}$$

From observation in the next section, we note that only zonal harmonics coefficients will remain after evaluating the integral in (4.27) (This will be further explained in Appendix A.8). This further reduces the rotation matrix to one that is similar to Zonal Harmonics rotation:

$$R = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & X_1 & 0 \\ 0 & 0 & X_2 & 0 \\ 0 & 0 & X_3 & 0 \end{pmatrix}$$

As we are performing a passive rotation of the SH coefficients, we use the R^{-1} matrix which is equivalent to R^T for orthogonal matrices.

$$R^{-1} = R^T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & X_1 & X_2 & X_3 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

The actual values for the rotation matrix can be obtained from a simple rotation matrix for rotating SH basis in Cartesian coordinates. The evaluation is equivalent to a dot product.

$$R^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \sin\alpha\sin\beta & \cos\beta & \cos\alpha\sin\beta \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

A.3 Cartesian values for spherical harmonics basis functions

For lower order Spherical Harmonics terms, the SH Basis functions can be computed analytically as found in Sloan's work [106].

$$y_0^0(x, y, z) = \frac{1}{2\sqrt{\pi}} \quad (\text{A.9})$$

$$y_{-1}^1(x, y, z) = -\frac{\sqrt{3}}{2\sqrt{\pi}}y \quad (\text{A.10})$$

$$y_0^1(x, y, z) = \frac{\sqrt{3}}{2\sqrt{\pi}}z \quad (\text{A.11})$$

$$y_1^1(x, y, z) = -\frac{\sqrt{3}}{2\sqrt{\pi}}x \quad (\text{A.12})$$

A.4 Rotation of clamped cosine lobe

A clamped cosine lobe is a spherical symmetric function, hence a simpler version of rotation using Zonal Harmonics (ZH) could be used for the SH lobe. Sloan [106] showed that a ZH function oriented on the z-axis can be rotated to an arbitrary direction $\vec{\omega}$, yielding a SH function. The SH coefficients c_m^l can be evaluated as:

$$c_m^l = \frac{y_m^l(\vec{\omega})}{n_l^* s_l} \quad (\text{A.13})$$

where s_l is the zonal harmonics coefficients of projecting the function on the l -band zonal harmonics basis defined on the z-axis. n_l^* is computed as $\sqrt{\frac{4\pi}{2l+1}}$. The ZH coefficients s_l of the clamped cosine lobe centered at z-axis can be computed analytically as:

$$s_0 = \int_{\vec{\omega}} \cos(\phi) y_0^0(\vec{\omega}) d(\vec{\omega}) = \frac{1}{2\sqrt{\pi}} \int_{\vec{\omega}} \cos(\phi) d\vec{\omega} = \frac{\sqrt{\pi}}{2} \quad (\text{A.14})$$

$$s_1 = \int_{\vec{\omega}} \cos(\phi) y_0^1(\vec{\omega}) d\vec{\omega} = \sqrt{\frac{\pi}{3}} \quad (\text{A.15})$$

We provide the source code listing A.1 of SHProjectCone which produces the coefficients, c_m^l , of the clamped SH cosine lobe oriented at direction $\mathbf{kDir}$. The source code is obtained from Børllum et al.s' work [7].

Listing A.1: Code for compressing a directional ray into its a SH cosine lobe

```
float4 SHRotate(float3 kDir, float2 kZHCoeffs)
{
    float2 kTheta12CS = make_float2(0.0f, 0.0f);
    float2 kPhi12CS = make_float2(0.0f, 0.0f);

    if (kDir.z * kDir.z < 0.99)
    {
        kTheta12CS = normalize(make_float2(kDir.x, kDir.y));
        kPhi12CS.x = sqrt(1.0 - kDir.z * kDir.z);
    }

    kPhi12CS.y = kDir.z;

    float4 kResult;
    kResult.x = kZHCoeffs.x;
    kResult.y = kZHCoeffs.y * kPhi12CS.x * kTheta12CS.y;
```

```

    kResult.z = kZHCoeffs.y * kPhi12CS.y;
    kResult.w = kZHCoeffs.y * kPhi12CS.x * kTheta12CS.x;

    return kResult;
}

float4 SHProjectCone(float3 kDir)
{
    const float2 kZHCoeffs = make_float2(0.886226f, 1.023326f);
    return SHRotate(kDir, kZHCoeffs);
}

```

A.5 Flux Propagation

The basic idea of flux propagation is to accumulate the contribution of flux from the six neighbouring voxels of each voxel. To compute the contribution of flux from a neighbouring voxel, we need to determine the amount of flux from the center of the neighbouring voxel that reaches each of the voxel faces, except for the bordering face connecting the two voxels, in the destination cell. Hence for one source cell s , its contribution to its neighbouring cell d is

$$\Phi_{f,d \leftarrow s} = \int_{\Delta\omega_{f,s}} I_s(\omega) d\vec{\omega}, \quad (\text{A.16})$$

where $\Delta\omega(f, s)$ is the solid angle subtended by center of the source cell to the face f of the neighbouring cell.

The flux computed in equation A.16 is projected as a new VPL in the destination voxel. The intensity of the newly added flux is once again converted into the form of a clamped cosine lobe centered towards face f as shown in equation A.17 in SH.

$$I_{f,d \leftarrow s} = \frac{\Phi_{f,d \leftarrow s}}{\pi} \max(0, n_f \cdot \vec{\omega}), \quad (\text{A.17})$$

where n_f is the face normal of the destination voxel face.

The flux contributions to every face f of the destination cell, from every neighbour voxel s are accumulated to the destination voxel's VPL.

In the original SSLPV paper, it is mentioned that the propagation process between two voxels can be computed using a matrix-vector multiplication. First, the flux transfer in A.16 can be expressed in its SH form as shown in A.20 since the radiance in the source cell is already in its SH form after the injection stage.

$$\Phi_{f,d \leftarrow s} = \int_{\Delta\omega_{f,s}} I_s(\omega) d\vec{\omega} \quad (\text{A.18})$$

$$= \sum_{l,m} c_s \int_{f,d \leftarrow s} y_{lm}(\vec{\omega}) d\vec{\omega} \quad (\text{A.19})$$

$$= v_{f,d \leftarrow s}^T \cdot c_s \quad (\text{A.20})$$

$$v_{f,d \leftarrow s}^T = \int_{f,d \leftarrow s} y_{lm}(\vec{\omega}), \quad (\text{A.21})$$

c_s is the SH coefficient vector of the radiance stored in the source voxel and $v_{f,d \leftarrow s}^T$ is the transfer vector which maps the source radiance (in SH) to the destination flux for a particular target voxel face. Next, the point light distribution in the destination cell is projected onto its SH coefficients (A.22) with its intensity proportional to the clamp cosine lobe centered around the face normal n_f . We substitute A.17 into A.22 to solve for the SH coefficients in the destination voxel.

$$c_{f,d \leftarrow s} = \begin{bmatrix} \int_{4\pi} I_{f,d \leftarrow s} y_{00}(\vec{\omega}) d\vec{\omega} \\ \int_{4\pi} I_{f,d \leftarrow s} y_{1-1}(\vec{\omega}) d\vec{\omega} \\ \int_{4\pi} I_{f,d \leftarrow s} y_{10}(\vec{\omega}) d\vec{\omega} \\ \int_{4\pi} I_{f,d \leftarrow s} y_{11}(\vec{\omega}) d\vec{\omega} \end{bmatrix} \quad (\text{A.22})$$

$$= \frac{1}{\pi} \begin{bmatrix} \int_{2\pi} n_f \cdot \vec{\omega} y_{00}(\vec{\omega}) d\vec{\omega} \\ \int_{2\pi} n_f \cdot \vec{\omega} y_{1-1}(\vec{\omega}) d\vec{\omega} \\ \int_{2\pi} n_f \cdot \vec{\omega} y_{10}(\vec{\omega}) d\vec{\omega} \\ \int_{2\pi} n_f \cdot \vec{\omega} y_{11}(\vec{\omega}) d\vec{\omega} \end{bmatrix} \cdot v_{f,d \leftarrow s}^T \cdot c_s \quad (\text{A.23})$$

The transfer matrix can thus be computed by:

$$M_{f,d \leftarrow s} = \frac{1}{\pi} \begin{bmatrix} \int_{2\pi} n_f \cdot \vec{\omega} y_{00}(\vec{\omega}) d\vec{\omega} \\ \int_{2\pi} n_f \cdot \vec{\omega} y_{1-1}(\vec{\omega}) d\vec{\omega} \\ \int_{2\pi} n_f \cdot \vec{\omega} y_{10}(\vec{\omega}) d\vec{\omega} \\ \int_{2\pi} n_f \cdot \vec{\omega} y_{11}(\vec{\omega}) d\vec{\omega} \end{bmatrix} \cdot \begin{bmatrix} \int_{\Delta\omega_{f,s}} y_{00}(\vec{\omega}) d\vec{\omega} \\ \int_{\Delta\omega_{f,s}} y_{1-1}(\vec{\omega}) d\vec{\omega} \\ \int_{\Delta\omega_{f,s}} y_{10}(\vec{\omega}) d\vec{\omega} \\ \int_{\Delta\omega_{f,s}} y_{11}(\vec{\omega}) d\vec{\omega} \end{bmatrix} \quad (\text{A.24})$$

which results in a 4x4 matrix. The computation of the cosine lobe in its SH coefficients is the same as the one done in the injection stage. As for the integral of the SH basis functions on the solid angle, the answer is just the product of the solid angle with the basis functions. We compute $M_{f,d \leftarrow s}$ for all five faces and sum all five matrices to get a transfer matrix $M_{f,d \leftarrow s}$. We can represent the final transfer matrix between two voxels as:

$$M_{d \leftarrow s} = \sum_f^5 M_{f,d \leftarrow s} = \sum z_{lm} \sum (y_{lm} \Delta\omega) \quad (\text{A.25})$$

The face to face flux transfer can then be computed as a Matrix-Vector multiplication:

$$M_{d \leftarrow s} c_{s,lm} = \sum_f^5 M_{f,d \leftarrow s} c_{s,lm} = \sum z_{lm} \sum (c_{s,lm} y_{lm} \Delta\omega) \quad (\text{A.26})$$

where z_{lm} is the SH coefficient of the cosine lobe and $c_{s,lm}$ is the SH coefficient of the source cell, s . We refer the reader to Appendix A.7 for a complete breakdown of the calculation for the transfer matrix.

A.6 Solid angle computation

We compute the solid angle for the front casing case (Figure A.1 (left)) by using the solid angle calculation for a four-sided right rectangular pyramid in the equation below:

$$\Delta\omega = 4 \arctan \frac{\alpha\beta}{2d\sqrt{4d^2 + \alpha^2 + \beta^2}}, \quad (\text{A.27})$$

where α and β are the base length of the pyramid and d is the distance between the center of the base rectangle to the apex of the pyramid. As for the side facing solid angle, we split the computation to two solid angle computation of the tetrahedron in equation A.28. The two patches which the solid angle are computed is shown in Figure A.2.

Figure A.1: Two possible solid angles that can be subtended from the center of the source cell. Front facing solid angle (left) and side facing solid angle (right). The solid angle is the volume shaded in blue.

$$\Delta\omega = 2 \arctan \frac{\vec{a} \cdot (\vec{b} \times \vec{c})}{(|\vec{a}||\vec{b}||\vec{c}| + (\vec{a} \cdot \vec{b})|\vec{c}| + (\vec{a} \cdot \vec{c})|\vec{b}| + (\vec{b} \cdot \vec{c})|\vec{a}|)} \quad (\text{A.28})$$

The vectors $\vec{a}, \vec{b}$ and $\vec{c}$ denotes the vector positions of the vertices A,B,C with respect to the center of origin at the center of the source voxel.

By solving equation A.27 and A.28, we obtain the solid angle of the two possible cases:

$$\Delta\omega_{front} = 0.4006696844, \quad (\text{A.29})$$

$$\Delta\omega_{side} = 0.423431354, \quad (\text{A.30})$$

Figure A.2: To compute the solid angle for the side facing case, we divide the solid angle to two tetrahedrons (green and blue) and use the solid angle computation for a tetrahedron.

Figure A.3: Axis directions for calculating flux transfer for each face.

A.7 Flux transfer between Voxels

In this section, we formulate the answer for $M_{d \leftarrow s} = \sum_f^5 M_{f, d \leftarrow s} = \sum z_{lm} \sum (y_{lm} \Delta\omega)$ in A.25, which can be precomputed as an offline input for the propagation process. We can evaluate the value of z_{lm} coefficient by substituting the normalized vector (d-s) into the SH basis function in Appendix A.4. We show the computation of the matrix M for the flux transfer of 2 voxels along the x-axis. The computation is done with reference to the axis directions defined in Figure A.3:

We first compute the flux contribution from the +Z face on the right voxel to the left voxel. We can evaluate the values of z_{lm} , y_{lm} and $\Delta\omega$ for that specific direction, the values

are provided as follow:

$$\begin{aligned}
 z_{lm} &= \begin{pmatrix} 0.88622689 \\ 0.0 \\ 1.0233266 \\ 0.0 \end{pmatrix}, \\
 y_{lm}(+Z) &= \begin{pmatrix} 0.28209481 \\ 0.0 \\ 0.25687355 \\ -0.41563013 \end{pmatrix}, \\
 \Delta\omega &= 0.42343
 \end{aligned} \tag{A.31}$$

The same procedure is repeated for the -Z, +Y, -Y, +X face. We ignore the computation for the face -X that is adjacent for both voxels.

For Face -Z,

$$\begin{aligned}
 z_{lm} &= \begin{pmatrix} 0.88622689 \\ 0.0 \\ -1.0233266 \\ 0.0 \end{pmatrix}, \\
 y_{lm}(-Z) &= \begin{pmatrix} 0.28209481 \\ 0.0 \\ -0.25687355 \\ -0.41563013 \end{pmatrix}, \\
 \Delta\omega &= 0.42343
 \end{aligned} \tag{A.32}$$

For Face +X,

$$\begin{aligned}
z_{lm} &= \begin{pmatrix} 0.88622689 \\ 0.0 \\ 0.0 \\ -1.0233266 \end{pmatrix}, \\
y_{lm}(-Z) &= \begin{pmatrix} 0.28209481 \\ 0.0 \\ 0.0 \\ -0.48860252 \end{pmatrix}, \\
\Delta\omega &= 0.40067
\end{aligned} \tag{A.33}$$

For Face -Y,

$$\begin{aligned}
z_{lm} &= \begin{pmatrix} 0.88622689 \\ 1.0233266 \\ 0.0 \\ 0.0 \end{pmatrix}, \\
y_{lm}(-Y) &= \begin{pmatrix} 0.28209481 \\ 0.25687355 \\ 0.0 \\ -0.41563013 \end{pmatrix}, \\
\Delta\omega &= 0.42343
\end{aligned} \tag{A.34}$$

For Face +Y,

$$\begin{aligned}
 z_{lm} &= \begin{pmatrix} 0.88622689 \\ -1.0233266 \\ 0.0 \\ 0.0 \end{pmatrix}, \\
 y_{lm}(-Y) &= \begin{pmatrix} 0.28209481 \\ -0.25687355 \\ 0.0 \\ -0.41563013 \end{pmatrix}, \\
 \Delta\omega &= 0.42343
 \end{aligned} \tag{A.35}$$

Performing the summation for $M_{d \leftarrow d+x} = \sum_f^5 M_{f,d \leftarrow d+x} = \sum z_{lm} \sum (y_{lm} \Delta\omega)$, we obtain the final transfer matrix.

$$M_{d \leftarrow d+x} = \begin{pmatrix} 0.167 & 0 & 0 & -0.240 \\ 0 & 0.070 & 0 & 0 \\ 0 & 0 & 0.070 & 0 \\ -0.037 & 0 & 0 & 0 \end{pmatrix}$$

A.8 Rendering incident radiance formulation

The section describes the incident radiance formulation described in Equation 2.55. The explanation can also be found in the original SSLPV paper [7]. The use of the point light source for rendering is not feasible since the probability for an eye ray to connect back to the point light source after a refraction event is infinitesimal. Hence, the solution proposed in the paper was to convert the point light source into a uniform area light source that was parallel to the surface direction with cross section area A . The flux ϕ_{point} emitted by the point light source must be equal to the flux emitted by the area light source $\phi_{area} = L_i A \pi$.

Figure A.4: (a) A VPL is defined in the center of the voxel. (b) It is difficult to perform a ray trace from a refracted eye ray into the VPL. (c) The VPL is treated like an area light source on the surface of the material. (d) The area light source is assumed to be parallel to the surface orientation. Image obtained from [7]

Hence $L_i = \phi_{point}/A\pi$ where the flux ϕ_{point} can be computed by $\phi_{point} = \int_{2\pi} I(\vec{\omega}) d\vec{\omega}$. This leads to the formulation in Equation 2.55. Figure A.4 gives a visual explanation of how the radiance is approximated using an area light source.

The simplification in Equation 2.55 comes from the fact that $\int_0^{2\pi} y_m^l(\theta, \phi) d\phi$ for $m \neq 0$ terms is equal to 0. For example:

$$\begin{aligned}
 \int_0^{2\pi} y_{-1}^1(\theta, \phi) d\phi &= \int_0^{2\pi} \sqrt{\frac{3}{4\pi}} \sin\phi \sin\theta d\phi \\
 &= \sqrt{\frac{3}{4\pi}} \sin\theta \underbrace{\int_0^{2\pi} \sin\phi d\phi}_0 \\
 &= 0
 \end{aligned} \tag{A.36}$$

The integral of $\int_0^{2\pi} y_m^l(\theta, \phi) d\phi$ for $m=0$ terms is proportional to 2π . The evaluation for the two bands is shown below:

$l=0$

$$\int_0^{2\pi} y_0^0(\theta, \phi) d\phi = 2\pi \frac{1}{4\pi} \tag{A.37}$$

$l=1$

$$\int_0^{2\pi} y_0^1(\theta, \phi) \sin\theta d\phi = 2\pi \sqrt{\frac{3}{4\pi}} \cos\theta \quad (\text{A.38})$$

Therefore it explains the simplification of the term $1/A\pi$ to $2/A$ in Equation 2.55.

A.9 Analytic comparisons of Henyey Greenstein Phase Function with its 2nd order SH form

We note that the Henyey Greenstein phase function (Equation 2.16) when projected into its Spherical Harmonics basis results in the following form:

$$P_{HG}(\theta, g) = \sum_{l=0}^{\infty} (2l+1) g^l P_l(\cos\theta), \quad (\text{A.39})$$

where g is the dimensionless scattering parameter ranging from -1 (back scattering) to 1 (forward scattering). P_l refers to the Legendre polynomials, for $m = 0$. θ refers to the scattering angle with respect to the incident light direction (refer to Figure A.5).

In our work, we only consider 2nd order SH for compression due to its computation efficiency. We show analytic plots (PDF against angular radians) for various scattering parameter g between the Henyey Greenstein phase function and its corresponding 2nd order ($l_{max} = 1$) Spherical Harmonics reconstructed function in Figures A.6, A.7 and A.8. From the plots in Figures A.7 and A.8 the difference in the probability density function produced by the SH plots greatly increased when $|g| > 0.3$. Much of the errors lie in the region of angular degrees of 150 and above for $g < -0.3$ and 0 to 30 degrees for $g > 0.3$.

Figure A.5: Incident light is scattered in a particular distribution when it collides with a particle in the participating media. θ describes the angle which the scattering light direction makes with the incident light direction.

Figure A.6: Plot of Henyey Greenstein Phase function (Magenta) against its 2nd order SH reconstructed phase function (Blue) for $g = 0$. The function is plotted as pdf (y-axis) against scattering angle (in degrees) θ (x-axis). Both plots have almost identical values.

(a) $g = -0.9$

(b) $g = -0.7$

(c) $g = -0.5$

(d) $g = -0.3$

(e) $g = -0.1$

Figure A.7: Plots of Henyey Greenstein Phase function (Magenta) against it's 2nd order SH reconstructed phase function (Blue) for $g < 0$. The function is plotted as pdf (y-axis) against scattering angle (in degrees) θ (x-axis).

(a) $g = 0.9$ (b) $g = 0.7$ (c) $g = 0.5$ (d) $g = 0.3$ (e) $g = 0.1$

Figure A.8: Plots of Henyey Greenstein Phase function (Magenta) against it's 2nd order SH reconstructed phase function (Blue) for $g > 0$. The function is plotted as pdf (y-axis) against scattering angle (in degrees) θ (x-axis).