Parallel and Adaptive Visibility Sampling for Rendering Dynamic Scenes with Spatially-Varying Reflectance

Rui Wang[1], Minghao Pan, Xiang Han, Weifeng Chen, Hujun Bao

State Key Lab of CAD&CG, Zhejiang University

Abstract

Fast rendering of dynamic scenes with natural illumination, all-frequency shadows and spatially-varying reflections is important but challenging. One main difficulty brought by moving objects is that the runtime visibility update of dynamic occlusion is usually time-consuming and slow. In this paper, we present a new visibility sampling technique and show that efficient all-frequency rendering of dynamic scenes can be achieved by sampling visibility of dynamic objects in an adaptive and parallel way. First, we propose a two-level adaptive sampling scheme to distribute sample points spatially and compute visibility maps angularly on each sample point. Then, we present a parallel hemispherical distance transform to convert these visibility maps into spherical signed distance fields. Finally, using such a distance-based visibility representation, we integrate our visibility sampling algorithm in the all-frequency rendering framework for scenes with spatially-varying BRDFs. With an entire GPU-based implementation, our algorithm enables interactive all-frequency rendering of moderate dynamic scenes with environment lighting and spatially-varying reflectance.

1. Introduction

Natural illumination, complex shadows and detailed reflections are all important in realistic image synthesis, but usually require high rendering costs. Precomputed radiance transfer (PRT) [1, 2], or precomputation-based rendering, has shown that such rendering costs can be greatly reduced by precomputing the light transport in static scenes. Various basis functions, such as spherical harmonics (SH) basis [1], wavelet basis [2], polynomials [3], spherical radial basis functions [4, 5], etc., have been proposed to approximate complex lighting, all-frequency visibility and spatially-varying BRDFs. However, the requirement of static scenes has so far excluded a lot of PRT methods from many important applications with dynamic objects.

One main challenge brought by dynamic scenes is that the runtime visibility update of dynamic occlusion is usually time-consuming. Some work has been done to extend the PRT framework to dynamic scenes [6, 7, 8, 9]. However, those methods more or less have some restrictions, such as handling only movements of rigid objects, rendering with low-frequency shadows, or requiring preprocess on scene's geometry. These restrictions make these methods unable to truly support all-frequency rendering of fully dynamic scenes.

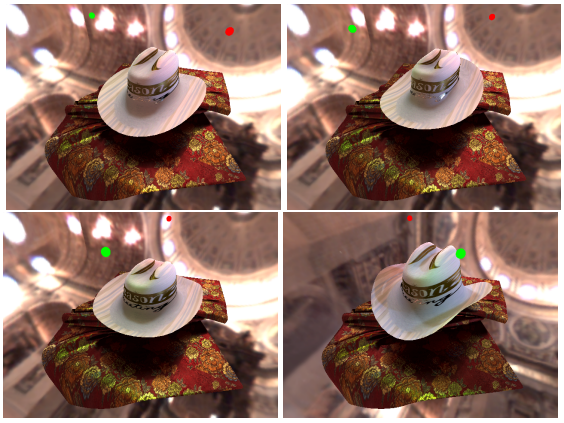


Figure 1: Rendering results of a dynamic scene at average 5.9 FPS. Both the cowboy hat and the piece of cloth are textured by spatially-varying reflectance and illuminated by local and environment lights.

In this paper, we present a new visibility sampling technique and show that efficient all-frequency rendering of dynamic scenes can be achieved by sampling visibility of dynamic objects in an adaptive and parallel way. Based on the observation in [10] that the light transportation in a scene is local and low rank, we first propose a two-level adaptive runtime visibility sampling scheme to distribute sample points spatially, and compute visibility maps angularly on these sample points.

Then, we convert these sampled visibility maps into hemispherical signed distance fields and interpolate between them for shading points. A parallel hemispherical distance transform algorithm is presented to make this conversion fast. Finally, using such a distance-based visibility representation, we integrate our visibility sampling in the all-frequency rendering framework for scenes with spatially-varying BRDFs. With an entire GPU-based implementation, our algorithm enables interactive all-frequency rendering of moderate dynamic scenes.

2. Related Work

Our paper focuses on all-frequency rendering of dynamic scenes. Our work is based on a PRT rendering framework and extends the precomputation of visibility to a runtime adaptive visibility sampling. Thus, we start with a briefly review of the PRT technique and then introduce some techniques that are most relevant to our work.

Precomputation-based rendering. PRT technique [1, 2] enables real-time rendering under natural lighting, complex shadowing and detailed materials. Various basis functions, such as spherical harmonics (SH) basis [1], wavelet basis [2], polynomials [3], spherical radial basis functions [4], etc., have been proposed to approximate lighting, visibility and BRDFs. A comprehensive survey of current PRT techniques can be found in [11]. Our work is based on the all-frequency rendering framework proposed in [5] and then extended in [12]. By approximating lighting and BRDFs into spherical Gaussians and representing visibility into signed spherical distance fields, real-time rendering is achieved for scenes with spatially-varying BRDFs. In our method, we replace the visibility precomputation by a parallel and adaptive visibility sampling algorithm so as to enable fast rendering of dynamic scenes.

The light transport response of vertices in a scene is local and of low rank [13]. It can be highly compressed by clustered principal component analysis (CPCA) [14] or wavelets [2]. Huang and Ramamoorthi [10] proposed a precomputing method to sample the light transport matrix adaptively and sparsely. Although they demonstrated faster precomputation time than previous methods, their sampling and reconstruction method is still too slow for dynamic scenes. Our work is inspired by their work but focuses on runtime update of visibility.

Rendering Dynamic Scenes under Complex lighting. The PRT technique has been extended to render dynamic scenes. Zhou et al. [6] proposed shadow fields to generate soft shadows in scenes with moveable objects. Sun et al. [15] generalized the idea to wavelet product to render scenes with dynamic glossy objects. However, these methods only handle the movement of rigid objects. Zonal spherical harmonics [7] with spheres approximation [8] enable rendering of deformable objects, but is limited to SH-represented low-frequency shadows. Iwasaki et al. [9] presented an all-frequency rendering method for dynamic scenes. But, their method still relies on a precomputed spheres approximation, which is unable to handle arbitrary movement of objects. Our adaptive visibility sampling method does not have such restrictions on scenes.

With the rapid development of computational power of GPU, some methods using GPU-based point lights rendering techniques have been proposed for dynamic scenes. Annen et al. [16] proposed a real-time method for all-frequency shadows in dynamic scenes based on convolution shadow maps. Ritschel et al. [17] presented imperfect shadow maps to compute indirect illuminations from a point approximation of the scene. Besides their work, there is a large body of recent work on GPU-based global illumination, such as [18, 19, 20]. These techniques use GPU to achieve fast illumination, thus support dynamic scenes. However, similar to most real-time point-based shadow map techniques, these methods do not support integrating the BRDF across the area light source domain, which limits its usage in all-frequency rendering of non-diffuse BRDFs, especially spatially-varying BRDFs.

Euclidean Distance Transformation. In this paper, we present a GPU-based parallel hemispherical distance transformation algorithm to generate spherical signed distance fields for visibility interpolation. Our work is inspired by the Euclidean distance transform (EDT) algorithms on 2D images. EDT is an important method in computer vision and geometry processing. A comprehensive survey on 2D image can be found in [21]. Generally, fast EDT algorithms are designed in a dimensionality reduction manner that it first computes in one dimension, e.g. for each row, and then computes in the second dimension, e.g. for each column. Such a dimensionality reduction idea was first proposed in [22] and then improved in [23, 24, 25, 26]. Recently, Cao et al. [26] has further extended the idea to GPUs and achieved fast 2D EDTs by dividing data into different bands. However, due to a non-uniform spherical parameterization, it is non-intuitive to directly apply these 2D EDT algorithms to the hemispherical domain. Michikawa and Suzuki [27] proposed a sequential algorithm for the spherical EDT but it is too slow for our application. In this paper, we introduce a new

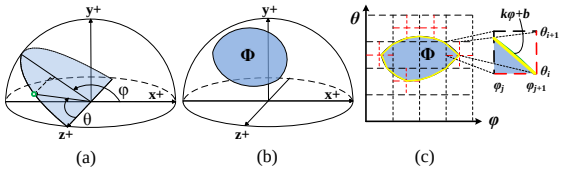


Figure 2: (a) Hemispherical parameterization of (θ, ϕ). (b) A visible region, Φ, on the hemisphere. (c) The visibility boundary approximation of the visible region Φ.

parallel distance transformation algorithm, which is designed specifically for the distance transform on the hemisphere.

3. Overview

Our visibility sampling algorithm is based on the all-frequency rendering framework for spatially-varying BRDFs, which was first proposed in [5] and then extended in [12] for better visibility approximations. In this section, we first briefly describe the all-frequency rendering framework and then introduce our visibility sampling algorithm.

3.1. Rendering Framework

The direct lighting at point $\mathbf{x}$ in view direction ω_o can be computed by the following integral:

$$L_o(\mathbf{x}, \omega_o) = \int_{\Omega_{2\pi}} L_i(\mathbf{x}, \omega) f(\mathbf{x}, \omega, \omega_o) V(\mathbf{x}, \omega) (\mathbf{n} \cdot \omega) d\omega \tag{1}$$

where L_o is the outgoing radiance, L_i is the incident lighting, $f(\mathbf{x}, \omega, \omega_o)$ is the BRDF, $V(\mathbf{x}, \omega)$ is the visibility function, and $(\mathbf{n} \cdot \omega)$ is the cosine term. To handle the all-frequency rendering with spatially-varying reflectance, Wang et al. [5] approximated the lighting and BRDFs by spherical Gaussians, and represented the visibility into spherical signed distance functions. In this paper, we follow the formation, where the incident lighting and BRDFs are computed as,

$$\begin{aligned} L_i(\omega) &\approx \sum_l G_l(\omega; \mathbf{p}_l, \lambda_l, \mu_l), \\ f(\omega, \omega_o) &\approx \sum_m G_m(\omega; \mathbf{p}_m^{\omega_o}, \lambda_m^{\omega_o}, \mu_m^{\omega_o}). \end{aligned} \tag{2}$$

in which a spherical Gaussian G has the following form:

$$G(\omega; \mathbf{p}, \lambda, \mu) = \mu . e^{\lambda(\mathbf{p} \cdot \omega - 1)} \tag{3}$$

where $\mathbf{p}$ is the lobe direction, λ is the lobe sharpness and μ is the lobe amplitude. The spherical signed distance

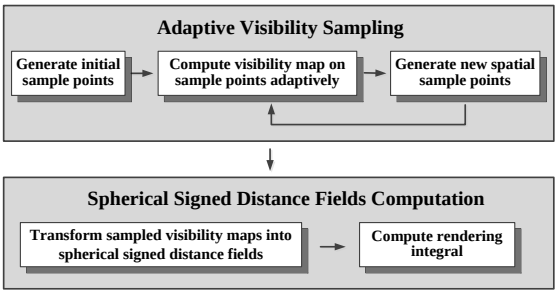


Figure 3: Our visibility sampling algorithm is split up into two steps: adaptive visibility sampling (described in Section 4) and hemispherical distance fields transformation (described in Section 5).

function (SSDF) [28] is used as an intermediate visibility representation, where for a direction $\mathbf{v}$, it stores a signed angular distance to the closest visibility boundary. The sign of the function encodes whether the direction $\mathbf{v}$ is occluded or not. It is defined as

$$D(\mathbf{v}) = \begin{cases} + \min\limits_{V(\mathbf{t})=0} \arccos(\mathbf{t} \cdot \mathbf{v}), \text{if } V(\mathbf{v}) = 1 \\ - \min\limits_{V(\mathbf{t})=1} \arccos(\mathbf{t} \cdot \mathbf{v}), \text{if } V(\mathbf{v}) = 0 \end{cases} \tag{4}$$

where $\mathbf{v}$ and $\mathbf{t}$ are vectors on a unit sphere, $V(\cdot)$ is the visibility function that returns 0 when being occluded and 1 otherwise. To better represent the visibility, in [12], the SSDF, $D(\mathbf{v})$, is extended to adaptively sampled distance field [29], where the domain of the SSDF is adaptively divided into a set of cells, and visibility boundaries are approximated by iso-lines in these cells, Figure 2(c). In this way, in each cell, the rendering integral is

$$L_o^{ij} = \int_{\varphi_i}^{\varphi_{i+1}} \int_{\theta_j}^{k\varphi+b} \sum_{l,m} G_{l,m}(\omega)(\mathbf{n} \cdot \omega) \sin\theta d\theta d\varphi. \tag{5}$$

where (φ, θ) is the 2D parameterizations of a sphere, Figure 2(a), $\varphi_i, \varphi_{i+1}, \theta_j, \theta_{j+1}$ are upper and lower bounds of a cell, and $k\varphi + b$ is the iso-line function in the cell, Figure 2(c). The final outgoing radiance of Eq. (1) can be computed by summing radiance from all cells. In this rendering framework, binary visibility maps are sampled at vertices and converted into distance functions to be interpolated for per-pixel shading.

Both in [5, 12], the spatially-varying field of SSDFs on vertices are computed in a preprocess. This makes them only applicable for static scenes. Our goal is to speed up the computation of these SSDFs.

3.2. Visibility Sampling for Dynamic Scenes

Based on the observation that these spatially-varying SSDFs on vertices can be heavily compressed by

PCA [5], the full visibility computation on all vertices is unnecessary and redundant.

We present a new parallel and adaptive visibility sampling algorithm that computes the visibility of dynamic scenes fast as well as updates the SSDFs at runtime. Our algorithm is split up into two main steps: adaptive visibility sampling and hemispherical distance transform (Figure. 3). In adaptive visibility sampling, we take a two-level adaptive scheme to compute the visibility spatially and angularly. At the first level, a set of sample points are initially distributed on vertices of objects, and then spatially refined in iterations. New sample points are placed in areas with high visibility variations. At the second level, on each sample point, rays are generated adaptively to compute the hemispherical visibility map according to sampled visible boundaries. The goal of these steps is to generate spatial sample points and angular rays to approximate the visibility distribution in the scene such that the visibility can be computed as close as possible to the real disctribution while using as few samples as possible. After computing visibility maps on sample points, we present a parallel hemispherical distance transformation (HSDT) algorithm to convert visibility maps into SSDFs. These SSDFs are interpolated per pixel to compute the render integral Eq. 5. We will discuss more details of these steps in following sections.

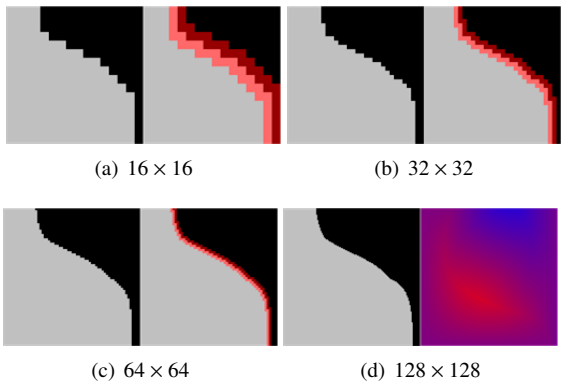
4. Adaptive Visibility Sampling

In this section, we introduce our two-level parallel and adaptive visibility sampling algorithm. One level is at adaptively inserting or erasing sample points on surface to spatially approximate the visibility distribution. The other level is at adaptively computing the visibility map, $V_{\mathbf{x}}$, of visibility sample point $\mathbf{x}$. To compute the visibility map efficiently, we use adaptive ray-tracing to evaluate the binary visibility map at spatial sample points and then estimate visibility differences among these sample points to adaptively insert new sample points or erase old sample points.

4.1. Generate Initial Sample Points

Before we start the adaptive sampling, a set of sample points are initially generated on geometry surfaces. To simplify the computation, we select sample points from geometry vertices. Since at this stage information of visibility is unavailable, we employ a geometric error metric proposed in [19], e_g, to find best candidates from vertices for sample points as,

$$e_g(x_i, x_k) = \alpha \|x_i - x_k\| + \sqrt{2 - 2(\vec{n}_i \cdot \vec{n}_k)} \qquad (6)$$



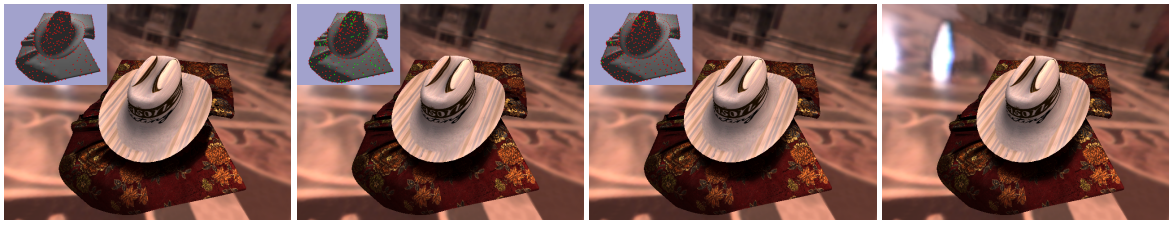
(a) 16×16 (b) 32×32 (c) 64×64 (d) 128×128

Figure 4: Adaptive angular visibility sampling. (a)~(d) are visibility maps with different resolution. Only boundary rays (in red) are generated for further visibility tests. In (d), we visualize the SSDF after our hemispherical distance transform.

where x_i and x_k are two vertices, $\vec{n}$ denotes a surface normal and α is a weighting factor that determines the relative importance of position and normal incurred changes, which typically varies between $0.1 \sim 0.5$ in our experiments (after the scene geometry scale is normalized). Based on such an error metric, we cluster vertices into groups by an iterative GPU-based KMean algorithm. After the clustering, we select the vertex that is closest to each cluster center as the sample point, in order to guarantee every sample point is on the surface.

4.2. Adaptively Compute Visibility Function

To adaptively sample visibility maps on sample points, we use a GPU-based ray-tracing method [30] to generate visibility rays. To utilize the parallel computational power, a multi-pass algorithm is taken. In the first pass, a small number, e.g 16×16, of rays, which are uniformly distributed on hemisphere, are generated for intersection tests. At following passes, these rays, producing visible boundaries, are sampled densely until the required resolution is reached. These boundary rays are detected from 8-neighborhood search on the visibility map generated in previous pass. If one neighbor pixel has different intersection status, e.g. the neighbor pixel is unocclued when the center pixel is occluded or the neighbor pixel is occluded when the center pixel is unoccluded, the center pixel is regarded as boundary pixel and the ray represented by the center pixel is regarded as a boundary ray. In Figure 4, we illustrate an example of the adaptive angular visibility sampling. It can be seen that our method only samples visibility at boundary regions, which are usually much sparser than the entire hemisphere space. However, if there are some small or slim objects in the scene, at the first pass, the initial



(a) Initial (b) Iteration 1 (c) Iteration 2 (d) Reference

Figure 5: Adaptive spatial sample points

16×16 may be not enough to capture all detailed occlusions. To improve the visibility sampling at each pass, we not only use visibility rays generated from this sample point but also use rays from adjacent sample points to detect boundary rays. Compared with local visibility rays, rays from adjacent sample points provide a good supplementary and reduce the possibility of missing small occluders. Furthermore, in some scenes, we increase the initial visibility rays from 16×16 to 32×32 to improve the sampling of visibility maps.

4.3. Adaptively Distribute Visibility Sample Points

After having the visibility maps on each sample point, we use these visibility information to update the spatial distribution of sample points. Our goal is to insert new sample points at regions with large visibility variations and erase sample points at places that has been approximated well by other sample points.

In order to fast evaluate the spatially-varying visibility among visibility sample points, we project visibility maps on sample points to spherical harmonics (SH) and then use the difference of SH coefficients to be the spatial visibility error, e_v.

$$e_v(x_i, x_k) = \sum_{l,m} \| c_i^{l,m} - c_k^{l,m} \| \tag{7}$$

where $c^{l,m}$ are SH coefficients and l and m are the band and index. In our method, we use total 16 SH coefficients.

Given the spatial visibility error, e_v, we take three steps to adaptively update visibility sample points. First, we build a KD-tree on existing sample points. Then, for a sample point, x_i, we take range search to find adjacent neighbors, x_j. From these neighbors, we interpolate a new sample point x_{avg} at the spatial position of current sample point x_i as

$$x_{avg} = \sum_{j=1}^{n} w_j x_j, w_j = \frac{\frac{1}{e_g(x_i,x_j)}}{\sum_{p=1}^{n} \frac{1}{e_g(x_i,x_j)}} \tag{8}$$

where w_j is an inverse distance weighting function based on the geometric error metric, e_g. Given the interpolated x_{avg}, we use Eq.(7) to compute the spatial visibility error between x_{avg} and x_i as $e_v(x_i, x_{avg})$. If the error is larger than a threshold, $E_{v,max}$, it indicates that there might be larger visibility variation between this sample point and its adjacent neighbors. In this case, new sample points will be added. If the error is smaller than a threshold, $E_{v,min}$, it indicates that this sample point x_i can be well interpolated by adjacent neighbors. In this case, it is marked as a candidate to be erased. If the error is between $E_{v,max}$ and $E_{v,min}$, this sample point will be kept in this iteration and without going through further operations.

The final step is to insert new sample points or erase sample points. When inserting new sample points, we first compute pairwise visibility error, e_v, between x_i and all its neighbors. The neighbors that produce errors larger than $E_{v,max}$ are collected. For each pair of x_i and the neighbor with large error, we average their spatial positions to obtain a candidate position, and select the vertex closest to the candidate position to be the new sample point. When erasing sample points, all candidates to be erased are collected. But we only erase one sample point, of which all neighbors are not in the candidate list. This is to avoid erasing too many sample points at one time. After erasing these sample points, further computations related to these samples, such as new sample point insertions and distance field interpolations, can be saved.

Figure 5 shows several iterations of our adaptive distribution of visibility sample points. Points in red are sample points generated in previous iteration and green ones are sample points inserted after one iteration. It can be observed that with more spatial sample points generated, the shadow under the hat gets improved.

5. Parallel Hemispherical Distance Transform

Given the visibility map V_x and the definition of the distance field in Eq. 4, our algorithm requires a fast dis-

tance transform method. The state-of-the-art EDT algorithm with best performance is computed in a dimensionality reduction manner. However, in our paper, the 2D parameterization from spherical visibility map to 2D dimensional domain is non-uniform. This makes us unable to directly apply these 2D dimensionality reduction EDT algorithms to our hemisphere distance transform. Fortunately, we observe that in the spherical cse, if we reduce dimensions in a certain order, i.e. first scan latitude and then scan longitude, the distance inequalities used in dimensionality reduction of 2D EDT algorithm still hold. Proofs can be found in our supplementary document. In this way, we restrict the scanning order in dimensions of latitude (parameterized in θ in our paper) and longitude (parameterized in ϕ), and use the spherical distance as the Euclidean distance to extend the EDT algorithm [24] from the 2D Euclidean space to the hemispherical space.

The main steps of the algorithm are listed in Algorithm 1. It takes two phases, the LatitudeScans and the LongitudeScans, to convert visibility maps into SSDFs. In our paper, each SSDF is represented in a square image with a 128×128 resolution, where one dimension is parameterized in latitude, ϕ_j, and the other dimension is in longitude, θ_i. In the LatitudeScans, the computation is taken in parallel for each θ_i. At each θ_i, we first scan ϕ_j from 0 to π and then scan backward. A temporary array, $g[i, j]$, is used to store the closest visible boundary point along latitude for each (θ_i, ϕ_j). The first scan will find the closest visible boundary point from one direction and then update from another direction. In the LongitudeScans, at each ϕ_j, a forward scan and a backward scan are taken on θ_i to extend all closest visible boundary in the latitude dimension to the longitude dimension. After these two phases, we obtain final distance field. In both phases, these scans can be parallelized and computed efficiently. More algorithm details can be found in the supplementary document.

Algorithm 1 Hemispherical Euclidean Distance Transform

procedure LatitudeScans()
for each θ_i **in parallel do**
  for $\phi_j = 0$ to π **do**
    if $V(\theta_i, \phi_j)$ is visible **then**
      $g[i, j] = (\theta_i, \phi_j)$
    else
      $g[i, j] = g[i, j-1]$
for each θ_i in **in parallel do**
  $s = (\theta_i, \pi)$
  for $\phi_j = \pi$ to 0 **do**
    if D$(g[i, j], (\theta_i, \phi_j))$¿D$(s, (\theta_i, \phi_j))$ **then**
      $g[i, j] = s$
    else
      $s = g[i, j]$
end

procedure LongitudeScans()
for each ϕ_j **in parallel do**
  $q = 0, t[0] = (0, \phi_j), s[0] = (0, \phi_j)$
  for $\theta_i = 0$ to π **do**
    while $q \geq 0$&&D$(t[q], s[q])$¿D$(t[q], (\theta_i, \phi_j))$ **do**
      $q = q - 1$
      if $q < 0$ **then**
        $q = 0, s[0] = (\theta_i, \phi_j)$
      else
        $\theta_w = \Theta(\phi_i, s[q], g[i, j])$
        $q = q + 1, s[q] = g[i, j], t[q] = (\theta_w, \phi)$
for each ϕ_j **in parallel do**
  for $\theta_i = \pi$ to 0 **do**
    if $\theta_i \geq w[q]$ **then**
      $d[i, j] =$ D$((\theta_i, \phi_j), s[q])$
    else
      $d[i, j] =$ D$((\theta_i, \phi_j), s[q-1]), q = q - 1$
end

6. Rendering

Our adaptive visibility sampling algorithm naturally fits in the all-frequency rendering framework proposed in [5, 12]. In this section, we introduce the implementation to integrate the visibility sampling algorithm in the rendering of dynamic scenes.

Before the runtime computation, in a preprocess, we approximate all environmental lighting and BRDFs by a set of spherical gaussians [5]. At runtime, we first update the KD-tree of the scene using the GPU-based KD-tree construction [30]. Then, we rasterize the dynamic scene into several frame buffers, where for each pixel, we store the spatial position, local coordinate frame, texture coordinate and the material index. After that, we apply the visibility sampling algorithm introduced in this paper to adaptively compute visibility maps and transform them into SSDFs. By interpolating these SSDFs from sample points to shading points, we compute the final rendering integral as that in [12]. At interpolating SSDFs, for each shading point, a local range search is employed to find adjacent sample points. The weight used in interpolation is the weight in Eq. 8.

Optimization. To efficiently compute the SH coefficients of visibility maps, we tabulate the SH basis on hemisphere into multi-resolution images and use short float (16 bits) to store them. We load higher levels of

this tabulated basis images in constant memory of GPU for fast data fetch, and try to keep lower levels in L2 cache to speedup the computation. Additionally, observing that shading points may not require all sample points generated in the initial step, we use rasterized frame buffer to collect sample points. Only these points that may be used for shading are taken for further adaptive computation. In each stage, computation in the rendering is organized into bathes. We always try to launch enough threads to utilize the parallel computational power and hide the latency.

7. Results

All results are generated in an image resolution of 640×480 and computed on a PC with an Intel Core™ i7 3770 CPU and a NVIDIA GeForce 680 GTX graphics card with 1G memory. The lighting probes are from [31]. The spatially-varying BRDF textures are available from [32, 33].

Our algorithm mainly introduces two kinds of parameters. One kind is the spatial visibility error thresholds to control the distribution of visibility sample points, $E_{v,\max}$ and $E_{v,\min}$. In our implementation, we use relative error thresholds, where $E_{v,\min} < e_v(x_i, x_{avg})/e_v(x_i, x_i) < E_{v,\max}$. The other kind is the resolution of visibility maps, r. To better visualize the changes of shadows under different parameters, we setup a simple scene that contains all diffuse objects and has one area light lightening from the top. Figure 6 shows some rendering results of such a scene under different parameter settings. From left to right, with the decrease of spatial visibility error threshold, $E_{v,\max}$, more visibility sample points are generated to better capture the shadow under the hat. From top to bottom, two different resolutions, 64×64 and 128×128 of visibility maps are used. It can be seen that with the increase of resolutions of visibility maps, shadows are better generated and with less ringing effects. In our following results, we use $E_{v,\max} = 10\%$ and $E_{v,\min} = 5\%$, and $r = 128 \times 128$ as default.

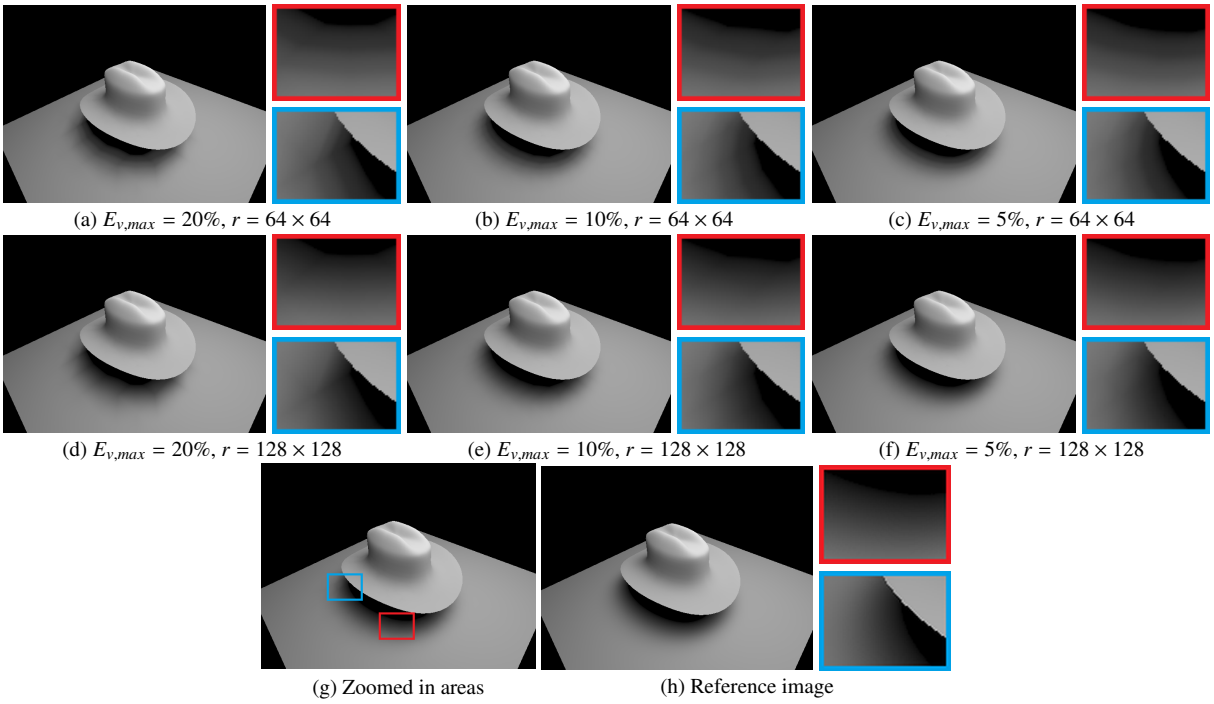
The statistics of our test scenes are listed in Table 1. The visibility map is computed with a resolution of 128×128 on each sample point. In order to reduce the storage sizes of visibility maps and SSDFs, we store visibility maps in 1 bit arrays and SSDFs in short float (16 bits) arrays. Hence, the total memory consumption for one sample point is 2KB of the visibility map and 32KB of the SSDF. To perform the GPU-based ray tracing, we keep a kd-tree in the memory, which consumes 15MB to 22MB for different scenes used in this paper. The total memory usage is reported in Table 1. For adaptive visibility sampling, we measure the time to compute visibility maps and that to take hemispherical distance transforms separately. It can be observed that most time is spent at computing visibility maps, t_V. The time to compute rendering integration, t_I, relates to the number of pixels. The time to take hemispherical distance transforms, t_S, is proportional to sample points.

To validate our method, we make further comparisons with two existing methods, the shadow map-based approach that replaces our adaptive visibility sampling by rendering shadow maps, and the GPU-based Monte Carlo ray tracing that directly computes the rendering equation by sampling rays. To compare with the shadow map-based approach, we use the scene shown in Figure 1 as the test scene and only compare the performance of computing visibility maps. Three different resolutions, 64×64, 128×128 and 256×256, of visibility maps are used and evaluated. Results are plotted in Figure 7. It can be seen that the shadow map-based approach has better performance when the resolution of visibility map is low. However, when the resolution of visibility map increases, our parallel and adaptive visibility sampling algorithm outperforms. It demonstrates the better scalability of our method. To compare with the GPU-based Monte Carlo ray tracing, we use the scene shown in Figure 9(a) as the test scene. We adapt the multiple importance sampling [34] to increase the sampling efficiency of Monte Carlo integration in scenes with environment maps and glossy BRDFs. Results generated by two methods are shown in Figure 8. It can be seen that at the same rendering time, our method has much less noise, and at the similar rendering quality, our method has about 10× speedup. The reason is mainly because our method only computes the visibility maps on certain visibility sample points, whereas in Monte Carlo ray tracing, for each pixel, a set of rays are required to generate for integration.

Figure 1 and Figure 9 show some screen shots of different dynamic scenes. The first scene is a deformed cowboy hat model on a piece of folded cloth. Both hat and cloth are textured by spatially-varying reflectance. Note the self shadow on the hat, and on the cloth and the shadow changes while hat deforms. Second scene is two balls rolling on a dish. Balls are rendered using Blinn-Phong model and the dish is used in Cook-Torrance model. Note the shadow changes while positions of balls move. Third scene is a walking man with a hat. Man and hat are both in diffuse BRDF and the plane is textured by spatially-varying reflectance. Due to different BRDFs, the specular shadows and diffuse shadows are different. Please refer to the supplementary video for the entire sequences of dynamic scenes. For these three scenes, our algorithm achieves interac-

Scene	#Pixels	#Verts	#Samples	Total Memory	t_I	t_V	t_S	FPS
Figure. 1	104k	21k	1282	57.5 MB	47ms	95ms	28ms	5.9
Figure. 9(a)-(c)	127k	30k	1824	81.5 MB	56ms	183ms	40ms	3.6
Figure. 8 and 9(d)-(f)	117k	32k	3527	139.1 MB	53ms	557ms	77ms	1.5

Table 1: Statistics of test scenes. From left to right, columns present different scenes, the number of average view pixels, vertices and average sample points, the average times to take integration, t_I, sample visibility, t_V, and take hemispherical distance transform, t_S. The final column is the average FPS.



(a) $E_{v,max} = 20\%$, $r = 64 \times 64$ (b) $E_{v,max} = 10\%$, $r = 64 \times 64$ (c) $E_{v,max} = 5\%$, $r = 64 \times 64$

(d) $E_{v,max} = 20\%$, $r = 128 \times 128$ (e) $E_{v,max} = 10\%$, $r = 128 \times 128$ (f) $E_{v,max} = 5\%$, $r = 128 \times 128$

(g) Zoomed in areas (h) Reference image

Figure 6: Results comparison using different parameters. From left to right, (a) to (c) and (d) to (f), with the decrease of spatial visibility error threshold, $E_{v,\max}$, more visibility sample points are generated to better capture the shadow under the hat. From top to bottom, (a) to (d), (b) to (e), and (c) to (f) , two different resolutions, 64×64 and 128×128, of visibility maps are used. It can be seen that with the increase of resolutions, shadows are better produced and with less ringing effects. (g) shows the zoomed in areas. (h) shows the reference image generated by per pixel shading with 256×256 visibility maps.

tive rendering at average 5.9, 3.6 and 1.5 FPS.

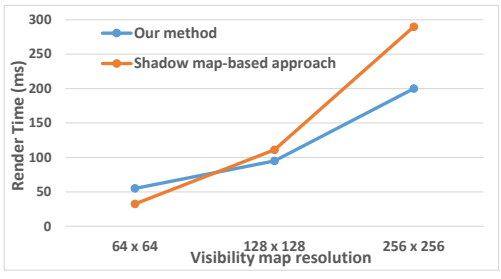
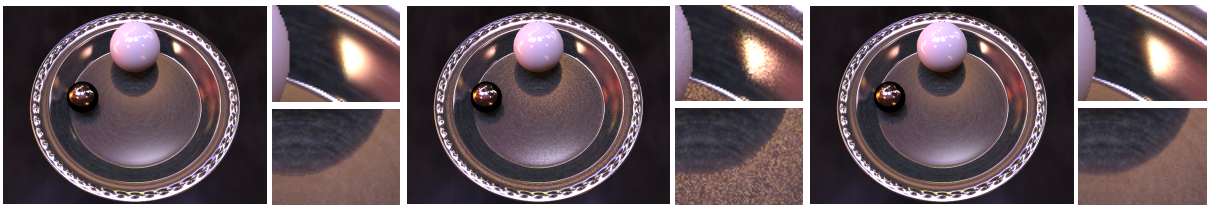


Figure 7: Performance comparison between our method and the shadow map-based approach in rendering the scene shown in Figure 1.

8. Discussion and Conclusion

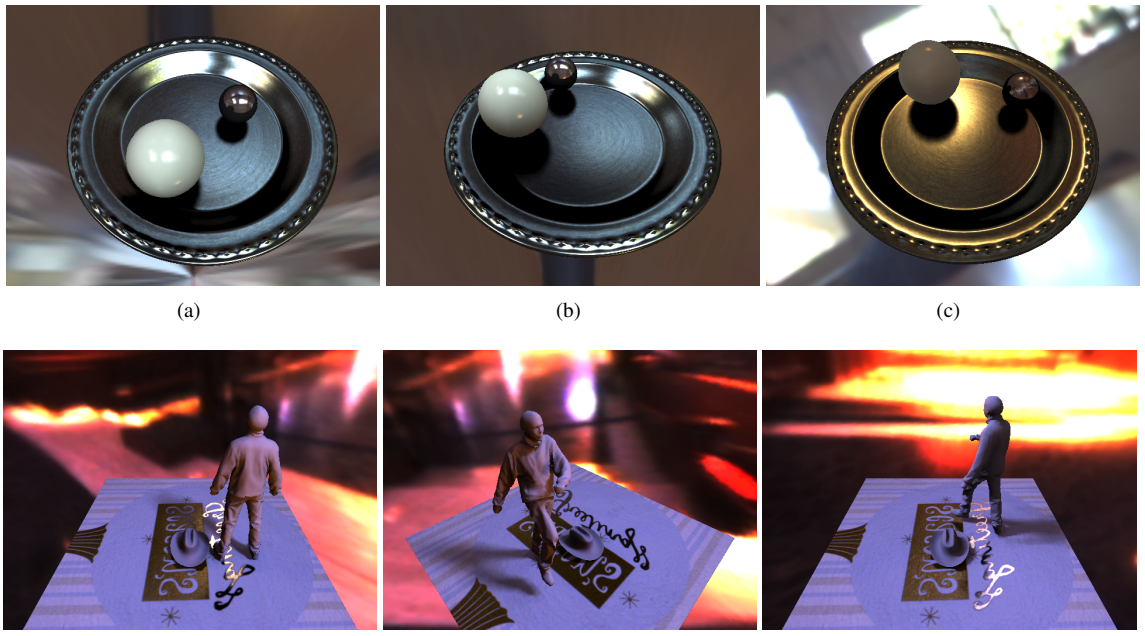
In this paper, we propose an adaptive visibility sampling algorithm for all-frequency rendering of dynamic scenes. Our approach mainly takes two steps, adaptive visibility sampling and hemispherical distance transformation. In the adaptive visibility sampling, we take a two-level adaptive scheme to compute the visibility spatially and angularly. The goal of such an adaptive sampling is to distribute spatial sample points and angular rays to represent true visibility in the scene while using as few samples as possible. In the hemispherical distance transform, we extend the 2D EDT algorithm to hemispherical space and achieve fast transformation. With an entire GPU-based implementation, our algorithm enables interactive all-frequency rendering of moderate dynamic scenes with environmental lighting and spatially-varying reflectance.

Some limitations imposed in our method might be good directions for future work. First, our method cannot handle semi-transparent surfaces. The visibility map



(a) Our result (b) Result of MC with the same rendering time (c) Result of MC with a similar quality

Figure 8: Comparison with GPU-based Monte Carlo ray tracing. (a) is generated by our method. The rendering time is about 253ms. (b) is generated by GPU-based Monte Carlo ray tracing with the same rendering time. The average samples per pixel is about 128. It can be seen that the image has much more noise than our result. For such a scene with glossy BRDF and environment lighting, it takes 2.48s and requires about 1024 samples per pixels to obtain a result with similar quality as ours, which is shown in (c). Our method has about 10× speedup.



(a) (b) (c)

(d) (e) (f)

Figure 9: Rendering results of more dynamic scenes.

only records visible status. It will be an interesting direction to integrate semi-transparent surfaces or multi-bounces light transports in our framework. Potential ways might be using separate or different visibility maps for different semi-transparent objects or multi-bounces. Second, we used the difference of SH coefficients as the spatial visibility error metric. Such an error metric may miss some high-frequency directional variations and result in insufficient sampling in some cases. It produced some slight flickering effects in rendering the entire sequences of dynamic scenes. More elegant error criteria will be an interesting topic in future. Third, since the compression and uncompression cost on SSDFs is high, we stored raw SSDFs for each sample point, which took tens to hundreds of megabytes storage size for our test scenes. In the future, fast compression or dynamic storage of SSDFs on sample points may be required to extend our method for a scene with a large number of sample points. Finally, the adaptive visibility sampling in our method is non-conservative. In some cases, slim or small occluders may be missed. But, given the complexity and computational cost of conservative visibility sampling algorithms, it is with great difficulties and challenges to adapt them for rendering scenes with natural illumination, all-frequency shadows and spatially-varying reflections. We believe that our adaptive visibility sampling algorithm provides an effective approximation on the visibility of the scene, and the fast computation of our method will be benefit for many applications.

Acknowledgements

We would like to thank the anonymous reviewers for their thoughtful comments. This work was partially

supported by NSFC (No. 60903037 and 61272301), the 863 program of China (No. 2012AA011902) and the Fundamental Research Funds for the Central Universities (No. 2012XZZX013).

References

[1] P.-P. Sloan, J. Kautz, J. Snyder, Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments, ACM Trans. Graph. 21 (3) (2002) 527–536.

[2] R. Ng, R. Ramamoorthi, P. Hanrahan, All-frequency shadows using non-linear wavelet lighting approximation, ACM Trans. Graph. 22 (3) (2003) 376–381.

[3] A. Ben-Artzi, R. Overbeck, R. Ramamoorthi, Real-time brdf editing in complex lighting, in: SIGGRAPH '06: ACM SIGGRAPH 2006 Papers, ACM, New York, NY, USA, 2006, pp. 945–954.

[4] Y.-T. Tsai, Z.-C. Shih, All-frequency precomputed radiance transfer using spherical radial basis functions and clustered tensor approximation, in: SIGGRAPH '06: ACM SIGGRAPH 2006 Papers, ACM, New York, NY, USA, 2006, pp. 967–976.

[5] J. Wang, P. Ren, M. Gong, J. Snyder, B. Guo, All-frequency rendering of dynamic, spatially-varying reflectance, in: ACM SIGGRAPH Asia 2009 papers, 2009, pp. 1–10.

[6] K. Zhou, Y. Hu, S. Lin, B. Guo, H.-Y. Shum, Precomputed shadow fields for dynamic scenes, in: SIGGRAPH '05: ACM SIGGRAPH 2005 Papers, ACM, New York, NY, USA, 2005, pp. 1196–1201.

[7] P. Sloan, B. Luna, J. Snyder, Local, deformable precomputed radiance transfer, in: Proc. of SIGGRAPH '05, 2005, pp. 1216–1224.

[8] Z. Ren, R. Wang, J. Snyder, K. Zhou, X. Liu, B. Sun, P.-P. Sloan, H. Bao, Q. Peng, B. Guo, Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation, in: ACM SIGGRAPH 2006 Papers, 2006, pp. 977–986.

[9] K. Iwasaki, W. Furuya, Y. Dobashi, T. Nishita, Real-time rendering of dynamic scenes under all-frequency lighting using integral spherical gaussian, Comp. Graph. Forum 31 (2pt4) (2012) 727–734.

[10] F.-C. Huang, R. Ramamoorthi, Sparsely precomputing the light transport matrix for real-time rendering, Computer Graphics Forum 29 (4) (2010) 1335–1345.

[11] R. Ramamoorthi, Precomputation-based rendering, Found. Trends. Comput. Graph. Vis. 3 (4) (2009) 281–369.

[12] R. Wang, M. Pan, W. Chen, Z. Ren, K. Zhou, W. Hua, H. Bao, Analytic double product integrals for all-frequency relighting, Visualization and Computer Graphics, IEEE Transactions on 19 (7) (2013) 1133–1142.

[13] D. Mahajan, I. K. Shlizerman, R. Ramamoorthi, P. Belhumeur, A theory of locally low dimensional light transport, ACM Trans. Graph. 26 (3).

[14] P. Sloan, J. Hall, J. Hart, J. Snyder, Clustered principal components for precomputed radiance transfer, in: Proc. of SIGGRAPH '03, 2003, pp. 382–391.

[15] W. Sun, A. Mukherjee, Generalized wavelet product integral for rendering dynamic glossy objects, ACM Trans. Graph. 25 (3) (2006) 955–966.

[16] T. Annen, Z. Dong, T. Mertens, P. Bekaert, H.-P. Seidel, J. Kautz, Real-time, all-frequency shadows in dynamic scenes, in: ACM SIGGRAPH 2008 papers, 2008, pp. 1–8.

[17] T. Ritschel, T. Grosch, M. H. Kim, H.-P. Seidel, C. Dachsbacher, J. Kautz, Imperfect shadow maps for efficient computation of indirect illumination, in: ACM SIGGRAPH Asia 2008 papers, 2008, pp. 1–8.

[18] S. Laine, H. Saransaari, J. Kontkanen, J. Lehtinen, T. Aila, Incremental instant radiosity for real-time indirect illumination, in: Proceedings of the 18th Eurographics conference on Rendering Techniques, EGSR'07, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2007, pp. 277–286.

[19] R. Wang, R. Wang, K. Zhou, M. Pan, H. Bao, An efficient gpu-based approach for interactive global illumination, ACM Trans. Graph. 28 (3) (2009) 91:1–91:8.

[20] M. Holländer, T. Ritschel, E. Eisemann, T. Boubekeur, Manylods: parallel many-view level-of-detail selection for real-time global illumination, in: Proceedings of the Twenty-second Eurographics conference on Rendering, EGSR'11, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2011, pp. 1233–1240.

[21] R. Fabbri, L. D. F. Costa, J. C. Torelli, O. M. Bruno, 2D Euclidean distance transform algorithms: A comparative survey, ACM Comput. Surv. 40 (1) (2008) 1–44.

[22] A. Rosenfeld, J. L. Pfaltz, Sequential operations in digital picture processing, J. ACM 13 (4) (1966) 471–494.

[23] M. N. Kolountzakis, K. N. Kutulakos, Fast computation of the euclidian distance maps for binary images, Information Processing Letters 43 (4) (1992) 181 – 184.

[24] A. Meijster, J. Roerdink, W. Hesselink, A general algorithm for computing distance transforms in linear time 18 (2002) 331–340.

[25] J. Maurer, C.R., R. Qi, V. Raghavan, A linear time algorithm for computing exact euclidean distance transforms of binary images in arbitrary dimensions, Pattern Analysis and Machine Intelligence, IEEE Transactions on 25 (2) (2003) 265–270.

[26] T.-T. Cao, K. Tang, A. Mohamed, T.-S. Tan, Parallel banding algorithm to compute exact distance transform with the gpu, in: I3D '10: Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games, ACM, New York, NY, USA, 2010, pp. 83–90.

[27] T. Michikawa, H. Suzuki, Spherical distance transforms, in: Computational Sciences and Its Applications, 2008. ICCSA '08. International Conference on, 2008, pp. 405–412.

[28] K. Xu, Y.-T. Jia, H. Fu, S. Hu, C.-L. Tai, Spherical piecewise constant basis functions for all-frequency precomputed radiance transfer, IEEE Transactions on Visualization and Computer Graphics 14 (2) (2008) 454–467.

[29] S. F. Frisken, R. N. Perry, A. P. Rockwood, T. R. Jones, Adaptively sampled distance fields: a general representation of shape for computer graphics, in: SIGGRAPH 2000 papers, 2000, pp. 249–254.

[30] K. Zhou, Q. Hou, R. Wang, B. Guo, Real-time kd-tree construction on graphics hardware, ACM Trans. Graph. 27 (5) (2008) 126:1–11.

[31] P. E. Debevec, J. Malik, Recovering high dynamic range radiance maps from photographs, in: SIGGRAPH '97 paper, 1997, pp. 369–378.

[32] J. Lawrence, A. Ben-Artzi, C. DeCoro, W. Matusik, H. Pfister, R. Ramamoorthi, S. Rusinkiewicz, Inverse shade trees for non-parametric material representation and editing, in: SIGGRAPH '06: ACM SIGGRAPH 2006 Papers, ACM, New York, NY, USA, 2006, pp. 735–745.

[33] J. Wang, S. Zhao, X. Tong, J. Snyder, B. Guo, Modeling anisotropic surface reflectance with example-based microfacet synthesis, ACM Trans. Graph. 27 (3) (2008) 1–9.

[34] E. Veach, Robust monte carlo methods for light transport simulation, Ph.D. thesis, Stanford University (1997).