Vis Comput (2015) 31:105–114
DOI 10.1007/s00371-013-0914-1

ORIGINAL ARTICLE

Adaptive cluster rendering via regression analysis

Xiao Dan Liu · Chang Wen Zheng

Published online: 1 January 2014

Abstract Monte Carlo ray tracing suffers noise and aliasing because of low sampling rate. We show that sparse samples can be used to generate high quality images based on feature cluster and regression analysis. Our algorithm has two main stages: adaptive sampling and polynomial reconstruction. In sampling stage, rendering space are organized into clusters based on their features. A feature vector is used to distinguish the different features, which contains gradient, variance and position. Clusters are progressively modified by adaptive sampling. In reconstruction stage, we model each cluster by smooth polynomial functions using regression analysis. The final image is synthesized by integrating these functions. The experiments show that our algorithm generates higher quality images than the previous methods.

Keywords Cluster sampling · Adaptive rendering · Feature vector · Polynomial function

1 Introduction

Photorealistic rendering of the real-world phenomena is important in Computer Graphics. It is widely used in research and industry. Monte Carlo ray tracing is a powerful technique to generate photorealistic images such as global illumination, motion blur and depth of field. But if we want to render high quality images, the consumption of ray tracing is expensive. Otherwise, if the sampling rate is low, there is always noise and aliasing in the results.

In order to reduce the noise and aliasing, early work focuses on sampling the local high frequency areas. These methods adaptively sample the rendering space using local variance [1]. Opposite to space analysis, Fourier transform of the spectrum is introduced to filter the samples in frequency domain. These kinds of methods sample and reconstruct the rendering space using its Fourier spectrum [2]. Because the derivation of general spectrum equation is hard, these frequency spectrum methods can only handle one- or two-dimensional space.

Monte Carlo ray tracing images are synthesized by integrating the light field through multidimensional rendering space including image dimensions, time dimension and lens dimensions. Most regions of the multidimensional space are smooth except the discontinuities such as the shadow edge and object border. One main reason of noise and aliasing is that the discontinuities always make the signal not band limited and cannot be entirely sampled. Recent work generates images by partitioning and reconstruction the rendering space. Bala et al. [3] separate the rendering space into edges and points to generate smooth results. This method needs special information to identify the edges. Hachisuka et al. [4] partition the multidimensional space into different parts and use anisotropic reconstruction to give high quality results. This method assumes the separated parts as a flat region, which may cause aliasing.

Motivated by the previous work, we propose a novel algorithm using feature cluster and regression analysis to generate smooth and high quality photorealistic images. The rendering space is partitioned into a grid of cells. Each cluster is a set of cells. A feature vector is introduced to identify the edge and

X. D. Liu (✉)
Integrated Information System Technology Laboratory,
Institute of Software, Chinese Academy of Sciences, Beijing, China
e-mail: lxdfigo@163.com

X. D. Liu
University of Chinese Academy of Sciences, Beijing, China

C. W. Zheng
Integrated Information System Technology Laboratory,
Institute of Software, Chinese Academy of Sciences, Beijing, China

distinguish different features of the rendering space. It contains gradient, variance and position. During sampling stage, our algorithm adaptively samples the rendering space and progressively reassigns cells to clusters using the error estimation and feature vector. After sampling stage, each cluster has similar features inside. To reconstruct the final image, feature clusters are modeled by polynomial functions using regression analysis. The color of each pixel is computed by integrating these polynomial functions. Our algorithm has the following contributions:

The feature vector In order to distinguish the different features of the rendering space, a feature vector is introduced. Our feature vector contains affine invariant gradient, local variance and position. Affine invariant gradient is used to identify the discontinuity. Local variance is used to estimate the feature difference. Position is used to combine the cluster.

Feature cluster According to the discontinuous and smooth regions of the rendering space, our algorithm organizes the multidimensional rendering space into clusters. The cluster is a set of cells. The shape of our cluster is formed by the feature vector. Each cluster has similar features inside. Different clusters have different features. During the sampling stage, we progressively reassign the cells into clusters by adaptive sampling.

Curve reconstruction In order to generate smooth and high quality images, regression analysis is employed to represent the feature clusters. The light field in each cluster is modeled by polynomial functions. The final image is computed by integrating these polynomial functions.

2 Related work

Most physics-based photorealistic methods integrate the light field of the scene to generate high quality images. There are three efficient ways to improve this kind of rendering methods: adaptive sampling, space partitioning and multidimensional reconstruction.

Adaptive sampling Since aliasing in rendering has been discussed by Crow [5], many adaptive sampling methods are given to solve this major problem in Monte Carlo ray tracing. Local variance is used to adaptively sample the image space or multidimensional rendering space [6,7]. Lepage [8] gives an adaptive multidimensional integration. Szécsi et al. [9] introduce an adaptive sampling method for environment mapping. Because they only focus on local frequency, it is very expensive to generate high quality images. To adaptively sample the rendering space beyond the image space, frequency analysis methods are introduced. Wavelet is employed to capture the variance in non-image dimensions [10]. Fourier transform is used to render high quality special effects such as motion blur or complex shadows [11]. Most this kind of methods analyze only one or two dimensions, because the derivation of general spectrum equation is hard.

Space partitioning Detecting the edges of the rendering space has been researched for a long time. Early researchers focus on finding the visibility and shadow of the scenes [12]. Some methods store the discontinuity information of textures to avoid aliasing [13]. Recently, f-divergence is used in the rendering method to analyze space and judge if more samples are necessary [14]. Bala et al. [3] propose a method based on combining edges and points to render high quality images, which need extra information to identify the edges. Kd-tree structure is used to divide multidimensional space [4]. The kd-tree separates the feature into multidimensional nodes and assumes light field in each node is flat, which may cause aliasing.

Multidimensional reconstruction Early reconstruction methods treat the pixel as isotropic area. They use simple filters to blur the noise and synthesize the images which may cause terrible aliasing on the edges [15]. In order to avoid this kind of aliasing, many reconstruction techniques are proposed. Some methods synthesize high quality image due to additional sample information such as the geometry and speed of the objects [16,17]. But the additional information limits their generality. Some methods analyze multidimensional rendering space to generate high quality images [4], but most of these methods suffer the curse of dimensionality. Li et al. [18] introduce an unbiased estimator SURE to reduce noise in image space. Durand et al. [11] analyze the light transport in frequency domain using Fourier transform. According to their research, many anisotropic filters are given to render effects such as motion blur, depth of field or soft shadows.

3 Overview

Photorealistic images are synthesized by computing the light transport equation (LTE) in Monte Carlo ray tracing systems [19]. In order to render the effects such as motion blur and depth of field, each pixel is computed by integrating the radiance through a multidimensional rendering space. In this article, we separate the rendering space and consider the integration as the sum of many separated subspaces' integrations.

$$\begin{aligned} P(i,j) &= \sum_{\Omega^k \in \mathrm{Pixel}(i,j)} L_{\Omega^k} \\ &= \sum_{\Omega^k \in \mathrm{Pixel}(i,j)} \int_{\Omega^k} l(u_1, u_2, \ldots, u_n) \mathrm{d}u_1 \mathrm{d}u_2 \ldots \mathrm{d}u_n \end{aligned} \tag{1}$$

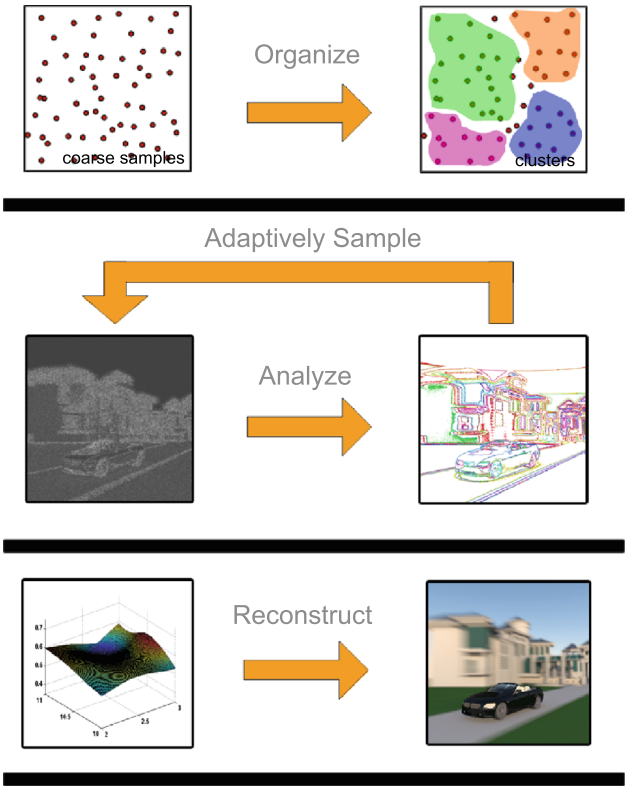

Fig. 1 The overview of our algorithm

The multidimensional rendering space is separated and organized into many clusters. L_{Ω^k} denotes the contribution of cluster Ω^k. l is the radiance function of the multidimensional cluster. To reconstruct the pixel value, polynomial function is used to evaluate the contribution of each cluster based on regression analysis.

The core idea of our algorithm is to organize the rendering space into reasonable parts and model each part by polynomial functions. Our algorithm has two stages: adaptively sampling based on cell clusters and reconstruction based on polynomial functions (Fig. 1). In order to separate and organize the rendering space, the sampling space is partitioned into a grid of cells. A cell is a cube or hypercube of the sampling space, its size is typically one pixel or quarter of a pixel. Each cell contains a feature vector. This vector contains the position of the cell, as well as the gradient and variance of the radiance in the local neighborhood of the cell. At the beginning of sampling stage, the rendering space is coarsely sampled. We compute the feature vector of each cell based on the coarse samples. The sampling space is organized into clusters by the feature vector. When a cluster is constructed, we estimate its error value. During the sampling stage, the cluster having the maximum error value is adaptively sampled. When a cell receives new samples, we recompute its feature vector and the cell may be reassigned to another cluster. Our algorithm repeatedly samples the scene until all the budget samples are used, typically we use 4–16 samples per pixel.

Table 1 The pseudo-code of our algorithm

```
function Render(P, S)
  // P is the rendering space
  // S is the sample budget
  InitClusters(P)
  {c} ← CoarseSample(t) // t is the coarse samples
  UpdateCluster({c})
  while using all the samples in S
    Ω ← SelectCluster()
    // Ω is the cluster having the maximum err
    {c} ← AdaptiveSampling(Ω , s)
    // we sample a certain samples for each time
    S ← S − s
    // update the sampled cells
    UpdateCluster({c})
  end
  Reconstruction(P)

function UpdateCluster({c})
  // {c} are the cells need to be updated
  // {u} are the changed cells
  // {Ω} are the changed clusters
  foreach cell c in {c}
    ComputeFeatureVector(c)
    {u} ← c
      foreach neighbor b around c
        ComputeFeatureVector(b)
        {u} ← b
    end
  end
  {Ω} ← ReassignCells({u})
  ComputeErrorValue({Ω})
```

After adaptive sampling, the rendering space has been divided into reasonable clusters. Because of the feature vector, each cluster has similar features, such as the similar material part of the object or the depth of field area of the same object. Smooth polynomial functions are introduced to represent the sampling space in each cluster. The contribution l of each cluster is modeled by polynomial functions. The samples are used to build the polynomial function based on regression analysis. The final image is generated by computing the light contribution in each cluster as shown in Eq. 1. The integrations of the cluster contribution are computed by the polynomial function. Table 1 shows the pseudocode of our algorithm.

4 The feature vector

In order to separate the different features from the rendering space, we identify a feature vector. Our algorithm assumes the rendering space is partitioned into a grid of

small equal cells. For example, if the rendering space is a two-dimensional image space, the cell is a square and the size of each cell is proportional to the pixel, like a quarter of pixel. Each cell stores a feature vector and the samples distributed in it. The feature vector represents the feature of the cell and it is used to build the cluster. It contains gradient, variance and position.

$$\mathbf{F} = \{g_{\text{aff}}, var, \mathbf{p}\} \tag{2}$$

The feature vector $\mathbf{F}$ is defined to represent the feature of different cells in the rendering space. It is constructed by the image affine invariant gradient g_{aff}, the local variance var and the cell's position $\mathbf{p}$. We use affine invariant gradient to identify the discontinuities, local variance to estimate the feature error and cell's position to combine the cluster. These values can all be computed from sample's contribution and position. These three parameters organize the cell reasonably and give a high quality result.

The key of cluster building is to find the edge of different features. In order to detect the boundaries of the sampling space's features, our algorithm is inspired by the active contour model [20]. We build the initial cluster using the coarse samples and progressively reassign the cells into reasonable clusters. We obtain the deformation by computing the feature vector. To detect edge in an affine invariant form, affine invariant gradient g_{aff} is introduced. It is useful to detect edges and corners in an image [21]. It computes the gradient in two-dimensional space. A gradient detector is performed to obtain the affine invariant contours. We first introduce the affine invariant gradient in image space, then we extend it into multidimensional space. In order to evaluate the gradient, two basic independent affine invariant descriptors H, J are calculated.

$$H_{xy} = I_{xx} I_{yy} - I_{xy}^2 \tag{3}$$

$$J_{xy} = I_{xx} I_y^2 - 2 I_x I_y I_{xy} + I_{yy} I_x^2 \tag{4}$$

Here, H_{xy} is an invariant descriptor of the image gradient which can be computed from light contribution I. J_{xy} is another invariant descriptor of the light gradient. They are used to identify the corners and edges in an affine invariant form. I_{xx}, I_{yy} and I_{xy} are the second-order derivatives in image space. I_x and I_y are the first-order derivatives. For example, I_x is the derivative of the light contribution in sampling space with respect to the x-axis. I_{xy} is the derivative of the I_x function with respect to the y-axis. The equations of I_x and I_{xy} are shown below:

$$I_x(i, j) = (\bar{I}(i+1, j) - \bar{I}(i-1, j))/2 \tag{5}$$

$$I_{xy}(i, j) = (I_x(i, j+1) - I_x(i, j-1))/2 \tag{6}$$

Here, $\bar{I}(i, j)$ denotes the light contribution in the sampling space. It is the mean sample value of the cell. i, j means the coordinates of the cell. The other derivatives are computed in the same way. After computing the invariant descriptors H and J, the affine invariant gradient in image space is evaluated as below:

$$g_{\text{aff}} = \sqrt{H_{xy}^2/(J_{xy}^2 + 1)} \tag{7}$$

Here, g_{aff} is the affine invariant gradient value in two-dimensional image space. It is computed by the two invariant descriptors. In order to render multidimensional space, we extend g_{aff} to a multidimensional gradient descriptor. The two basic affine invariant descriptors H, J are extended to multidimensional descriptors.

$$\bar{H} = \sum_{i=0, j=0, i \neq j}^{n} H_{ij} \qquad \bar{J} = \sum_{i=0, j=0, i \neq j}^{n} J_{ij} \tag{8}$$

They are computed by summing independent two-dimensional descriptors along every two dimensions. Here, n is the dimension of the rendering space. $\bar{H}$ and $\bar{J}$ give the basic independent affine invariant descriptors of the multidimensional rendering space. i, j are two different dimensions of the multidimensional space. H_{ij} and J_{ij} are the invariant descriptors of two-dimensional space. The g_{aff} in Eq. 7 is computed by replacing H_{xy}, J_{xy} by $\bar{H}$, $\bar{J}$.

This affine invariant gradient value helps to identify the rapid and flat changing regions in the rendering space and separates the space into different features. After computing the edge of different features, we use the following equation to compute the local variance of a certain cell.

$$\text{var} = \frac{1}{N-1} \sum_{i=0}^{N} (I_i - \bar{I})^2 \tag{9}$$

Here, I_i is the contribution of sample i in the cell. $\bar{I}$ is the mean value of these sample contributions in a cell. N is the number of samples in a certain cell. In order to assemble the cells in clusters, the feature vector needs the position information $\mathbf{p}$. The space of the position coordinate is normalized to [0, 1]. The feature vector controls the shape of the cluster. Affine invariant gradient and the local variance are computed from the sample contributions, the vector position is the cell location. Before sampling stage, each cell has an initial feature vector, typically it is a zero vector.

5 Adaptive Sampling

Our algorithm has two main stages: adaptive sampling and polynomial reconstruction. In sampling stage, the cells in rendering space are progressively reassigned into clusters. These clusters are built by using the feature vector. An error

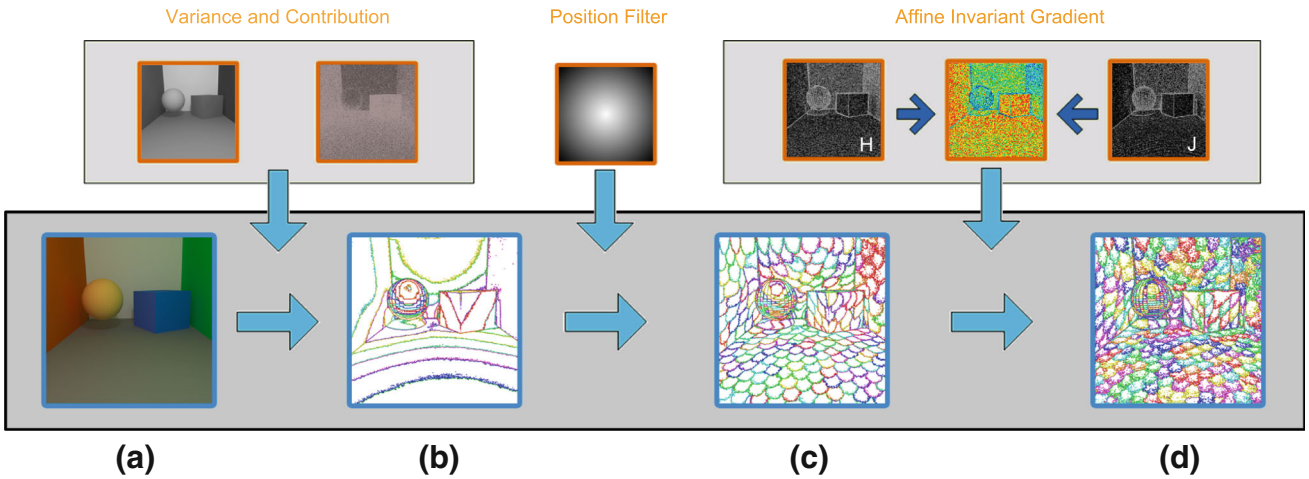

Fig. 2 The process of building the clusters. In order to separate different clusters in the rendering space, our algorithm considers different characters of the feature. **a** The rendering space. **b** The clusters which organized by only considering the variance values. **c** The space is separated into several small clusters by including the position vector. **d** The final result using affine invariant gradient. The gradient is computed from H and J invariant descriptors

estimation is used to adaptively sample the rendering space. After sampling stage, each cluster has similar features inside.

At the beginning, the feature vector of each cell is initialized to 0. In order to initially evaluate the rendering space, the rendering space is coarsely sampled using random strategy. We compute the feature vectors of the cells which receive new samples. Then, we build the initial clusters. The entire sampling space is one cluster. The feature vector is used to organize the cell of the rendering space. g_{aff} indicates the edges of the features. var implies the variance in sampling space. Its position term **p** controls that each cluster is continuous. These cells are organized into clusters by the following equation:

$$\Omega = \{C_i : \omega ||\mathbf{F}_i - \bar{F}|| \leq \text{err}\} \tag{10}$$

Each cluster Ω is composed of cells C_i and has a standard vector $\bar{F}$ which is the mean feature vector. If the distance between the feature vector $\mathbf{F}_i$ of cell C_i and the standard vector is less than a threshold, C_i is considered to be in cluster Ω, otherwise C_i is refused. $\omega \in (0, 1]$ controls the cluster's deformation. If ω is larger, the cluster will be smaller. Otherwise, the cluster will be larger. The cluster cannot overlap. The cell will assign to the cluster with the most similar standard feature vector around it (Fig. 2).

After initializing the clusters, to adaptively sample the rendering space and progressively reassign the cells, an error value is estimated from feature vector.

$$\text{err} = \frac{1}{N^2} \sum_{C_i, C_j \in \Omega} ||\mathbf{F}_i - \mathbf{F}_j|| \tag{11}$$

The differences of feature between every two cells are used to estimate the error value err. err is defined by the mean value of all the differences. N is the count of cells in cluster Ω. The error value is used for adaptive sampling.

After computing the error value of each cluster, the cluster having the maximum error value is randomly distributed a certain number of samples, typically is 4–16. When a cell receives new samples, the feature vectors of this sampled cell and only its neighbors are need to be recomputed. Our algorithm reorganizes these cells by their new feature vectors. According to Eq. 10, each cell is assigned an error value. If one cell is refused by one cluster, it is reassigned to another cluster around it. If it does not belong to any clusters, it builds a new cluster. After the reorganization, the error value of the modified clusters is recomputed. Based on the adaptive sampling stage, the cluster is progressively deformed. The sampling stage will not stop until all the budget samples are used.

6 Curve reconstruction

After adaptive sampling, each cluster should have similar features inside. It means the light field varies slowly in each cluster. The light contribution of one cluster can be evaluated by a smooth multidimensional function. We assume that the contribution in the cluster changes independently along each dimension. The multidimensional contribution function can be evaluated by many smooth polynomial functions for each rendering space dimension.

$$l(u_1, u_2, \ldots, u_n) \approx \frac{1}{n}(f_1(u_1) + f_2(u_2) + \cdots + f_n(u_n)) \tag{12}$$

Here, l is the contribution function. The count of rendering space dimensions is n. $u_1, u_2, \ldots, u_n$ are the position of multidimensional space. $f_1, f_2 \ldots f_n$ are the polynomial

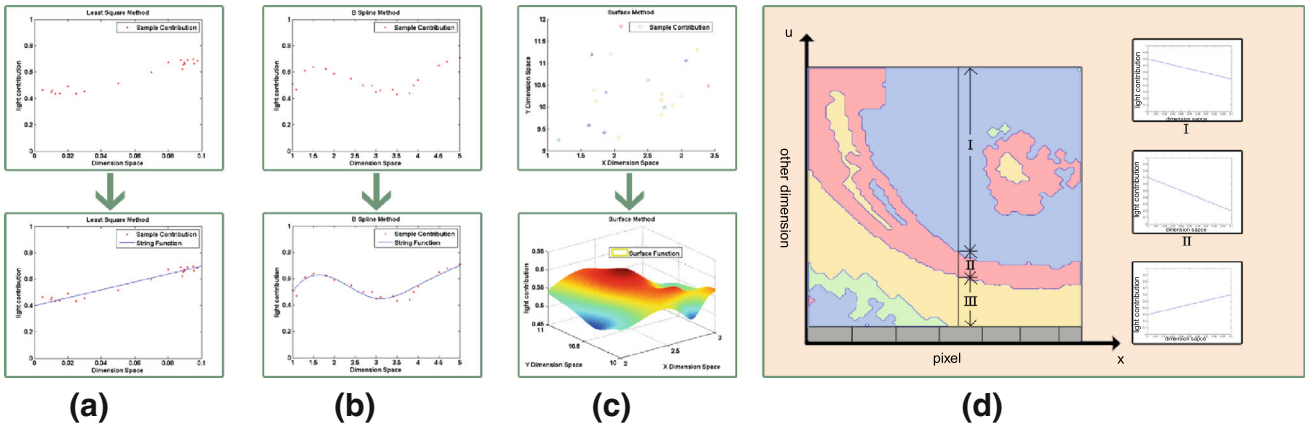

Fig. 3 In order to compute the smooth integration efficiently, different polynomial functions are used. **a** The linear function computed by the least-square method. **b** The polynomial function built by the cell contributions. **c** The curve surface which is built by polynomial and linear functions. **d** Algorithm that computes the pixel value by integrating the polynomial functions along the cluster, the three figures on the *left* are the regression functions in different clusters according to the pixel position

functions along each dimension. According to these functions, we can compute the cluster's contribution as below:

$$L_\Omega \approx \frac{1}{n} \int_\Omega f_1(u_1) + f_2(u_2) + \cdots + f_n(u_n) du_1 \ldots du_n \quad (13)$$

Because the polynomial functions $f_1, f_2, \ldots, f_n$ are independent, the cluster contribution can be combined by integrating each function independently.

$$P(i,j) \approx \frac{1}{n} \sum_{d=1}^{n} \sum_{\Omega_{\text{sub}} \in P(i,j)} F^d_{\text{sub}} \Big|_{\Omega^{\min}_{\text{sub},d}}^{\Omega^{\max}_{\text{sub},d}} \quad (14)$$

Here, F is the integrated function of f in Eq. 13. Ω_{sub} is the parts of clusters which only covers the area of pixel $P(i,j)$. F^d_{sub} is the integrated function of cluster Ω_{sub} along dimension d. $\Omega^{\max}_{\text{sub},d}$ and $\Omega^{\min}_{\text{sub},d}$ are the maximum and minimum value in dimension d of cluster Ω_{sub}. The pixel value of final image in Eq. 1 can be calculated. In order to generate the final image, each pixel is computed by summing all the cluster contribution in its space. And the clusters are modeled by polynomial functions. The contribution of each pixel is computed by integrating these polynomial functions (as shown in Fig. 3).

6.1 Least-squares estimation

The light contribution in the cluster is modeled by many polynomial functions along each dimension. Because clusters are organized by similar features of the rendering space, the light contribution of each cluster can be approximated by simple polynomial functions along each dimension. These functions are continuous, smooth and integrable. Due to these characteristics, we use regression analysis to estimate the them. Least-squares estimation is employed in our algorithm. The function f in Eq. 13 is shown below:

$$f(x) = \sum_{i=0}^{n} a_i x^i \quad (15)$$

The polynomial function $f(x)$ has $n+1$ elements. a_i is the ith coefficient of the function. x^i is the ith element. The sparse samples in each cluster are used for fitting. To calculate the coefficient a_i, we use the following equation.

$$\begin{aligned}
\sum_{j=0}^{m} c(x_j) f(x_j) &= \sum_{i=0}^{n} \sum_{j=0}^{m} a_i c(x_j) x_j^i \\
\sum_{j=0}^{m} c(x_j) f(x_j) x_j &= \sum_{i=0}^{n} \sum_{j=0}^{m} a_i c(x_j) x_j^{i+1} \\
&\cdots \\
\sum_{j=0}^{m} c(x_j) f(x_j) x_j^n &= \sum_{i=0}^{n} \sum_{j=0}^{m} a_i c(x_j) x_j^{i+n}
\end{aligned} \quad (16)$$

Here, n indicates the number of elements in $f(x)$. m is the number of x value used to build the function. $c(x)$ is the weight of x. According to the least-squares estimation, we use Eq. 16 to compute coefficient a_i in Eq. 16. We use these n equations to compute a_i. Because of the division strategy,

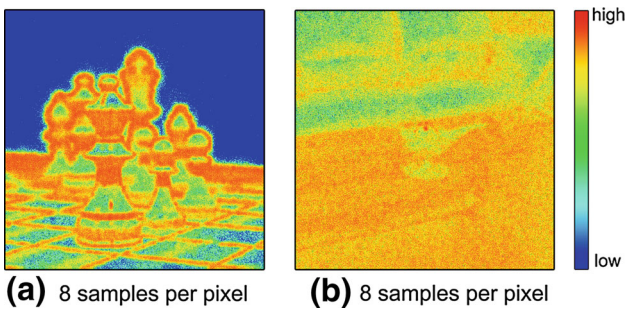

Fig. 4 The sample distributions of chess scene and magic lamp scene using our algorithm

the feature in each cluster is smooth and simple. We use linear function or parabola function to model the cluster. Linear function has two elements ($n = 1$), and parabola function has three elements($n = 2$). They can be easily computed and integrated in constant time. In implementation, the number m is all the cell position along each dimension. x is the contribution value of the cell. The weight $c(x)$ is the count of ray tracing samples in cell x.

7 Results and discussion

Our algorithm and previous methods are implemented in LuxRender 1.0 [22]. All results are rendered on a Intel Core i7 CPU at 2.8 GHz with 2 GB RAM. First, we analyze the sample distribution in our algorithm. Second, we compare the convergence rate using different parameters and analyze the polynomial function. Third, our algorithm is compared with the previous methods such as adaptive wavelet

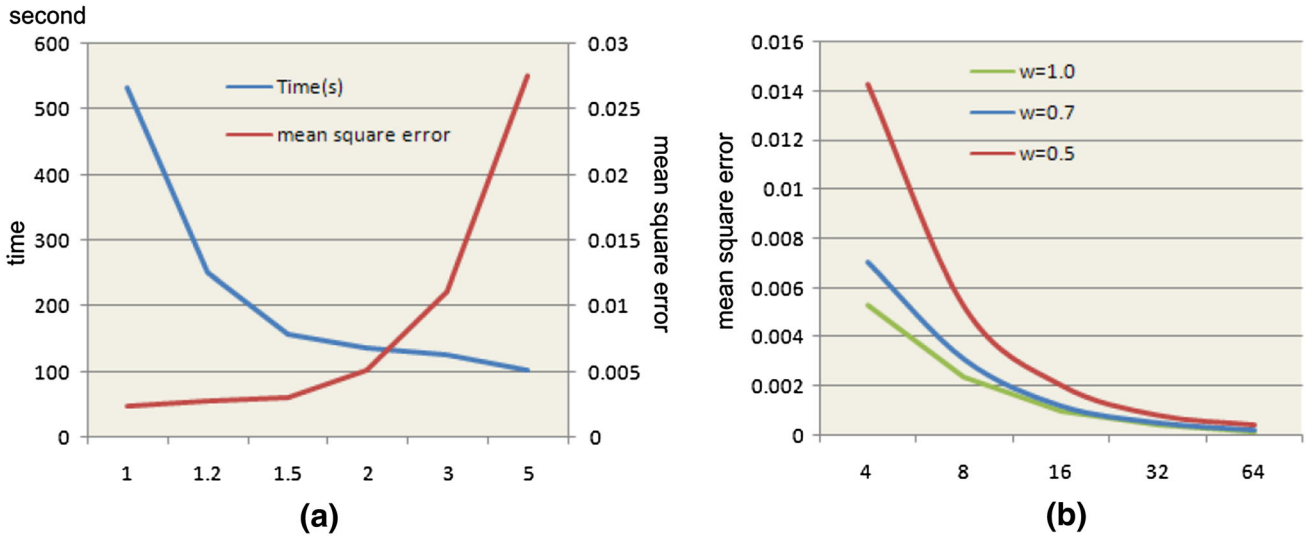

Fig. 5 The analysis of parameter ω in our algorithm. **a** The time and mean square error using different parameter values. **b** The mean square error using different values of ω with different samples per pixel

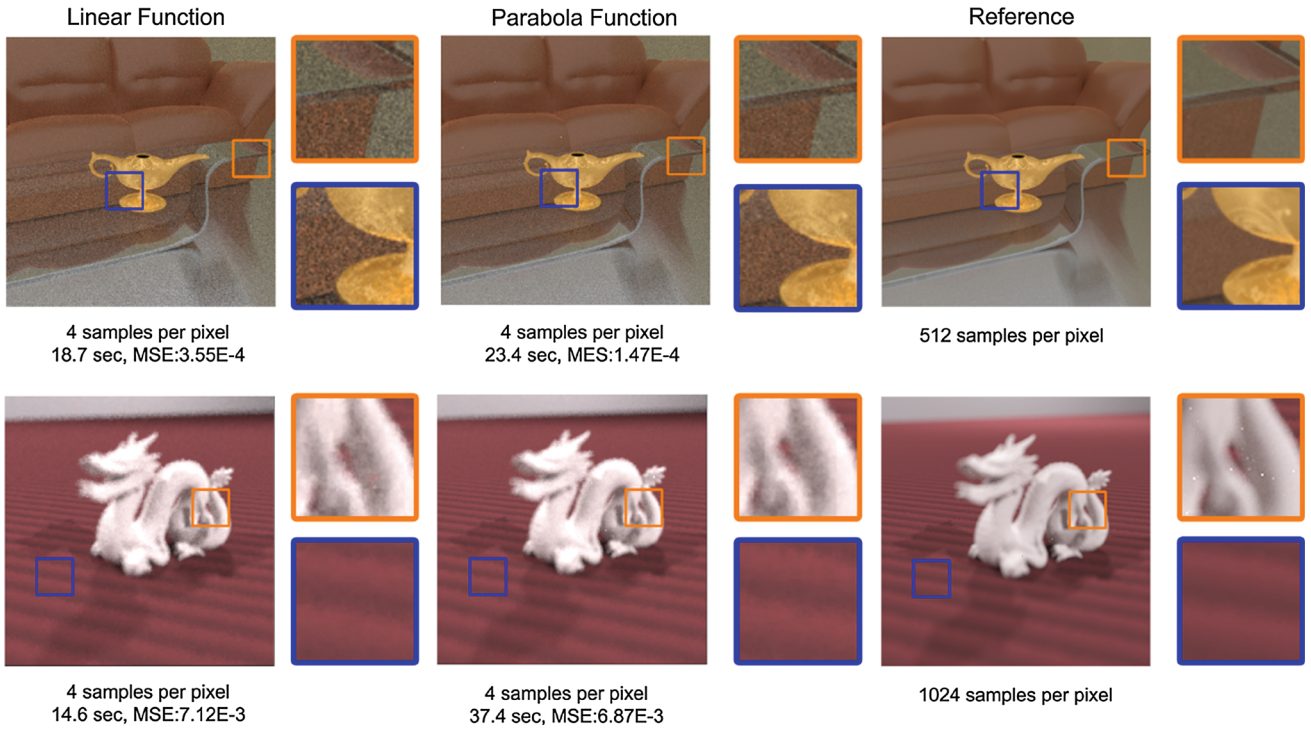

Fig. 6 The images rendered by our algorithm using different polynomial functions

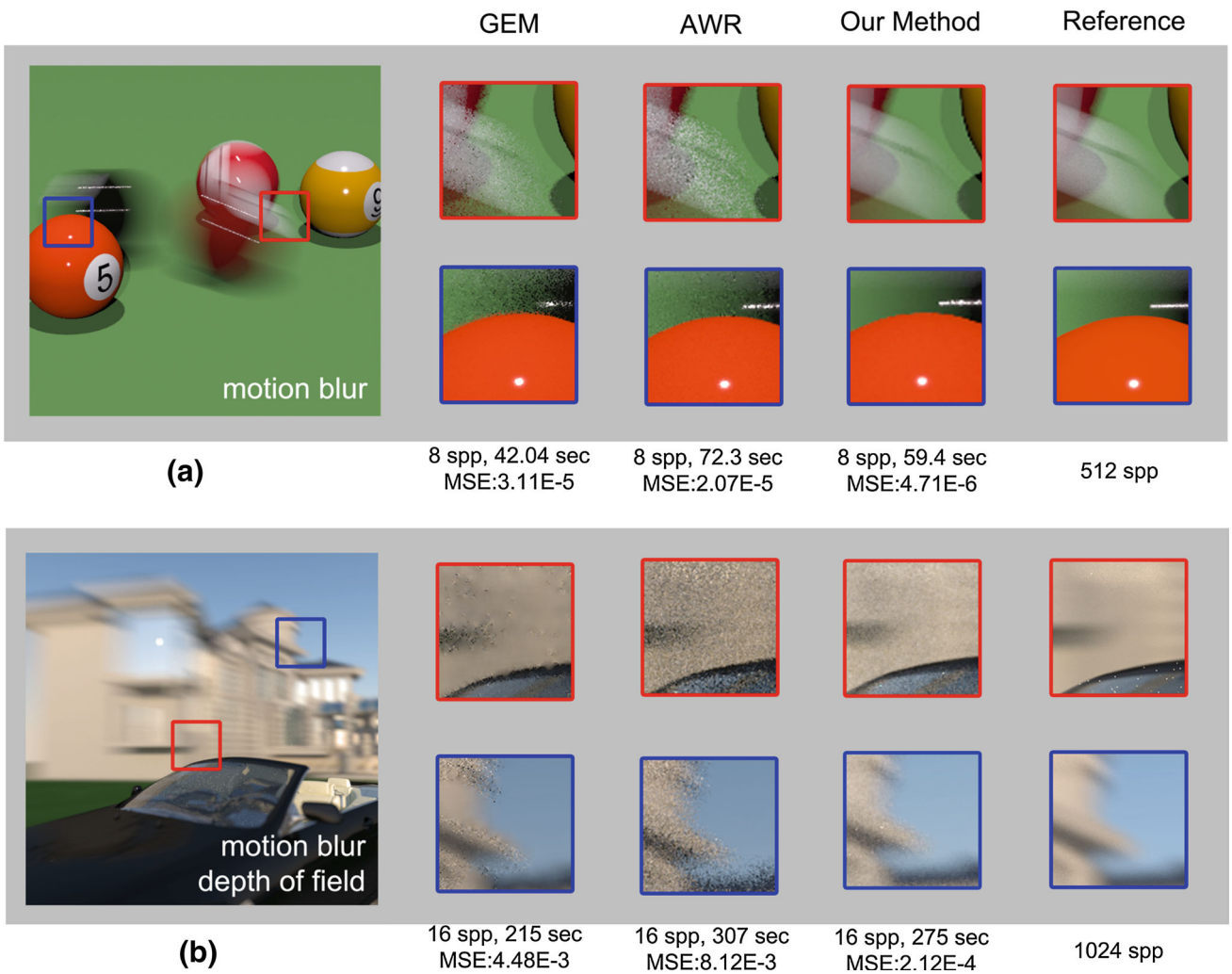

Fig. 7 **a** A 3D scene with motion blur effect. **b** A 5D scene with depth of field and motion blur effects. Our algorithm generates better images than the previous image space adaptive rendering methods

rendering (AWR) [10], greedy error minimization method (GEM) [23] and the multidimensional adaptive rendering method (MDAS) [4].

7.1 Sampling distribution

Figure 4 shows the sample distribution of chess scene and magic lamp scene rendered by our algorithm. Both images use eight samples per pixel. Our algorithm adaptively samples the cluster of the rendering space whose error value is higher. The sample distribution of chess scene shows that our algorithm can focus on the edge of the object and the depth of field regions. The sample distribution of the magic lamp scene shows that in global illumination scene, the samples are smooth because they are receiving indirect illumination. The sample distribution also indicates the cluster in the rendering space. Low sample density means low density clusters. High sample density means high density clusters.

7.2 Convergence analysis

We analyze the convergence rate of our algorithm by rendering the same scene with different ω parameter in Eq. 10. We render the magic lamp scene in 512×512 resolutions (with 8 samples per pixel in Fig. 5a). According to the threshold in Eq. 10, different values of ω give different clusters during the sampling stage. ω should be equal or greater than 1. Figure 5a shows that when ω is high, the mean square error value is high and the time consumption is low. This experiment shows that when ω is 0.7, the algorithm gives a high quality result and the time consumption is acceptable. Figure 5b shows the convergence rate of our algorithm using different ω. The mean square error shows that when ω is small, the convergence rate is fast.

7.3 Comparing curve functions

Our reconstruction strategy is analyzed by rendering the magical lamp scene and the dragon scene using different polynomial functions. All images are rendered with four samples per pixel at resolution 512 × 512. Both scenes are reconstructed using linear functions or parabolic functions. The magical lamp scene is rendered only in image space. Compared with the reference image, the reconstruction method using parabolic function gives a higher quality result. The dragon scene is rendered in four-dimensional space including image and lens dimensions. The results show that the image using parabolic function is as similar to the reference images as the one using linear function, but the parabolic function takes much more time. If we adaptively render the image space, the features in sampling space are complex. The parabolic function gives better results. If we adaptively render the multidimensional space, the features along each dimension is simple. The results using these two kinds of functions are similar (Fig. 6).

7.4 Results

Our algorithm can be used in both image space rendering and multidimensional space rendering. When the number of dimensions increases, the computing time and memory consumptions of the feature vector and polynomial functions also increase. Therefore, we use different strategies in different dimensional rendering spaces. In image space rendering, we use parabola in reconstruction. In multidimensional rendering, we use linear function in reconstruction.

In image space rendering, we compare our algorithm with previous methods such as AWR and GEM. Figure 7a shows a billiard scene with motion blur effect. This scene contains different color balls. Two of them are moving and one ball is rotating. The resolution of these rendered images is 1,024 × 1,024. Each image uses eight samples per pixel. GEM method uses multiscale filters to smooth the image, it filters a better result than AWR on the region of motion blur. AWR method uses local frequency information to sample and remove the noise, it reconstructs smooth edges in which GEM has aliasing. Compared with these two methods, our method achieves near reference quality image. Figure 7b shows an outside car scene with depth of field and motion blur effect. There is a moving car in the front and the camera is moving with the car. The focus is on center of the car. The resolution of the rendered images is 1,024 × 1,024. Each method uses 16 samples per pixel. GEM method filters the blur based on minimizing the local numerical error in square regions, it has a better result of blur regions than AWR method. AWR reconstruct higher quality edges than GEM. Our algorithm reconstructs image by clusters due to the features. It renders a better result than these previous methods in both the motion blur and depth of field blur regions. The images and MSE values of the two scenes show that our algorithm gives high quality images in image space rendering.

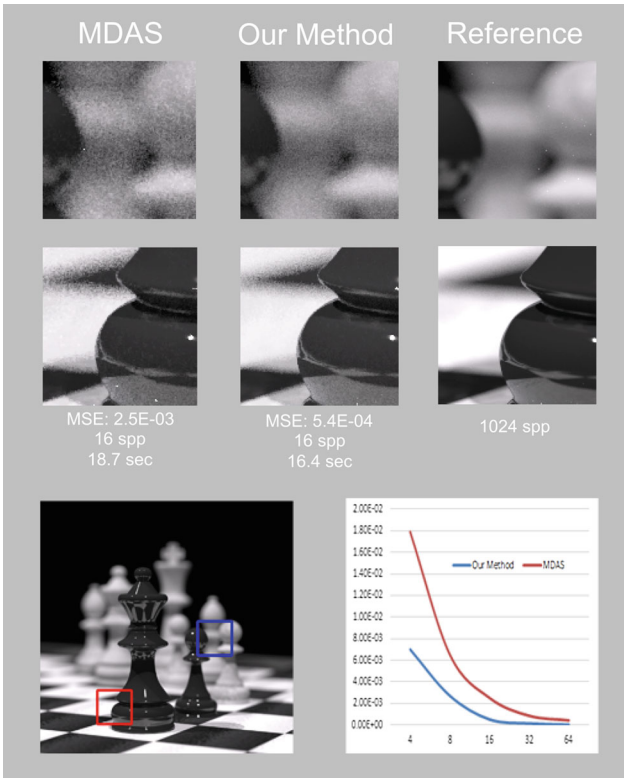

Fig. 8 We compare our algorithm with the MDAS in a 4D scene with depth of field effect

In multidimensional space rendering, we compare our algorithm with MDAS method. Figure 8 is a chess scene with depth of field effect. This scene is combined with white and black chessmen. The focus of the camera is on the black bishop in the center. The resolution of the rendered images is 1,024 × 1,024. Each image uses 16 samples per pixel. Because of the polynomial reconstruction, our algorithm gives a smoother result in depth of field area than MDAS. The visual images and MSE show that our algorithm renders a high quality result in multidimensional rendering.

7.5 Limitations

Our algorithm has some limitations. First, we tile the rendering space to cells. It limits the precision of the rendering. Second, similar to most multidimensional methods, our algorithm suffers the curse of dimensionality. The integrands of the polynomial functions should be simple when the number of dimensions increases. In implementation, we only consider the time and lens dimensions. Third, we only use a few

easy integrated functions to model the cluster. There are still some special features that are hard to be represented by these functions.

8 Conclusions

In this article, we propose a novel adaptive rendering method. A feature vector which contains affine invariant gradient, variance and position is introduced to organize the rendering space into clusters. These clusters are used to adaptively sample the scene. After sampling, each cluster has similar features inside. In order to reconstruct the rendering space, each cluster is modeled by smooth polynomial functions. We integrate these functions to generate the final image. Our algorithm gives higher quality images in both visual quality and numerical error than previous methods.

References

1. Mitchell, D.P.: Generating antialiased images at low sampling densities. Computer Graphics Proceedings. Annual Conference Series, ACM SIGGRAPH, vol. 21, pp. 65–72. ACM, Anaheim (1987)
2. Clarberg, P., Jarosz, W., Akenine-Möller, T., Jensen, H.W.: Wavelet importance sampling: efficiently evaluating products of complex functions. In: Proceedings of ACM SIGGRAPH 2005. ACM Press, Los Angeles (2005). http://graphics.ucsd.edu/papers/wis/
3. Bala, K., Walter, B., Greenberg, D.P.: Combining edges and points for interactive high-quality rendering. ACM Trans. Graph. **22**(3), 631–640 (2003). http://doi.acm.org/10.1145/882262.882318
4. Hachisuka, T., Jarosz, W., Weistroffer, R.P., Dale, K.: Multidimensional adaptive sampling and reconstruction for ray tracing. ACM Trans Graph (Proceedings of the SIGGRAPH Conference) **27**(3), 33:1–33:10 (2008)
5. Crow, F.: The aliasing problem in computer-generated shaded images. Commun. ACM **11**, 799–805 (1977)
6. Mitchell, D.P.: Spectrally optimal sampling for distribution ray tracing. In: Computer Graphics Proceedings. Annual Conference Series, ACM SIGGRAPH, vol. 25, pp. 157–164. ACM, Las Vegas (1991)
7. Liu, X.D., Wu, J.Z., Zheng, C.W.: Kd-tree based parallel adaptive rendering. Vis. Comput. **28**(6–8), 613–623 (2012)
8. Lepage, G.P.: An Adaptive Multidimensional Integration Program. Cornell University, NY, CLNS-80/447 (1980)
9. Szécsi, L., Szirmay-Kalos, L., Kurt, M., Csébfalvi, B.: Adaptive sampling for environment mapping. In: Proceedings of the 26th Spring Conference on Computer Graphics, pp. 69–76. ACM, New York (2010). http://doi.acm.org/10.1145/1925059.1925073
10. Overbeck, R.S., Donner, C., Ramamoorthi, R.: Adaptive wavelet rendering. ACM Trans. Graph. (Proceedings of the ACM SIGGRAPH Asia Conference) **28**(5), 1–12 (2009)
11. Durand, F., Holzschuch, N., Soler, C., Chan, E., Sillion, F.X.: A frequency analysis of light transport. ACM Trans. Graph. **24**(3), 1115–1126 (2005). http://doi.acm.org/10.1145/1073204.1073320
12. Durand, F.: 3D Visibility: analytical study and applications. PhD thesis, Grenoble University (1999). http://www-imagis.imag.fr
13. Sen, P.: Silhouette Maps for Improved Texture Magnification. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, pp. 65–73. ACM, New York (2004). http://doi.acm.org/10.1145/1058129.1058139
14. Rigau, J., Feixas, M., Sbert, M.: Refinement criteria based on F-divergences. In: Proceedings of the 14th Eurographics Workshop on Rendering, pp. 260–269. Eurographics Association, Switzerland (2003). http://dl.acm.org/citation.cfm?id=882404.882442
15. Gamito, M.N., Maddock, S.C.: Accurate multidimensional Poisson-disk sampling. ACM Trans. Graph. **29**(1) (2009)
16. Sen, P., Darabi, S.: On filtering the noise from the random parameters in Monte Carlo rendering. ACM Trans. Graph. **31**(3), 1–15 (2012). http://doi.acm.org/10.1145/2167076.2167083
17. Lehtinen, J., Aila, T., Chen, J., Laine, S., Durand, F.: Temporal light field reconstruction for rendering distribution effects. ACM Trans. Graph. **30**(4) (2011)
18. Li, T.M., Wu, Y.T., Chuang, Y.Y.: Sure-based optimization for adaptive sampling and reconstruction. ACM Trans. Graph. (Proceedings of ACM SIGGRAPH, Asia 2012) **31**(6), 186:1–186:9 (2012)
19. Kajiya, J.T.: The rendering equation. Comput. Graph. (Proceedings of ACM SIGGRAPH 86) 143–150 (1986)
20. Kass, M., Witkin, A., Terzopoulos, D.: Snakes: active contour models. Int. J. Comput. Vis. **1**, 321–331 (1988)
21. Sapiro, G.: Geometric Partial Differential Equations and Image Analysis. Cambridge University Press, New York (2006)
22. http://www.luxrender.net/ (2008)
23. Rousselle, F., Knaus, C., Zwicker, M.: Adaptive sampling and reconstruction using greedy error minimization. ACM Trans. Graph. (Proceedings of the SIGGRAPH Asia Conference) **5**(3), 1–10 (2011)

Xiao Dan Liu is a Ph.D. candidate in the National Key Laboratory of Integrated Information System Technology, Institute of Software, Chinese Academy of Sciences. He received his B.Sc. degree from Xiamen University in 2009. His research interests include computer graphics, realistic rendering, game developing and computer simulation.

Chang Wen Zheng is a Professor in the National Key Laboratory of Integrated Information System Technology, Institute of Software, Chinese Academy of Sciences. He received his Ph.D. degree from Huazhong University of Science and Technology. His research interests include computer graphics, virtual reality, computer simulation and artificial intelligence.