Real-Time Camera Walks Using Light Fields

Biswarup Choudhury, Deepali Singla, and Sharat Chandran

Indian Institute of Technology Bombay
http://www.cse.iitb.ac.in/∼{biswarup, deepali, sharat}

Abstract. An interesting alternative to traditional geometry based rendering is Light Field Rendering [1,2]. A camera gantry is used to acquire authentic imagery and detailed novel views are synthetically generated from unknown viewpoints. The drawback is the significant data on disk.

Moving from static images, a walkthrough or a *camera walk* through the implied virtual world is often desirable but the repeated access of the large data makes the task increasingly difficult. We note that although potentially infinite walkthroughs are possible, for any given path, only a subset of the previously stored light field is required. Our prior work [3] exploited this and reduced the main memory requirement. However, considerable computational burden is encountered in processing even this reduced subset. This negatively impacts real-time rendering.

In this paper, we subdivide the image projection plane into "cells," each of which gets all its radiance information from the cached portions of the light field at select "nodal points." Once these cells are defined, the cache is visited *systematically* to find the radiance efficiently. The net result is *real-time* camera walks.

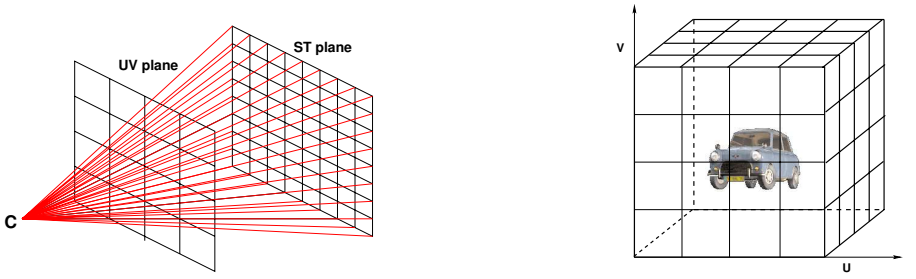
1 Introduction

In contrast with traditional geometry based rendering, a somewhat recent approach for "flying" through scenes is Image-Based Rendering (IBR) ([4], [5], [6]) which uses a confluence of methods from computer graphics and vision. The IBR approach is to generate novel views from virtual camera locations from pre-acquired imagery ([7], [8]). Synthetic realism is achieved, so to speak, using real cameras.

Light Field Rendering [1] (or Lumigraphs [2],[9],[10]) is an example of IBR. The approach is to store samples of the plenoptic function [11] which describe the directional radiance distribution for every point in space. The subset of this function in an occlusion-free space outside the scene can be represented in the form of a four-dimensional function. The parameterization scheme is shown in Fig. 1(a). Every viewing ray, computed using a *ray-shooting* technique, from the novel camera location C passing through the scene is characterized by a pair of points (s, t) and (u, v) on two planes. By accessing the previously acquired radiance associated with this four tuple, the view from C is generated.

In the general case, C can be anywhere in three-dimensions. So six light slabs are combined so that the entire scene is covered (Fig. 1(b)). An unfortunate consequence of this scheme is the huge datasize of the lightfield. The authors in

P. Kalra and S. Peleg (Eds.): ICVGIP 2006, LNCS 4338, pp. 321–332, 2006.



(a) Two-Plane Parametrization.

(b) Complete Lightfield: Six slabs of two-plane lightfields.

Fig. 1. The light field is a description of all light rays in a region of interest

[12] propose an interactive rendering based on Lumigraphs [2], by either using a smaller set of textures (with compromise in quality) or by storing the reconstructed image as a new texture for subsequent nearby images (with additional geometric information).

1.1 Problem Statement and Contributions

The beauty of image based rendering lies in re-sampling and combining the pre-acquired imagery. In a typical walkthrough situation, a person is expected to walk along a trajectory in three space and "suitably" sample the input images (in our case, light field). The problem we pose in this paper is *"Given the huge light field on disk, and a camera walk, how efficiently can the scene as seen by the camera be rendered?"*

In interactive walkthroughs, light field rendering is impacted by the sampling density of acquired images. For example, lightfield generated in [1] is sampled at 0.125 units; for a 512x512 image, the size of the lightfield is about 4.8GB. With increase in the resolution and density of acquired images, the size of the light field increases dramatically. For *interactive* rendering of the scene, one needs to store the complete light field in volatile memory, and perform computationally heavy [13] ray shooting operations.

Earlier in [3], we observed that for a camera walk, only a subset of the complete lightfield is needed. We computed the *optimal* location of a sparse set of "nodal points," suitable for the camera walk. The *lightweight* light field stored at these nodal points is enough to render the scene from any of the infinite points — termed *query* points — on the camera path. The advantage of this was that at the time of camera walk, accesses to the hard disk were reduced or absent. However, considerable computational burden was encountered in processing the input light field to obtain this subset. In addition, the number of ray shooting operations required for rendering an image from a query point on the camera walk was a function of the resolution of the rendered image size. Thus, rendering time increases considerably with increase in image size. In this paper, we show that efficiently caching the subset of light field, appropriate for the camera walk,

and further *dividing the image plane into "cells"* results in rendering an image from a query point in real-time. Specifically,

1. We partition the image plane into "cells," each of which gets all the information (radiance values) from a specific nodal point, thereby avoiding the necessity to perform ray-plane intersections. Further, we show that for unknown, on-line camera walks, the nodal points once used can be discarded paving the way for memory efficient real-time implementation.
2. The correctness of our scheme is shown using a mathematical characterization of the geometry of the light field.
3. A new light field dataset, using a Mini Cooper car model, has been generated and experiments have been performed on it. Results validate our technique.

2 Our Approach

As in the original work [1], the field of view of the query camera is expected to be identical to the cameras that generated the light field. Likewise, sheared perspective projection handles the problem of aligning the plane of projection with the lightfield-slab. The center of projection of the camera moves along a plane parallel to the UV and the ST plane. For brevity, consider a setup similar to the **two slab setup** (Fig. 1(a)) where planes, UV and ST are replaced by lines U and S. We call this as the **two-line setup** (Fig. 2(a)). The query points q lie on line C, which in turn replaces the camera plane. As in [1], nearest neighbor approximation is employed for determining the radiance corresponding to q. We provide the complete mathematical framework with respect to this setup.

The rest of this paper is organized as follows. Section 2.1 and Section 2.2 summarize [3] for coherence (If proofs are not desired, this paper is self-contained). Section 2.3 develops the mathematical framework for our new algorithm. In Section 3 and Section 4, we give details of our approach and present our algorithm. Experimental results and analysis are discussed in Section 5. Finally in Section 6, we provide our concluding remarks.

2.1 Fixed-Direction Algorithm

In this section, we provide a brief summary of the concept of nodal points for a query point q. Later, we use these concepts in the mathematical characterization of the rest of the paper.

Denote Δl to be the constant distance $d[G_i, G_{i+1}]$ between two consecutive grid points on the U line, i.e., the distance between the input lightfield camera locations. For a specific s, denote `assoc(`q`)`, where q is a point on C, to be the closest grid vertex G (on U) to the ray $\overline{qs}$. In Fig. 2(a), `assoc(`q`)` $= G_1$. Given q, we use Algorithm 2.1 to compute nodal points N_1 and N_2. The radiance $L[q]$, in the direction of s, is obtained from these nodal points (presumably cached).

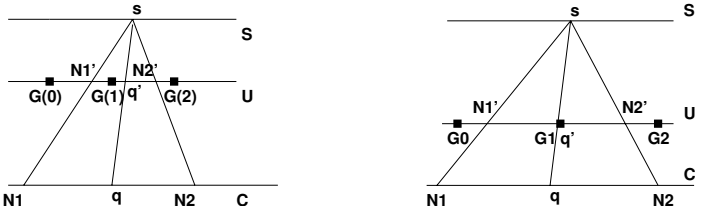


Fig. 2. (a): N_1 and N_2, the nodal points for q are marked such that $d[q'N_1{}'] = d[q'N_2{}'] = \frac{\Delta l}{2}$. (b): `assoc(`$N_1$`) is` G_0 `and assoc(`N_2`) is` G_2.

Algorithm 2.1. **Fixed-Direction** (q, s)

Shoot a ray from q to s to obtain q' on U. Mark points $N_1{}'$ and $N_2{}'$ on U at a distance $d = \frac{\Delta l}{2}$ apart on either side of q'. This determines the nodal points N_1 and N_2 on C.
if `assoc`$(N_1) == G_1$ **then**
  $L[q] = L[N_1]$
else
  $L[q] = L[N_2]$
end if

In the case of two-plane parametrization, given q, one may compute four nodal points N_1, N_2, N_3 and N_4. Shoot the ray from q to s for a given s to obtain q' on UV. Now, mark four points $(q'.u \pm \frac{\Delta l}{2}, q'.v \pm \frac{\Delta l}{2}, z_{uv})$, where $q'.u$ and $q'.v$ represent the component of q' along u and v respectively, and z_{uv} is the z coordinate of the UV plane. These four points correspond to four nodal points on the camera COP (center of projection) plane. We use `assoc` of these nodal points to determine $L[q]$.

Notice that if the distance d in Algorithm 2.1 is more than $\frac{\Delta l}{2}$, as in Fig. 2(b), an incorrect value of $L[q]$ is computed. When d is as specified in Algorithm 2.1, it is easy to observe that either `assoc(`N_1`)` $= G_1$ or `assoc(`N_2`)` $= G_1$; it cannot be the case that `assoc(`N_1`)` $= G_0$ and `assoc(`N_2`)` $= G_2$. A choice less than $\frac{\Delta l}{2}$ might be suitable to maintain correctness, but will increase the number of nodal points, and hence decrease our efficiency. *Also note that the* `if` *condition in* Algorithm 2.1 *cannot be dispensed with. We cannot simply pick the nearest nodal point.*

2.2 All-Directions Algorithm

Algorithm 2.1 is "backward" in that it computes nodal points given a query point; in a sense it appears useless. However using this algorithm as the basis, in [3] we proved that,

- The nodal points N_1, N_2 corresponding to a query point q are sufficient for determining the radiance of *any query point* in the interval $[N_1, N_2]$. This generalizes to three dimensions.
- Choice of nodal points is independent of the direction (of s).

Using the above results, we use Algorithm 2.2 to compute the radiance corresponding to any query point q in the interval $[N_1, N_2]$. For simplicity, the algorithm has been presented for the two line setup.

Algorithm 2.2. **All-Directions** (q)

```
Determine nodal points N1, N2 bounding q.
for all s ∈ S do
   Shoot a ray from query point q to s.
   if assoc(N1) == assoc(q) then
      L[q] = L[N1]
   else
      L[q] = L[N2]
   end if
end for
```

2.3 Image Plane Intervals

In Algorithm 2.2, the scene as rendered from a query point q is determined expensively by shooting N rays (to the N sample points on S), followed by a lookup of `assoc` for each of the N rays. This computational burden increases dramatically with the increase in query points on a camera walk. Expectedly, the situation for the two-plane setup is worse. (The number of shot rays are $N \times N$ for a query point.) With an independent increase in the number of query points, the computational requirements prohibit real-time rendering.

In this section, we show how to subdivide the image plane into cells and thereby derive a deterministic "square wave" pattern of using nodal points for each cell. For the sake of exposition, we consider the two-line setup wherein cells degenerate to intervals. The following lemma depends on *mid grid points*, which are defined as the points lying in the middle of any two adjacent grid points. For a query point q bounded by nodal points $[N_j, N_{j+1}]$, consider (respectively) rays from q, N_j and N_{j+1} through the mid grid points on U. These divide S into intervals $[S_i, S_{i+1}]$ (i is a whole number) (see, for example, Fig. 3(a)).

Lemma 1. Range Lemma: *The radiance values corresponding to all s points in an interval* $[S_i, S_{i+1}]$ *can be determined from a single nodal point.*

Proof: Without loss of generality, let the interval under consideration be $[S_2, S_3]$ (Fig. 3(a)). Let s_p be any point in the interval $[S_2, S_3]$. Observe that for s_p, `assoc`(q)=G_2. By construction, $\forall s \in [S_2, S_4]$, `assoc`(N_1)=G_2. Since $[S_2, S_3]$ is a subset of $[S_2, S_4]$, so $\forall s \in [S_2, S_3]$, `assoc`(N_1)=G_2. Thus, for s_p, `assoc`(q)=`assoc` (N_1) and therefore $L(q)$=$L(N_1)$. □

Thus, in general, we can avoid ray shooting for a range of s values in any interval since the radiance $L(q)$ will come from *some* fixed nodal point. The next lemma tells us that even the choice of nodal point is deterministic.

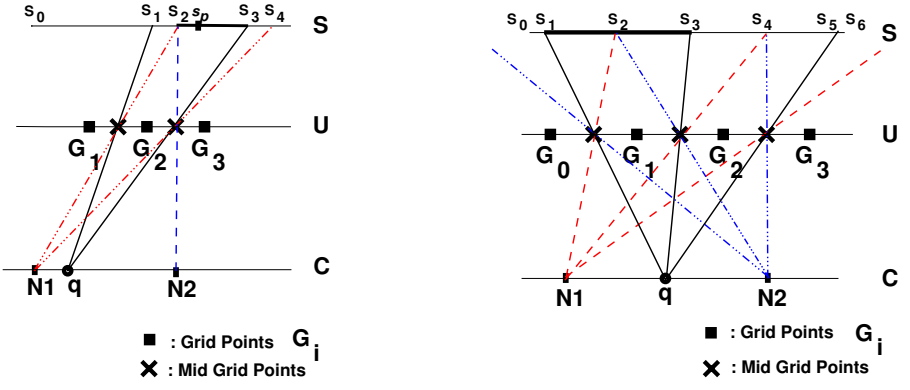


Fig. 3. (a): Radiance $L(q, s)$ for $\forall s \in [S_2, S_3]$ can be determined from N_1. (b): $\forall s \in [S_1, S_2]$, $L(q)=L(N_2)$; $\forall s \in [S_2, S_3]$, $L(q)=L(N_1)$.

Lemma 2. Toggle Lemma: *If* $L(q)=L[N_j]$ *for some* $s \in [S_i, S_{i+1}]$, *then* $\forall s \in [S_{i+1}, S_{i+2}]$, $L(q)=L[N_{j+1}]$ *and vice-versa.*

Proof: Without loss of generality, let $[S_1, S_2]$ and $[S_2, S_3]$ be the two intervals under consideration (Fig. 3(b)). By construction, we observe that $\forall s \in [S_1, S_2]$, $\texttt{assoc}(q)=G_1$ and $\texttt{assoc}(N_2)=G_1$. So $\forall s \in [S_1, S_2]$, $L(q)=L[N_2]$. The lemma claims that $\forall s \in [S_2, S_3]$, $L(q)=L[N_1]$.

By construction, $\texttt{assoc}(q)=G_1, \forall s \in [S_1, S_3]$. Also, $\texttt{assoc}(N_1)=G_1, \forall s \in [S_2, S_4]$. The intersection of intervals $[S_1, S_3]$ and $[S_2, S_4]$ is $[S_2, S_3]$. Hence, $\forall s \in [S_2, S_3]$, $\texttt{assoc}(q)=\texttt{assoc}(N1)$, or, $L(q)=L[N_1]$.

The situation when we consider the intervals $[S_2, S_3]$ and $[S_3, S_4]$ is similar; we find that $\forall s \in [S_3, S_4]$, $L(q)=L[N_2]$. □

Thus the lemma asserts that, for a query point, the radiance corresponding to each interval in S, is deterministic as a toggle between bounding nodal points. This is best visualized as a square wave (Fig. 4) and is exploited in our algorithm (Algorithm 2.3).

Algorithm 2.3. **Interval Algorithm** (q)

Determine the nodal points N_j, N_{j+1}, bounding q.
Determine all intervals $[S_i, S_{i+1}]$ on S using q, N_j and N_{j+1}.
Shoot a ray from query point q to the first s in $[S_0, S_1]$.
`Toggle` = (`assoc`(N_j) == `assoc`(q)) ? N_j : N_{j+1}
for all intervals **do**
  for all $s \in [S_i, S_{i+1}]$ **do**
    $L[q]$ = L[`Toggle`]
  end for
  `Toggle` = (`Toggle`==N_j) ? N_{j+1} : N_j
end for

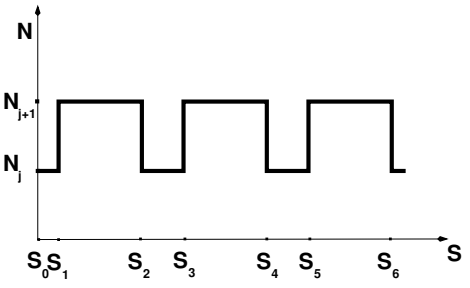


Fig. 4. Nodal points are accessed in a "toggle" manner for any query point. The duty cycle of the square wave is dependent on which query point is used. Size of the image plane determines the end conditions.

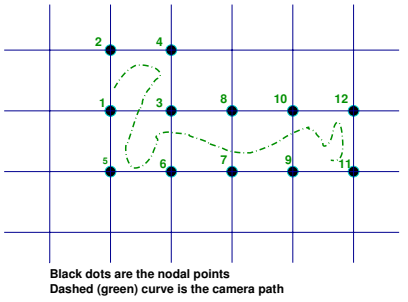
3 The Algorithm

We now have the apparatus to select the nodal points and divide the image plane into cells, given a query point. For the sake of exposition, we consider two types of cases (Fig. 5). In the first case, the input is an unrestricted camera walk and nodal points are computed on the fly (Fig. 5(a)). As we traverse along the camera walk, nodal points "used" can be discarded. In the second case, the first input is a domain, and nodal points are computed after the domain is given (Fig. 5(b)). The second input are camera walks restricted to be in the domain. In this case, multiple walks can be rendered efficiently without recomputing new nodal points, and in parallel.

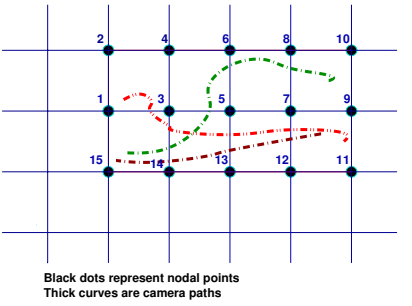
3.1 Case 1

Algorithm 3.1. Incremental-Camera Walk (`walk`)

1. Starting from the initial query point on the camera walk, mark four nodal points at a distance $\Delta x = \Delta l \times R$, where R is the ratio of the distance between the camera plane and the ST plane, and the distance between the UV and ST plane. For simplicity, the nodal points are selected parallel to the u and v directions as shown in Fig. 5(a). A cell thus is created.
2. The light field is cached at four nodal points in the grid enclosing the query point (The precise computation of the light field at the nodal points can take advantage of the methods suggested in Section 4, instead of the method in [1].)
3. Apply Algorithm 2.3 iteratively to calculate the radiance at all query points (along the camera walk) inside the cell.
4. As the walk exits the cell, update the nodal points and go to Step 2.



(a) Incremental nodal points along a camera walk.



(b) Domain-based nodal points.

Fig. 5. Rendered scene as viewed from a camerawalk can be computed from nodal points

3.2 Case 2

Next, if we are given several camera walks lying in a domain, we pick domain-based nodal points, as shown in Fig. 5(b). Scene from any query point, on any camera walk, or even at random, in the rectangular region defined by the bounding box of the nodal points can be rendered efficiently as shown below.

Algorithm 3.2. Domain-Camera Walk (`domain`)

1. Determine the bounding box of the domain specified.
2. Mark nodal points at a distance $\Delta x = \Delta l \times R$, where R is the ratio of the distance between the camera plane and the ST plane, and the distance between the UV and ST plane along the complete bounding box. For simplicity, the nodal points are selected parallel to the u and v directions as shown in Fig. 5(b). A grid is thus created.
3. The light field is cached at all the nodal points in the grid (The precise computation of the light field at the nodal points can take advantage of the methods suggested in Section 4, instead of the method in [1].)
4. Apply Algorithm 2.3 to calculate the radiance at any query point inside any cell of the grid.

In summary, the incremental algorithm is more suitable when the user does not want to specify the camera walk in advance. The domain-based algorithm, on the other hand, is useful when the user has a number of camera walks, or random query points in a domain.

4 Caching Radiance at Nodal Points

In Algorithm 2.2, the radiance computation for nodal points was done using the method in [1]. Caching of nodal points can be less expensive by using the ideas in

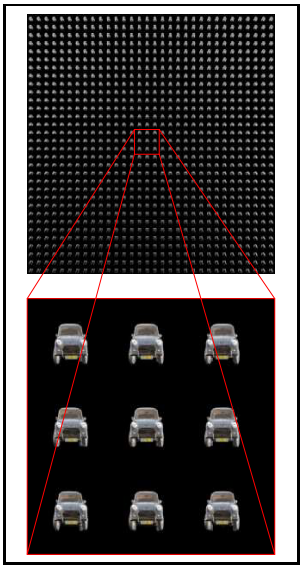


Fig. 6. Lightfield for the MiniCooper

Fig. 7. Rendered novel images (best seen in color)

Lemma 1, i.e., by dividing S into intervals (or the image plane ST into cells). The radiance values for a number of image pixels is computed from a fixed camera grid point. Further, the relationship between the intervals and the camera grid points can be easily determined. As a result, in our method for nodal points, ray shooting has to be done only *once* to find the intervals and their corresponding grid points.

5 Experiments and Results

In this section, we first describe the implementation details and then show the significant computational advantage. For baseline comparisons, and since it is a standard, and freely available, we use [1] to contrast our method. Various simulations on different paths confirm our claims to superiority.

Mini Cooper Dataset: We have generated a new lightfield dataset for purposes of our experimentation (Fig. 6). It consists of images of a Mini Cooper car model captured from cameras placed on a 32x32 grid. The Mini Cooper, a CSG model, was rendered by performing radiosity computation with 3 different light sources using Povray. Some features of the Mini Cooper include specular metal body, reflective window glasses, inner details of the car (car seats, rear-view mirror). Resolution of the images is 256 x 256.

Table 1. Time in seconds for different camera walks

|q|	[1]	§3.1
164	82.617	5.020
335	165.015	10.468
550	281.126	14.640
741	376.040	17.997
1023	521.605	24.417

Table 2. Number of disk accesses for camera walks on different planes

zCam	[1]	§3.1
10	160338	23244
20	211730	20640
30	222284	15360
40	231329	12512
50	238442	10240

Table 3. Time in seconds for camera walks on different planes

zCam	|p|	[1]	§3.1
10	170	317.424	30.349
20	86	413.178	27.141
30	60	432.255	20.717
40	46	444.092	17.173
50	40	462.085	14.764

Quality: We downloaded the reference implementation [1], obtained the input lightfield dataset after the decompression stage, and then "hooked" our modification. The rendered images (Fig. 7) using our method are identical to those generated in [1]. The output is devoid of any artifacts — `diff` under Gnu-Linux reports the null set.

Theoretical Analysis: The advantages of our technique arise due to efficient nodal light field caching, and division of the image plane into cells. For a path with q query points, let the number of nodal points needed be p. Note that for a camerawalk, the number of query points is much larger than the number of nodal points, i.e., $q \gg p$ (e.g., for $|q|$ =886, $|p|$=40 on a plane at **zCam**=50). Let the average number of grid points required for generating an image from a query point be g. We penalize disk access to the lightfield data (densely sampled and at a high resolution) by a factor of d.

In [1], the number of rays shot is of the order of the resolution of an image (NxN). Theoretically, the total time taken is $q(k_1N^2 + dg)$, where k_1 is the time taken by each ray shooting operation and corresponding computations. In our method, time taken for caching each nodal point is less than that taken by a query point in [1] (Section 4). For each nodal point, the initial computation of determining cells is constant (c_1) but again, the time taken for disk accesses is dg. For a query point, computation of cells takes constant time (c_2). So the theoretical computational gain is

$$\frac{q(k_1N^2 + dg)}{p(c_1 + dg) + qc_2} = \frac{\mathsf{O}(q(N^2 + dg))}{\mathsf{O}(pdg + q)} \tag{1}$$

Computational Advantage: All our experimentation was performed on an Intel Pentium IV 2.4GHz, 1 GB RAM Linux based computer. Our results confirm the theoretical computational advantage in Equation 1.

1. Table 1 depicts the results we obtained using different camera walks. The two techniques compared are [1] and Algorithm 3.1 (with caching of nodal points as in Section 4). The distance between two successive query points on the camera walk is constant for all the experiments. With increase in the number of query points, the rate of increase of time taken in [1] is more than

Table 4. Total time in seconds for loading light field into a domain and rendering from random query points. The rendering time after loading is nominal.

$\|q\|$	[1]	§3.2
64	29.949	12.35
130	61.294	12.587
212	99.255	12.682
300	140.166	13.019
464	215.86	13.44

Table 5. Number of disk accesses for random query points on a plane

$\|q\|$	[1]	§3.2
64	15987	9968
130	32229	9968
212	52400	9968
300	74360	9968
464	114707	9968

that of our method. Note that the number of frames rendered per second ranges from 30 to 40.

2. In Table 2 and Table 3, we show the results using camera walks on parallel planes at varying distances from the lightfield setup. The total number of query points was kept constant, in this case it happened to be 886. The computational gain increases with increase in the value of **zCam**, because the number of nodal points decrease with increase in **zCam**.
3. Table 4 and Table 5 compares the results of Algorithm 3.2 (with caching of nodal points as in Section 4) with [1]. The experiments have been performed on a fixed domain (42 nodal points) and with random number of query points. The rate of increase in the time taken by our algorithm, as the number of query points increases is very low, because most of the computational time is spent generating the (fixed number of) nodal point images. Rendering of images from the query points takes nominal computational time.
4. We also compared our method with the technique in our previous work [3] and observed a significant computational gain. For instance, on a camera walk with $|q|$=370 (and **zCam** ranging from 10 to 50), our method was on an average 10 orders of magnitude faster.

6 Conclusion

In this paper, we have looked at the problem of reducing the computational burden in dealing with the rich and densely sampled light field when a user walks through a virtual world. We have achieved this by recognizing that instead of considering the complete light field, it is enough to consider a sparse set of nodal points. We have proved that the division of the image plane into cells and thereafter, deriving a deterministic pattern of the use of the nodal points for each of these cells, has increased the computational efficiency significantly. The proofs of the mathematical characterizations of these concepts have been provided. A new lightfield dataset for purposes of experimentation has been generated and experimental results have been shown to validate our technique.

Our description does not explicitly deal with decompression issues (indeed, in the first stage [1] of rendering, the entire light field is decompressed as it is read

into memory from disk). However, there is no conceptual blockade in applying the general caching strategy and the mathematical elements even in these cases.

Acknowledgements. This work was funded by an Infosys Ph.D. fellowship grant. The base Light field code was downloaded from `graphics.stanford.edu` The Mini Cooper model was taken from `www.oyonale.com` and the lightfield generated with Povray. We thank the members of ViGIL, IIT Bombay for useful discussions.

References

1. Levoy, M., Hanrahan, P.: Light field rendering. In: SIGGRAPH 1996: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, New York, NY, USA, ACM Press (1996) 31–42
2. Gortler, S.J., Grzeszczuk, R., Szeliski, R., Cohen, M.F.: The lumigraph. In: SIGGRAPH 1996: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, New York, NY, USA, ACM Press (1996) 43–54
3. Pandey, A., Choudhury, B., Chandran, S.: Efficient lightfield based camera walk. In: Fourth Indian Conference on Computer Vision, Graphics and Image Processing ICVGIP. (2004) 302–307
4. Debevec, P., Gortler, S.: Image-based modeling and rendering. In: SIGGRAPH 98 Course Notes. ACM SIGGRAPH, Addison Wesley, July. (1998)
5. Shum, H.Y., Wang, L., Chai, J.X., Tong, X.: Rendering by Manifold Hopping. International Journal of Computer Vision **50** (2002) 185–201
6. Chen, S.E.: Quicktime VR: an image-based approach to virtual environment navigation. In: SIGGRAPH 1995: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, New York, NY, USA, ACM Press (1995) 29–38
7. Unger, J., Wenger, A., Hawkins, T., Gardner, A., Debevec, P.: Capturing and rendering with incident light fields. In: EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering, Aire-la-Ville, Switzerland, Switzerland, Eurographics Association (2003) 141–149
8. Shum, H.Y., He, L.W.: Rendering with concentric mosaics. In: SIGGRAPH 1999: Proceedings of the 26th annual conference on Computer graphics and interactive techniques, New York, NY, USA, ACM Press (1999) 299–306
9. Isaksen, A., McMillan, L., Gortler, S.J.: Dynamically reparameterized light fields. In: SIGGRAPH 2000: Proceedings of the 27th annual conference on Computer graphics and interactive techniques, New York, NY, USA, ACM Press (2000) 297–306
10. Buehler, C., Bosse, M., McMillan, L., Gortler, S.J., Cohen, M.F.: Unstructured lumigraph rendering. In: SIGGRAPH 2001: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, ACM Press (2001) 425–432
11. Adelson, E.H., Bergen, J.R.: The Plenoptic Function and the Elements of Early Vision. In: Computational Modeling of Vision Processing. MIT Press (1991)
12. Sloan, P.P., Cohen, M.F., Gortler, S.J.: Time critical lumigraph rendering. In: SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics, New York, NY, USA, ACM Press (1997) 17–ff.
13. Sharma, P., Parashar, A., Banerjee, S., Kalra, P.: An Uncalibrated Lightfield Acquisition System. In: Third Indian Conference on Computer Vision, Graphics and Image Processing ICVGIP. (2002) 25–30