Efficient Image Updates Using Light Fields[*]

Biswarup Choudhury
Computer Science & Engg. Dept.
IIT Bombay
Mumbai, India 400076
biswarup@cse.iitb.ac.in

Aviral Pandey
Institute of Technology
Banaras Hindu University
Varanasi, India 221005
aviral.pandey@cse05.itbhu.org

Sharat Chandran
Computer Science & Engg. Dept.
IIT Bombay
Mumbai, India 400076
sharat@cse.iitb.ac.in

Abstract

The light field rendering method is an interesting variation on achieving realism. Once authentic imagery has been acquired using a camera gantry, or a handheld camera, detailed novel views can be synthetically generated from various viewpoints.

One common application of this technique is when a user "walks" through a virtual world. In this situation, only a subset of the previously stored light field is required, and considerable computation burden is encountered in processing the input light field to obtain this subset. In this paper, we show that appropriate portions of the light field can be cached at select "nodal points" that depend on the camera walk. Once spartanly and quickly cached, scenes can be rendered from any point on the walk efficiently.

1. Introduction

The traditional approach for "flying" through scenes is by repeated rendering of a three-dimensional geometric model. One well known problem with "geometry-based" modeling is that it is very difficult to achieve photo-realism due to the complex geometry and lighting effects present in nature. A relatively newer approach is Image-Based Rendering (IBR [2], [3]) 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. Synthetic realism is achieved, so to speak, using real cameras.

Light Field Rendering (LFR) [5] (or Lumigraphs [4]) is an example of IBR. The approach here is to store samples

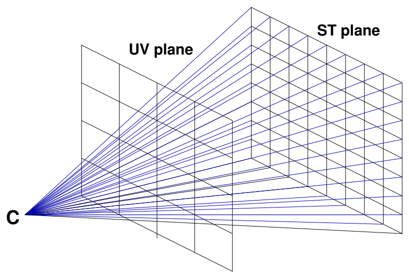


Figure 1. Two-Plane Parametrization

of the plenoptic function[1] which describes 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 Figure 1. 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, we are able to generate the view from *C*. In order to view a scene from any point in surrounding space, six light slabs are combined so that the six viewpoint planes cover some box surrounding the scene.

1.1. Statement of the Problem

The beauty of LFR lies in re-sampling and combining the pre-acquired imagery. In a typical walk-through situation, a person is expected to walk along a trajectory in three space and "suitably" sample the light field. The problem we pose in this paper is *"Given the light field on disk, and a walk using a camera, how fast can the images seen by the camera be updated?"*

[*]An earlier version of this paper appeared in ICVGIP 2004 [8]

For best results in light field based IBR, we expect that the size of the light field data-structure drastically increases with the increase in the resolution and the density of acquired images. As mentioned above, ray-shooting is performed as an intermediate step of the rendering procedure, a reportedly computationally intensive operation [9].

1.2. Contributions

For interactive rendering of the scene, one needs to store the complete light field in volatile memory, whereas only a subset of this is needed for a specific camera walk. Prior methods do not effectively address this issue. In this paper, we show how caching the light field *suitable* for the camera walk, dramatically reduces the computational burden, this is seen in Figure 10. Specifically,

- We compute the optimal location of a sparse set of "nodal points." The lightweight "light field" stored at these nodes is enough to render the scene from any of the infinite points – termed *query* points – on the camera path.
- The method in [9] uses homography to reduce the ray shooting computational burden in producing the image from one query point; multiple query points are treated afresh since no notion of nodal points was required therein. We use an alternative Taylor series method for reducing the computational burden of ray shooting queries; this is particularly applicable when the center of projection of the camera moves along a plane parallel to the UV and ST planes.
- The correctness of our scheme is shown using a mathematical characterization of the geometry of the light field. Experimental results validate our scheme.

The rest of this paper is organized as follows. Section 2 develops the mathematical framework for the problem. In Section 3, we give details of our approach and present our algorithm. Sample results and analyses are given in Section 4. We end with some concluding remarks in the last section.

2. Our Approach

As in the original work [5], 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 [5] handles the problem of aligning the plane of projection with the light-slab. Coming to the camera walk, in this section we provide the mathematical basis for the location and spacing of nodal points.

For brevity, we initially restrict the center of projection of the camera to move along a plane parallel to the UV and the ST plane. We later relax this condition and show that, the mathematical framework still holds. For motivation, consider a setup similar to the two slab setup where planes, UV and ST are replaced by lines U and S. We call this as the two-line setup. The query points lie on line C, which in turn replaces the camera plane. We provide the complete mathematical framework with respect to this setup.

2.1. Fixed Direction

The algorithms in this section tell us where to place nodal points for a specific query point q assuming a fixed direction determined by some point s. This condition is relaxed later.

Denote Δl to be the constant distance $d[G_i, G_{i+1}]$ between two consecutive grid points on the UV plane, i.e., the distance between the acquired camera locations.

2.1.1. Fixed Direction Algorithm

Given q, Algorithm 2.1.1 computes nodal points N_1 and N_2 (Figure 2). The radiance $L[q]$, in the direction of s, is presumably cached at points N_1 and N_2. We need to make use of this cache. Denote `assoc(p)`, where p is a point

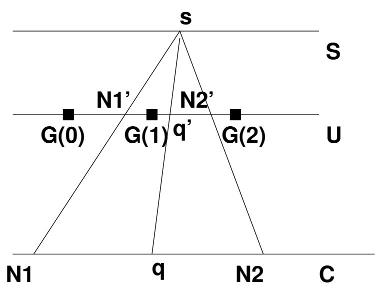


Figure 2. N_1 and N_2, the nodal points for q are marked such that $d[q'N_1'] = d[q'N_2'] = \frac{\Delta l}{2}$.

on C, to be the closest grid vertex G (on U) to the ray $\overline{ps}$. Suppose `assoc(q)` is G_i.

Algorithm 2.1.1: **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($N_1$) is $G_i$ then
    $L[q] = L[N_1]$
else
    $L[q] = L[N_2]$
end if
```

In the two-dimensional case, given q, our algorithm computes 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. 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 $L[q]$.

2.1.2. Comments

Notice that if the distance d in Algorithm 2.1.1 is more than $\frac{\Delta l}{2}$, as in Figure 3, an incorrect value of $L[q]$ is computed. When d is as specified in Algorithm 2.1.1, 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.

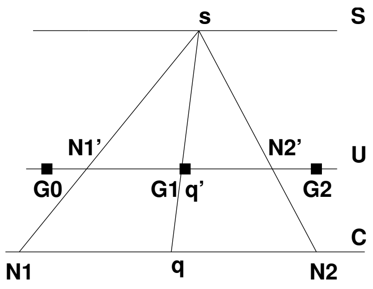


Figure 3. assoc(N_1) **is** G_0 **and** assoc(N_2) **is** G_2**.**

2.2. Changing the view direction

The results in this section assert that the nodal points may be chosen by arbitrarily picking any direction and applying Algorithm 2.1.1. That is, the selection of nodal points is independent of the direction.

Lemma 2.2.1 *The set of nodal points {N1, N2} for a given point s on S, serves as the set of nodal points for all s.*

Proof: For any query point q, consider two directions corresponding to $s1$ and $s2$. Let N_1 be any nodal point determined by Algorithm 2.1.1 for q and $s1$. The lemma asserts that nodal point N_1 serves as a nodal point for q and $s2$. Figure 4 illustrates the idea.

The basic idea is to prove that $N_1'q'$ is equal to $N_1''q''$. Applying similar triangles on $\Delta[N_1, S_1, q]$ and $\Delta[N_1, S_2, q]$, this can easily be proved. □

Next, we consider the lemma for the two-dimensional case:

Lemma 2.2.2 *Given a query point, nodal points may be decided using any point (s, t) provided the camera planes are parallel.*

Proof: As in Figure 5, let N_1, N_2, N_3, N_4 be the nodal points for query point q as determined by Section 2.1.1. Let $S_a = (s_1, t_1, z_s)$ be the intersection point of the query ray from q on the ST plane, and let $X_a = (x_1, y_1, z_u)$ be the intersection point of that ray on the UV plane. Similarly, define S_b and X_b. The proof uses the relationships (equations 1 and 2) below, to prove equation 3.

$$\overline{S_b N1} = \overline{S_a N_1} + \overline{(S_a - S_b)} \quad (1)$$

$$\overline{S_b q} = \overline{S_a q} + \overline{(S_a - S_b)} \quad (2)$$

$$\overline{S_b N1} = k(\overline{S_b X_b} + (-\frac{\Delta l}{2}, \frac{\Delta l}{2}, 0)) \quad (3)$$

□

2.3. The Power of Nodal Points.

Once nodal points are selected, there are a range of query points for which these nodal points are valid, as stated below.

Lemma 2.3.1 *The nodal points N_1, N_2 of a query point q_1 are sufficient for determining the radiance of any query point in the interval $[N_1, N_2]$.*

Proof: Consider any point q_2, between N_1 and N_2, and let the nodal points as determined by Algorithm 2.1.1, be N_3 and N_4. The lemma asserts that, for q_2, the radiance values stored at N_1 and N_2 are sufficient.

Without loss of generality, assume q_2 to be nearer to N_2 than N_1. We observe that either $d[N_1', \texttt{assoc(N}_3\texttt{)}] < \frac{\Delta l}{2}$ or $d[N_2', \texttt{assoc(N}_3\texttt{)}] < \frac{\Delta l}{2}$.

- **Case 1:** $d[N_1', \texttt{assoc(N}_3\texttt{)}] < \frac{\Delta l}{2}$ (shown in Figure 6(a)) $\Rightarrow$ assoc(N$_3$) = assoc(N$_1$). So assoc(N$_2$) = assoc(N$_4$), i.e., $L[N_1] = L[N_3]$, $L[N_2] = L[N_4]$. Thus, the radiance of q_2 can be obtained from the set [N_1,N_2].

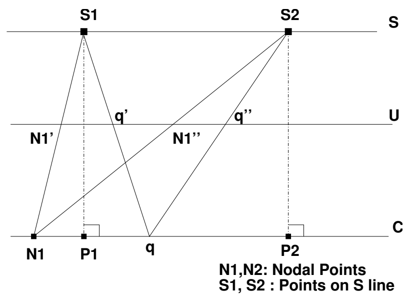
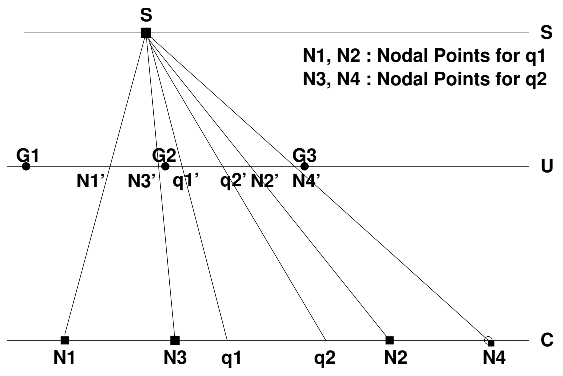
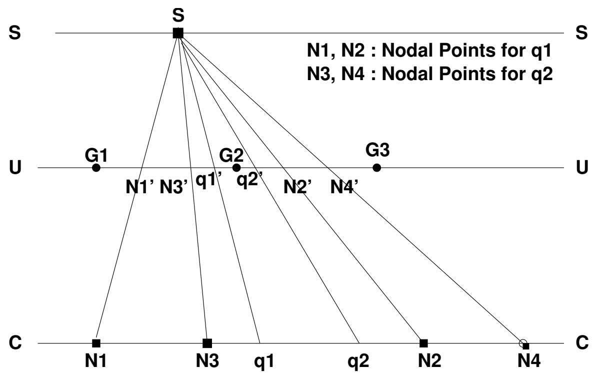


Figure 4. The choice of nodal points is independent of the direction. $N_1{}'q'$ **is equal to** $N_1{}''q''$**.**



(a) assoc(N_3) is closer to N_1 than N_2. Notice that $\mathtt{assoc(N_3)} = \mathtt{assoc(N_1)}$.



(b) assoc(N_3) is closer to N_2 than N_1. Notice that $\mathtt{assoc(N_3)} = \mathtt{assoc(N_2)}$.

Figure 6. Power of Nodal Points

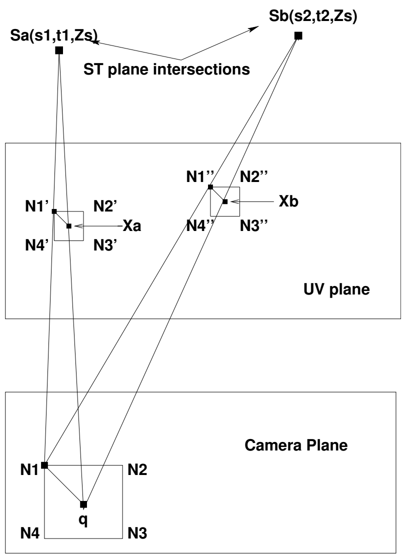


Figure 5. Choice of nodal points is independent of direction

- **Case 2:** $d[N_2', \mathtt{assoc(N_3)}] < \frac{\Delta l}{2}$ (Shown in Figure 6(b)) $\Rightarrow$ $\mathtt{assoc(N_3)} = \mathtt{assoc(N_2)}$. So, $L[N_3] = L[N_2]$. Further, $d[q_2', \mathtt{assoc(N_1)}] > \frac{\Delta l}{2}$ and $d[q_2', \mathtt{assoc(N_4)}] > \frac{\Delta l}{2}$. So in this case, $\mathtt{assoc(q_2)} = \mathtt{assoc(N_2)}$. Thus, the radiance of q_2 can be obtained from N_2.

Thus the lemma asserts that, for q_2, the radiance values stored at N_1 and N_2 are sufficient. □

Notice that, unlike in Figure 2, the nodal points N_1 and N_2 are not *equidistant from q_2. Also note that the two cases are not symmetrical.*

Lemma 2.3.1 enables us to determine the radiance at any query point q_i using the following algorithm:

Algorithm 2.3: **All-Directions (q_i)**

```
1. Determine nodal points bounding $q_i$.
2. for all $s \in$ S do
       Shoot a ray from query point $q_i$ to $s$.
       Apply Algorithm 2.1.1 and output radiance $L[q]$, in the direction of $s$.
3. end for
```

We denote the time taken for this operation as k_2 (Section 4.2).

2.3.1. Correctness of Algorithm 2.3

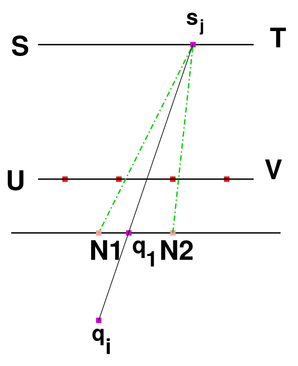
In either of the cases in Lemma 2.3.1, for any query point q_i lying in the region bounded by $[N_k, N_{k+1}]$ (determined in Algorithm 2.3: Step 1),

$$\mathtt{assoc(q_i)} = \begin{cases} \mathtt{assoc(N_k)} & \text{or} \\ \mathtt{assoc(N_{k+1})} \end{cases}$$

The result follows. □

Nodal points on the plane also enable multiple query points. Specifically,

Lemma 2.3.2 *The nodal points N_1, N_2, N_3 and N_4 of a query point q_1 on the camera plane are sufficient for determining radiance at any query point in the rectangular region bounded by these nodal points.*



$L(q_i) = L(q_1) = L(N_1 \mid N_2)$, for fixed s_j

Figure 7.

Proof: The proof of this lemma is similar to that of 2.3.1. Replacing the lines U, S and C by planes UV, ST and Camera, we get four different cases, which asserts that for determining the radiance at any query point the nodal points bounding it are sufficient. For more information, see [7]. □

Finally, when the query point is placed away from the plane containing nodal points we have,

Lemma 2.3.3 *The nodal points N_1, N_2 of a query point q_1 are sufficient for determining the radiance of any query point q_i along a direction s_j, as long as the points $[q_1, q_i, s_j]$ are collinear.*

Proof: Without loss of generality, let q_i be the query point and s_j denote a direction. From [1], and for the direction s_j, the radiance $L[q_1] = L[q_i]$ since q_1 is visible from q_j. From Lemma 2.3.1, $L[q_1]$ can be obtained from N_1 and N_2. The result follows. □

2.4. Ray Intersection

Algorithm 2.3 requires costly [9] ray intersections and should be avoided. In this section we show how to avoid ray intersection using the Taylor's theorem. Specifically, consider $X_1 = [X_c, Y_c, Z_c]$, which is center of projection for a virtual camera, and $I = (I_x, I_y, C_1)$, which is a point on the ST plane. Then a ray from X_1 to I, intersects UV plane at $g(X_1) = [g_x, g_y, C_2]$. Moving the COP to the location $X_2 = X_1 + \Delta X$, the change in x co-ordinate of the point of intersection with the UV plane is given by:

$$\Delta g_x = \frac{C_2 - C_1}{Z_c - C_1}\Delta X_c + \frac{(I_x - X_c)(C_1 - C_2)}{(Z_c - C_1)^2}\Delta Z_c \quad (4)$$

A similar equation is derived for Δg_y. The error associated with approximation is given by E_{g_x},

$$E_{g_x} = g_x(X_1 + \Delta X) - \overline{g}_x(X_1 + \Delta X) \quad (5)$$

where, $\overline{g}_x(X_1 + \Delta X)$ is the Taylor's estimate. From the first order analysis

$$E_{g_x} = \frac{2(I_x - X_c)(C_1 - C_2)}{(C_1 - Z_c)^2}\Delta Z_c \quad (6)$$

A similar equation is derived for E_{g_y}. If the camera motion is on any arbitrary path in a plane parallel to the ST plane ($\Delta Z_c = 0$), then the error $E_{g_x} = 0$, $E_{g_y} = 0$. Higher order Taylor error also are zero. In addition, the computational complexity involved in calculating the new UV intersection point decreases substantially, as (4) reduces to

$$\Delta g_x = \frac{C_2 - C_1}{Z_c - C_1}\Delta X_c \quad (7)$$

which is independent of the direction of the ray. This implies that a regular camera motion results in a regular shift of intersection points. In other words, we avert expensive ray intersection computations.

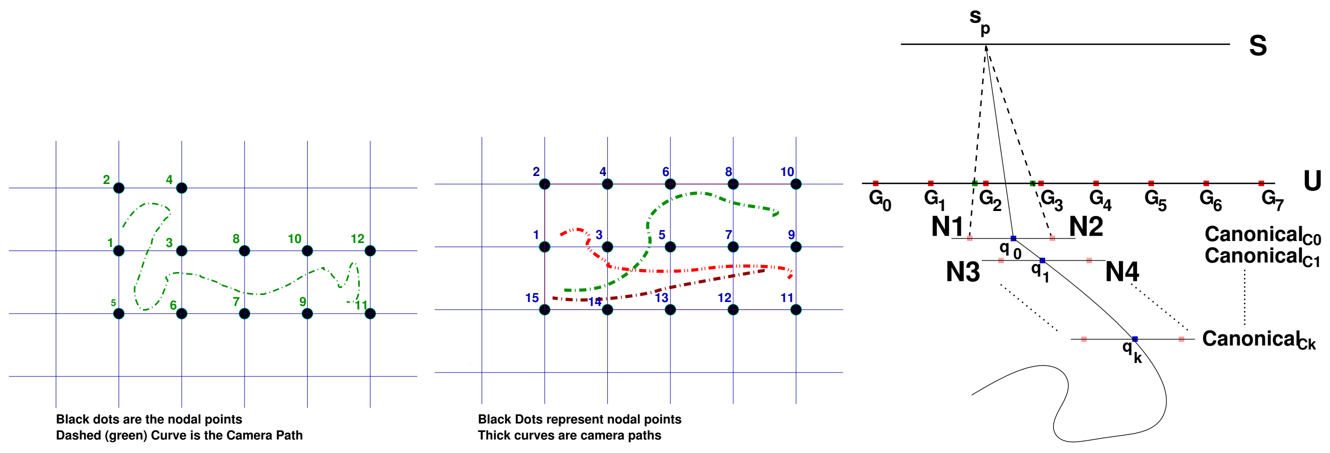
3. Caching Nodal Points

We now have the mathematical apparatus to select the nodal points given a camera walk. For the sake of exposition, we consider two orthogonal cases.

- Case 1: *Camera Walk on a plane parallel to the UV plane.*

Algorithm 3: 2D-Incremental-CameraWalk (`walk`)

1. Starting from the initial position on the camera path curve, 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. For simplicity, the nodal points are selected parallel to the u and v directions as shown in Figure 8(a). A grid is thus 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 2.4, instead of the original method [5].)
3. Apply Algorithm 2.3 to calculate the radiance at any query point inside the initial cell.
4. As the walk exits a grid cell, update the nodal points and go to Step 2.


Black dots are the nodal points
Dashed (green) Curve is the Camera Path
Black Dots represent nodal points
Thick curves are camera paths
s_p
S
U
G_0 G_1 G_2 G_3 G_4 G_5 G_6 G_7
N1
N2
N3
N4
q_0
q_1
q_k
Canonical_{C0}
Canonical_{C1}
Canonical_{Ck}

(a) 2D-Camera Walk: Incremental Nodal Points along the camera walk

(b) 2D-Camera Walk: Domain-based Nodal Points

(c) 3D-Camera Walk: Incremental Nodal Points

Figure 8. Beauty of Nodal Points

If we are given several camera walks lying in a domain, an alternate, version of the algorithm is to pick domain-based nodal points, as shown in Figure 8(b). Any query, on any camera walk, or even at random, in the rectangular region defined by the convex hull of the nodal points can be answered efficiently.

- Case 2: *Camera Walk along any arbitrary path in 3-dimension*.
  For brevity, we again consider a setup similar to the two slab setup where planes (UV and ST) are replaced by lines U and S as in Section 2 (Figure 8(c)). We now have an arbitrary camera walk in the 3D coordinate frame. The incremental algorithm is as follows:

Algorithm: 3D-Incremental-CameraWalk (`walk`)

1. Compute nodal points N_i and N_{i+1} on the line (termed $Canonical_{c_i}$) parallel to U using Algorithm 2.1.1.
2. Cache light field at these nodal points as in Step 2 of Algorithm 3.
3. Compute interval I_j on $Canonical_{c_i}$ defined by the intersections of the two line segments from the two corners of S to q_j on `walk`. If $I_j \in [N_i, N_{i+1}]$, use Algorithm 2.3 to calculate $L[q_j]$ (Lemma 2.3.3). Increment j, go to Step 3.
4. Otherwise, increment i and go to Step 1.

The two cases described above paves the way for the algorithm for the camera walk. For variety, we describe a domain based algorithm.

3.1. The Algorithm

```
Algorithm 3.1: All-CameraWalk

1. Determine $q_0$ the point on the arbitrary camera walk
   corresponding to the minimum $z$ coordinate (Figure 9).
2. Determine the plane (termed Canonical$_c$) parallel to
   the UV plane at $q_0$.
3. Determine the convex hull (the dotted-dashed quadrilateral
   in Figure 9) of the path.
4. Determine tangent lines from the two corners of S to
   the convex hull and thus points of intersections with
   Canonical$_c$.
5. Determine nodal points on Canonical$_c$ using Step 1
   of Algorithm 3 in the interval defined by the points.
6. For a query point $q_i$ on the camera walk,
     • For a point $s_j$ on S, determine $q_i^{'}$, the intersection
       of the ray $\overline{q_i s_j}$ with Canonical$_c$.
     • Use Algorithm 2.1.1.
```

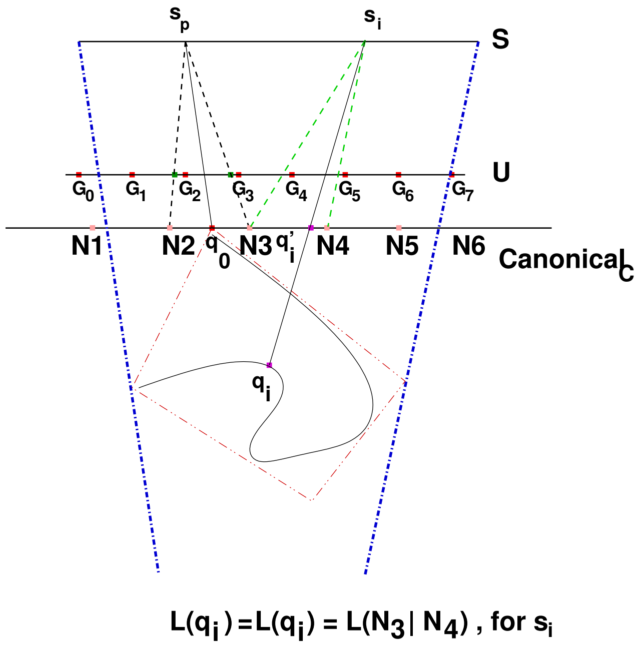
The domain-based algorithm described above is useful when there are a number of camera walks, or random query points in the region bounded by the domain. The incremental algorithm, on the other hand, is more suitable when we want to conserve memory for the nodal points.

3.2. Quadrilinear Versus Nearest Neighbor Approximation

Once nodal points are known, nearest neighbor approximation or quadrilinear interpolation can be used. In generating views using the nearest neighbor approximation, four nodal points will suffice for all information that is needed for intermediate query points. For quadrilinear interpolation 16 nodal points are needed to provide information (radiance) for a query point.

4. Sample Results

In this section, we first provide evidence that the results obtained by the use of our method matches those obtained by the implementation given in [5]. Later we show that our method requires less resources.



s_p
s_i
S
U
G_0 G_1 G_2 G_3 G_4 G_5 G_6 G_7
N1 N2 q_0 N3 $q_i^{'}$ N4 N5 N6
Canonical$_C$
q_i
$L(q_i) = L(q_i) = L(N_3 | N_4)$, for s_i

Figure 9. Domain-based Nodal Points

4.1. Buddha and Dragon

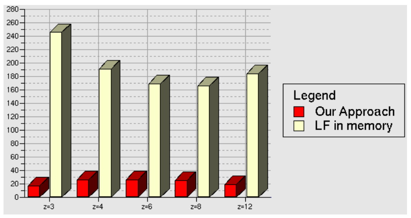
Figure 11 shows results obtained using our method from Section 3. The images are identical to those generated using [5], as returned by `diff` in Unix. The virtual camera viewpoint was (for example in Figure 11(g)) at $(0, 0, 3)$ and the nodal points were situated at $(\pm 0.09375, \pm 0.09375, 3.00000)$, where the origin was at the center of the ST plane. The input images were those obtained using 32×32 cameras.

Identical behavior is observed when we render the dragon (Figure 12); the light field for this was acquired using 8×8 cameras. Here the virtual camera was (for example in Figure 12(g)) located at $(0.03, 0.02, 2.00)$ and the nodal points at $(\pm 0.5, \pm 0.5, 2)$.

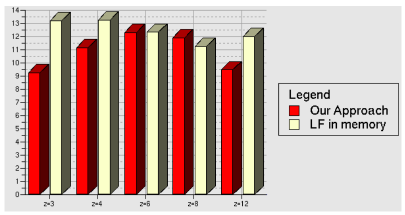
4.2. Computational Advantage

We now proceed to show the computational advantage when a camera walk is introduced. As discussed earlier, advantages arise due to nodal light field caching, and avoiding ray intersection calculations. Let n be the number of query points, and p the number of nodal points. Denote k_1 to be the time taken for ray intersection computations in the original method [5] for one query; if we use homography (from [9]), then this value is negligible.

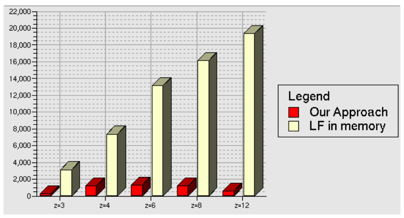
When the input light field is densely sampled, and is at a high resolution, it may or may not be possible to place the light field in memory. We penalize access to the light field



(a) Experiments performed with 32MB RAM. The time taken by the proposed method is considerably lesser than the original method.



(b) Experiments performed with 1GB RAM. Time taken by our approach is comparable. However, our approach uses only a very tiny fraction (2%) of the system memory.



(c) Number of disk accesses. Our approach performs significantly better.

Figure 10. All results are for 100 query points.

(for both methods) by the factor t in the following equation. In Algorithm 3: 2D-Incremental-CameraWalk, the expected gain is

$$\frac{n(t + k_1)}{(p(t + k_1) + nk_2)} \tag{8}$$

If the light field does not fit in memory, t represents disk access time. Therefore $t \gg k_1$, and the gain is approximately $\frac{n}{p}$.

4.2.1. Resource Usage

For the purpose of comparison, and to hand an advantage to the original light field implementation, we have chosen not to use the optimization in Section 2.4 in the experiments. Nevertheless, the results are worth noting. Our time results are based on an Intel Pentium IV 2.4GHz Linux based computer with 1 GB memory.

1. To simulate low RAM situations, we used only 32 MB of the 1 GB available and rendered Buddha on various camera paths located at different distances from the original camera gantry.

   We note that there is considerable gain as seen in figure 10(a), where the real time taken by the two approaches is plotted with respect to the z co-ordinate of COP. The average value of p for $n = 100$ is approximately 19, so the time gain is approximately 5.26, which is what theoretically equation 8 promises.

2. When the memory is sufficiently large to accommodate the huge light field, Figure 10(b) shows that the time taken by our method is comparable to the original method. This indicates that computing and going to the nodal cache is not very expensive. However, the total memory that we used was even less than 2% of the memory requirement of method in [5]. This is due to the fact that the method in [5] uses $u \times v \times r$ units of memory for a $u \times v$ camera gantry with an image resolution of r.

3. To quantify disk access, we rendered Buddha on various camera paths located at different distances from the original camera gantry. Starting at $(-1.5, -1.5, zCord)$ and going to $(1.5, 1.5, zCord)$, the virtual camera was made to follow different zig-zag paths at different values of z co-ordinate denoted $zCord$. The query points were chosen randomly along these paths. In the experiments on the Buddha image, the origin of the co-ordinate system was located at the center of the ST plane. Figure 10(c) shows the relative gain in terms of disk accesses. The graph shows the number of accesses to the disk storage required by various techniques when the nearest neighbor approximation is used, at $z = 3, 4, 6, 8, 12$, where z is the distance of the COP from the ST plane.

   As a point to note, when the value of z increases, the number of nodal points required for the same camera path decreases and so we get a quantitative difference in the number of disk accesses.

5. Final remarks

In virtual reality and in gaming applications, the light field is useful because no information about the geometry or surface properties is needed. However, there are some disadvantages.

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. The number of nodal points, and the distance between them have been characterized to ensure that the rendering of the scene is identical to what may have been done without the cache. The proofs of these characterizations have also been given.

Our description does not explicitly deal with decompression issues (indeed, in the first stage [5] of rendering, the entire light field is decompressed as it is read into memory from disk.) However, there is not any conceptual blockade in applying the general caching strategy and the mathematical elements even in this case.

References

[1] E. H. Adelson and J. R. Bergen. *Computational Modeling of Vision Processing*, chapter The Plenoptic Function and the Elements of Early Vision. MIT Press, 1991.

[2] D. Burschka, G. D. Hager, Z. Dodds, M. Jgersand, D. Cobzas, and K. Yerex. Recent methods for image-based modeling and rendering. In *Proceedings of the IEEE Virtual Reality 2003*, page 299. IEEE Computer Society, 2003.

[3] P. Debevec and S. Gortler. Image-based modeling and rendering. In *In SIGGRAPH 98 Course Notes. ACM SIGGRAPH, Addison Wesley, July*, 1998.

[4] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The Lumigraph. *Computer Graphics*, 30(Annual Conference Series):43–54, 1996.

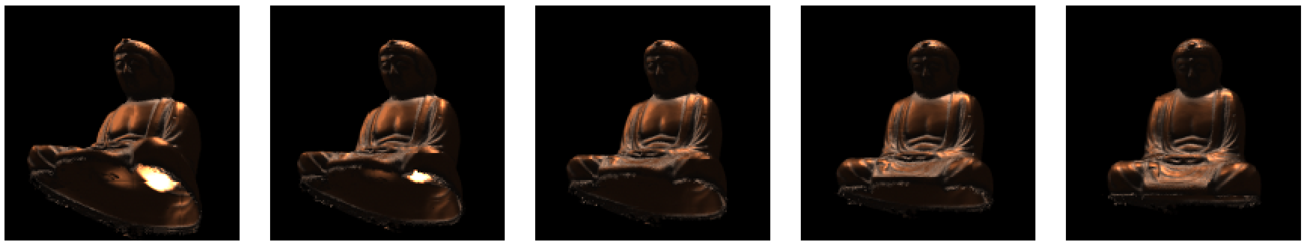
[5] M. Levoy and P. Hanrahan. Light field rendering. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 31–42. ACM Press, 1996.

[6] P. J. Narayanan. Image based rendering: A critical look. In *Second Indian Conference on Computer Vision, Graphics and Image Processing ICVGIP*, pages 216–222, 1998.

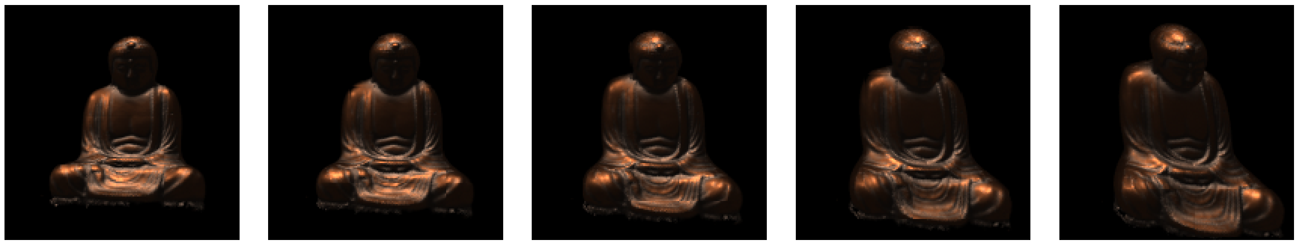
[7] A. Pandey, B. Choudhury, and S. Chandran. Light Field Based CameraWalk. `http://www.cse.iitb.ac.in/˜biswarup/research/icvgip2004/`.

[8] A. Pandey, B. Choudhury, and S. Chandran. Efficient light field based camerawalk. In *Fourth Indian Conference on Computer Vision, Graphics and Image Processing ICVGIP*, pages 302–307, 2004.

[9] P. Sharma, A. Parashar, S. Banerjee, and P. Kalra. An uncalibrated lightfield acquisition system. In *Third Indian Conference on Computer Vision, Graphics and Image Processing ICVGIP*, pages 25–30, 2002.

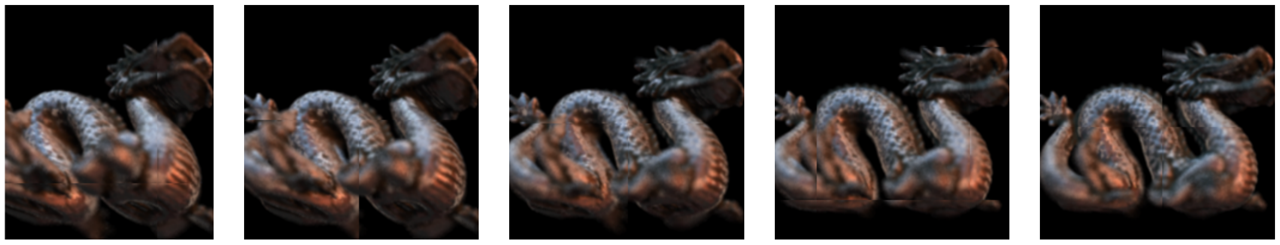


(a) (b) (c) (d) (e)

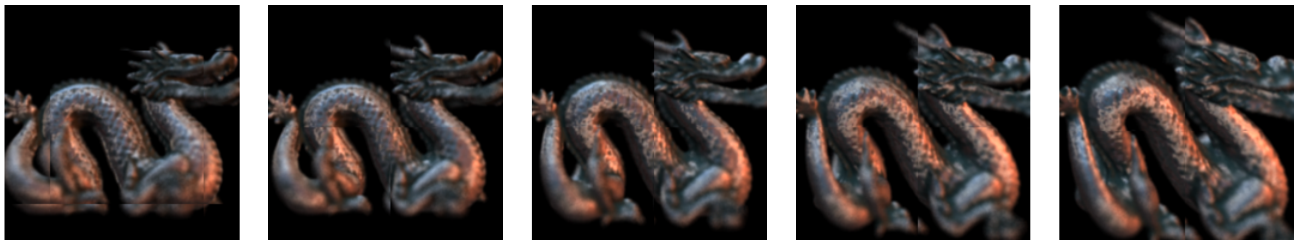


(f) (g) (h) (i) (j)

Figure 11. Rendered images of Buddha on a camera walk using our method



(a) (b) (c) (d) (e)



(f) (g) (h) (i) (j)

Figure 12. Rendered images of Dragon on a camera walk using our method