An Adaptive Sampling Method for Layered Depth Image

Ravinder Namboori, Hung Chuan Teh, Zhiyong Huang
Department of Computer Science
School of Computing
National University of Singapore
Singapore 117543
{namboori, tehhc, huangzy}@comp.nus.edu.sg

Abstract
Sampling issue is an important problem in image based rendering. In this paper, we propose an adaptive sampling method to improve the Layered Depth Image framework. Different from the existing methods of interpolating or splatting neighboring pixels, our method selects a set of sampling views based on the scene analysis that can guarantee the final rendering quality. Furthermore, in our method, the rendering speed is accelerated by the pre-computed patch lookup table, which simplifies the reference view selection process to a simple lookup of a hash table. We have implemented our method. The experiment study shows the advantage of the method.

Keywords: image based rendering, layer depth images, data sampling, image warping.

1. Introduction

Image based rendering (IBR) is an image synthesis framework where rendering is generated from a pre-sampled set of images of a scene, also called the reference images. The quality of the rendering result is now governed by the reference images. Thus, their sampling is an important problem. It is desirable that a robust solution be formulated to determining the exact set of reference images of proper sampling rate required in the rendering.

Our work is to address the sampling on one IBR framework, the Layered Depth Images (LDI). Our goal is to enhance the rendering quality and performance by adaptive sampling. The idea is to introduce a filtering stage after densely over-sampling the real world. In this stage, the method effectively computes the required reference viewpoints from the dense samples to avoid possible loss or redundancy of data. It not only improves the quality of the rendering, in terms of getting rid of holes and occlusion artifacts, but also enables quick generation of images, owing to the tabulation of only the necessary sampled imagery.

We have implemented the adaptive sampling method in LDI. In our experiment study, we can demonstrate the advantages of the method by using a test model with occlusions and a number of non-uniform surfaces. We can see that even an object of such complexity, which could have otherwise been difficult to render without an extremely dense uniform sampling, can be rendered much more accurately than using interpolating or splatting neighboring pixels. The rendering speed is also faster, comparable to using sparse sampling.

The remaining of the paper is organized as follows. First, we present the related work. Then, we start to describe our method followed by the implementation and experiment study. Finally, we conclude the paper with a brief discussion of future work.

2. Related work

IBR can be classified into four distinct categories: pixel, block, reconstruction and mosaicing based [Kang, 1997]. They vary largely in the knowledge of the geometry of the scene and the number of samples of the scene.

Our work follows an IBR framework based on the LDI [Shade et al., 1998]. We briefly describe the various techniques employed to address the sampling problem. While some of these techniques look at remedying the damage caused by the problem like splatting the holes during rendering, others attempt to find the best next view to sample, assuming the first sample is ideal. All these techniques aim to exploit the geometrical knowledge to improve the photo-realism of the synthetically generated scene.

Splatting is a technique, which aims to remedy the effects of the sampling problem. The layered depth image, which is created from uniformly sampled images, is splat into the output image by estimating the projected area of the warped pixels [Shade et al., 1998]. This estimation is computed differentially based on the distance between the sampled surface point and the LDI camera, the field of view of the camera, the dimensions of the LDI and the angle between the surface normal at the sampled surface point and the line of sight to the LDI camera.

As splatting is a post-sampling step, care has to be taken that it does not slow down the rendering engine. However, our method generates a lookup table during precomputing. In rendering, the reference view selection process is reduced to a rapid lookup of a hash table.

Furthermore, in [Shade et al., 1998], the reference images may be undersampled. The LDIs hence created are not substantial in quality. The splatting technique covers up most of the holes but with possibly incorrect data. It also accounts for much of computation time. Our method takes care of the completeness of the reference set of images required for rendering. The proper selection of reference images eliminates the need of splatting to ensure the quality of real time rendering.

The LDI Tree employed by [Chang et al., 1999] is a technique that adaptively selects an LDI from the LDI cluster for each pixel. However, since nothing is done to prevent gaps in rendering, the sets of samples at each resolution may be inadequate. In our method, we tackle the issue of gaps by sampling from not just one sampling circle, but a set of concentric sampling circles. The issue of gaps of the synthetic view is solved.

Another technique samples all visible surfaces, an attempt is made to record a series of images that, collectively, capture all visible surfaces of the object. One such heuristic is to segment the object to exemplify hierarchical visibility [Stuerzlinger, 1998]. The scene is assumed to be a set of surface polygons organized in a hierarchy. The hierarchical visibility method subdivides the scene hierarchy depending on the relative visibility of objects. Yet another heuristic is to cover all possible surfaces, masking reference images as each surface is considered [Fleshman et al., 1999]. In this approach, the set of scene polygons visible from a viewing zone is approximated and then a greedy algorithm is employed to select a small number of camera positions that together cover every polygon in the geometric model.

The best next view problem [Pito 1999] is to select the next view for the sampling system to take, given some already acquired views of the object. Two criterions are often considered in solving this problem. The visibility criterion attempts to maximize the number of surfaces not seen thus far, by adding the next image to the sampled set, while the quality criterion aims to improve the quality of the surfaces sampled. The quality criterion prioritizes an image, which samples a decent number of surfaces, covering most areas of these surfaces, over an image, which samples a lot of surfaces but obliquely.

The best next view method relies heavily on the set of steps taken previously as the greedy algorithm. A wrong choice by the heuristic at one stage would imply an inefficient solution. Also, since the quality criterion encourages the inclusion of reference images until a particular threshold is reached, there is no check on the redundancy whilst sampling. The selection of an image owing to some visibility criterion only means that it has the most number of surfaces not seen thus far. It does not dictate that the surfaces are not present in this image. On the contrary, in our method, we consider the whole scene before deciphering which samples to use for rendering. This eliminates the problem of incorrect intermediate steps of a greedy algorithm. Since the overlap between the patches is minimal and most of it is eliminated during patch merging, the issue of redundancy is almost non-existent. The process of finding the critical sampling arcs ensures that the sample data collected for the stage of rendering is minimal.

3. Our work

In this section, we give an overview of our work in the LDI framework. Then, we describe the major techniques in detail.

3.1 Overview

The original LDI framework is essentially classified into three main phases: scene sampling, scene geometry and photometry extraction and scene re-sampling as shown in Figure 1. Our work is on the four pre-computation stages as depicted in Figure 2 as flow chart. We briefly go through each of them as follows:

(1) Patch categorization and polygonization (subsection 3.2)

Based on the range and color image samplings of a scene, the normal vector of each point of the objects is estimated. Then, from the sampling points, this step attempts to identify the uniform patches, defined by certain constraints. They are polygonized to rectangles.

(2) Contour formation (subsection 3.3)

The output of the previous stage, i.e., the rectangular patches, is used to identify unique 2D-contours along the vertical axis. It detects the parts that can be approximated as planes and subsequently summarizes them as a 2D-contour. Details will be discussed in subsection 3.3.

(3) Identifying visibility and sampling regions (subsection 3.4)

In this step, we find the visibility and sampling regions for each of the contours. Visibility region for a particular edge in a 2D-contour is defined as that region from which the whole edge is visible, if there is no occlusion.

A sampling region for a particular edge is defined as the region where it is appropriate to sample that particular edge, ensuring that all of the data visible on the edge is captured. A sampling region is determined by formulae dependent on the size of the edge, the camera calibrations and the sampling camera trajectory.

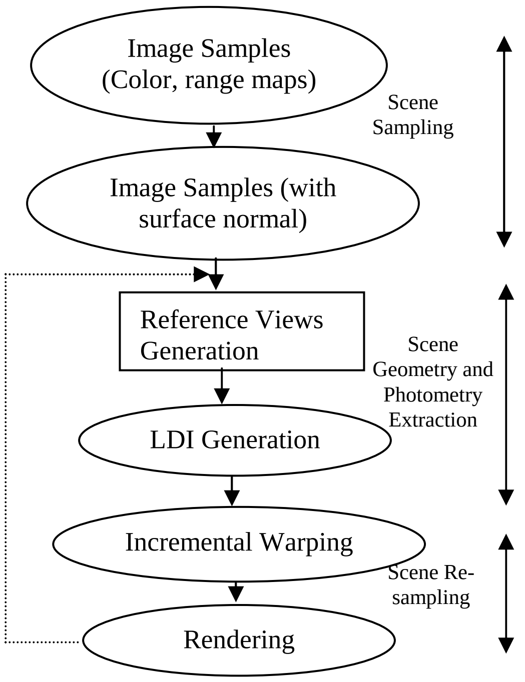


Image Samples (Color, range maps)
Scene Sampling
Image Samples (with surface normal)
Reference Views Generation
Scene Geometry and Photometry Extraction
LDI Generation
Incremental Warping
Scene Re-sampling
Rendering

Figure 1. LDI system flow

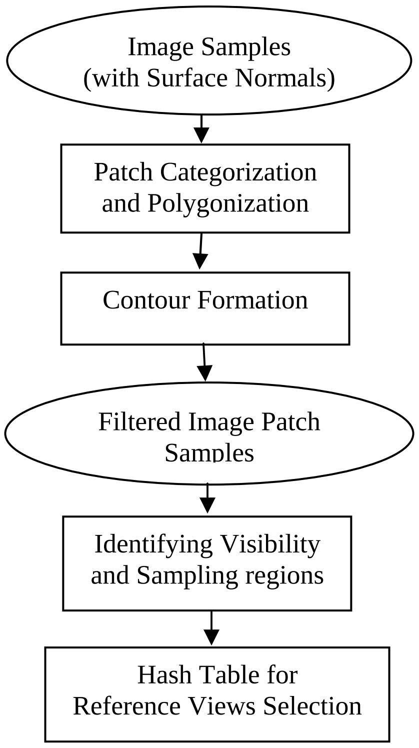


Image Samples (with Surface Normals)
Patch Categorization and Polygonization
Contour Formation
Filtered Image Patch Samples
Identifying Visibility and Sampling regions
Hash Table for Reference Views Selection

Figure 2. The flow chart of the proposed method

(4) Patch lookup table for reference view selection (subsection 3.4)

This step generates the patch lookup table for selection of reference views in rendering. The whole sampled object is categorized into uniquely defined sampled regions or rectangularised patches. We have associated each of these sampled regions with a particular view point. In rendering, the set of reference images needed to generate the required synthetic view can be obtained, if the sampled regions for that view point are known.

3.2 Patch Categorization and polygonization

This step starts from the point cloud to understand the geometry of the scene and proceeds to adaptively sample the object.

Before going any further with the procedure of patch categorization, we explain the concept of a patch, in the context of data sampling. A patch is regarded as any uniform surface on the object (a surface without uneven bumps), which can wholly fit into the field of view of the camera under consideration.

The purpose of defining a patch is to be able to summarize the geometry of the object in a 2D plane, so as to get a representation sketch of the uniform sections of the object. An intuitive way of thinking about adaptive sampling is that more/less samples are needed for non-uniform/uniform areas of the object.

The normal at each point of the object surface serves as the key parameter to categorize the patch. The normal $\mathbf{n}_C$ at the corresponding object point in the camera reference frame is computed from the x, y, z coordinates of the range map. It is approximated by averaging the 8 normals that correspond to the surrounding 8 neighboring triangular surfaces around the given pixel. The corresponding normal $\mathbf{n}_w$ in the world coordinates is computed by:

$$\mathbf{n_w} = (\mathbf{C}^{-1})^T\, \mathbf{n_c}, \qquad (1)$$

where **C** is the transformation matrix from the camera (from which the range map is obtained) to the world coordinates.

We now define a patch as a surface where every point of the surface satisfies the following constraints:

(1) The normals between any two neighboring points of a patch, in the spherical co-ordinate system do not differ by a preset δn_φ and δn_θ.

(2) The normals between the extreme two points of a patch, in the spherical co-ordinate system do not differ by more than a preset Δn_φ and Δn_θ.

(3) The z values of any two neighboring points of a patch do not differ by more than a preset δz. This ensures that areas on two objects with closely equal normal values are not grouped as a patch.

(4) The sizes of a patch in the screen coordinate system both horizontally and vertically should not exceed the maximum size that the field of view θ of the camera permits at that z value.

The size of a patch is computed in this way: Take a top view and denote the size of the patch in any scan line whose first and last points of this edge are S_{max} and S_{min}. Suppose the orthogonal bisector of the edge intersects the sampling circle at a point, and let the distance from the midpoint of the edge to this point be denoted as D. The maximum size d of the patch (Figure 3), constrained by the filed of view, is:

$$d = 2D \tan(\theta/2) \qquad (2)$$

The same criteria apply for the vertical extent, with the corresponding angle φ. Given the point cloud and these constraints, the patches are obtained and the whole point cloud is categorized.

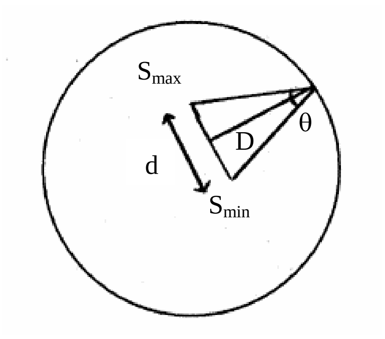


Figure 3. Patch size constraint (top view)

The patch derived so far is still non-uniform in shape in a 2D plane. We need to subdivide them so that they can be summarized in one dimension as a line. We choose the rectangle to represent the sampled regions for the fact that it can be reduced to a line along a scanning direction.

The patch polygonization step subdivides the patches into rectangles. Finally, all the patches are approximated by rectangular components.

Note that storing the entire sampled point cloud in a heap may not be feasible, owing to the highly dense sampling. We consider one reference image at a time. After the rectangular patches are formed, they are checked for overlaps with patches found from previous reference images in order to merge them. Merging the patches is done by a global image registration technique [Stockman et al., 1982]. Care should be taken to make sure that the patch, as it is being merged, still satisfies the patch constraints, over the border and as a whole.

3.3 Contour formation, sampling and visibility graphs

A scan line traversal is now performed to the patches. At the end of this process, we can associate a set of patches with each scan line. For each set, we can derive a contour in the plane defined by the viewpoint and the scan line (Figure 4). We list the pseudo code of the contour formation algorithm applied to the rectangular patches in Table 1.

```
procedure FormContours (patch [ ] )
  for k←0 to patch.size-1
  //find patch demarcations in the vertical direction
    demarcations.add(patch[k].min_y)
    demarcations.add(patch[k].max_y)

  sort(demarcations)     //in ascending order

  for j←0 to demarcations.size-1
    for k←0 to patch.size-1
        if patch[k].min_y == demarcations[j]
        //new  patch starts at this demarcation
          temp.add(patch[k])
        else if patch[k].max_y = demarcations[j]
        //old patch ends at demarcation
          temp.remove(patch[k])

    if j = 0
        prev_demarcation = -1000
    else
        prev_demarcation = demarcation[j-1]

    // form a contour with the patches in temp,
    // and applicable for y from prev_demarcation
    // to the current
    createContour (temp, prev_demarcation,
                        demarcations[j])
end procedure
```

Table 1. Contour formation algorithm.

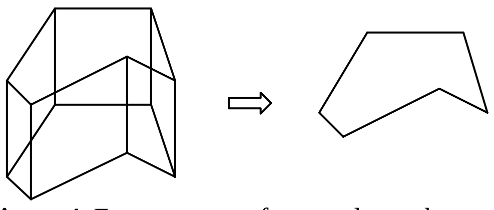


Figure 4. From a group of rectangle patches associated with one scan line to a 2D contour

Now the sampling problem is reduced to adequately sampling all the edges in each of these contours. In this context we define the sampling arc. For any contour, we sample the edges from the circumference of a circle, lying on the plane of the contour, with its center at the object's origin and a radius which defines how close we can get to the object during camera walkthrough. We call this the sampling circle. We can have multiple concentric sampling circles for various resolutions.

For any edge, a sampling arc is defined as the arc of the sampling circle, such that from any point on the arc, the edge under consideration has maximum visibility. Intuitively, when we project a scene to a viewing plane, the number of pixels does not necessarily have a one-to-one mapping with the number of actual points in the world coordinate system. The sampling arc of any edge is the arc from which the sampling rate is the same as that of the orthogonal view.

Figure 5 depicts a contour, the object's sampling circle and the sampling arc of a particular edge labeled "e1" of the contour.

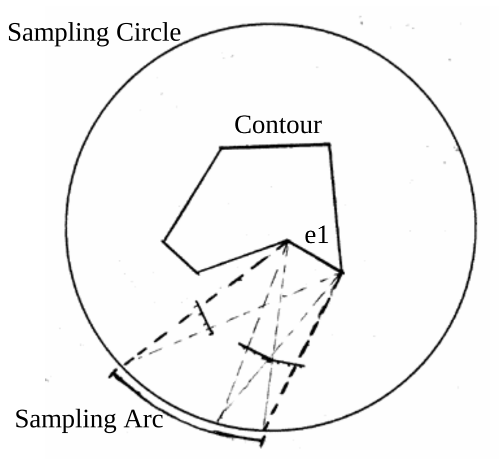


Figure 5. Illustration of sampling arc and sampling circle

We now describe how to find the sampling arc given an edge and a sampling circle. Suppose we choose a Cartesian st-coordinate system for the sampling circle with the center of the circle placed at its origin and the s-axis parallel to the edge. First, let us look at how the edge depth z, varies with respect to the camera motion, as depicted in Figure 6. We define the edge depth, as the distance between the midpoint of the edge and the camera position placed on the sampling circle. A prime sampling point (s_o,t_o), is the point of intersection of the edge's orthogonal bisector and the sampling circle. For any given edge, its prime depth, z_o, is its distance to (s_o,t_o). The edge depth function F_z, which is the edge depth as a function of the sampling point, can be formulated as:

$$z' = F_Z(s',t') = \begin{cases} z_0, & \text{if } \Delta s = 0 \text{ and } \Delta t = 0 \\ \Delta s / \cos(\tan^{-1}(z_0 - \Delta t / \Delta s), & \text{otherwise} \end{cases} \tag{3}$$

where $\Delta s = s' - s_0$ and $\Delta t = t' - t_0$.

Our aim is to find a sampling arc, such that the same pixel resolution as seen from (s_o,t_o) can be obtained for all points within the sampling arc. We observe from Figure 6 that as we move away from (s_o,t_o), the number of points on the edge projected to a pixel on the view plane will reduce. The critical arc is derived in this way: from (s_o,t_o), we locate the most left and right points on the arc so that the sampling rates are the same as that from (s_o,t_o). From Figure 6, we can see that the more right/left a segment on the edge, the more reduction of its size on the image plane.

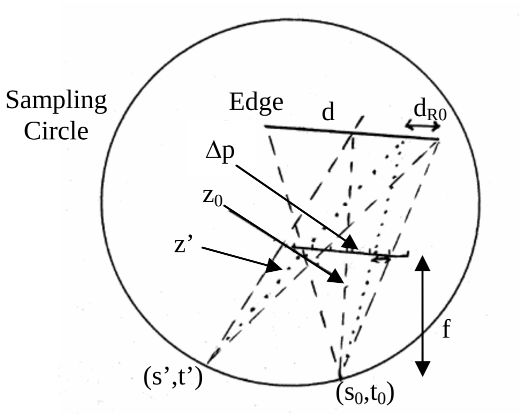
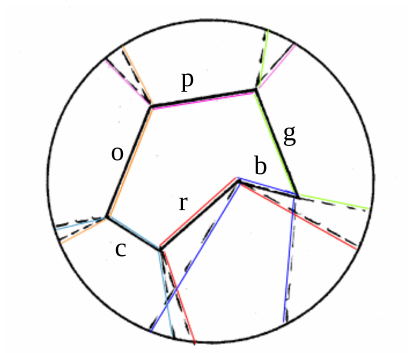


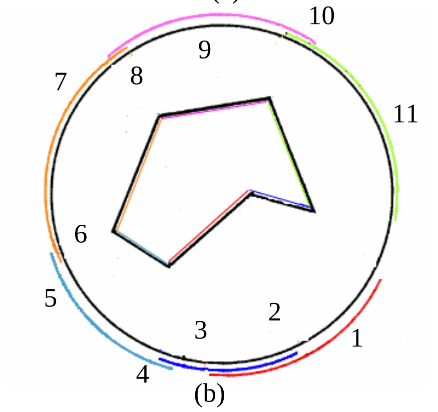
Figure 6. Illustration of deriving the sampling arc

Hence, the problem of finding the sampling arc becomes finding the points on the sampling circle where the two critical segments occupy two pixels. There are two camera dependent parameters: the pixel size Δp corresponding to edge segment Δp_w and the camera focal length f, the distance between the camera and the view plane. d_{R0} is the length of the right critical fragment when seen from the prime sampling point. We can now define a right and left *pixel occupancy functions* F_{WR} and F_{WL}, which are the number of pixels occupied by their respective critical segments. The sampling arc is thus determined by the set (s',t') such that $F_{WR}(s', t') > 1$ and $F_{WL}(s', t') > 1$.

Having established how to find the sampling arc, given any edge, we shall now look at how we use these sampling arcs to sketch the sampling graph and hence find the points to sample from. For all of the edges in the contour, the sampling arcs are transformed from their st-coordinate system to the world coordinate system. Figure 7 (a) depicts the sampling graph sketched by the sampling arcs obtained for all the edges. We number the various arc regions formed on the circumference of the sampling circle, as depicted in Figure 7 (b).



(a)



(b)

Figure 7. Sampling graph

Next, we tabulate every arc segment against its associated edges in the contour, as in Table 2. A directed graph is sketched with each of these arc segments as nodes. Node-A directed to Node-B in the graph indicates that the edges associated with the arc segment represented by Node-A, is a subset of the edges associated with the arc segment represented by Node-B. Figure 8 depicts the directed graph plotted for the example in Figure 7.

Arc segment	Associated edges
1	r
2	r,b
3	b
4	b,c
5	c
6	c,o
7	o
8	o,p
9	p
10	p,g
11	g

Table 2. Tabulation of arc segments and their associated edges

The set of critical arc segments is the set of minimum number of arc segments required to cover all the edges of the contour. This is determined by considering the set of arc segments represented by the leaf nodes in the directed graph. We perform a greedy algorithm on this set of arc segments, by selecting one arc segment at a time, to maximize the number of edges covered thus far. The result of this step is the set of critical arc segments, which in the case of the example depicted in Figure 8, is {2, 6, and 10}.

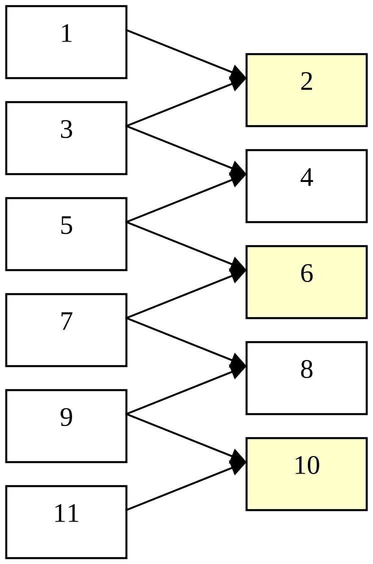


Figure 8. Directed graph of the arc segments

We now have a set of critical arc segments and the list of edges that can be sampled from each of the critical arc segments. The set of adequate samples for the contour are obtained by sampling from selected points on these critical arc segments, trying to maximize the number of edges sampled in a single sample, ensuring that all the edges of the contour are covered. A greedy algorithm, in pseudo code depicted in Table 3, is devised on a set of edges to be covered by critical arc segments. We first find that sampling point (position and direction) on the arc segment, in which the maximum number of edges can be sampled. The *criticalarcs* is an array of all critical arc segments. The *criticalarcs* object contains the array of edges that can be sampled. The *edgescovered* is a temp array which keeps track of the edges that can be sampled from the given point. The *solution* is a set of edges and point (position and direction) from which these edges are sampled using the greedy algorithm. If we find a larger set of edges that can be covered from the sampling point, the algorithm replaces the previous solution with this one. Typically, a contour of n edges will result in n/3 to n adequate samples.

```
procedure **MakeSampleSet** (criticalarcs [ ])
  for k ← 0 to criticalarcs.size-1      //for all critical arcs
    while criticalarcs[k].edges.size > 0
                //as long as some edge is yet to be sampled
      for j ← criticalarcs[k].beginpt to criticalarcs[k].endpt
        for i ← 0 to 180
          for a ← 0 to criticalarcs[k].edges.size-1
            calculate(pixel_occupancy for
                           criticalarcs[k].edges[a])
                 // z'and d/2 change with angle i and edge
            if pixel_occupancy satisfactory
              edgescovered.add(criticalarcs[k].edges[a])
                //this edge can be sampled from here
            if edgescovered.length > max
               //if this is the most optimum solution thus far
             solution.remove(edgescovered)
             solution.add(j, i, edgescovered)
               //point and angle to sample the edge set
            max = edgescovered.size
      criticalarcs[k].remove(solution.edgescovered)
               //these edges are sampled
  return solution
end procedure
```

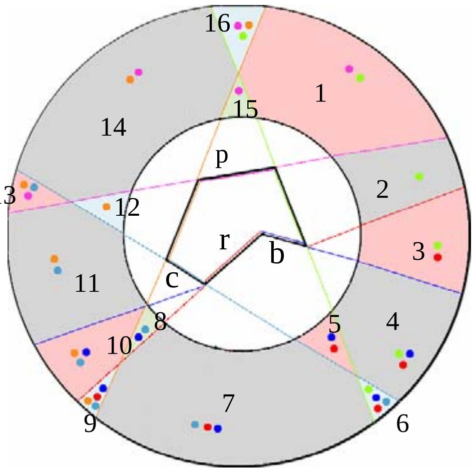
Table 3. The algorithm to derive the sample set

3.4 Visibility region and visibility graph

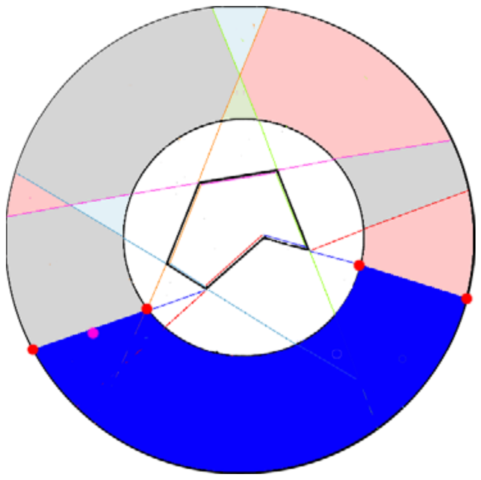
This subsection generates the patch lookup table for selection of reference views in rendering. The whole sampled object is categorized into uniquely defined sampled regions or rectangularised patches. We have associated each of these sampled regions with a particular view point. In rendering, the set of reference images needed to generate the required synthetic view can be obtained, if the sampled regions for that view point are known.

Towards this aim, we define potential visibility region of an edge as the region in the plane of the edge, where at least some part of the edge is visible. As an edge can be seen from any region in front of it, the potential visibility region for an edge is as depicted in Figure 9. A visibility graph for a contour is defined as the graph sketching the potential visibility regions of all the constituent edges of the contour, highlighting the overlap of various visibility regions.

We see from Figure 9 (a) that a number of potential visibility regions are formed, given any contour, with each region having a clearly defined set of visible edges. Note that a region formed by the overlap of visibility regions of two edges, defines the region from which both the edges are visible, *e.g.* edges c, r and b are potentially visible in visibility region 7.



(a)



(b)

Figure 9. The visibility graph

It might be worthwhile to note that, while sketching the visibility graph, we need not worry about the visibility regions inside the sampling circle, as by the definition, the sampling circle sets the boundary for walkthrough. For example, a visibility region is highlighted in blue in Figure 9 (b) whose two arcs are defined by four end points (r,θ_{i1}), (r,θ_{i2}), (R,θ_{o1}), (R,θ_{o2}) highlighted as red points in a cylindrical coordinate system. Its origin is at the center of the circle. Table 4 depicts its sample patch lookup table generated for the visibility graph.

Radii index (length unit in mm)	Angle index (degrees) $(\theta_{o1}, \theta_{o2})$	Angle index (degrees) $(\theta_{i1}, \theta_{i2})$	edge
1000, 2000	(40,150)	(45,135)	p
1000, 2000	(215,345)	(220,350)	b

Table 4. Patch lookup table (y = 10 to 18)

This table can be used to lookup the patches that can be seen from any point (y', r', θ') in rendering, where y' is the scan line and (r', θ') are the cylindrical coordinates at that plane. The visibility arcs on the inner and outer bounding circles define a visibility region for an edge. All of the sampling regions depicted in Figure 9 (a) are formed by the overlapping of these regions. In cylindrical coordinates, if the visibility arcs for edge e are defined as (r, θ_{i1}), (r, θ_{i2}), (R, θ_{o1}), (R, θ_{o2}), a point (r', θ'), during walkthrough would be able to see all edges, whose index satisfy the constraints of Table 5.

(1) $r<r'<R$,
(2) $\theta' \in [\theta_{o1}, \theta_{o2}]$,
(3) let θ_{oc} be closer to θ', then
    if $|\theta_{o1} - \theta_{o2}| \leq 180$, $h' \geq h$
    if $|\theta_{o1} - \theta_{o2}| \geq 180$, either $\text{int}(\theta'/90) \neq \text{int}(\theta_{o1}/90)$ and $\text{int}(\theta'/90) \neq \text{int}(\theta_{o2}/90)$ or $h' < h$,

where $h=R.\sin(\tan^{-1}(r.\sin(|\theta_{ic}-\theta_{oc}|)/(R-r.\cos(|\theta_{ic}-\theta_{oc}|))))$,
$h'=R.\sin(\tan^{-1}(r'.\sin(|\theta'-\theta_{oc}|)/(R-r'.\cos(|\theta' - \theta_{oc}|))))$.

Table 5. The constraints for visibility graph

For example, a point (r', θ') in figure 9 (b), marked in pink, where r' = 1500 and θ' = 217 degrees, is found to be in the visibility region of edge b, by satisfying the following constraints.

(1) $1000<1500<2000$
(2) $217 \in (215,345)$
(3) $(345-215) < 180$ and $(h'=309) > (h=173)$

We have several patch lookup tables (one for each contour), and given any view point in cylindrical coordinates during the walkthrough, the table can be used along with the constraints in table 5, to lookup the edges/patches that can be seen from it. The radii index is stored, in case we have more concentric sampling circles for different resolutions (like the LDI Tree).

3.5 Rendering engine

Our rendering engine generally follows that of [Shade et al, 1998]. It is broadly a composition of identifying the user position in the 3D space, selecting the reference images required to render the synthetic view at the view point, and finally generating the synthetic view from the reference images. We also adopt the re-projection of the pixel from the reference patches to the LDI and the incremental warping to avoid redundant calculations when generating the re-projected pixels in the scanlines. However, in our method, as the visibility graphs has been derived from the 2D contours, during pre-computation, the process of reference view selection now becomes a mere look up of the patch lookup table, based on the user's coordinates, to elicit the patches visible from those co-ordinates.

4. Implementation

In this section, we shall go over the various components used for the system implementation. We describe some of the issues dealt with, during the system implementation.

4.1 Hardware Components

The Range Scanner used in our sampling process is the Minolta Vivid 900, which samples both a color map and a range map at the same time for any given viewpoint.

We rotate the object instead of a much heavier range scanner. A mannequin object was chosen and the diffuse lighting environment was set up. A mount is built and placed over the turntable to hold the object to be sampled. This apparatus has two degrees of freedom; rotation about a vertical axis with precision of ±0.2 degrees and vertical translation with a precision of ±0.2 cm. Figure 10 illustrates this apparatus and the setup used for sampling.

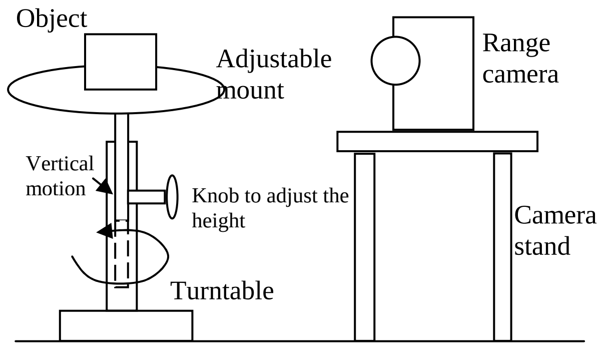


Figure 10. Set-up for sampling

We also considered the occlusion. The mannequin was placed behind two vertical rods which catered for most of the occluded regions. The existence of several non-uniform surfaces on the mannequin was also noticeable. Figure 11 illustrates a few snapshots of this object.

4.2 Software Components

The program was developed in Open GL, with C++ using Microsoft Visual C++ development environment. The object was scanned by the Range scanner, and the scanner software output format was converted to a simpler color map and depth map format. The color map was represented as Portable Pixel Map images (.ppm files), while the range map was represented using text files (.txt files). A one to one correspondence could be found between the (r,g,b) pixels in the color map and the (x,y,z,flag) surfels of the range map.

The camera, often taken for granted in most rendering systems, constitutes one of the key components of the rendering engine. We represent the camera as a 4x4 transformation matrix, such that given a camera placed at a view point c_1 and given its representation as matrix $\mathbf{C}_1$, any point in the global co-ordinate system could be re-projected into the camera's view plane by a simple matrix multiplication between the point co-ordinates and the camera matrix $\mathbf{C}_1$.

4.3 Other issues

In this subsection, we shall look at some of the issues involved in the system implementation. They are the range map, surface normals, and uniform dense sampling.

Each range image file has a three-line header giving the number of rows and columns in the image followed by four image sections. The first is the so-called 'flag' image, where a pixel value of 1 means the corresponding (x, y, z) values at that pixel are valid, and vise versa. Following the flag image are the image of X-coordinates, the image of Y-coordinates, and the image of Z-coordinates in floating point. The X and Y images are required only when calculating the normals. The Z-values are the range/depth under consideration. A very high value was used for the depth, in cases where the pixels do not correspond to the object but to some background.

There is one consideration that ought to be discussed in approximating the surface normals as discussed in section 3.2. The 8 neighboring points may not exist or even if they do, they may not lie on the same surface. We have addressed the special cases. For example, when the direction of a normal vector differs a lot from the rest of the normals, imply that they are the normals for an edge or for a different surface. In such a case, we discard those points and do not involve them in calculating the normal.

We place the object over a turntable, as depicted in Figure 10, to ensure an accurate rotational motion to 0.2-degree precision. The mount underneath the object helps the object to move up and down to an accuracy of 0.2cm. Together, these two motions simulate the positioning of the camera anywhere on a cylindrical surface surrounding the object. We can sample the object densely, as the redundant data will be filtered out.

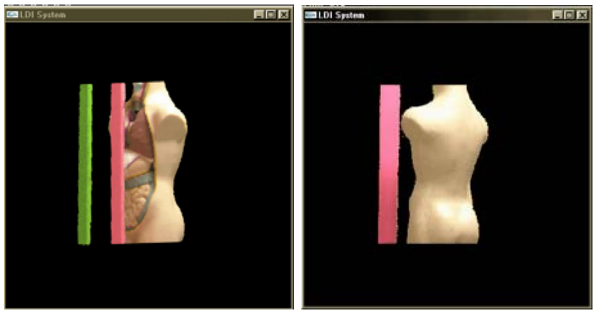
For the sample implementation, we used the MIDDLE Lens and sampled along only one sampling circle. We can sample along various concentric sampling circles, to ensure that a proper circle is always available during rendering. Samples taken with a TELE Lens along a sampling circle close to the object, and with a WIDE Lens along a sampling circle far from the object, are added to the existing sample set, to further enhance the rendered output and frame rate.

5. Results

In this section, we will analyze and discuss the results and improvements of the proposed system. We shall also compare and discuss the results with those of other systems.

The results observed from the sample implementation are promising. As can be seen from Figure 11, the quality of the rendered output is comparable to current systems using splatting. No holes are observed, and unlike splatting, the rendered output is completely a result of the original sampled data, and not of any interpolation or synthetic approximations.

The rendering speed can be qualified as quite fast, as a real time walkthrough shows no signs of processing lag. Such a high speed would have been impossible, if all of the data initially sampled were to be retained for rendering the synthetic views during walkthrough. With our method, we retained the adequate data to ensure the high quality of the rendered output, and disposed the redundant data, to ensure the real time rendering speed noticed in the camera walkthrough.



(a) (b)

Figure 11. Synthetic Views generated by our system

Table 6 shows the frame rate observed with our improved system during walkthrough and other statistics of the system.

System Configuration:	CPU: Pentium-4, 1.6 GHz system RAM: 256 MB
Input Data:	Adaptively filtered set of patches from the sampled set of reference images of 296x222 resolution, sampled along a circle around the object, at regular intervals of 2.82^o Size: 97.3 Mb
Frame Rate:	10.7 fps
Patch loading time:	Proportional to the size of a patch. Typical Patch size: 2 Kb Time to load a patch: 0.71 milliseconds

Table 6. Statistical information for our system

Now we compare the results obtained from our system with a sample implementation of the original LDI system [Shade et al., 1998], with uniform sparse sampling (with and without splatting) and uniform dense sampling.

With a uniform sparse sampling and reference images taken along a circular orbit around the object at regular intervals of 19.74 degree the following results were obtained.

It is observed that our result (Figure 11) is better than the results of sparsely sampled, both non-splatted and splatted systems (Figure 12). The renderings of the non-splatted system (Figure 12 (a) and (b)), if magnified a few times their present sizes, will reveal the holes along the vertical green rods and between the two lungs for the uniform sparse sampling without splatting. In the case of the splatted system (Figure 12 (c) and (d)), both the systems have no holes. However, the rendering results of the splatted system are not based on the original samples. Some textures on the object when viewed closely are not continuous.

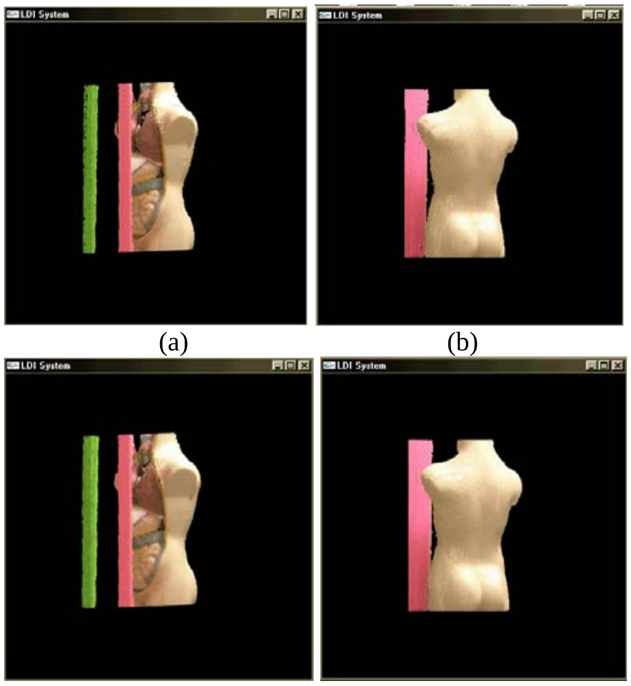


Figure 12. (a) and (b) Synthetic Views generated by the sparsely sampled LDI system, without splatting; (c) and (d) synthetic Views generated by the sparsely sampled LDI system, with splatting

The speed of our system is comparable to the sparsely sampled non-splatted system, while it is better than the splatted version (see Table 7).

System Configuration:	CPU: Pentium-4, 1.6 GHz system RAM: 256 MB
Input Data:	Reference images of 296x222 resolution, sampled along a circle around the object, at regular intervals of 19.74^o Size: 40.6 Mb
Frame Rate:	12.2 fps
Image loading time:	Proportional to the size of an image. Reference Image size: 2.25 Mb Time to load an image: 0.8 seconds

(a)

System Configuration:	CPU: Pentium-4, 1.6 GHz system RAM: 256 MB
Input Data:	Reference images of 296x222 resolution, sampled along a circle around the object, at regular intervals of 19.74^o Size: 40.6 Mb
Frame Rate:	7.1 fps
Image loading time:	Proportional to the size of an image. Reference Image size: 2.25 Mb Time to load an image: 0.8 seconds

(b)

Table 7. Statistical information for the sparsely sampled LDI system: (a) without splatting; (b) with splatting

Now, we compare our system with the highly dense uniform sampling with reference images taken along a circular orbit around the object, at regular intervals of 2.82 degrees, the results obtained are as depicted in Figure 13.

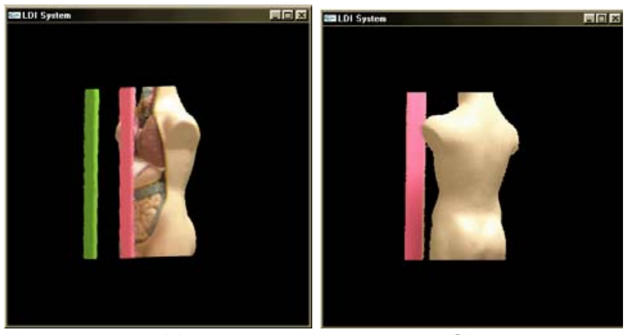


Figure 13. Synthetic Views generated by the densely sampled LDI system

It is observed that the quality of the rendered views of our system is comparable to that of the dense sampled system, despite the fact that the dense sampled system has a lot more input data at its disposal.

The speed of our system is much faster than the dense sampled system. Table 8 illustrates the statistical information observed with the uniform dense sampled system.

System Configuration:	CPU: Pentium-4, 1.6 GHz system RAM: 256 MB
Input Data:	Reference images of 296x222 resolution, sampled along a circle around the object, at regular intervals of 2.82^o Size: 286 Mb
Frame Rate:	1.3 fps
Image loading time:	Proportional to the size of an image. Reference Image size: 2.25 Mb Time to load an image: 0.8 seconds

Table 8. Statistical information for the densely sampled LDI system

Table 9 summarizes the statistical difference among all the systems. A video sequence of the walkthrough can be found in http://www.comp.nus.edu.sg/~tehhc/sldi.html.

Attributes	Our Improved System	Sample Implementation of Original LDI System [Shade et al., 1998]		
		Sparsely Sampled (no splatting)	Sparsely Sampled (with splatting)	Densely Sampled
Input Data Size	97.3 Mb	40.6 Mb	40.6 Mb	286 Mb
Renderin g Speed / Frame Rate	Fast 10.7 fps	Fast 12.2 fps	Average 7.1 fps	Slow 1.3 fps
Reference View Selection	Pre-computed Patch Lookup Table	Closest Reference Images during walkthrough		
Splatting	No	No	Yes	No
Holes	No	Yes	Mostly No	No
Quality	Good	Poor	Average	Good

Table 9. Comparison of the different systems

6. Conclusion and future work

Sampling issue is an important problem in IBR. In this paper, we proposed a method to improve the LDI, by adaptively sampling a scene, to avoid various computations during rendering. The quality of the output is enhanced, owing to the fact that the synthetic view is not generated by interpolating or splatting neighboring pixels, but with original sampled data. In addition to the improvisation of the quality during rendering, the rendering speed is enhanced by the pre-computed patch lookup table, which simplifies the reference view selection process to a simple lookup of a hash table.

To evaluate our method, we demonstrated the advantages of this approach by considering an object with occlusions, and quite a number of non-uniform surfaces. It was established that even an object of such complexity, which could have otherwise been difficult to render without an extremely dense uniform sampling, was rendered much more accurately than a splatted synthetic image. The rendering speed was comparable to sparse sampling, and better than the splatted system.

For the future work, we will include the reflectance properties of the materials and lighting effects.

7. References

[Chang et al., 1999] Chun-Fa Chang, Gary Bishop and Anselmo Lastra. LDI Tree: A hierarchical representation for image-based rendering. Proceedings of the SIGGRAPH 1999 annual conference on Computer graphics, 1999, pp. 291 – 298.

[Fleshman et al., 1999] Shachar Fleishman, Daniel Cohen-Or and Dani Lischinski. Automatic camera placement for image-based modeling. Proceedings of the Pacific Graphics ’99, 1999.

[Kang, 1997] Sing Bing Kang. A survey of image-based rendering techniques. Cambridge Technical Report Series, August 1997.

[Pito 1999] R. Pito. A solution to the next best view problem for automated surface acquisition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 21(10):1016--1030, 1999.

[Shade et al., 1998] Jonathan Shade, Steven Gortler, Li-wei He and Richard Szeliski. Layered depth images. Proceedings of the 25th annual conference on Computer Graphics, 1998, pp. 231 - 242.

[Stockman et al., 1982] G.C. Stockman, S. Kopstein and S. Benett. Matching images to models for registration and object detection via clustering, IEEE Trans Pattern analysis and Machine intelligence 4, 1982, pp 229-241.

[Stuerzlinger, 1998] Wolfgang Stuerzlinger. Imaging all visible surfaces. Computer Science Technical Report TR98-010, Mar 1998.