Efficient Constraint Evaluation Algorithms for Hierarchical Next-Best-View Planning

Kok-Lim Low
National University of Singapore
lowkl@comp.nus.edu.sg

Anselmo Lastra
University of North Carolina at Chapel Hill
lastra@cs.unc.edu

Figure 1. (a) The eight 3D views in a computed view plan. The views may be at different height. (b)&(c) Two views of the polygonal model constructed from the eight range scans.

Abstract

We recently proposed a new and efficient next-best-view algorithm for 3D reconstruction of indoor scenes using active range sensing. We overcome the computation difficulty of evaluating the view metric function by using an adaptive hierarchical approach to exploit the various spatial coherences inherent in the acquisition constraints and quality requirements. The impressive speedups have allowed our NBV algorithm to become the first to be able to exhaustively evaluate a large set of 3D views with respect to a large set of surfaces, and to include many practical acquisition constraints and quality requirements. The success of the algorithm is greatly dependent on the implementation efficiency of the constraint and quality evaluations. In this paper, we describe the algorithmic details of the hierarchical view evaluation, and present efficient algorithms that evaluate sensing constraints and surface sampling densities between a view volume and a surface patch instead of simply between a single view point and a surface point. The presentation here provides examples for the design of efficient algorithms for new sensing constraints.

1. Introduction

With the advent of affordable *active range sensing* devices, reconstructing detailed 3D digital models of real-world objects and environments has become more common. A typical reconstruction would require multiple range scans made from different scanning locations. The set of scanning locations must be chosen carefully so that each location satisfies a set of *acquisition constraints* and the reconstructed 3D digital model can meet a set of *quality requirements*. This task is known as *view planning*. For 3D reconstruction, a priori knowledge of the scene geometry is not available to the automatic view planner. The first scan is made from a view selected by a human operator, and for each subsequent scan, the planner must determine its best view based on the information collected from the previous scans. This is often called the *next-best-view (NBV) problem*.

The NBV problem is inherently a local optimization problem since global geometric information is unknown. It is NP-hard, and since it can be reduced to the set-covering problem, it is often solved approximately using a greedy approximation algorithm. A greedy NBV algorithm selects the view that maximizes a *view metric* as the best view for the next scan.

The major challenge to a practical NBV solution is an efficient method to evaluate the view metric for a large set of views, using information provided by a partial model of the scene. Each evaluation can be computationally very expensive, since a large amount of information of the partial model may be involved, and visibility computations and other constraint evaluations are expensive. This apparent computation difficulty has limited many previous NBV algorithms to simple and small objects, incomplete search space, incomplete set of acquisition constraints and reconstruction quality requirements, and low-quality acquisition. Some early algorithms even ignore self-occlusion of the objects [2].

However, an efficient algorithm to evaluate the view metric is actually possible. Recently, we proposed a hierarchical approach [5] to adaptively exploit the various *spatial coherences* inherent in the acquisition constraints and quality requirements. Results show that the hierarchical approach can speed up view evaluation by one to two orders of magnitude over the straightforward method used in the previous NBV algorithms. These impressive speedups have allowed our NBV algorithm to become the first to be able to exhaustively evaluate a large set of 3D views with respect to a large set of surfaces, and to include many practical acquisition constraints and quality requirements.

Our proposed hierarchical view evaluation algorithm was inspired by the hierarchical radiosity algorithm [4], which can be generalized to evaluate pair-wise interactions between extended objects. The success of the hierarchical view evaluation is due to the evaluation of the sensing constraints and surface sampling densities between a view volume and a surface patch, instead of between a single view point and a single surface point at a time. The straightforward, inefficient, single-view-point-to-single-surface-point approach is used in almost every previous view planning algorithms [2, 3, 6, 7, 8]. The purpose of this paper is to describe the algorithmic details of the hierarchical approach, and present efficient algorithms that evaluate sensing constraints and surface sampling densities between a view volume and a surface patch. The presentation here provides examples for the design of efficient algorithms for new acquisition constraints.

Section 2 presents a summary of the material in [5]. The reader is encouraged to refer to the paper for more details. In Section 3, we present the details of the hierarchical view evaluation algorithm, and the efficient implementation of the individual constraint and sampling density evaluation algorithms. Section 4 presents some results of our hierarchical view evaluation. We conclude the paper in Section 5.

2. The Hierarchical NBV Algorithm

For our NBV algorithm, we have formulated a view metric and used an adaptive hierarchical method to efficiently evaluate the view metric for a large set of views. The view metric incorporates the reconstruction quality requirements and acquisition constraints.

Our view metric takes into account the following two reconstruction quality requirements.

(1) **Completeness.** The NBV algorithm tries to maximize the amount of surface area acquired so that the reconstructed model can be more complete.

(2) **Surface sampling quality.** Our algorithm tries to maximize the surface area that reaches a required surface sampling density.

Several acquisition constraints must be observed when planning a view of the scanner. Each constraint can be classified as one of the following types.

(1) **Positioning constraints.** The physical construction of the scanner and the capability of the positioning device can constrain the scanner's physical position. A view that satisfies all the positioning constraints is called a *feasible view*.

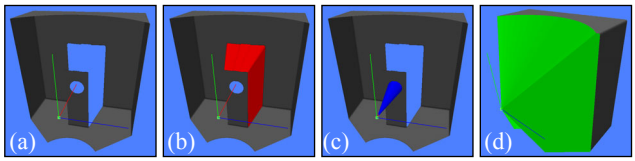

Figure 2. Different types of surfaces in a partial model. (a) True surfaces. (b) Occlusion surfaces (red). (c) Hole-boundary surfaces (blue). (d) Image-boundary surfaces (green).

(2) **Sensing constraints.** These constraints determine whether a surface point in the scene can be measured from a view. For example, a surface point cannot be measured by the scanner if it is not visible from the range sensor.

(3) **Registration constraint.** Due to positioning error of the scanner, each new scan has to be explicitly aligned to the previous scans. However, this registration is not guaranteed to be successful. Our view planning algorithm can ensure that the new scan to be acquired from the planned view can be successfully registered with the previous ones. However, we will not discuss the formulation and evaluation of the registration constraint in this paper.

2.1. Partial Model

The partial model consists of the acquired surfaces (called *true surfaces*) and three types of *false surfaces*: (1) *occlusion surfaces*, (2) *hole-boundary surfaces*, and (3) *image-boundary surfaces*. They are shown in Figure 2. These false surfaces are added to connect holes caused by occlusions, missing samples, and range image boundaries, respectively. These surfaces enclose a volume of known empty space. The false surfaces provide clues to how the unknown volumes can be resolved by subsequent scans.

One partial model can be merged to another of the same environment by performing the union of their known empty volumes and the union of their true surfaces.

In our implementation, the partial model is represented using an octree. All surface types and empty space are represented.

2.2. View Metric

Our view metric is shown in Eq. (1), where $h(v)$ is the score of view v.

$$h(v) = f(v) \cdot r(v) \cdot \int_{p \in S} c(v,p) \cdot w(p) \cdot t(v,p)\, dp \qquad (1)$$

where

- S is the set of all surface points in the current partial scene model; it includes all true and false surfaces;
- $f(v)$ is 1 if view v is a feasible view, otherwise $f(v)$ is 0;
- $r(v)$ is 1 if the registration constraint is satisfied at view v, otherwise $r(v)$ is 0;

- $c(v,p)$ is 1 if all the sensing constraints between view v and surface point p are satisfied, otherwise $c(v,p)$ is 0;
- $w(p)$ is the weight or importance value assigned to the surface type (false or true surface) of p;
- $t(v,p)$ is the improvement to the recorded sampling density at p if a scan is made from view v.

We use the following definition for $t(v,p)$.

$$t(v,p) = \max(0,\ \min(s(v,p),\ D) - q(p)) \qquad (2)$$

where

- $s(v,p)$ is the sampling density at p if it is scanned from view v; this is referred to as the *new scan sampling density*;
- D is the sampling density requirement for all surfaces;
- $q(p)$ is the maximum sampling density at which p has been scanned previously; this is referred to as the *recorded sampling density*; if p is on a false surface, then $q(p)$ is 0;

2.3. Algorithm Overview

Our strategy to evaluate $h(v)$ for all the views is to evaluate Eq. (1) in pieces, from least to most expensive to compute. Figure 3 shows the major steps in the evaluation of $h(v)$. We first evaluate $f(v)$ for all views to eliminate the infeasible views. Next, we use our hierarchical view evaluation method to evaluate the integral part of Eq. (1) for all the feasible views. The feasible views are then ranked by their current scores. Starting from the highest-score view, the registration constraint function, $r(v)$, is evaluated. The first view found to satisfy the constraint is output as the next best view.

To support the hierarchical view evaluation, surface voxels in the partial octree scene model are grouped into planar patches. The planar patches are then ranked in descending order of importance, so that the most important ones can be used to evaluate the views first.

Computing feasible views. An octree is used to represent the feasible view volumes. This *feasible view octree* contains a subset of the empty space in the cumulative partial model, and it is "carved" out using the positioning constraints.

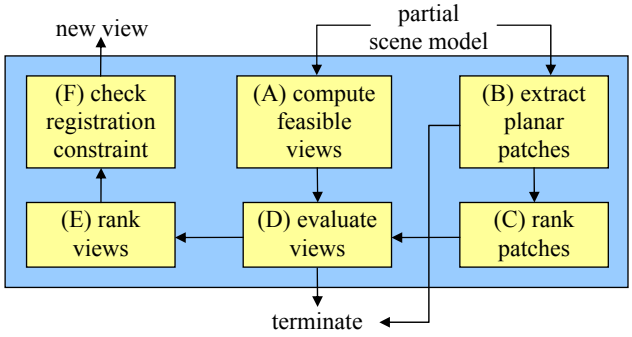

Figure 3. The major steps in the NBV algorithm.

Extracting planar patches. Surface voxels that have not reached the required sampling density are grouped into planar patches. Each patch has the following attributes: (1) a bounding rectangle, (2) an approximate surface area, (3) the average recorded sampling density, and (4) the *sampling deficit*. The sampling deficit is defined as the number of samples needed to make the average recorded sampling density equal to the sampling density requirement D.

Ranking patches. The most important patch should have the greatest potential impact on the value of the view metric in Eq. (1). This leads to the following: the *patch importance value* of P = $w(P)$ × sampling deficit of P, where P is the patch, and $w(P)$ is the weight assigned to the surface type of P, which is the same as the weight $w(p)$ in Eq. (1). The patches are then sorted in descending order on their patch importance values.

2.4. Hierarchical View Evaluation

Let $g(v)$ be the integral part in Eq. (1), i.e.

$$g(v) = \int_{p \in S} c(v,p) \cdot w(p) \cdot t(v,p)\, dp \qquad (3)$$

The next step of the NBV algorithm is to evaluate $g(v)$ for all the feasible views. Due to the potentially large area of surfaces in the partial scene model, a brute-force approach would be impractical. However, the amount of computation can actually be reduced by exploiting the spatial coherences in the sensing constraints and the sampling quality function.

The idea is that if a constraint is satisfied between a view v and a surface point p on the partial model, very likely the same constraint is also satisfied between another view v' and p, provided v' is near to v. The same constraint is also very likely to be satisfied between v and another surface point p' that is near p. We exploit these spatial coherences using a hierarchical approach. Neighboring views are first grouped into view volumes, and neighboring surface points are grouped into surface patches. The constraint is evaluated between each view volume V and a patch P. If it is entirely satisfied or entirely not satisfied between V and P, then the constraint evaluation is considered completed between every view in V and every surface point in P. If the constraint is partially satisfied between V and P, then we subdivide either V or P, and continue the evaluation on the children.

2.4.1. Formulation

Suppose all the false surfaces and under-sampled true surfaces in the partial model have been partitioned into N patches $\{P_i \mid i = 1,\ldots,N\}$, then Eq. (3) can be rewritten as

$$g(v) = \sum_{i=1}^{N} g(v, P_i) \qquad (4)$$

where

$$g(v,P) = \int_{p \in P} c(v,p) \cdot w(p) \cdot t(v,p)\, dp \qquad (5)$$

Now, we will focus on evaluating views with respect to a patch P, instead of with all the surface area in the partial model. Suppose the values of $c(v,p)$ and $t(v,p)$ remain constant between a view volume V and a patch P, where $v \in V$ and $p \in P$, then $g(v,P)$ can be computed as

$$g(v,P) = g(V,P) = c(V,P) \cdot w(P) \cdot t(V,P) \cdot a(P) \qquad (6)$$

where

$$t(V,P) = \max(0,\ \min(s(V,P),\ D) - q(P)) \qquad (7)$$

and $c(V,P)$, $w(P)$ and $s(V,P)$ are similarly defined as $c(v,p)$, $w(p)$ and $s(v,p)$; $a(P)$ is the patch area of P, and $q(P)$ is the average recorded sampling density of P.

In actual fact, the value of $s(v,p)$ does not stay constant between V and P. However, if every $s(v,p)$ between V and P is bounded within a small interval, then we consider it *approximately constant*. The value of $s(v,p)$ between V and P is considered approximately constant if

$$\frac{s_{\max}(V,P) - s_{\min}(V,P)}{s_{\max}(V,P)} \le \varepsilon_s \qquad (8)$$

where $s_{\min}(V,P)$ and $s_{\max}(V,P)$ are the minimum and maximum $s(v,p)$ between V and P, respectively. We have chosen to let $s(V,P) = s_{\min}(V,P)$, and compute $g(V,P)$ using Eq. (6). If any sensing constraint is found entirely not satisfied between V and P, then $s(V,P)$ need not be computed and $g(V,P) = 0$.

If $c(v,p)$ is not constant or $s(v,p)$ is not approximately constant between V and P, then we cannot compute $g(V,P)$ using Eq. (6). We can subdivide either V or P, and apply Eq. (6) on the sub-volumes or the sub-patches. If patch P is subdivided, then

$$g(V,P) = g(V,P^1) + \cdots + g(V,P^k) \qquad (9)$$

where $P^1,\ldots,P^k$ are the sub-patches of patch P. If view volume V is subdivided, then $g(V,P)$ is replaced with $g(V^1,P),\ldots,g(V^m,P)$, where $V^1,\ldots,V^m$ are the sub-volumes of V. In this case, $g(v,P) = g(V^i,P)$ if $v \in V^i$. The subdivision stops when $c(v,p)$ is constant and $s(v,p)$ is approximately constant between the view volume and the patch.

3. Hierarchical View Evaluation Algorithm

The following describes the algorithmic details and implementation of the hierarchical view evaluation to evaluate the integral part of the view metric.

It is assumed that the range sensor is *monostatic*, and all the samples in a range image are measured from a single center of projection or viewpoint. This assumption is true for many commercial mid-range and long-range laser scanners that use *time-of-flight* range sensing. Many of such scanners also have 360° horizontal FOV but limited vertical FOV. Since we assume that the scanner is always in the upright orientation, each view of the scanner is effectively only a 3D position. We use the feasible view octree to represent the 3D view volumes.

In our implementation, $c(v,p)$ consists of four separate sensing constraints:

(1) The *maximum-range constraint*, represented by $c_0(v,p)$. If the distance between view v and surface point p is more than the maximum effective range of the range sensor, then $c_0(v,p) = 0$, otherwise $c_0(v,p) = 1$.

(2) The *vertical-field-of-view constraint*, represented by $c_1(v,p)$. If the surface point p is outside the vertical field of view of the scanner at view v, then $c_1(v,p) = 0$, otherwise $c_1(v,p) = 1$.

(3) The *angle-of-incidence constraint*, represented by $c_2(v,p)$. If the angle between the surface normal vector at p and the direction vector from p to v is greater than a threshold angle, then $c_2(v,p) = 0$, otherwise $c_2(v,p) = 1$.

(4) The *visibility constraint*, represented by $c_3(v,p)$. If the line of sight from v to p is occluded, then $c_3(v,p) = 0$, otherwise $c_3(v,p) = 1$.

Consequently, the binary function $c(v,p)$ is defined as $c(v,p) = c_0(v,p)\ c_1(v,p)\ c_2(v,p)\ c_3(v,p)$. Similarly, $c(V,P)$ is made up of $c_0(V,P)$, $c_1(V,P)$, $c_2(V,P)$, and $c_3(V,P)$, where $c_i(V,P) = 1$ if $c_i(v,p) = 1$ for all $v \in V$ and $p \in P$, or $c_i(V,P) = 0$ if $c_i(v,p) = 0$ for all $v \in V$ and $p \in P$, otherwise $c_i(V,P)$ is undefined. Intuitively, when $c_i(V,P)$ is undefined, it means that the corresponding constraint is only partially satisfied between V and P.

In Figure 4 is a simplified C-like procedure to evaluate $g(V,P)$. Here, input viewcell V is a feasible view volume. Each input Boolean element `C_in[i]` is `true` if $c_i(V,P)$ is already known to be 1, otherwise `C_in[i]` is `false` to indicate that $c_i(V,P)$ is unknown. The input Boolean argument `S_const` is `true` if the relative error of $s(V,P)$ is known to be bounded by ε_s. The input argument `S` is $s(V,P)$ if `S_const` is `true`. Initially, the procedure `EvaluateView()` is called with all `C_in[i]=false` and `S_const=false`.

The function `EvaluateConstraint(i,V,P)` evaluates $c_i(V,P)$ and returns 0 or 1 to indicate $c_i(V,P) = 0$ or $c_i(V,P) = 1$, respectively, or returns any other integer values to indicate $c_i(V,P)$ is undefined. The function `EvaluateSamplingDensity(V,P,`

`&SMin,&SMax)` evaluates the minimum and maximum new scan sampling densities between V and P. The details of both functions are described in the subsections below.

In our implementation, a viewcell is subdivided into eight equal sub-viewcells, whereas a patch is subdivided into four sub-patches by splitting its bounding rectangle into four equal parts. A viewcell is not subdivided if it has reached the *minimum viewcell size*. Similarly, a patch is not subdivided if it has reached the *minimum patch size*. When either a viewcell or a patch is to be subdivided, we subdivide the patch if its longer side is larger than the viewcell's width, otherwise the viewcell is chosen.

It is important to note that when $c_i(V,P)=1$, it is also true that $c_i(V,P^k)=1$ and $c_i(V^m,P)=1$ for all sub-patches P^k of P and all sub-viewcells V^m of V. Therefore, when $c_i(V,P)=1$, and `EvaluateView()` is called with the sub-patches P^k or the sub-viewcells V^m, there is no need to recompute $c_i(V,P^k)$ and $c_i(V^m,P)$. This is similar for $s(V,P)$, when its relative error has been determined to be bounded by ε_s. This important observation can eliminate a large amount of computation since once a constraint is determined to be 0 or 1 for V and P, it needs not be evaluated anymore for their descendents.

After all patches have been evaluated (or the allotted view evaluation time is up), a viewcell's score is not yet propagated down to its children. Since each child viewcell contains part of the view volume of its parent viewcell, the scores in the children should include the parent's score. Therefore, the score of each viewcell should be updated by adding to it the scores of its ancestors. The center point of each leaf node of the feasible view octree is a *candidate view*, to be ranked and tested for the registration constraint. The first candidate view that satisfies the registration constraint is chosen as the best view for the next scan.

```
EvaluateView( Viewcell *V, Patch *P,
              bool C_in[4], bool S_const, float S )
{
  bool C[4] = { C_in[0], C_in[1], C_in[2], C_in[3] };

  for ( int i = 0; i < 4; i++ )
    if ( !C[i] )
    {
      int t = EvaluateConstraint( i, V, P );
      if ( t == 0 ) return;
      if ( t == 1 ) C[i] = true;
    }

  if ( !S_const )
  {
    float SMin, SMax;
    EvaluateSamplingDensity( V, P, &SMin, &SMax );

    if ( ( SMax - SMin ) / SMax <= epsilon_S )
    {
       S_const = true;
       S = SMin;
       if ( MIN( S, D ) - q(P) <= 0 ) return;
    }
  }

  if ( C[0] && C[1] && C[2] && C[3] && S_const )
  {
    V->score += w(P) * ( MIN( S, D ) - q(P) ) * a(P);
  }
  else if ( ToSubdividePatchFirst( V, P ) )
  {
    SubdividePatch( P );
    for ( int k = 0; k < P->numChildren; k++ )
      EvaluateView( V, P->child[k], C, S_const, S );
  }
  else
  {
    SubdivideViewcell( V );
    for ( int m = 0; m < V->numChildren; m++ )
      EvaluateView( V->child[m], P, C, S_const, S );
  }
}
```

Figure 4. A procedure to evaluate *g*(*V*, *P*).

3.1. Constraint and Sampling Density Evaluations

This section describes the implementation of `EvaluateConstraint()` and `EvaluateSampling Density()` to evaluate $c_i(V,P)$ and $s(V,P)$, respectively. The success of the hierarchical view evaluation depends on how efficiently they can be evaluated.

In actual fact, `EvaluateConstraint(i,V,P)` need not evaluate $c_i(V,P)$ precisely, in the sense that when `EvaluateConstraint(i,V,P)` returns 1 or 0, it implies that $c_i(V,P)=1$ or $c_i(V,P)=0$, respectively, but the inverse implication need not be true. When $c_i(V,P)=1$ or $c_i(V,P)=0$, `EvaluateConstraint (i,V,P)` may return undefined. This is preferred when it is expensive to precisely determine whether $c_i(V,P)=1$ or $c_i(V,P)=0$. By returning undefined, the precise evaluation of the constraint is left to the sub-patches of P or the sub-viewcells of V, and because of their smaller sizes, they are more likely to belong to one of the easy cases. Of course, when $c_i(V,P)$ is undefined, `Evaluate Constraint(i,V,P)` must return undefined.

For the same purpose, `EvaluateSampling Density(V,P,&SMin,&SMax)` need not return the precise minimum and maximum new scan sampling densities between V and P. It is allowed to underestimate the minimum new scan sampling density and overestimate the maximum new scan sampling density.

The following sections describe the algorithms. There are certainly some other efficient ways to accomplish these operations, but these provide examples for the implementation of new constraints. When V is indivisible or P is indivisible, where they are treated as points, the algorithms are generally trivial, so they are not described here.

3.1.1. Maximum-Range Constraint

Let the maximum effective range of the range sensor be $R_{\max}$. When $c_0(V,P)=1$, the distance between any point in patch P and any view in V is equal to or less than $R_{\max}$. To determine this, four imaginary spheres of radius $R_{\max}$ are centered at the four corners of the patch's bounding rectangle. If the entire viewcell V is inside all the four spheres, then `EvaluateConstraint(0,V,P)` returns 1. The viewcell can be approximated with a bounding sphere to speed up the computation. If the viewcell (or its

bounding sphere) intersects or is inside some but not all the four spheres, undefined is returned.

When $c_0(V,P)=0$, the distance between any point in patch P and any view in V is greater than $R_{\max}$. This can be determined as follows. Let C be the imaginary convex hull of the four spheres of radius $R_{\max}$ that are centered at the four corners of the patch's bounding rectangle. If the viewcell is entirely outside the convex hull C, then $c_0(V,P)=0$. By approximating the viewcell with a sphere, it is not hard to efficiently determine whether the sphere is outside the convex hull. When the viewcell (or its bounding sphere) has been determined to be outside the convex hull, `EvaluateConstraint(0,V,P)` returns 0. For all other cases, `EvaluateConstraint(0,V,P)` returns undefined to indicate that the actual value of $c_0(V,P)$ is still uncertain, or that the constraint is satisfied by only some, but not all, pairs of views and patch points.

3.1.2. Vertical-Field-of-View Constraint

The scanner is assumed to have a 360° horizontal FOV, but a limited vertical FOV, as shown in Figure 5. To determine whether a surface point is within the vertical FOV, we compute the angle between the y-axis (vertical axis) and the vector from the view position to the surface point. If the angle is less than θ_{top} or more than $180°-\theta_{\text{bot}}$, then the surface point is outside the vertical FOV.

When $c_1(V,P)=0$, every point in patch P is outside the FOV of every view in V. Figure 6 illustrates a method to determine whether $c_1(V,P)=0$. If the directions of all the four directed lines in Figure 6(a) are in the bottom outside region of the vertical FOV, then `EvaluateConstraint(1,V,P)` returns 0. Similarly, in Figure 6(b), if the directions of all the four directed lines are in the top outside region, `EvaluateConstraint(1,V,P)` also returns 0. If some of these directed lines are inside and some are outside the vertical FOV, then `EvaluateConstraint(1,V,P)` returns undefined.

Figure 7 illustrate how to determine whether $c_1(V,P)=1$. If all the four corners of the patch's bounding rectangle are on the positive sides of both planes A and B, then the patch is entirely inside the vertical FOV of every view in the viewcell and the value to be returned by `EvaluateConstraint(1,V,P)` is 1. Otherwise, `EvaluateConstraint(1,V,P)` returns undefined.

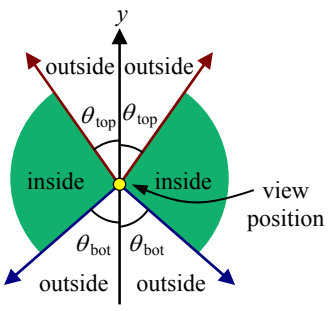

Figure 5. The vertical field of view of the scanner.

3.1.3. Angle-of-Incidence Constraint

The angle of incidence, ϕ, of a surface point p from a view position v is the angle between the surface normal vector at p and the direction vector from p to v. If this angle is greater than a threshold angle $\phi_{\max}$, then $c_2(v,p)=0$, otherwise $c_2(v,p)=1$.

To determine whether $c_2(V,P)=1$, four open-ended cones are set up at the four corners of the patch's bounding rectangle as shown in Figure 8. The base of each cone extends infinitely in the direction of the patch's normal vector, and the half angle at the apex of each cone is $\phi_{\max}$. If the viewcell V (or its bounding sphere) is entirely inside all four cones, then $c_2(V,P)=1$, and `EvaluateConstraint(2,V,P)` returns 1. If the

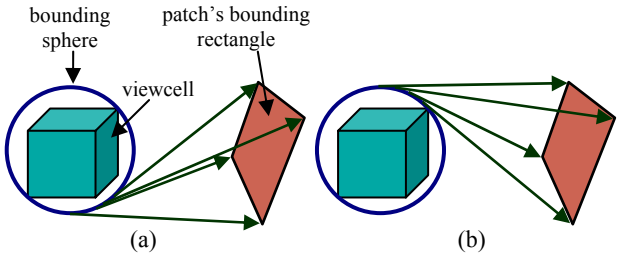

Figure 6. (a) The four directed lines are tangent to the sphere and point at the four corners of the patch's bounding rectangle. Each directed line touches the sphere at the lowest point where it is still tangent to the sphere. If the angles between the *y*-axis and all the directed lines are larger than $180°-\theta_{\text{bot}}$, then the patch is entirely in the bottom outside region of the vertical FOV of the viewcell. (b) Each of the four directed lines touches the sphere at the highest point where it is still tangent to the sphere. If the angles between the *y*-axis and all the directed lines are less than θ_{top}, then the patch is entirely in the top outside region of the vertical FOV of the viewcell.

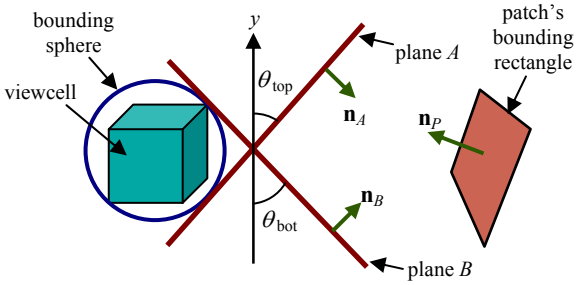

Figure 7. Determining whether a patch is entirely inside the vertical FOV of every view in the viewcell. Planes *A* and *B* are both tangent to the bounding sphere of the viewcell. The planes' normal vectors $\mathbf{n}_A$ and $\mathbf{n}_B$ are coplanar with the normal vector $\mathbf{n}_P$ of the patch. If all the four corners of the patch's bounding rectangle are on the positive side (the side where the normal vector is pointing) of both planes *A* and *B*, then the patch is entirely inside the vertical FOV of every view in the viewcell.

viewcell intersects or is inside some but not all four cones, then `EvaluateConstraint(2,V,P)` returns undefined.

To determine whether $c_2(V,P)=0$, the viewcell V (or its bounding sphere) must be entirely outside the open-ended convex hull that encloses all the four cones in Figure 8. In this case, `EvaluateConstraint (2,V,P)` returns 0, otherwise it returns undefined.

3.1.4. Visibility Constraint

Here, we are testing the visibility between a viewcell and a rectangle-bounded patch.

When $c_3(V,P)=1$, it implies the viewcell and the patch are *totally visible* to each other. To determine that, one has to ensure that there is no occluder in the *shaft* between the viewcell and the patch, which is the 3D volume occupied by the line segments connecting every point in the viewcell to every point on the patch. To determine $c_3(V,P)=1$, bounding planes are constructed to enclose the shaft between the viewcell and the patch's bounding rectangle. The non-empty-space voxels in the partial octree model are then tested, in a top-down traversal, against all the bounding planes to find out if any of the voxels intersects the volume bounded by the bounding planes. If not, `EvaluateConstraint (3,V,P)` returns 1.

When $c_3(V,P)=0$, the viewcell and the patch are *totally invisible* to, or *totally occluded* from, each other. Determining *total invisibility* or *total occlusion* between two extended objects is a difficult problem because the total occlusion may be caused by multiple occluders that are not connected to each other [1].

In the absence of an efficient algorithm, we have chosen to use a probabilistic approach to estimate total occlusion. The method is illustrated in Figure 9. On the viewcell's bounding sphere, the great circle parallel to the patch is first identified. Then, an equal number of random points is generated in each quadrant of the disc bounded by the great circle, and in each quadrant of the patch's bounding rectangle. Rays are shot from the random points on the disc to the points on the patch's bounding rectangle. 16 random rays are generated in this way. If all the random rays are occluded, then it is estimated that the patch is totally occluded from the viewcell, and `EvaluateConstraint(3,V,P)` returns 0. Otherwise, it is assumed that the patch is *partially visible*

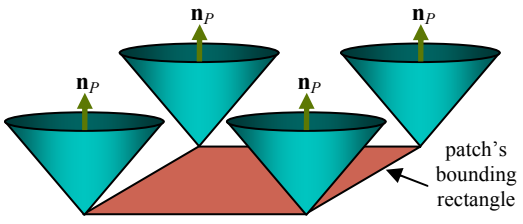

Figure 8. The four open-ended cones set up at the four corners of the patch's bounding rectangle. $\mathbf{n}_P$ is the normal vector of the patch.

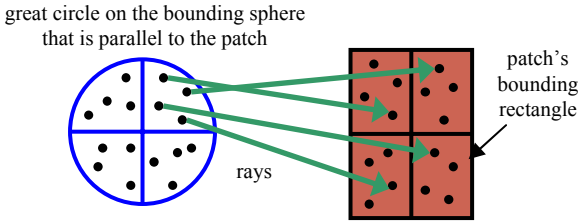

Figure 9. Generating random rays from the viewcell's bounding sphere to the patch's bounding rectangle to estimate total occlusion.

from the viewcell, and `EvaluateConstraint (3,V,P)` returns undefined. The hierarchical structure of the partial octree model is exploited to accelerate the determination of whether a ray intersects any non-empty-space voxels.

There is no ill-effect when total occlusion is erroneously declared as partial visibility, except that it may cause an unnecessary subdivision of the viewcell or the patch. On the other hand, it may be undesirable when partial visibility is erroneously declared as total occlusion, since the patch will be disregarded. However, since the method is probabilistic, the "missed" patch might still be reconsidered in a later acquisition cycles.

3.1.5. New Scan Sampling Density

The function `EvaluateSamplingDensity(V,P, &sMin,&sMax)` outputs the minimum and the maximum new scan sampling densities between V and P. Let α be the angle interval between two successive samples acquired by the range scanner, and r be the distance from the view position v to the surface point p. The surface sampling density around point p is $s(v,p)=d=\cos\phi/\alpha r$ where ϕ is the angle of incidence of the laser beam at surface point p.

When the values of α and d are fixed, the locus of the view position is the surface of a sphere with radius $1/(2\alpha d)$, and the sphere is tangent to the surface at p. All points inside the sphere have sampling densities greater than d, and points outside have sampling densities less than d. We call it the *sampling density sphere* of p.

Since the function `EvaluateSamplingDensity()` is allowed to under-estimate the minimum sampling density, it is sufficient to construct, for each corner of the patch's bounding rectangle, the smallest sampling density sphere that encloses the viewcell, and let S be the largest of the four spheres. The minimum sampling density from the viewcell V to the patch P is estimated as $1/(2\alpha R)$, where R is the radius of S. The viewcell may be approximated by a bounding sphere.

The function `EvaluateSamplingDensity()` is allowed to over-estimate the maximum sampling density. Let p be the point on the patch's bounding rectangle that is closest to the viewcell's bounding sphere. Then, let S be the largest sampling density sphere of p that touches a point of the viewcell's bounding sphere but does not enclose the bounding sphere. The maximum sampling density from the viewcell V to the patch P is estimated as

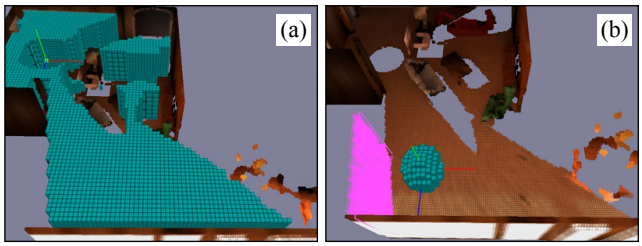

Figure 10. (a) The feasible view volumes to be evaluated. (b) The results of evaluating the patch (magenta) with the feasible view volumes. The best 500 viewcells are shown.

$1/(2\alpha R)$, where R is the radius of S.

4. Results

Figure 10(a) shows the feasible view volumes computed for a partial model of a large living room. These feasible view volumes are then evaluated with a patch shown in magenta color in Figure 10(b). The resulting best 500 viewcells are shown. The minimum viewcell size used is 4×4×4 inch^3, and the minimum patch size is 2×2 inch^2. A brute-force method took 259.6 seconds to evaluate the feasible views with the patch, while our hierarchical algorithm took just 11.9 seconds—a difference of more than 20 times. Typical speedups for indoor scenes are between 10 to 100 times. Generally, larger and simpler scenes, and smaller minimum viewcell and minimum patch sizes result in larger relative speedups.

We have tested our NBV planning system in simulations and on real scenes. The scanners used in both cases have only 3D translational poses, and have full horizontal FOVs but limited vertical FOVs. The simulated scanner has pose errors, and produces range data with range errors, drop-outs and outliers. Figure 1(a) shows the computed view plan for acquiring scans of a synthetic living room. The acquisition was manually terminated after the eighth scans. Figure 1(b)&(c) show the polygonal models reconstructed from the eight range images. Every cycle, the hierarchical view evaluation was able to evaluate almost all the patches with all the feasible views within the allotted two minutes.

Figure 11 shows the acquisition of a real building interior using a DeltaSphere-3000 laser scanner. The acquisition process was manually terminated after the fifth scan. Figure 11(a) shows the view plan. Every cycle, more than 75% of the patch areas can be evaluated with all the feasible views. This experiment shows that our NBV planning system is robust for real-world applications.

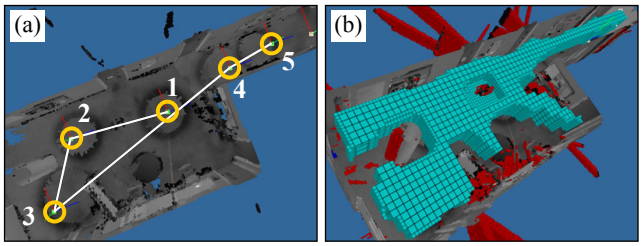

Figure 11. (a) The view plan computed for a real scene. (b) The final partial model and the feasible view volume.

5. Conclusion

Our hierarchical view evaluation method has made exhaustive 3D view evaluation for greedy NBV planning practical. This is mainly due to the evaluation of the sensing constraints and surface sampling densities between a view volume and a surface patch, unlike previous NBV algorithms, which simply evaluate between a single view point and a single surface point at a time. We have presented efficient algorithms to evaluate the individual sensing constraints and sampling quality between view volumes and surface patches. The descriptions also serve as examples for the design of efficient algorithms for new acquisition constraints.

Acknowledgements

We thank John Thomas, Herman Towles, Lars Nyland, and 3rdTech Inc. for their technical help, and thank Andy Wilson and the UNC Walkthrough Group for the beautiful house model. This work is supported by NSF grant number ACI-0205425.

References

[1] Daniel Cohen-Or, Yiorgos Chrysanthou, Claudio Silva, and Frédo Durand. A Survey of Visibility for Walkthrough Applications. IEEE Transaction on Visualization and Computer Graphics, 9(3):412–431, July 2003.

[2] C. I. Connolly. The Determination of Next Best Views. Proceedings of IEEE International Conference on Robotics and Automation, pp. 432–435, 1985.

[3] H. González-Baños, E. Mao, J.-C. Latombe, T. M. Murali and A. Efrat. Planning Robot Motion Strategies for Efficient Model Construction. Proceedings of International Symposium on Robotics Research, pp. 345–352, 1999.

[4] Pat Hanrahan, David Salzman, Larry Aupperle. A Rapid Hierarchical Radiosity Algorithm. ACM SIGGRAPH Computer Graphics (Proceedings of SIGGRAPH '91), 25(4):197–206, July 1991.

[5] Kok-Lim Low and Anselmo Lastra. An Adaptive Hierarchical Next-Best-View Algorithm for 3D Reconstruction of Indoor Scenes. Technical Report TR06-003, Department of Computer Science, University of North Carolina at Chapel Hill, January 2006.

[6] N. A. Massios and R. B. Fisher. A Best Next View Selection Algorithm Incorporating a Quality Criterion. Proceedings of British Machine Vision Conference, 1998.

[7] R. Pito. A Sensor Based Solution to the Next Best View Problem. Proceedings of IEEE International Conference on Pattern Recognition, pp. 941–945, 1996.

[8] J. M. Sanchiz and R. B. Fisher. A Next-Best-View Algorithm for 3D Scene Recovery with 5 Degrees of Freedom. Proceedings of British Machine Vision Conference, 1999.