Generating a Complete and Economical Set of Images from a Scene

Faisal Zubair Qureshi

November 12, 2001

1. Introduction

In recent years, computer graphics have generated a lot of interest. This interest is partly the result of the applications of computer graphics with in the realms of entertainment, medicine, and data visualization etc. Computer graphics primarily deals with the synthesis of images. In the case of 3D objects, one scheme to generate such images is 3D model-based rendering. Here, some sort of 3D representation of the scene is stored. At the time of rendering, this information is used to project the scene on the view-plane from a particular viewpoint. The process of projection has to be repeated for every new image of this scene. This process is complex. The time it takes to generate an image from the scene increases with the scene complexity. Generation of photo-realistic images adds an other dimension to the complexity. Not only that, for some scenes (natural or outdoor scenes) it is non-trivial to generate realistic 3D models that can be used to generate images of these scenes. These problems prompted researchers to look for other means to generate images. One such scheme is Image-based Rendering (IBR).

1.1. Image-based Rendering

In IBR, the information about the scene is stored as a set of images. This information is then used to generate novel images (from different viewpoints) of the scene. A primary difference between IBR and 3D model-based rendering is that the later stores the 3D information about the scene (usually in the form of sets of polygons and texture maps etc.) whereas the former only stores information about the scene in the form of pictures.

The main advantage of IBR is that the time it takes to generate a novel image from a set of images is independent of the scene complexity[1]. As a consequence, we can deal with scenes of arbitrary complexities without any penalty[2]. IBR techniques also readily yield themselves to generating photo-realistic images. As a matter of fact, how photo-realistic the novel image is depends solely on the quality of the set of images that describe the scene; hence, there is no cost associated with generating photo-realistic images unlike the 3D model-based rendering techniques.

One of the main issues in IBR is how to choose the set of images that completely describes the scene without any redundancy. We want this set to exhibit the follow two properties.

1. Completeness—none of the important details of the scene should be lost.
2. Economical—should not have redundant information.

It happens that finding such a set is a non-trivial problem, and in general, it is up to the discretion of the person involved, to choose this set of images for a given scene. In this project, I have attempted to automate this process. In the next section, I will present my algorithm (at the time of developing this algorithm I was not aware of any related work; however, recently some work has been done to address this problem).

Our problem is very similar to the art gallery problem introduced by Victor Klee in 1973. This problem is, "Where will you place the cameras in a way that the whole gallery is guarded? How many cameras are required? What is the minimum number of cameras required to view every point in the gallery?". A simple case of this problem treats the art gallery as a simple polygon and assumes that the field of view of each camera is 360 degrees. The solution of this problem is straightforward and it is known that for a polygon of n vertices, we need at most floor$\left(\frac{n}{3}\right)$ cameras. Our problem, though similar to the art gallery problem, is much harder. For one thing, we are working in 3D, and for another, we have limits on the various parameters of our cameras like near and far plane and field of view etc.

[1]Ofcourse it depends upon the number of images we are dealing with.

[2]Taking a picture of a complex real scene is almost instantaneous with the current digital or analog cameras. Even for a synthetic scene, in which case we first need to generate images using 3D model rendering to use IBR techniques, we save time; since, we just need to generate a representational set of images and then can use IBR techniques to generate novel images. In general, rendering an image esp. a photorealistic image using 3D model-based rendering techniques takes much longer than generating a novel image using IBR techniques.

2. Algorithm

This algorithms tries to compute a set of view points and the corresponding camera parameters. Images are generated (in the case of synthetic scenes) or taken (in the case of natural scenes) from these view points. We believe that this set of images is both complete and in some sense economical (it might be possible to come up with a smaller complete set of images although it is not obvious how to achieve this). The completeness of this set is ensured by guarantying that each point on a surface in the scene will be visible in at least one of the images (the algorithm can enforce constraints such as "each surface in the scene be visible from at least n different view point"—this might be useful for generating a 3D model of the scene from these images using stereo-vision techniques.). We also guarantee that the viewing angle for any surface in the scene is within a certain threshold.

The economical nature of this set is supported by our greedy approach to prune redundant images. The guarantee that the set of images will remain complete under the removal of redundant images is embedded in our definition of redundancy. We define an image to be redundant if every surface(patch) visible in this image is also visible in at least one other image (this definition of redundancy can be trivially modified to enforce other constraints as the user might deem fit.).

2.1. Input

1. 3D model of the scene. Although, in the current implementation, the scene is represented as a polyhedra, this is in no way a limitation of the algorithm. The algorithm just requires a way to sample points on the various surfaces in the scene. For synthetic scenes 3D models are readily available. But even for a natural scene (like an outdoor scene) it is not hard to construct a coarse 3D model—it is of course extremely hard to generate an accurate 3D model.

2. Other parameters like limits on cameras field of view, camera near and far planes etc.

3. Constraints such as 1) viewing angle constraint or 2) each point in the scene be visible from 2 cameras constraint etc.[3]

[3]In theory we can also enforce volume constraints. This type of constraints will only allow camera locations to be within a certain region. In the current implementation we have forced

2.2. Output

A set of view points with the corresponding camera parameters. Pictures taken or rendered from these view points form the complete and economical set of images for the IBR algorithms. IBR techniques can be used to generate novel images from this set of images.

2.3. Scheme

```
1: for all vertices $v$ of the scene do
2:     Generate a camera $c$ whose optical axis is along the surface normal $n$ at
       vertex $v$.
3:     Place camera at $p$ such that $nearplane \leq \sqrt{(v-p)^2}$
4:     Move camera away from $v$ along $n$ so as to maximize the number of vertices
       viewed from this camera. (Optimization Step 1)
5:     Move camera $c$ left and right and choose an angle that maximize the number
       of vertices viewed from this camera. (Optimization Step 2)
6:     Move camera $c$ up and down and choose an angle that maximize the number
       of vertices viewed from this camera. (Optimization Step 3)
7:     Adjust camera's $c$ field of view to increase the scene's project on the view-
       plane. (Optimization Step 4)
8: end for
9: Generate a bi-partite graph. One set of nodes represent the cameras and the
   other set of nodes represent the vertices in the scene. The nodes in the sets
   have an edge if a vertex in the first set is visible from a camera in the second
   set.
10: Apply minimum-dominating set algorithm on the graph generated in the pre-
    vious step. (I have implemented a greedy algorithm for finding the minimum
    dominating set in bipartite graphs). This algorithm will prune away redun-
    dant cameras.
11: Only keep cameras corresponding to the remaining camera nodes in the graph.
    Delete all other.
```

Optimizations 1, 2, 3 can be seen as the search for a camera's parameters like distance from the vertex and direction so as to maximize the number of vertices viewed from this camera. Optimizations 4 ensures that the projected image covers most of the viewplane's render area. A vertex is viewed from a camera if it satisfies

the camera locations to lie on the normal of a vertex.

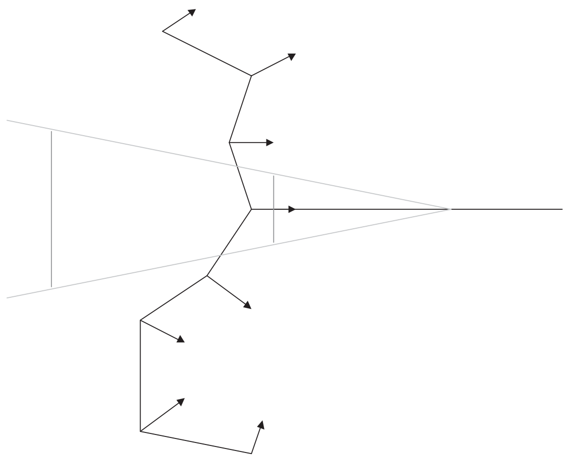


Figure 2.1: For each vertex a camera is created with the default properties (field of view, near and far plan etc.). The optical axis of this camera is aligned with the normal of the vertex.

following conditions

1. Visibility condition: the vertex is not hidden from the camera and it falls within the camera's view volume.

2. Angle condition: the angle between surface normal at the vertex and the camera's optical axis is with in certain threshold.

3. Complexity

The minimum-dominating set algorithm is NP-Complete. We therefore suggest intelligent sampling of points on the scene surfaces. The first part of the algorithm (steps 1-8), the time complexity is $O(n^2k)$. Here, n is the number of vertices and k is the maximum iterations allowed for Optimization steps (steps 4-7). The most costly step is evaluating the visibility condition. Algorithms exists that can speed up the evaluation of this condition.

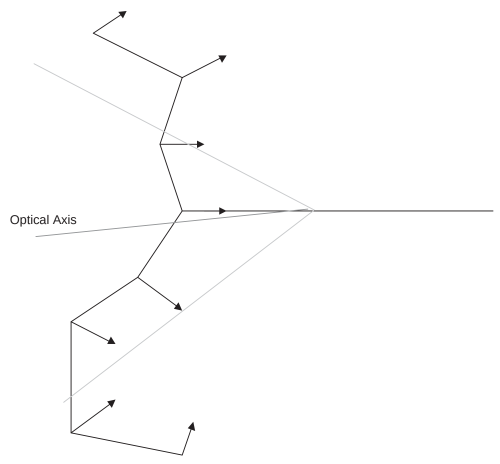


Figure 2.2: The parameters of this camera are tweaked to maximize the vertex cover. Essentially we change camera's position, direction, and field of view to view as much of the scene's surfaces as possible without compromising any constraints.

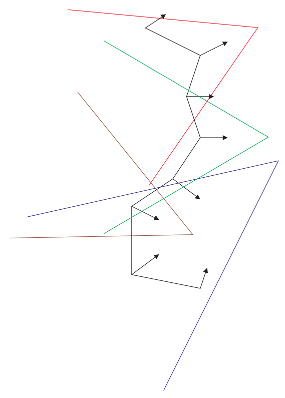


Figure 2.3: We end up with n cameras. We then use minimum dominating set algorithm to get rid of the redundant cameras.

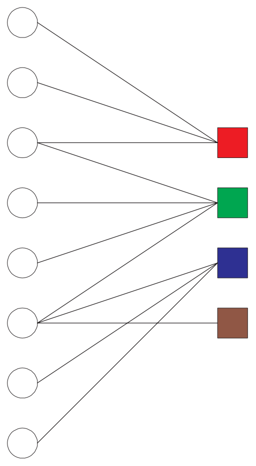


Figure 2.4: The bi-partite graph corresponding to figure 2.3

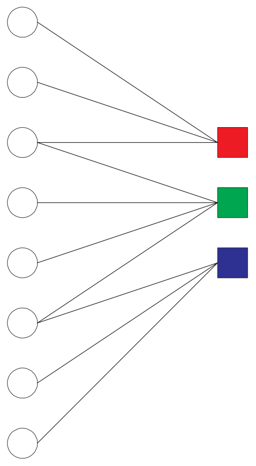


Figure 2.5: After applying minimum dominating set algorithm over the graph shown in figure 2.4, we are left with only three cameras.

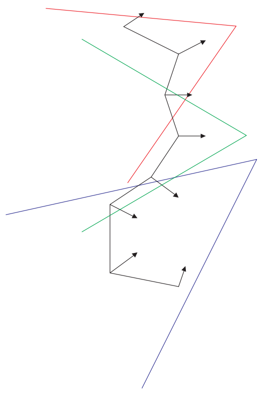


Figure 2.6: After deleting the redundant camera (brown) we are left with only three cameras.

4. Implementation

The system is implemented in C++ using Qt[4], Tcl/Tk and Coin3D[5] libraries . Once the scene is loaded, the systems computes the locations and parameters of the cameras. Images can then be rendered from these camera locations using 3D model-based rendering techniques. These images form the complete and economical set of images for the scene. Novel images can be generated from this set.

5. Results

5.1. Test 1

In this test, we attempt to generate a small number of images that completely represent the room (in the sense that every surface is visible in at least one of the images). As clear, the algorithm successfully choose three camera positions from which each surface is visible (Figure 5.1). Once the cameras locations are selected, user can then generate images using these cameras. Although we have not modified the camera positions and directions selected by the algorithm, the

[4]Qt is a platform independent MFC like gui library. It is available for both windows and linux/unix platforms.

[5]Coin3D is a free Open Inventor clone.

user can easily modify one of the camera parameters to further improve the quality the images' set.

```
% surface loadmn roominside room1.mn;
% surface show test;
% viewall;
% set camera fovdeg 120;
% set camera nearplane 0.1;
% set camara farplane 1.8;
% set camera aspect 1;
% set adjustdir 1;
% set adjustfov 1;
% set testangle 1;
% set testvisible 1;
% minview roominside;
Number of camera (Before MinDomSet) = 13
Making graph applying minimum dominating set algorithm.
Number of camera (After MinDomSet) = 3 Savings = 10
% camera showall;
% viewall;
```

5.2. Test 2

If we set the outside flag to be true.

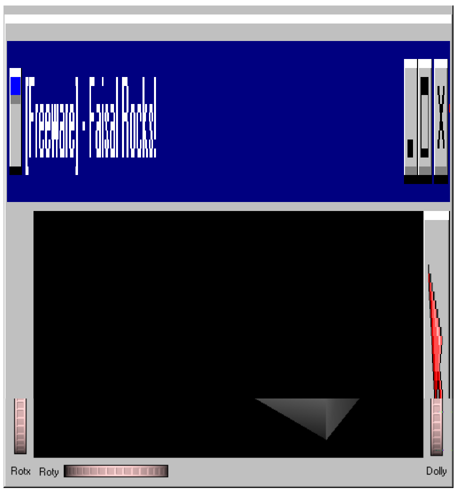
```
% set side outside;
% minview roominside;
Number of camera (Before MinDomSet) = 13
Making graph applying minimum dominating set algorithm.
Number of camera (After MinDomSet) = 2 Savings = 11
```

We get two cameras that view the outside of the room as shown in Figure 5.2.

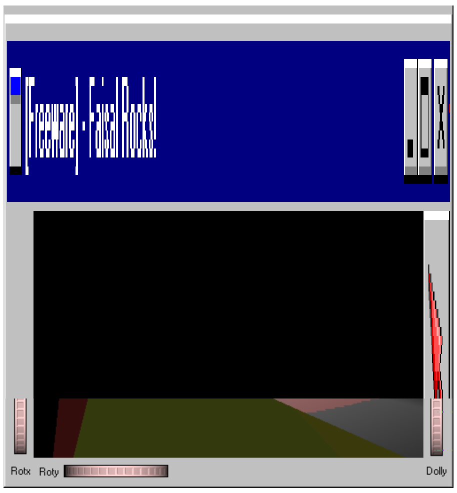
5.3. Test 3

Figure 5.2 show that one of the camera is viewing the floor from beneath. We can get of rid of this problem surface regions by putting constraints on the camera positions or by identifying surfaces which contain no useful information.

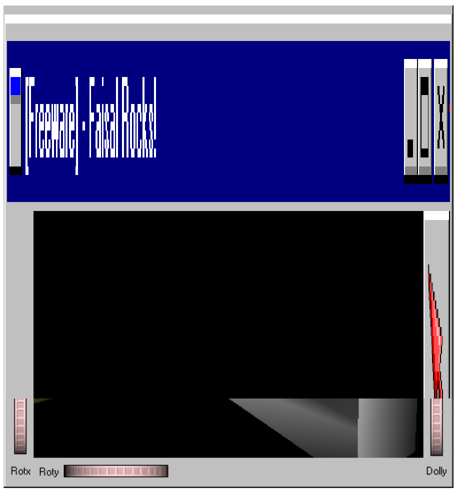
```
% camera positiony 0 1; // constrains the camera y-position to lie
between 0 and 1.
% minview roominside;
```



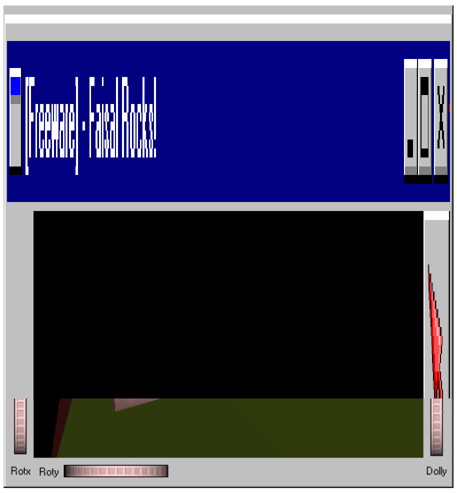
(a) The room



(b) Camera 1

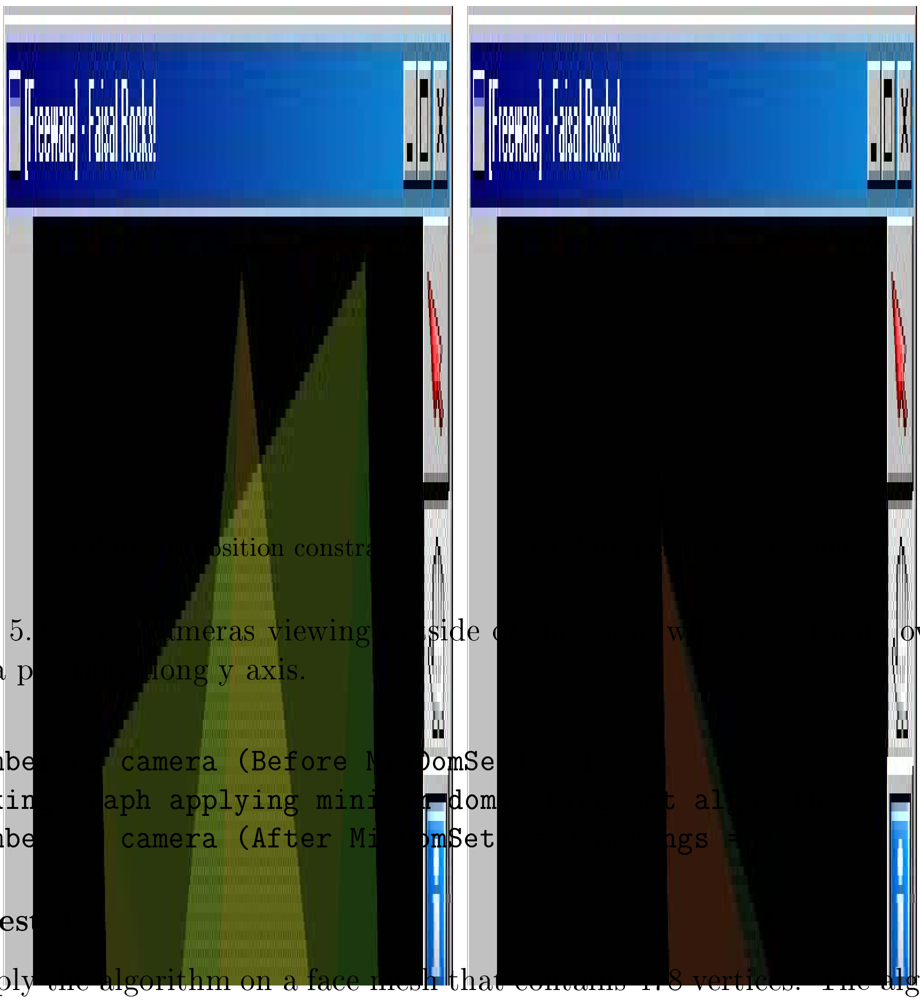


(c) Camera 2



(d) Camera 3

Figure 5.1: Applying the above algorithm on the room shown in (a) gives us 3 cameras. The images rendered with these cameras forms a representational set of images.



[illegible] position constra[illegible]

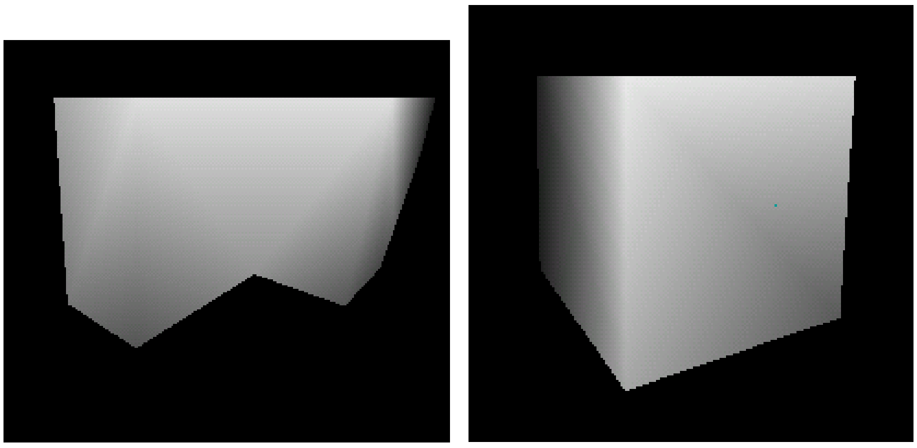
Figure 5.[illegible]meras viewing [illegible]side [illegible] over the camera p[illegible]along y axis.

```
Numbe[illegible] camera (Before M[illegible]omSe[illegible]
Makin[illegible]aph applying mini[illegible]dom[illegible]t al[illegible]
Numbe[illegible] camera (After Mi[illegible]omSet[illegible]ngs =[illegible]
```

5.4. Tes[illegible]

We apply [illegible] algorithm on a face [illegible] that [illegible] vertices. [illegible]algorithm comes up with two camera positions from which all of these vertices are viewed Figure 5.4. It is interesting to point out that the same face mesh is used by Lee [2] to model the human faces. It means that we can use this algorithm to come up camera locations required to completely view a human face. Following scripts runs the algorithm.

```
% surface loadmn face generic_face_mesh.mn;
% surface show face;
% viewall;
% set camera fovdeg 120;
% set camera nearplane 3;
% set camara farplane 15;
```



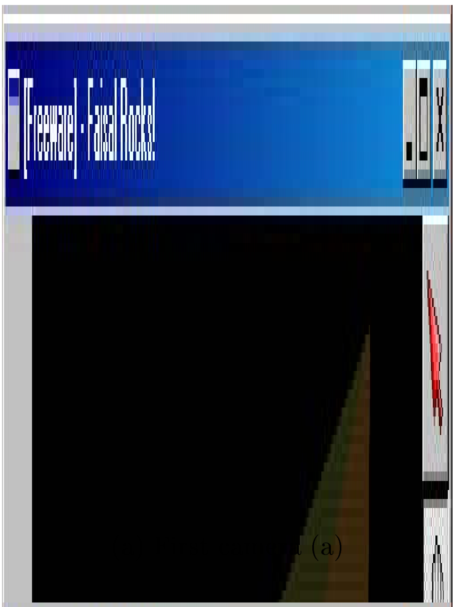
(a) Image from the first camera (b) Image from the second camera

Figure 5.3: Images rendered from the two cameras positioned outside the room. The camera positions satisfy the position constraints.

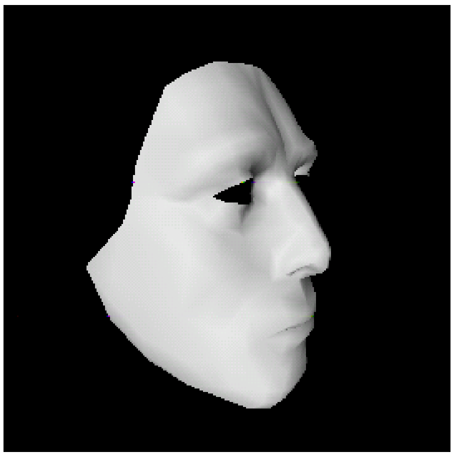
```
% set camera aspect 1;
% set adjustdir 1;
% set adjustfov 1;
% set testangle 1;
% set testvisible 1;
% minview face;
Number of camera (Before MinDomSet) = 438
Making graph applying minimum dominating set algorithm.
Number of camera (After MinDomSet) = 2 Savings = 436
% camera showall;
% viewall;
```

6. Future Directions

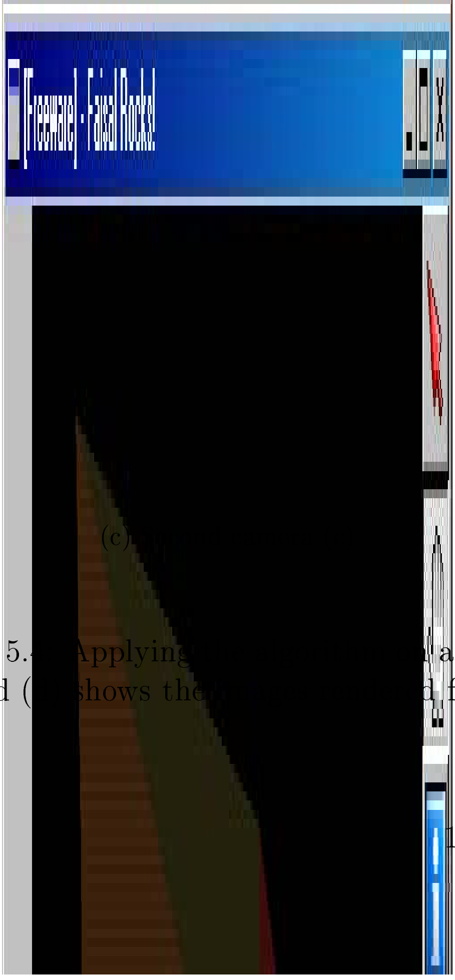
In this project we have just attempted to develop an algorithm for generating a representational set of images that can be used by IBR techniques to generate novel images. The current implementation is restricted in the sense that instead of sampling points on the scene's surfaces, it uses the vertices. The performance



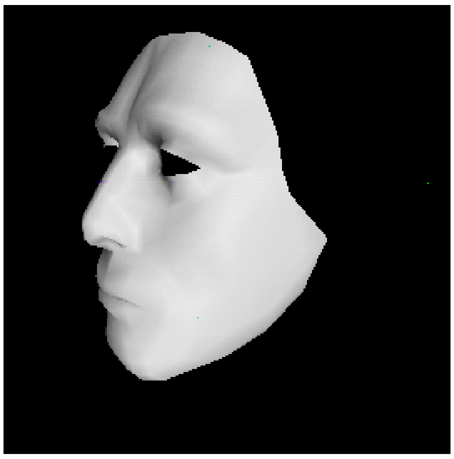
(a)



(b) Image from the first camera (b)



(c)



(d) Image from the second camera (d)

Figure 5.4 Applying a face mesh. We get two cameras. Figures (b) and (shows the d from these camera positions.

of the algorithm can be improved tremendously if we use smaller number of vertices. We therefore suggest that intelligent sampling of the surface that takes into account surface curvature and camera parameters will yield improvements in algorithm time characteristics. We have not implemented an IBR algorithm to generate novel image of the scene but we believe that the set of generated images should be sufficient for this purpose.

The algorithm can be readily applied to computing optimum light source positions (both directional and otherwise) such that the whole scene is illuminated.

References

[1] J. D. Foley, van D. Andreas, S. Feiner, and J. K. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley, 1990.

[2] Y. V. Lee. The construction and animation of functional facial models from cylinderical range reflectance data. Master's thesis, University of Toronto, Department of Computer Science, 1993.

[3] N. Megiddo and U. Vishkin. On finding a minimum dominating set in a tournament. *Theoretical Computer Science*, 61:307–316, 1988.

[4] J. O'Rourke and G. In. *Computational Geometry in C.* Cambridge University Press, 1994.

[5] J. Urrutia. Art gallery and illumination problems. In *Handbook on Computational Geometry.* 1997.

[6] B. Welch. *Practical programming in Tcl & Tk.* NJ, 2 edition, 1997.

[7] J. Wernecke. *The Inventor Mentor.* Addison-Wesley, 1994.