中華民國八十六年全國計算機會議

環境漫遊系統中之影像式三維物體瀏覽器
An Image-Based 3D Object Viewer for Environment Walk-Through

Jiunn-Jia Su, Shyei-De Lee, Zhi-yi Chaio, Ming Ouhyoung
蘇俊嘉 李協德 邱志義 歐陽明
Communications and Multimedia Lab.
Dept. of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan, R.O.C.

摘要

本文提出一個以影像為基礎的 3D 物件瀏覽系統，使用者可以利用本系統來觀看 3D 物件。本系統是利用拍攝所得的照片加以內插處理，然後根據使用者的控制即時播放，讓使用者能自由觀看 3D 物件。另外本文中也提出兩個物件編輯工具，讓物件製作過程變的更簡單。目前我們系統除了 Windows95 單機程式之外，也支援 Netscape Navigator 的 plug-in 。在 WWW 上更可以把本系統的物件瀏覽能力發揮到極致。

Abstract

This paper presents an image-based 3D object viewer system, Object Viewer, for environment walk-through. With this system, a user can interactively observe an object from different directions in real time, as if he moves or rotates the object with his own hands.

Our method is first to get source images of an object from some pre-selected viewing directions. Then intermediate images are interpolated from these source images according to manually-selected matching features. The actual display sequence contains both the source images and the intermediate ones. Furthermore, to build up an object model file for the object viewer, we need to remove the background from the source images, mark up matching features, and solve the occluding area problem for the interpolated intermediate images.

We have implemented the system on Windows 95 both as a stand-alone application and as a Win 32 plug-in module for Netscape Navigator. We also implemented two authoring tools that can help to remove the background from the source images and to mark up matching features. Object Viewer has been applied in several cases, and can be used in Internet applications, like virtual malls.

1. Introduction

Traditionally, computer graphics or virtual reality systems are mostly model-based. However, a major problem of the model-based system is the trade-off between rendering time and rendering quality. To get around this problem, some image-based methods are used. There are some advantages for these image-based systems, including constant rendering time and higher rendering quality. In addition, sometimes it is difficult to represent objects by using polygon models, for example, a hairy teddy bear. Therefore, we chose to develop an object viewer system using image-based rendering. This object viewing system, or Object Viewer, allows a user observe an object interactively from different directions in real time. Combined with other environment walk-through systems, e.g; "Photo VR" [TSAO96A; TSAO96B; TSAO96C], Object Viewer would be very useful in virtual mall applications.

2. System Organization

2.1 System Architecture

The method used in our Object Viewer system is view interpolation with matching features, which is somewhat similar to the method used by Chen and Williams [CHEN93]. However, our method uses selected features to interpolate intermediate images from original image data, so no depth information or range data are required.

The whole object viewer system can be divided into two major phases: the object model construction phase and the object model display phase. In the object model construction phase, source images must be collected first, then these source images need to be processed to remove the background and to identify matching features between two adjacent ones; the object model display phase includes the display of source images and interactively interpolated intermediate images. The architecture of the object viewer system is shown in Figure 1.

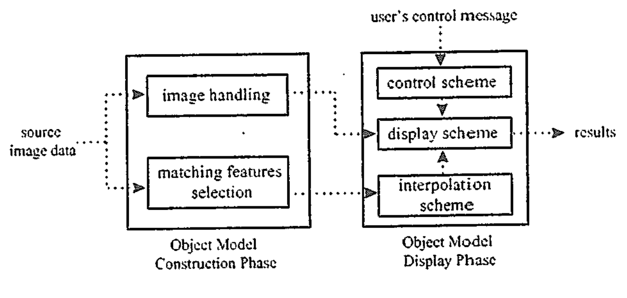

Fig 1 System Architecture

2.2 The Acquirement of Image Data

The first step to construct an object model is the acquirement of the source image data. The source images can be captured by a camera or a camcorder, and must cover all desired view directions, as shown in Figure 2:

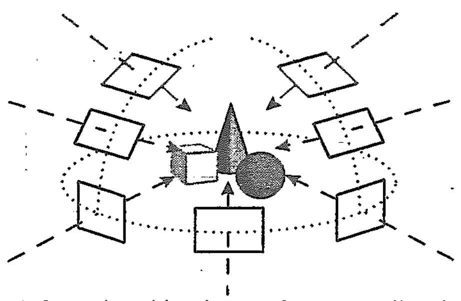

Fig 2 Capturing object images from some directions

The basic idea of our system is to show these source images sequentially according to the viewing direction of a user. The display process will be similar to the rotation of an object. The more source images used, the better result will be because the difference between two adjacent images are small. However, a large volume of data will take too much memory and disk space. One alternative method stated is: only a few source images are obtained from some pre-selected positions, say, every 30 degrees, and then some intermediate images interpolated are used to simulate the transition process between two source images.

2.3 Image Data Handling

Image data handling includes two tasks: the background removal and the selection of matching features. We have implemented tools for both of these tasks.

The reasons for background removal are:
1. Sometimes there may be a lot of textures or other objects on the background, so the viewers can not concentrate themselves on the foreground (the object that we want to show them).
2. If we fix the object and move our camera to different positions to get the source images, the change (or the movement) of the background between two adjacent source images will be greater than the one of the foreground (since it is further away from the camera). If the background is not removed, the display process will not be natural and smooth.

We have developed a useful tool to aid the users to do background removal task. We call this tool as **boundary detector and removal**. With this tool, a user can determine the boundary of an object easily. The determined boundary data can also be used to detect the boundaries in adjacent images automatically.

The second task is to select matching features of the two adjacent images. These selected features are used to interpolate intermediate images from source images. The matching features of each two adjacent source images are stored in files separately. These matching features are the basic components of the interpolation of intermediate images, so they strongly affect the quality of intermediate images. We also developed tools to help users to deal with this task. After processing, these images are then combined to form an object model file which will be used in the display phase.

2.4 Display Method

In the object model display phase, which is the major part of the object viewer system, the system retrieves data from the model file and from corresponding image files and matching features files, and then displays the source images and intermediate images according to the user's viewing direction interactively.

The control method allows a user to treat the mouse cursor as his own hand. When the user moves the cursor, Object Viewer will show the corresponding images. The result will be similar to the rotation of objects. Thus, a user can rotate an object by a cursor just as he rotates the object by his own hand.

We have implemented the object viewer system on two different platforms, one is a Microsoft Windows 95 stand-alone application and the other is the Win 32 plug-in module for Netscape Navigator.

3. Boundary Detector

In the above, we have mentioned that sometimes the source image data must be handled for background removal. For such a task, the boundary between the foreground (the object) and the background must be obtained. It will be a hard and boring task for a user to select the boundary manually. Therefore, we implemented a tool that can aid the user in the detection of the object boundary. Furthermore, since the difference of two adjacent source images may be small, the boundaries of these two images also be similar. So, our tool can use the detected boundary of the first images to aid the boundary detection of the other ones. We named this tool boundary detector.

The basic idea of boundary detector is referred to "Intelligent Scissors for Image Composition," as proposed by Mortensen and Barrett [MORT95], which uses dynamic programming for boundary detection. Boundary detection via dynamic programming can be formulated as a graph searching problem where the goal is to find an optimal path from the start node to the goal node. Each pixel in the image represents a node of the graph, and an edge between two nodes of the graph represents a link between two adjacent pixels in the image. Here, each link is assigned a local cost value, and optimality is defined as the minimum cumulative cost path from a start node to a goal node. As applied to the image boundary detection, the graph search consists of finding a globally optimal path from the start pixel to the goal pixel.

Since a minimum cost path should correspond to a boundary, pixels that exhibit strong edge features should be assigned low local costs. The local cost function is the combination of components created from various edge features, including the strength and the direction of the edge sections.

After the local cost function is defined, the dynamic programming can be used to work out the graph search problem. First, a start or seed pixel must be determined. Then, a desired boundary from the start pixel, which is the minimum cost path at the same time, can be choose dynamically via a free point. The user can move the mouse cursor, and the position of the cursor in the image will be treated as the free pixel, and the boundary from the start pixel to the free pixel can be choose interactively. Thus, based on these algorithms, an interactive tool can be implemented.

Our tool can aid the user to determine the boundary of a single image. The user select a group of pixels, and our tool will find out boundary sections between them. If all boundary sections forms a close loop, the images will be divided into different regions. Some of them are background, and can be removed.

Since the boundary sections of two adjacent images are similar, the determine boundary of one image can be applied to the detection of the boundary of the adjacent images. Our tool can load the previous detected boundary, and map it to the second image with some processing. The user will not need to choose all the points again. What the user need to do is to fix some error parts. Thus, the human time is much saved. The user can create a batch process to handle a sequence of images. After the batch process is completed, the user correct the error parts. Since the error rates are low, the user will not spend a lot of time to fix them.

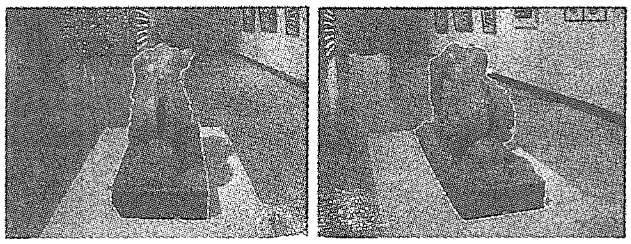
Fig 3 Two adjacent source images with boundary determined

The above, as Figure 3 shows, is the boundaries of two adjacent images, where the boundary of the second image is chosen based on the boundary of the first one. Figure 4 is the results after the background parts are removed.

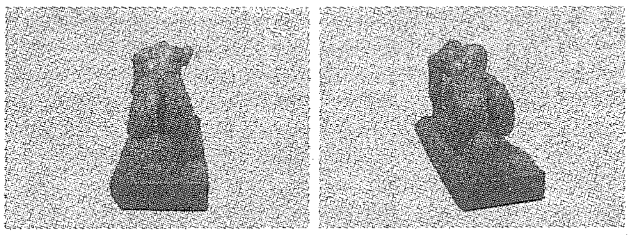
Fig 4 Two images that have been removed background

4. Intermediate Images

To reduce the amount of source images without maiming the smoothness of the display results, the intermediate images interpolated from the source images are necessary.

We use matching features to interpolate intermediate images from the source ones. These matching features are selected by the user. A key point we must emphasize here is that the intermediate images are not the exact images from the views within the original views. They are just used to approximate the images from those views.

4.1 The Extraction of Matching Features

The matching features used by the object viewer to interpolate intermediate images are selected by the user. These features are stored as the form of triangles. First, the user must select corresponding points of the two images. To help the user to select these corresponding points precisely, we strongly recommend that these points should be feature points such as corners or edges. It will be easier for the user to select corresponding points from these cases.

After the corresponding points are selected, the next work is to combine three selected points to form a feature triangle. These feature triangles are the basic components of the interpolation process. Thus, the selection of the feature triangles will highly affect the qualities of interpolated intermediate images. We strongly recommend that the following points must be noticed when selecting these feature triangles:
1. For a large region, the triangles should be arranged radially.
2. If there are complex textures on the object, it would be better to cover the whole portion of the textures with a single triangle.
3. Smaller triangles will get better interpolation results.
4. The triangles must cover the whole area of the object in the image, or there will be leaks in the intermediate images.

4.2 Interpolating Intermediate Images

The method used to interpolate the intermediate images is somewhat similar to morphing and texture-mapping. In short, the method used is to get the feature triangles from the two source images, transform them to the new shapes and new positions in the intermediate images, then fill them with the image data of the corresponding source image pixels.

The method to interpolate the intermediate images is a bi-directional interpolation method, where the shape and the texture data of the intermediate image come from both the source images. The major challenge for the generation of intermediate images is the handling of occluding areas. The occluding areas occur when two different parts of the object are projected onto the same position in one of the two source images, but to different positions on the other. In such a situation, only the part which is nearest the camera is shown in the occluding image. In other words, the image information of the occluded part, such as color data and position data, is lost in the occluding image. If the object is only moved but not rotated, the occluding problem seldom occurs. However, the occluding problem often occurs in the rotation case.

Two forms of error will be derived from this problem; one is the overlap, and the other is the hole. The overlap means that two pixels in the source image will move to the same position in the intermediate image, and the hole means that no pixel on the source image can be used to fill up a certain position in the intermediate image. We use the following way to solve the occluding problem: when an occluding part appears in one source image but disappears in the other one, the feature triangle of the occluding part is assigned normally in the appearing source image; but in the other source image, the feature triangle would be assigned to be a narrow area, or a line, or even a point. In interpolation, when the system finds that there is large disparity between the areas of the two feature triangles, it will interpolate the shape data in the intermediate image from both source images, but interpolate the color data from only one of the source images. The other source image will not be used because the image data of the occluding part is lost in the other source image. The following is an example.

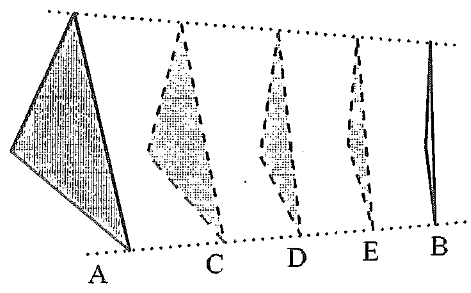

Fig 5 A, B are source triangles and C, D, E are interpolated results

As shown in Figure 5, A and B are corresponding feature triangles in different source images, and C, D, and E are interpolated triangles in intermediate images. The area of A is far more than the area of B. Thus, the shape data of C, D, and E are interpolated from both A and B, but their color data are generated from only triangle A.

Figure 6 are two adjacent source images of an object model. After selecting feature triangles, intermediate images are generated, as shown in Figure 7 and Figure8 respectively.

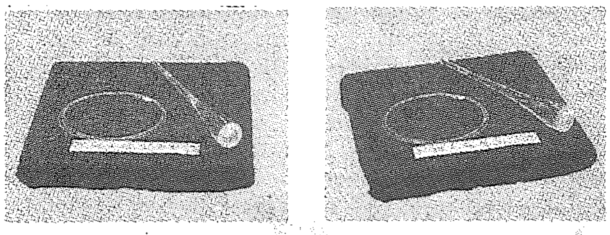

Fig 6 Two adjacent source images

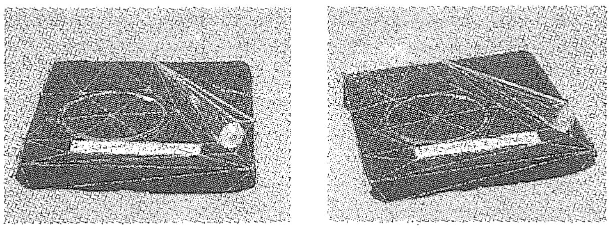

Fig 7 Two adjacent images with features selected

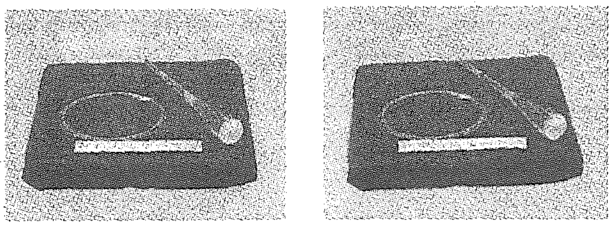

Fig 8 Intermediate images

5. Plug-in Module Implementation

We have implemented our Object Viewer system both as a Windows 95 standalone application and as a Nescape plug-in module. Plug-ins are software programs that extend the capabilities of Netscape Navigator in a specific way. In other words, users can execute plug-ins on Internet via the browser. By such a method, our Object Viewer system can be executed not only in Windows 95 desktop but also on Netscape Navigator platform.

When a user opens a page containing Object Viewer plug-in, Navigator will start up the plug-in. If the plug-in has been installed correctly, the user now can view objects freely the same as in Object Viewer Windows 95 standalone system.

Because of involvement of network, implementing a plug-in module is more difficult than implementing a standalone application. These problems include how to retrieve data stored on the remote server, the browser dependency problem, the cache problem and so on.

Data retrieval problem

A standalone application retrieves data and consumes them on the same local machine. However, a plug-in module retrieves data on the server via Internet and consumes them on the client machine. That is, when the plug-in module needs image data, it must issue a data request to the remote server and wait for the data, instead of reading these data directly from the local disk.

Retrieving data from server brings some problems. Because of waiting for data stream from the server, the plug-in module sometimes stalls. If the network jams, users will spend a lot of time waiting. In addition, the sequence of issuing requests is not the same as that of responding data stream. This will introduce some troubles in programming.

In order to solve these problems, our plug-in module recognizes the responding data file, so the display sequence is correct. Object Viewer shows images that have been transmitted and allows a user to rotate object progressively. As more and more images are gotten, a user views objects more freely. Such a data transmitting method avoids long waiting time. In addition, because of using intermediate images in Object Viewer, the amount of the data that needs to be transmitted is little, and ,therefore, Object Viewer plug-in module is suitable for Internet.

Plug-in modules must access data on Internet via browsers such as Netscape Navigator, and so it is restricted in data handling. Plug-in SDK supports two modes of transferring data. One mode is transferring files between the server and the client. The other is transferring byte streams. However Microsoft Internet Explorer only supports streaming mode. In this mode, it is not convenient for the client plug-in module to utilize some utility libraries which treat a data file as a unit. Our object viewer plug-in uses the utility library to handle JPEG files, and uses file-transferring mode so that our plug-in module cannot yet be executed in Microsoft Internet Explorer until now.

Now we are developing our Object Viewer AtiveX Control for Internet Explorer and are approaching the completion stage.

Browser dependency problem

As the above said, plug-in modules are specific to Netscape Navigator. A plug-in module designed for running with Netscape Navigator doesn't always work in other browsers. Besides, plug-in programming must follow some rigid rules specified by Netscape. Plug-in modules have to fill a group of function place holders which is prefixed by NPP_. In order to write these place holders and to transfer parameters between them, the programming is quite restricted. In addition, since plug-in modules are dynamic linkage files and run in Navigator address space, it is very difficult to debug.

Cache problem

In order not to download pages every time and to speed up the browser responding time, Navigator always caches these pages. Similiarly, the data consumed by plug-in modules also need to be cached. In Netscape Navigator directory structure, there is a sub-directory called Cache, and this is where Navigator caches pages. To conform to the behavior of Navigator, our plug-in also caches data in this directory. However, the volume of this directory is restricted in Navigator option, and we have no rights to clear the cache. Once the amount of data requested exceeds the volume of Cache directory, Navigator will crash. Now this problem has been brought up as a bug of Plug-in SDK. But it will still give rise to some trouble in our plug-in module.

Combination with Photo VR on Internet

As mentioned, we have implemented another image-based system called Photo VR. Photo VR is an image-based environment walk-through system. It uses textured cylinder-like prisms or sphere-like polyhedrons as the environment maps, where the texture data are generated from photo-realistic images. The combination of both systems is via a natural way. When an object is allowed for the viewer to observe via Object Viewer, a link to the object model file is added on the scene. The object can be selected via the mouse, and will then be shown by Object Viewer.

The Photo VR also has a Win 32 plug-in module for Netscape Navigator, thus the combination can be put on the Internet. A user can first open a page containing a Photo VR plug-in, and then wander around in the virtual scene. When an object is selected, the Navigator forwards to another page containing the object viewer plug-in. After the user observing the object, he can close the page and backward to Photo VR.

Such a combination will be suitable for various applications on the Internet, such as a virtual mall. In the case of virtual mall, the whole scene can be a large shopping mall. Each shop can be represented as a prism or a polyhedron, and can be shown through Photo VR. Each exhibit item can be represented as an object, and be displayed via Object Viewer. The other types of media supported by WWW can provide useful assistance. For example, the text can provide the details about the introduction of the shop and the exhibit; the audio makes the whole page more attractive; the URL anchors provide the users a way to get more related information. Finally, the E-Mail function provides a way for the user to contact with the dealer. This may become an important part of the electronic commerce.

In addition to Photo VR, the object viewer plug-in module can also combine with other media types, such as VRML. Since the plug-in module is embedded in an HTML page, thus, the link to Object Viewer plug-in can be invoked via the way of the anchor. For example, the whole scene can be constructed by geometry models in VRML. Such a model may be rough, and some objects within the scene may need image data to give a user realistic impression. In such a case, links to the object viewer may be added onto these scene objects. Via these links, the user can invoke the object viewer to observe these objects more clearly.

Demonstration

The following are some snap shots of our object viewer system for Windows 95 stand alone application and plug-in module.

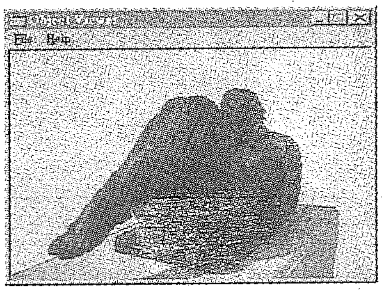

Fig 9 Our object viewer standalone application

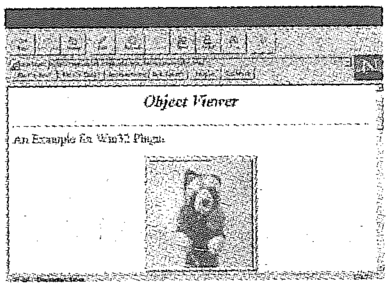

Fig 10 Object viewer win32 plug-in module

6. System Performance

In this section, we will discuss the system performance of Object Viewer. First, we will discuss the image volume data. If interpolation for intermediate images is used, we recommend that the a source image should be taken for every 30 degrees; otherwise, a source image should be taken for every 15 degrees or less. For general case, if the background is removed and JPEG file format is used, the file size of a source image with resolution 320*200 be no more than 30k bytes. In fact, the file size will be smaller most of time. Thus, for one latitude with 24 source images (one source per 15 longitude degrees), the total file size is about 700k, or less. Thus it is suitable for network transmission. We will take some real scenes as example. The raccoon contains 72 source images (3 latitudes, 24 images for each), with resolution 200*200. Its total file size is about 540k bytes. Another ant piece , theTransformation, which is a sculpture within the scene of Taipei Fine Art Museum, contains 24 source images (1 latitude), with resolution 320*240. Its total file size is about 440k bytes.

For two 320*240 source images, our system will need about 80 milliseconds to generate an intermediate image, on a computer with Pentium 133 CPU, 32M RAM. Thus the frame rate is about 15 frames per second (include source images and intermediate images.) If the intermediate images are not used, the frame rates is more than 25 frames per second.

Object Model Name	No. Of Longitude	No. of Latitude	Image Size	No. of Image	Volume of Total Image Data
Raccoon	24	3	200*200	72	538k
Transformation	24	1	320*240	24	439k
Sunflower	24	1	200*200	24	214k

7. Conclusions

This paper proposes an image-based system — Object Viewer — for users to interactively observe an object from different view directions in real time. This system can be executed on Microsoft Windows 95 as a stand-alone Windows 95 application or as a Win32 plug-in module for Nescape Navigator without any special hardware support. We have also developed authoring tools that can be used to remove the background of the source images and mark up the matching features.

The advantages of our methods are as follows:
1. Since source images are acquired from some pre-selected viewing directions, the volume of the image data is small. Therefore it is suitable for transmition through network.
2. We generate interpolated intermediate images to approximate the actual images for the in-between views between two adjacent source images.
3. The user can fully control the system, and the system will react interactively in real time.

Furthermore, this system can be combined with our Photo VR, which is an image-based environment walk-through system. The combination of Object Viewer and Photo VR would form a general purpose image-based environment walk-through system, and supports the new media type of image-based virtual reality for WWW. This will be a very useful tool for applications in entertainment, education, business, and so on.

8. Reference

[APPL95] Apple Computer, Quick Time VR software package, 1995.
[BEIE92] Beier, T. and Neely, S., "Feature-Based Image Metamorphosis," Proceedings of SIGGRAPH '92, Chicago, Illinois, 1992. In *Computer Graphics*, 26, 2 (July 1992), ACM SIGGRAPH, pp. 35–42.
[CHEN93] Chen, S.E. and Williams, L., "View Interpolation for Image Synthesis," Proceedings of SIGGRAPH '93, Anaheim, California, 1993. In *Computer Graphics* Proceedings, Annual Conference Series, 1993, ACM SIGGRAPH, pp. 279–288.
[CHEN95] Chen, S.E., "QuickTime® VR – An Image-Based Approach to Virtual Environment Navigation," Proceedings of SIGGRAPH '95, Los Angeles, CA, 1995. In *Computer Graphics* Proceedings, Annual Conference Series, 1995, ACM SIGGRAPH, pp. 29–38.
[MORT95] Mortensen, E.N. and Barrett, W.A., "Intelligent Scissors for Image Composition," Proceedings of SIGGRAPH '95, Los Angeles, CA, 1995. In *Computer Graphics* Proceedings, Annual Conference Series, 1995, ACM SIGGRAPH, pp.191–198.
[NETS] Netscape Communications Corporation, "Plug-In Guide,"
<URL:http://home.netscape.com/eng/mozilla/3.0/handbook/plugins/pguide.htm>
[SEIT96] Seitz, S.M. and Dyer, C.R., "View Morphing," Proceedings of SIGGRAPH '96, New Orleans, LA, 1996. In *Computer Graphics* Proceedings, Annual Conference Series, 1996, ACM SIGGRAPH, pp. 21–30.
[TASO96A] Tsao, W.K., Su, J.J., Chen, B.Y., and Ouhyoung, M., "Photo VR: A System of Rendering High Quality Images for Virtual Environments Using Sphere-like Polyhedral Environment Maps," *Proceedings of Real-time And Multimedia System*, Taipei, R.O.C., 1996, pp. 397–403.
[TSAO96B] Tsao, W.K., Chen, B.Y., Su, J.J., and Ouhyoung, M., "Rendering Real World Scenes and Objects for Virtual Environment Navigation," *Proceedings of International Computer Symposium*, Kaohsiung, R.O.C, 1996, pp. 1–8.
[TSAO96C] Tsao, W.K., *Rendering Scenes in the Real World for Virtual Environments Using Scanned Images*, Master thesis, Department of CSIE, NTU, 1996.
[Su97] Su, J.J., Master thesis, National Taiwan University, 1997.