Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

Rendering Real World Scenes and Objects for Virtual Environment Navigation

Wen-Kae Tsao Bing-Yu Chen Jiunn-Jia Su Ming Ouhyoung
Communication and Multimedia Labs.
Dept. of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan, R.O.C.

Abstract

In the first part of this paper, we describe a novel method of panoramic view rendering for real-time interactive applications trying to present a real-world like environment. The proposed method is different from the approach of QuickTime® VR, in that the scene is rendered by generating a sphere-like polyhedral environment map from photo-realistic images and using the generated maps to render the scene by techniques of computer graphics. In the second part, we purpose an image-based method for interactively observing a real-world object from arbitrary view point. Although some precision in alignment is lost, the high complexity, high quality and high frame rate performance on a Pentium PC without any hardware supported makes this approach attractive in low cost systems.

Introduction

In traditional geometry-based rendering system, a scene is generated by rendering a model composed of geometric elements, such as triangles, and describe the scene. However, this method usually works well only when the quantity and complexity of objects are relatively low, often far below those in the real world. Furthermore, the shading is either too slow or unrealistic. In recent years, some method of image-based rendering has been developed, such as "image warping" used by "QuickTime® VR" of Apple Computer [1][2], or texture mapping [3]. This method renders the scene by using cylinderic or prismatic environment maps generated from overlapping images taken with a regular camera. However, this method has a common problem: the environment map used does not cover the top view and the bottom view, hence the vertical viewing range is limited. In this paper, we propose a method that renders the scene by sphere-like polyhedral environment maps.

But, how about the objects in the scene? In the VR application, when an user walks in the rendered scene, if he wants to get some objects to observe more clearly, we must find another method to do this. The traditional method for observing an object from all views is to generate a polygon model of the object, and then the model is rendered from different view points. However, it is difficult or impossible to generate a model for some objects, such as a fuzzy teddy or a valuable artifact. We propose an image-based method to solve this problem.

Texture Mapping Approach

To render the scene in a room, we first put a camera in a proper position, such as the center of the room, and then take images of the whole view from the camera position. These images are arranged as a sphere-like polyhedron consisting of textured trapezoids. This polyhedron forms the environment map of the scene and is used as the 3D model for rendering. It is clear to see that, when moving around the center of this sphere-like polyhedron, the image presented to the user will approximate the one rendered by traditional method used in computer graphics or even the one seen in the real world if all objects in the room are a certain distance away from the center.

To render such a sphere-like polyhedron is quite easy: no shading required. No hidden surface problem occurs. (Polygonal) texture mapping is usually contained in rendering packages, libraries or graphic hardware accelerators. Thus high complexity, high quality and high frame rate become possible even on low cost PC systems.

To allow a user walk around the scene, one can generate several such sphere-like polyhedrons by taking images from several different positions, each responsible for a "visible area" in the scene. One polyhedron is rendered at a time, and only when the user is in the corresponding "visible area" will the polyhedron be rendered. Adjacent visible areas will be

overlapped instead of providing a clear border (Fig. 9), and if a user is in the overlapped area, only the previously rendered polyhedron will be rendered to avoid frequent switching of 3D models when the user moves around the border. We will describe this more clearly with an example later in this paper.

The switching between 3D models will be observed by the user, just like the switching of acts in a film. Because the switching is performed when the viewpoint is moving, it will not be so annoying to the user. However, it is a problem remaining unsolved by us yet. Linear interpolation is, perhaps, not a practical solution. Not only because the time for rendering will be at least doubled (for at least 2 cylinder-like prisms need to be rendered), but also because the interpolation may involve theories in computer vision: objects in images must be identified and located for interpolation. The complexity of these interpolated images taken from the two real environment images usually makes this process (object identification) too slow and even impossible. However, if geometric information of the scene is available and objects in the images are identified in advance, interpolation (such as morphing) might be possible.

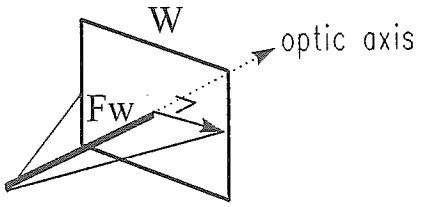


Fig. 1 $Fw = \frac{W}{2} cot \frac{horizontal\ viewing\ angle}{2}$

Image Registration

If we do not know the absolute orientation of the camera of each image, we can register the image by hand, or by corresponding points through the following method:

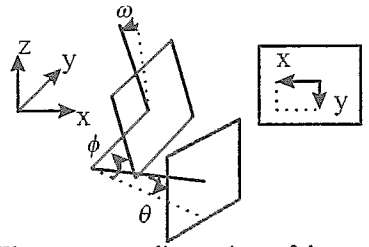


Fig. 3 The corresponding points of the two images

Generating the Sphere-like Polyhedron

Assumptions

We make some basic assumptions of our system:

1. Geometric distortion of the camera lens is negligible (ideal perspective projection).
2. The size, camera constants, horizontal and vertical view angles of different images are the same.
3. All objects in the scene are far enough from the camera such that the movement of the COP (center of projection) caused by panning the camera is negligible. Thus all optic axes of the images can be regarded as intersected at the COP.
4. Each image is perpendicularly intersected by optic axis at its center.

Camera Calibration

First, we have to measure the horizontal and vertical viewing angle of the camera. Then we convert horizontal viewing angle to the camera constant *Fw*, and convert vertical viewing angle to the camera constant *Fh*:

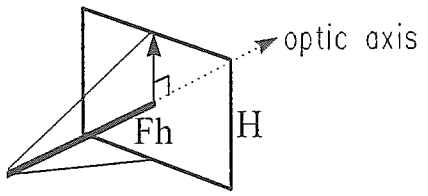


Fig. 2 $Fh = \frac{H}{2} cot \frac{vertical\ viewing\ angle}{2}$

If the corresponding points are:

(x_1,y_1) of image$_1$ = (x_2,y_2) of image$_2$

(x_3,y_3) of image$_1$ = (x_4,y_4) of image$_2$

where (x,y) are defined as displacements from center of pixmap (as the above figure).

We have to decide the three rotation angles:

θ: the paning angle

ϕ: the tilting angle

ω: the swinging angle

where all the three angles are from image 1 to image 2.

(1) Normalize $X_1 - X_4$:

$$X_i = X_i \times \left(\frac{Fh}{Fw} \right), i = 1...4$$

For convenient, in the following discussion, we will use *Fh* instead of using *Fw* and *Fh*.

(2) Define $r_1 = \frac{\sqrt{Fh^2 + x_1^2 + y_1^2}}{\sqrt{Fh^2 + x_2^2 + y_2^2}}$

$$r_2 = \frac{\sqrt{Fh^2 + x_3^2 + y_3^2}}{\sqrt{Fh^2 + x_4^2 + y_4^2}}$$

(3) For convenient, we put image$_1$ paralleling *y-z* plane with its center at (*Fh*,0,0), and then we have:

$$-y_1 = r_1\left[Fh \cdot \sin\phi - (y_2 \cos\omega + x_2 \sin\omega)\cos\phi\right]$$
$$-y_3 = r_2\left[Fh \cdot \sin\phi - (y_4 \cos\omega + x_4 \sin\omega)\cos\phi\right]$$

(*z* equal)

Let $A = \sin\phi, B = \cos\omega\cos\phi, C = \sin\omega\cos\phi$, we have equations of the following form:

$$aA + bB + cC = d$$
$$aA + eB + fC = g$$
$$A^2 + B^2 + C^2 = 1$$

where *a*,*b*,*c*,*d*,*e*,*f* and *g* are know.

The first two equation form a line (or a plane, or empty set in the space in the extreme cases), and the third is a sphere. So there may be two, one or no solutions for (A,B,C). However, subtract the second equation from the first, we have $(b-e)B + (c-f)C = (d-g)$. So we can convert (A,B,C) into $(a_0 + a_1 t, b_0 + b_1 t, c_0 + c_1 t)$ except the extreme cases. We can solve *t* from the third equation. Generally, there are two solutions. Each set of (A,B,C) decides a pair of (ω,ϕ).

For each pair of (ω,ϕ), we can derive θ by solving one of the following equations:

$$-x_1 = r_1\left[x_2' \cos\theta + (Fh \cdot \cos\phi + y_2' \sin\phi)\sin\theta\right]$$
$$-x_3 = r_2\left[x_4' \cos\theta + (Fh \cdot \cos\phi + y_4' \sin\phi)\sin\theta\right]$$

(*y* equal)

where
$$y_2' = y_2 \cos\omega + x_2 \sin\omega$$
$$y_4' = y_4 \cos\omega + x_4 \sin\omega$$
$$x_2' = x_2 \cos\omega - y_2 \sin\omega$$
$$x_4' = x_4 \cos\omega - y_4 \sin\omega$$

If the two θs derived from two different equations differ from each other too much, the pair of (ω,ϕ) should not be used.

Practically, the corresponding points given by user usually have errors. Not only because images are digitized: *x* and *y* is quantized, but also because it is not easy for human eyes to mark corresponding points for a photo-realistic image that contains no clear border or corner at all.

In our implementation, we have provided a tool allowing a user to manually register images or modify the registration of images registered by corresponding points with the related portion of the environment map changing in almost real-time.

Model Generation

After all images are registered, a texture mapped sphere-like polyhedron is generated. The essential concept of model generation is simple: generate a sphere-like polyhedron with its textures generated by ray-casting on original images. The original images are arranged as textured polygons in the space by their registrations, as shown in the following figure.

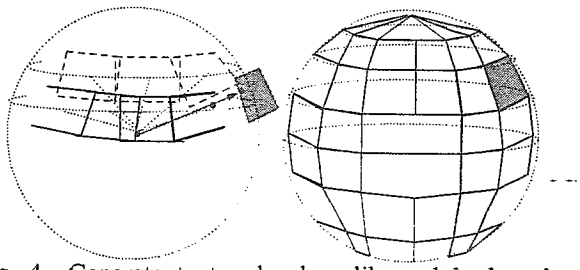


Fig. 4 Generate textured sphere-like polyhedron by ray-casting Texture

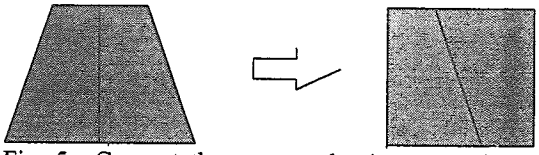


Fig. 5 Convert the non-regular image to the regular one

Polygons on the sphere-like polyhedron are all trapezoids or triangles, so the storage of texture is a problem: non-regular images are harder to handle than regular ones. We simply divide the texture vertically through the center and re-combine the two halves into a rectangle as indicated above.

As for determining the resolution of the texture, we simply let the resolution (pixel/degree) at the center of

the trapezoid be the same as the resolution at the center of the original image.

Smooth Intensity Discontinuity

Generally, there are intensity discontinuity between adjacent images, so the environment map generated by the previous ray-casting method will consist of patches. To solve the intensity discontinuity, it is suggested that there are overlapped portions between adjacent images, and the ray-casting method is modified for this purpose: instead of get color from the image first hit by the ray, we get colors from **all images** hit by the ray, and average these colors with the weight of each one. The weight is the square of the distance (on image) between the corresponding pixel and the nearest border, as indicated below.

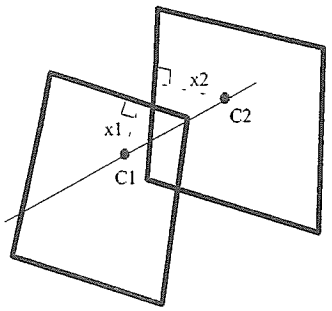


Fig. 6 result color = $\frac{x_1^2 \cdot C_1 + x_2^2 \cdot C_2}{x_1^2 + x_2^2}$, (where x_1, x_2 are distancesfrom the hit pixels to their nearest borders.)

By using such weighted average, the intensity discontinuity on the generated environment map will be smoothed. However, for adjacent images of large intensity difference, the overlapped portions may be insufficient. So the exposure should still be properly controlled to avoid large intensity difference between adjacent images when taking them.

Sometimes the border of the images may have noise, so we can ignore colors of pixels too near to their nearest borders by using the weighted average.

Image Data Handling

The data volume of images is generally tremendous. A 512x512 digitized NTSC resolution image with R:G:B = 5:6:5 bits (2 bytes per pixel) will take 0.5 MB (Megabytes). A typical sphere-like polyhedron made of about 90 such NTSC resolution images (with adjacent images overlapped) consists of 216 trapezoids will take about 21.2 MB! Thus the data transfer between main memory and storage device is relatively heavy, making image compression necessary not only for saving storage space but also for reducing data transfer. In our implementation, the JPEG image compression is applied to dramatically reduce the data size of textures. For the previous example polyhedron, the data size of texture is dramatically reduced from about 21.2 MB to about 3 MB, making it suitable for some large but relatively slow storing device such as CD-ROM, or for being transferred through existing network such as Internet. However, the image decompression usually takes time, so it is performed only when necessary (for images used by polygons that are going to be rendered right away) or when the CPU is idle. The priority to determine which image is decompressed first when CPU is idle or which decompressed image is released first when memory is not enough is simply by their relative location with respect to the viewing direction: the nearer image has higher priority.

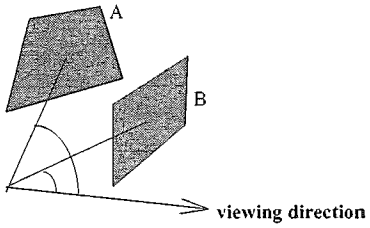


Fig. 7 Priority: image B > image A: (when CPU is idle, image B is decompressed before image A; when memory is insufficient, image A is released before image B.)

Object Viewer

The goal of "Object Viewer" is to let users interactively observe an object from an arbitrary view point in real time. In other words, the system must show an arbitratry part of an object in real time. Thus, the essential idea of image-based method is that the system sequentially shows the images taken from the same object from all view points, such as images A, B, and C in the following figure.

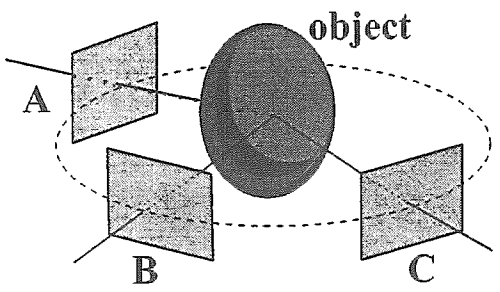


Fig. 8 A 2D prototype of object viewer

There is one major problem: how many images should be taken? For high resolution, the images should be as many as possible, but this will take too much disk space. Thus we can just take a few images, and generate the intermediate frames from these reference images. The typical methods for the generation of the intermediate frames are computer vision techniques, such as multi-view stereo and 3-D scene representation [7][8]. These methods generate reasonably good results, but they are time-consuming, and cannot be used in real-time applications. We find that a much simpler method is needed and described below. In short, the intermediate frame, say A', between A and B in the above figure is generated by interpolation. That is, the color of each pixel on A' is the weighted sum of the color of the corresponding pixel on A and B. Then we sequentially show images A, A', and then B.

With this simple method, we find that for a 2-D implementation, that is, the reference views are arranged in a circle around the object in the horizontal plane, one image per 15 degrees may be enough. One intermediate frame is generated between two reference images. (Of course, we can also generate more than one intermediate frame, but one is enough.) The sequential display of the reference images and intermediate frames is good enough to fool human eyes. Thus the result would look like arbitrary rotation of an object.

Result

We have made three textured sphere-like polyhedrons. Each polyhedron is made of 94 NTSC resolution images (size of 640x480 pixels, with R:G:B = 5:6:5 bits per pixel). These images are taken from the yard of the Agriculture college at National Taiwan University. The polyhedron consists of 216 trapezoids. By applying the JPEG image compression technique, the data size of the whole texture is dramatically reduced from 21.2 MB to 3 MB. A walk through demo system is implemented on an IBM Pentium-133 PC (a 486-DX2 66 is also suitable), under the MS-DOS operating system, using the Watcom C++ compiler (version 10.0a), and an ET-4000 SuperVGA card. The MS-Windows95 version is still under development.

A demonstration of the rendered results is given at the end of the paper. However, there is a practical problem remaining: it is almost impossible for some images to be registered without objects on the border being duplicated or lost in adjacent images because the camera was moved during panning.

Frame rate measurement is shown below as a table and wes measured on a Pentium-133 PC. Image decompression time is not included, where φ is the latitude of viewing direction, as indicated in Photo 4. The frame rate decreases while latitude increases because the number of trapezoids increases.

Frame size φ	319x199 No double buffer	639x479 No double buffer	319x199 VESA double buffer	639x479 VESA double buffer
0	44.31	16.46	34.76	13.65
15	42.73	16.38	33.80	13.00
30	41.14	15.44	32.99	12.13
45	38.13	14.66	27.70	11.94
60	35.26	13.14	23.33	10.33
65	33.80	12.60	23.16	10.27
70	32.76	12.35	22.74	9.88
75	30.33	11.83	22.70	9.10
80	27.64	10.92	19.85	8.23
85	27.30	10.57	17.59	8.09
90	25.28	10.19	16.99	8.08

Table 1 Result table

The third and forth columns are the frame rates when we use the VESA standard to access ET4000 card and make one more page in video memory for double buffer. Double buffer provides better output quality, but there are additional overheads.

We also implement the "Object Viewer" in MS-Windows95. As mentioned above, one image is taken per 15 degrees, and one intermediate frame is generated between two reference images. With Pentium-133 PC, 16MB RAM, the frame rate is about 6-8 frames per second. Porting to MS-Windows95 by using Microsoft Direct X has not been finished yet, and the final frame rate is expected to increase dramatically .

Conclusion

Although our approach appears to be simple, the method of generating textured sphere-like polyhedron does work well, and the rendering result is surprising: the illusion is pretty good when the user is kept near the center of the sphere-like polyhedron enough and no objects in the images are too near to the user. So it is easy to fool human eyes. Simplicity is beautiful.

Future Work

There are at least four items to be investigated in the future, and are listed below:

1. How to select the positions for the cameras and the corresponding "visible area"? Besides intuition, is there any rules to follow or to assess, or even algorithms to make the decision? This remains to be studied.
2. A good editing tool for spherical environment maps is essential. Currently the whole process of manual editing takes 2-3 hours for just one sphere .
3. The images taken by a camera are static. How about taking several images in the same direction from the same position if the scene is dynamic and choosing them according to the same order and timing in rendering? The moving objects may reveal the flaws of our "trick" of replacing the real objects by a textured wall; but on the other hand, a dynamic scene may be more realistic. And the user may be attracted by the movement and thus more easily to be fooled by our "trick".
4. Shading not being necessary in our approach implies specular lights are not handled. In an environment where specular lights are obvious and important (such as a room with a mirror) this may be a serious problem. Can this approach be extended to handle specular lights?

References

[1] Apple Computer, "*QuickTime® VR*" software package, 1995.

[2] Shenchang Eric Chen, "*QuickTime® VR-An Image-Based Approach to Virtual Environment Navigation*", ACM SIGGRAPH '95 p.29-p.38, 1995

[3] Wen-kae Tsao, Ming Ouhyoung, "*An Alternative Approach of Rendering High Quality Images for Virtual Environments Using Scanned Images*", IEEE HDTV '95, p.7B-1-p.7B-8,1995. (Also appears in proceedings of RAMS' 95, p.71-p.78)

[4] Richard Szeliski, "*Video Mosaics for Virtual Environments.*", IEEE CG&A Mar. 1996, p.22-p.30, 1996.

[5] Shenchang Eric Chen, Lance Williams, "*View Interpolation for Image Synthesis.*", ACM SIGGRAPH '93, p.279-p.288, 1993.

[6] Foley, van Dan, Feiner, Hughes, "*Computer Graphics: Principles and Practice*", 2nd Edition, Addison Wesley.

[7] R.M.Haralick, L.G.Shapiro, "*Computer and Robot Vision*", Volume I and II, Addison Wesley, Reading, MA, 1992.

[8] Stèphane Laveau, Olivier Faugeras, "*3-D Scene Representation as a Collection of Images and Fundamental Matrices*", Technical Report 2205, INRIA, 1994.

[9] Wen-kae Tsao, "*Rendering Scenes in the Real World for Virtual Environment Using Scanned Images*", MS thesis, Dept. Of CSIE, National Taiwan University, June 1996.

Appendix: Demonstration

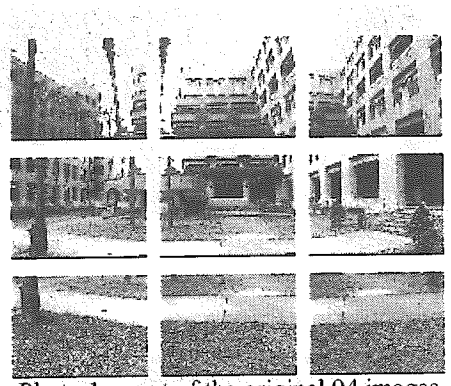
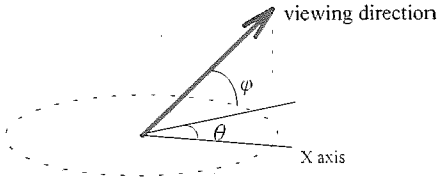


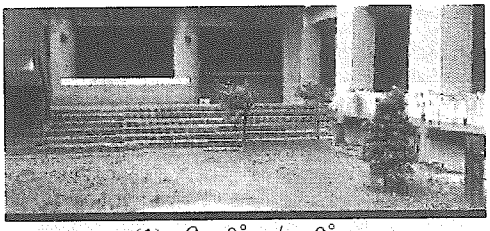
Photo 1 . part of the original 94 images

Photo.2 Part of the environment map generated by ordinary ray-casting. There is obvious intensity discontinuity.

Photo.3 By applying the proposed modified version of ray-casting, the intensity discontinuity is smoothed.

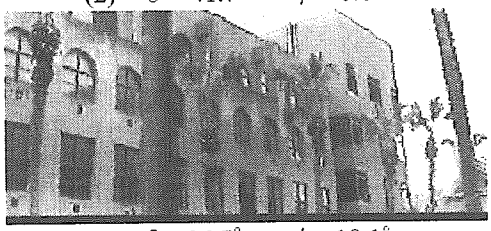
Photo 4 Rendered results when the viewpoint is at the center of the sphere-like polyhedron



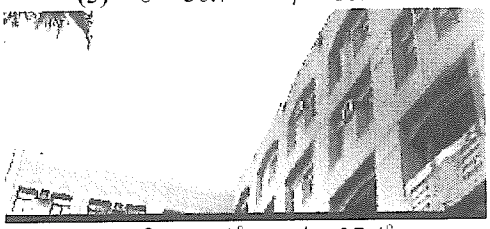


(1) $\theta = 0^\circ$ $\phi = 0^\circ$

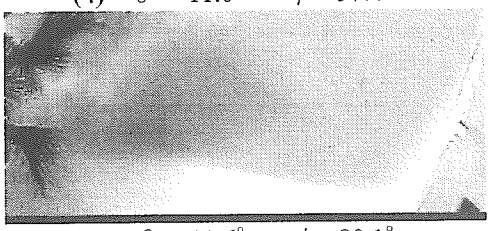
(2) $\theta = 41.7^\circ$ $\phi = 0.6^\circ$



(3) $\theta = 36.7^\circ$ $\phi = 18.1^\circ$

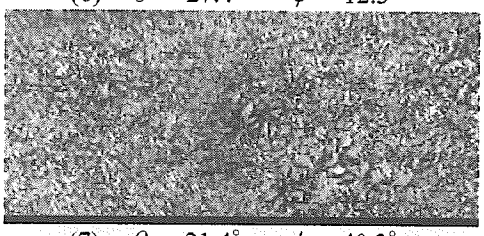


(4) $\theta = -11.0^\circ$ $\phi = 37.4^\circ$

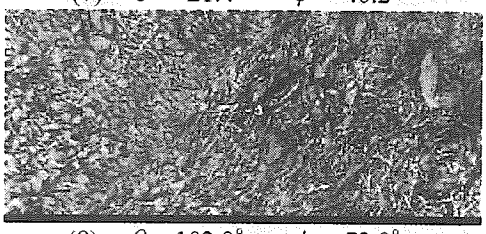


(5) $\theta = -11.6^\circ$ $\phi = 80.1^\circ$

(6) θ = -27.4° ϕ = -12.3°



(7) θ = -21.4° ϕ = -40.2°



(8) θ = 103.9° ϕ = -72.8°

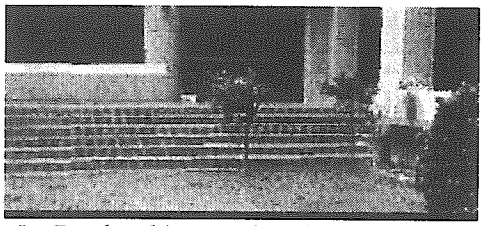


Photo.5 Rendered image when the viewpoint of Photo 4-(1) is moved 1/3 "radius" (distance from the center of the sphere-llike polyhedron to any face) away from the center of the polyhedron

Photo.6 Rendered image when the viewing direction of Photo 5 is turned 45 degrees away from the original direction

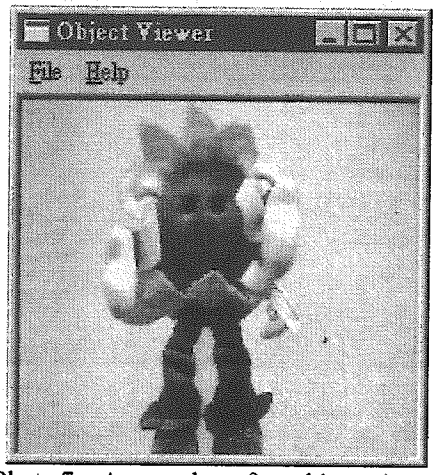


Photo.7 A snap shot of an object viewer