

# The Lumigraph

## I. Overview

My project was based on the work of the paper “The Lumigraph” and “Light Field Rendering” SIGGRAPH 96. The ray-traced images were generated to construct the lumigraph, some straightforward compression algorithm was applied to the original data. The program can reconstruct the 3D scene at the user-specified virtual camera location with 3 options of interpolation method: Nearest, Quadralinear and Depth corrected Quadralinear. Both Nearest and Quadralinear interpolation can be done in an interactive speed, the depth corrected quadralinear interpolation can be done in reasonably speed.

## II. Scene generation

The lumigraph was generated from a series of ray-traced images of size 512X512. The sampling rate of the s-t parameter plane is 16X16. The scene contains a bottomless teapot of 28 bezier patches (See the figure on the right). To do the depth correction, the depth file was generated at the same time the scene was rendered. This was really a time consuming and CPU intensive job. It took about 10 hours to render all these 256 images and to generate the depth files. The attached page shows all the images for the lumigraph.



## III. Straightforward compression of data

The original size of the images is about 192MB, the size of all the depth file is about 256MB (for 4-byte float point numbers). Note that the image here only contains a tiny fraction of all the 24-bit colors, I decided to use a palette and let each pixel only store the 16-bit index into the palette. This straightforward compression algorithm will compress the images to 128MB. Also note the depth range of the teapot is  $[-2, 2]$  (I retrieve this range from a program after I got all the depth files). An 8-bit precision is reasonably sufficient for the depth value between -2 and 2. Thus, only 1-byte was used for the depth, which compressed the depth data into 64MB. Obviously, these data (128M of image data + 64MB of depth data) have large redundancy in it (while using zip, they can be zipped into 31MB). But in order to randomly access the data, at least 192MB memory will be used to run the program that uses depth correction.

## IV. Reconstructing the image

When reconstructing the image, three interpolation methods were used for comparison: Nearest, Quadralinear, and Depth-corrected Quadralinear. The user can specify the camera location and film location.

## V. Result and discussion

The next page shows the application for demonstrating the program. The user can scroll to move the virtual camera and the film, or select from the three interpolation options. Nearest and Quadralinear can be done in real time with little delay and the delay for the depth correction is acceptable.

The block effect of the Nearest interpolation is very obvious. Quadralinear removes the block effect but introduces the blurry effects. While depth correction can reconstruct a fairly nice image except on the boundaries. Figure 3, 4, 5 show the comparison between the three interpolation methods (at the same camera location and film location).

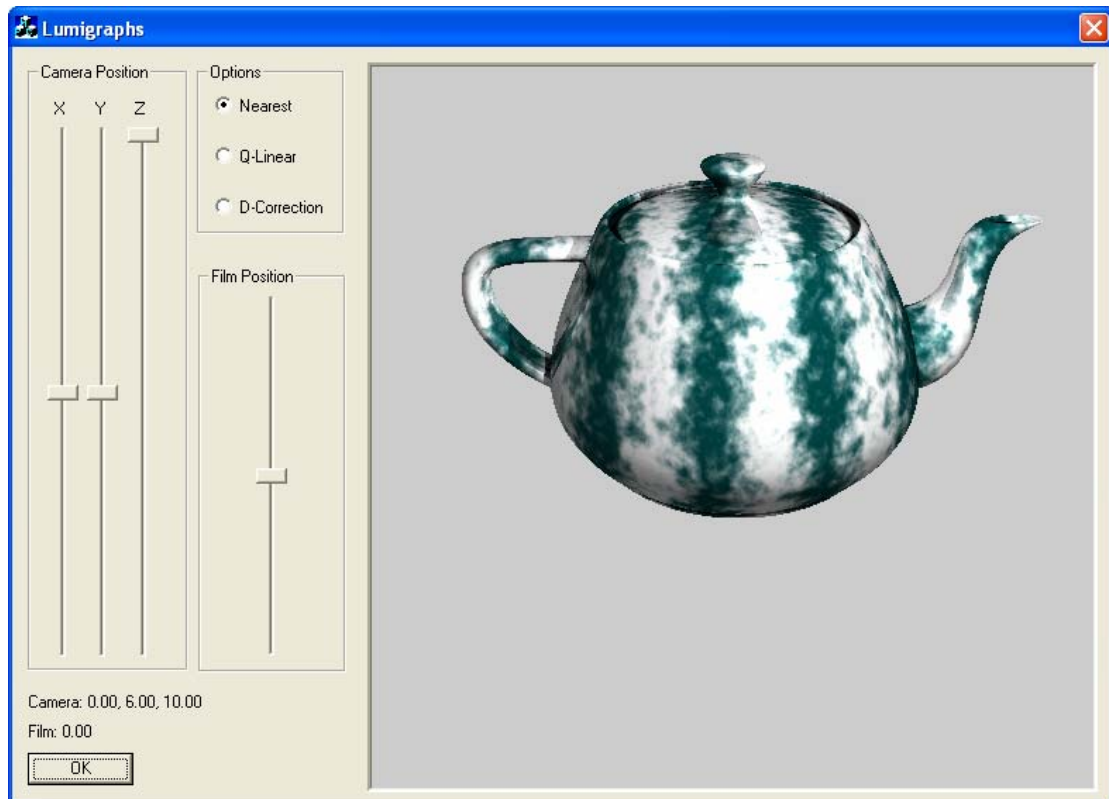


Fig. 2. Snap shot of the application



Fig. 3. Reconstructed image from Nearest interpolation



Fig. 4. Reconstructed image from the Quadralinear interpolation



Fig. 5. Reconstructed image from the Depth corrected Quadralinear interpolation

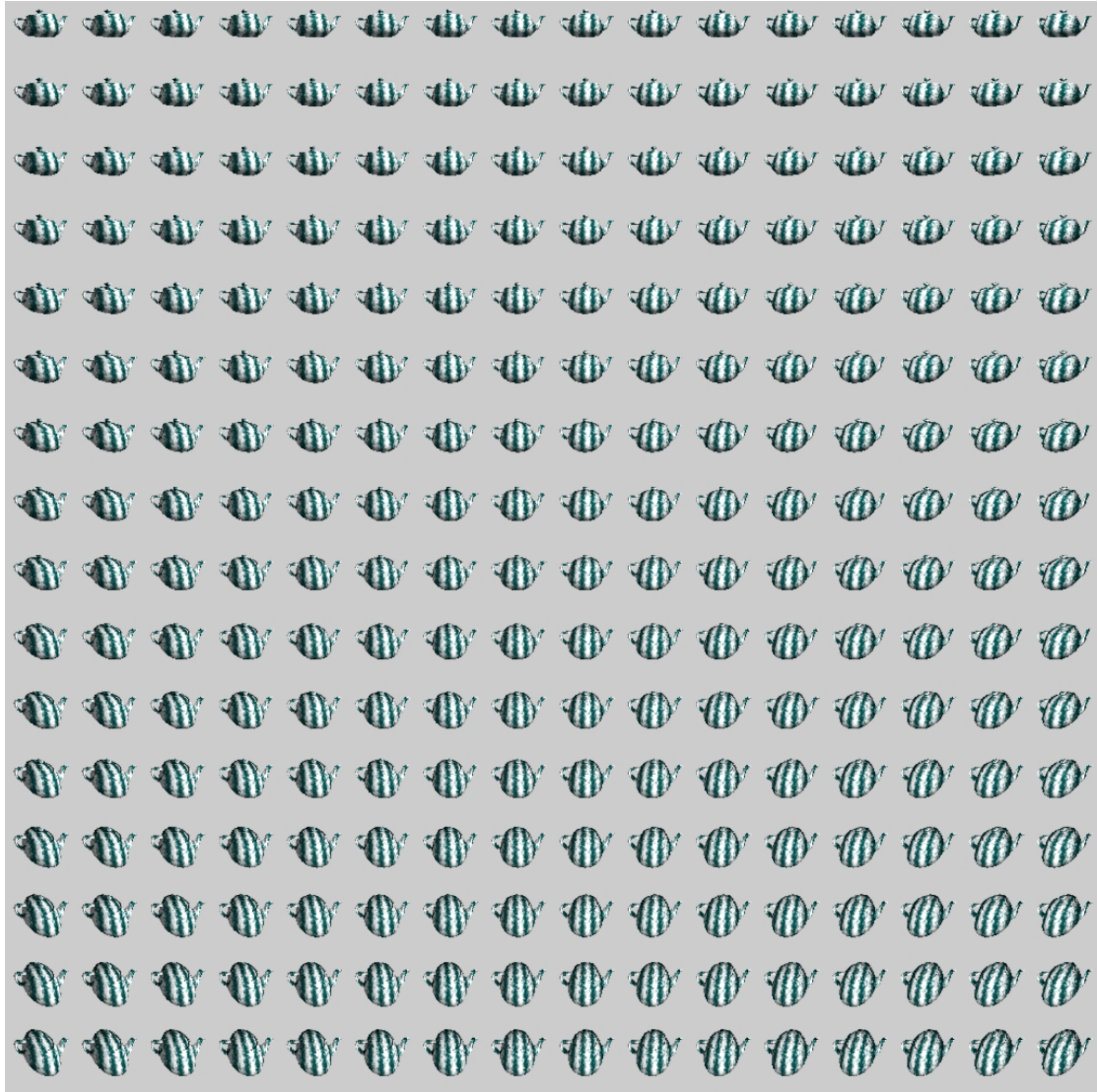


Fig 6. The 2D array of 2D rendered images