Color and Depth Image Based 3D Object Model Generation

YANG-KEUN AHN, KWANG-SOON CHOI, YOUNG-CHOONG PARK
Korea Electronics Technology Institute
121-835, 8th Floor, #1599, Sangam-Dong, Mapo-Gu, Seoul
REPUBLIC OF KOREA
ykahn@keti.re.kr

Abstract: - This study examines methods for creating a perfect 3D model by obtaining the values of R, G and B from color/depth map images, conducting a point-based 3D modeling in OpenGL for objects and images, and filling the holes created on the front or flank sides based on depth values. The most important part in this process is to fill the front and side holes by employing a differentiated hole-filling method. 3D models allow users to watch images drawn from the left and right sides of a 3D television by means of a pair of 3D glasses, and this study explores how an OptiTrack camera can search for users and then show them objects and images that fit the users'viewpoint from their own location.

Key-Words: - 3D Object Model, 3D Model-FHD, 3D Model, Lenticular, Hole Filling, Glasses-free 3D Model

1 Introduction

This study aimed to go beyond the existing 3D modeling system which tracks the location of users and creates real-time multi-view images based on such location information, and to focus instead on creating a practical 3D model in real-time.

The prior system for our third-year project employed the depth-image-based rendering (DIBR) method to produce a variety of multi-view images in real time that suited the location of users. We created multi-view images by rendering the texture images, in order to produce a virtual viewpoint for different user locations based on the RGB texture images and depth images. By tracking user locations, we could warp the multi-view images in real time to produce images based on virtual viewpoints for different locations. This results in lively 3D effects by presenting the warp images as if they moved according to each user's movement, thereby creating more active contents.

That technology, however, did not make users actually feel they were watching a 3D model. Users merely have an impression that 2D images are moving with them, because 2D images are used instead of real 3D models. Against this backdrop, this fourth-year development focused on utilizing depth map images and color images in the OpenGL windows as a means to create 3D models and provide them in real time to users to correspond to their varied viewpoints, eventually delivering a more lively experience of 3D models.

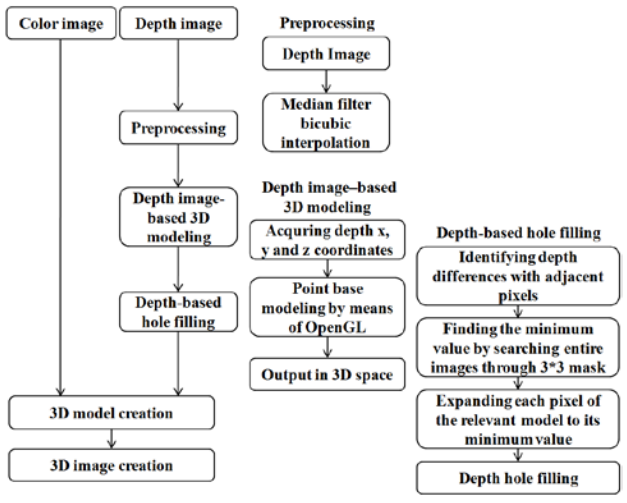

Color image
Depth image
Preprocessing
Depth-image-based 3D modeling
Depth-based hole filling
3D model creation
3D image creation
Preprocessing
Depth Image
Median filter bicubic interpolation
Depth image–based 3D modeling
Acquring depth x, y and z coordinates
Point base modeling by means of OpenGL
Output in 3D space
Depth-based hole filling
Identifying depth differences with adjacent pixels
Finding the minimum value by searching entire images through 3*3 mask
Expanding each pixel of the relevant model to its minimum value
Depth hole filling

Fig. 1 Algorithm for 3D model creation

Such technology also aimed at maintaining the quality of the existing images to the maximum possible extent, the details of which will be explained in the next chapter. The issues discussed in the next chapter are: (1) Camera calibration technology based on point correspondences; (2) Preprocessing of depth maps; (3) Color image-based texture creation technology; (4) Color/depth image-based mesh creation technology; (5) Multi-view color/depth image matching technology; and (6) Glasses-free 3D model creation by using lenticular images and synthesizing eight viewpoints. These issues will be further examined in the following chapters with experiments and conclusion. The references used for this study are provided in the last chapter.

2 Discussion

2.1 Camera Calibration Technology Based on Point Correspondences

The first step involves matching the ratio and location of color images and depth map images. We conducted calibration based not on depth map images, but based on color images, because calibrating the size of color images based on depth map images is likely to mar the resolution of color images and therefore lower the resolution of the final outcome. Merely resizing the images often produces holes due to unmatched sizes, and so bicubic interpolation was used in order to address the issue.

This interpolation involves a cubic polynomial, in which we presume that we know about two points and the slope of the tangent at those two points. A double polynomial often produces an unnatural curve or does not draw a curve at all, but a cubic polynomial makes a highly natural curve that smoothly links each section. For bicubic interpolation, if the original image’s coordinates for each pixel of the target image have real numbers when the original image is expanded, the 16 pixel values surrounding the relevant pixel are used for interpolation.

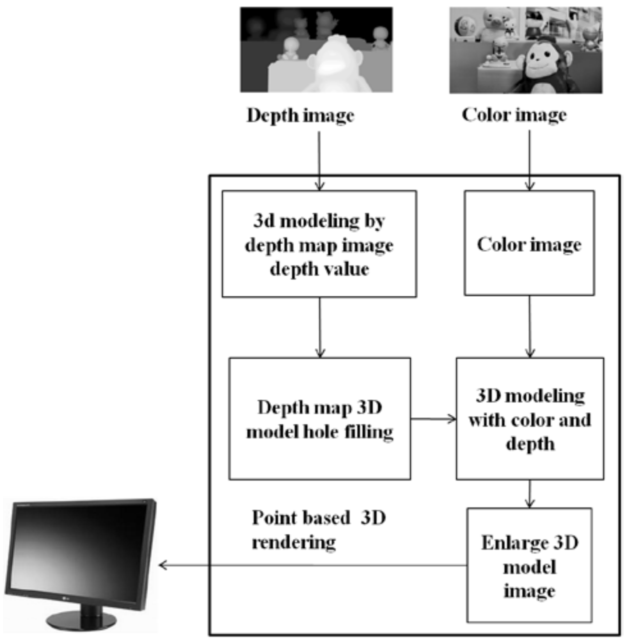

Depth image
Color image
3d modeling by depth map image depth value
Color image
Depth map 3D model hole filling
3D modeling with color and depth
Point based 3D rendering
Enlarge 3D model image

Fig. 2 Color/depth 3D modeling method

2.2 Preprocessing of Depth Maps

Since we expanded the depth maps based on the color images, the holes already lost and nonblocking parts grew larger. To address this issue, we used the median filter to eliminate the holes seen from the front side of the object.

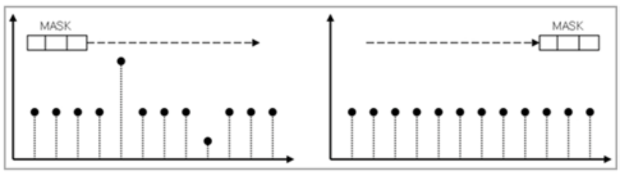


Fig. 3 Median filter

2.3 Color Image-based Texture Creation Technology

We obtained the depth values from depth maps based on the color images, and then obtained the values of R, G and B for x-coordinate, y-coordinate and depth. If we are to obtain the color values from color images based on depth maps, it is impossible to obtain all color values even if we conduct interpolation and preprocessing, since the depth values are distributed sporadically, unlike color values. Therefore, we are likely to see lowered resolution of the objects, and holes tend to remain if we proceed with 3D modeling in the OpenGL window.

However, if we use depth values based on the color values, we can utilize all of the depth values and maintain the quality of the color images’resolution. We can have the values of X, Y and Z (depth) as well as R, G and B after obtaining the color values from the color images based on X and Y coordinates, as well as depth values that correspond to the X and Y coordinates. These coordinates can be rendered in the OpenGL based on the order of depth values and X and Y values for the purpose of point-based 3D modeling.

If we are to obtain X, Y and Z values from depth map images and B, G and R values from color images by means of forward mapping, it is likely that pixels corresponding to the output model will overlap with one another, or empty holes will be created with no matching pixels. This is because we employ backward mapping to obtain depth map values based on color images as a means of matching each pixel of the output model to new pixels of the input images. This way, we can resolve, to a certain extent, the issue of overlapping pixels and holes created in the output image as a result of forward mapping.

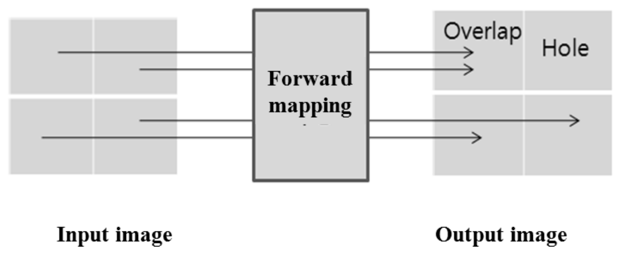


Fig. 4 Forward mapping

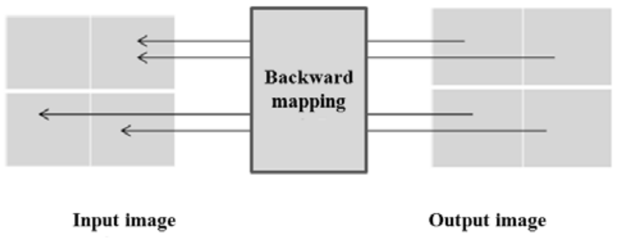


Fig. 5 Backward mapping

2.4 Color/depth Image-based Mesh Creation Technology

The rendering of information derived from the color and depth maps eventually creates a point cloud-based 3D model. However, this 3D model is not a perfect model due to the existence of holes and nonblocking space. In order to address this problem, it is necessary to fill the holes and nonblocking sections with appropriate values. Using the same method to fill the holes in the front and flank sides of an object and the background will not fill the holes perfectly, since holes filled in the front side will be emptied when the object revolves around, constantly leaving empty holes. These holes, created due to the differences of depth values, must be filled based on depth, as illustrated in the following Fig. 6.

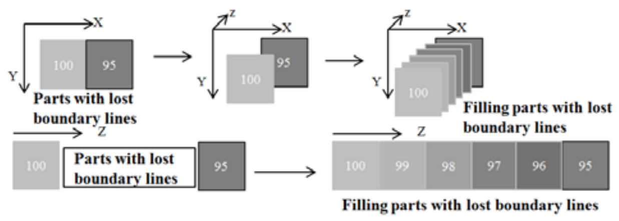

X
Y
Z
100
95
Parts with lost boundary lines
Filling parts with lost boundary lines
100 99 98 97 96 95

Fig. 6 Depth map-based hole filling

As shown in the above figure, holes are created on the boundary lines or the flank side of an object as a result of the differences of depth values, because points indicating depth values in the three-dimensional space of the OpenGL is bound to have empty spaces between points, unless the depth differences exist in order.

By not creating space between points, it is possible to fill the holes and nonblocking sections on the boundary lines and flank side of an object. A point-based 3D rendering model forms a perfect 3D mesh when its holes and nonblocking sections are completely filled, eventually becoming a perfect 3D model. This study particularly concentrated on the inner parts and boundary lines of objects. The inside holes are mostly filled during preprocessing, presenting a perfect model when viewed from the front side. The holes on the boundary lines tend to be created because one or more differences of depth values exist, and therefore points cannot exist in order, rather than because points are lost. Failing to understand such elements before filling the holes will prevent creating a perfect 3D model.

In addition, since the infrared rays of the camera are dispersed on the boundary lines, the points exist at a location with lower depth values. If this is not resolved, viewers may feel as if the object is dispersed backwards when the 3D model revolves around. In order to address this issue, we set threshold values for each point after averaging the depth values of adjacent points, and then cut the values far behind the threshold value and pushed inside the points within a certain permissible range toward the object, by the difference from the threshold value. By cutting and moving points in such a way, we were able to realize smoother and more natural boundary lines of the 3D object.

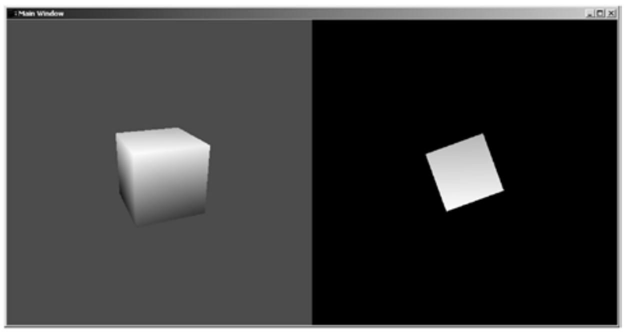


Fig. 7 OpenGL subwindow

2.5 Multi-view Color/depth Image Matching Technology

There are a number of methods of watching 3D images from a 3D television. The method used in this study involves adjusting the images with different viewpoints for the left and right sides and showing the left image to the left eye and the right image to the right eye. Human beings grasp a sense of distance, or the degree of depth, by using the parallax of their two eyes. In a similar way, viewers can experience a sense of depth when a 3D model created in the OpenGL is rendered onto a screen differently in the left and right sides, and the y-axis of the object is rotated to show another image.

In this program, we used the OptiTrack camera to identify the location of the viewer's eyes and revolve both the left- and right-side 3D model by the distance needed to match the eye level based on the y-axis. Since the object is rotated just to the point of a viewer's eyes, the viewer can have the experience of watching 3D objects as if they actually revolved around.

The OpenGL has either a single window, or a multi-window. This project required both the left and right images at the same time. For this, we chose the subwindow and multi-window modes of the OpenGL. We employed the subwindow mode in 3DModel-FHD and separately output the object of such window in a 1920*1080 television.

There are certain circumstances when we have to take caution in drawing two or more windows or consecutively drawing a 3D model in a subwindow: Since the OpenGL draws points or objects in order, the image drawn earlier could be distorted. This is because the models or their points drawn at a different timing could be seen as a mingled form based on the priority of points, not the priority of depth values.

In order to address this issue, the glEnable(GL_DEPTH_TEST); function is used, so that the OpenGL can organize a 3D model again based on depth values even when the model is mixed based on the order of points. Using a number of windows can solve the problem of 3D model objects being distorted.

There is another aspect to pay attention to when drawing points or objects in OpenGL: the use of glbegin(); and glend(); functions. The former must be called before drawing an object in OpenGL and the latter at the end. However, using these functions indiscriminately is likely to cause the programs'performance to decline. Since the point cloud has a great number of points during the point-based 3D rendering, using those functions every time such a point is drawn is significantly damaging to the performance.

The method of solving this problem is to use the glbegin(); function by sending the point cloud to the for loop and use the glend(); function after finishing drawing all points. When drawing points, the number of which spans between 90,000 and 600,000, there is a considerable difference between calling the two functions up to 600,000 times and just once.

The three-dimensional effect is not created instantly by the existence of left and right images to produce a stereoscopic 3D effect. The left and right images must have parallax when viewers actually watch them; if a 3D model with the same viewpoint in the left and right sides is presented, viewers just see an image with no three-dimensional effect. Thus, it is necessary to present the images actually seen from the left and right eye of a viewer as left and right images. In order to create an image with different viewpoints by using a single model, the 3D models in the left and right windows each need to move based on the y-axis. Therefore, it is possible to create left and right images that actually have left-right parallax if the left image is rotated towards the right and the right image to the left.

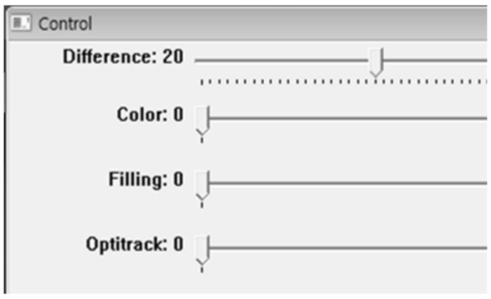

Control
Difference: 20
Color: 0
Filling: 0
Optitrack: 0

Fig. 8 3DModel-FHD control UI

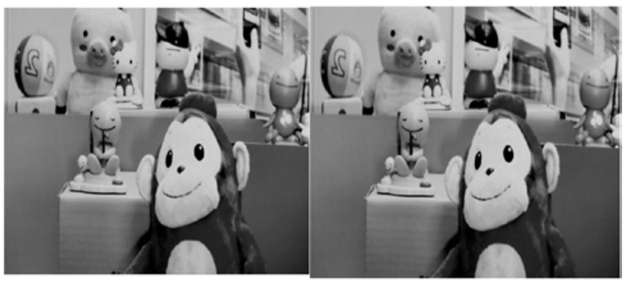


Fig. 9 3D television UI

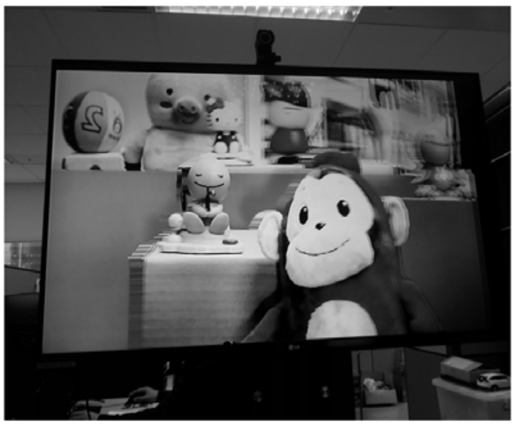


Fig. 10 Television image with 3D mode

2.6 Lenticular Images Synthesizing Eight Viewpoints for Glasses-free 3D Model

A lenticular lens is required in order to watch a 3D model without glasses. A lenticular lens presents an image matching the direction from which a viewer watches. This project used the lenticular 3D display of Alioscopy3DHD24, which automatically presents up to eight viewpoints in the form of a stereoscopic 3D display. This display shows images adjusted to a users'location because the curved lenticular lens can deliver one of the eight viewpoints synthesized beforehand, and users watch the refracted image.

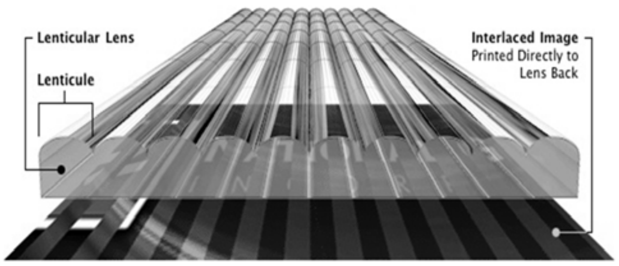


Fig. 11Cross-sectional diagram of the lenticular lens

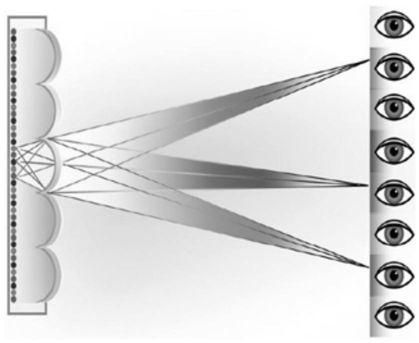


Fig. 12 Method of watching an image from eight viewpoints through a lenticular lens

In order to watch a 3D model without glasses in a multi-view lenticular 3D display, it is necessary to synthesize eight-viewpoint images into a single image. What distinguishes an eight-viewpoint image from the existing one-viewpoint image is that R, G and B values for each pixel are obtained from eight viewpoints in order, not from a single viewpoint. For example, each R, G and B value for Pixel 1 comes from each R, G and B value of first, second and third viewpoints, respectively. Again, each R, G and B value for Pixel 2 comes from each R, G and B value of fourth, fifth and sixth viewpoints, respectively. Then, the G value for Pixel 3 comes from the G value of eight viewpoints, and the B value for Pixel 1 from the B value of the first-viewpoint image. This way, one image can be synthesized with images with eight different viewpoints.

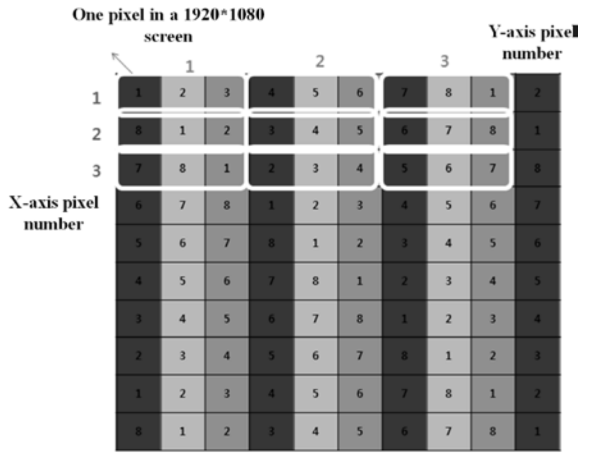


Fig. 13 Method of synthesizing eight-viewpoint images

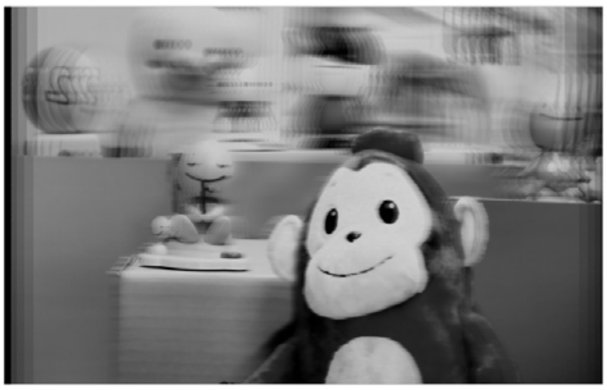


Fig. 14 Lenticular image of synthesizing eight viewpoints

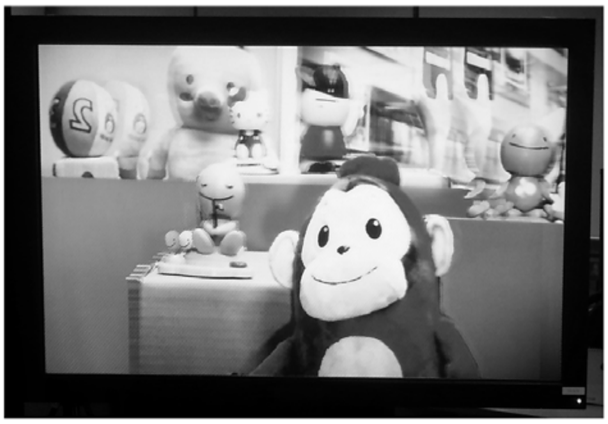
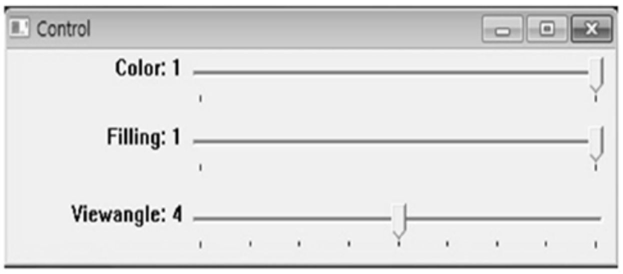


Fig. 15 Image of synthesizing eight viewpoints seen from Alioscopy3DHD24



Color: 1

Filling: 1

Viewangle: 4

Fig. 16 Control UI

3 Experiment

The development environment included the operating system of Windows7, Tool Visual Studio 2010 and C++, while the hardware was organized with reflective marker 3D glasses, OptiTrack Flex 13 infrared camera, 3D display, desktop with Intel i7-2600K with 3.48GBRAM.

As the measurement values for comparing image quality, we used the values of peak signal-to-noise ratio (PSNR), which indicates the power of noise against the maximum electric power that signals can have. These values are mostly used for assessing information on the loss of image quality due to lossy video compression and allows a quantitative understanding of differences of two different images. In this study, the PSNR values were obtained by means of the formula given in Fig. 17. In the formula, MSE means the variation of the relevant image, and MAX refers to the maximum value of the relevant image and can be obtained by subtracting the minimum values from the maximum value of the relevant image channel. The PSNR is expressed in the unit of dB. Since the value is measured at the log scale, lower loss is expressed with higher numerical values.

$$MSE = \frac{1}{m\,n}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2$$

The PSNR is defined as:

$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right)$$
$$= 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right)$$

Fig. 17 PSNR measurement formula

The resulting value of 34.09dB is considered appropriate.

The rate of 3D model creation was measured to be 0.608sec, approximately five seconds faster than the time required for a 3D model creation in the fourth-year evaluation. The program renewed the record of the relevant 3D model at an average of 15fps.

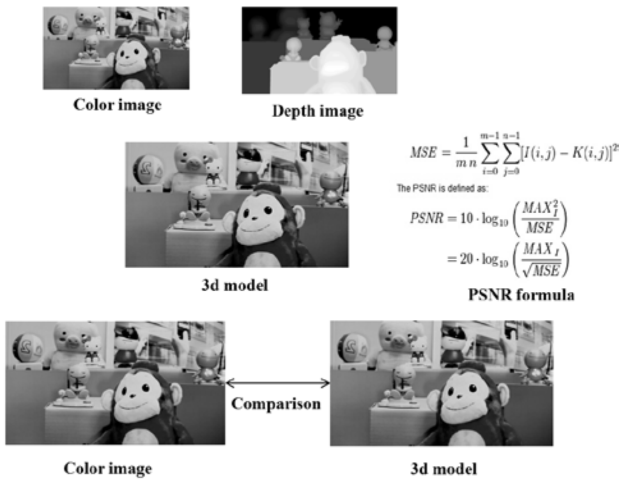


Fig. 18 PSNR experiment process

4 Conclusion

The main purpose of this project was to enable users to feel the 3D space and object in real time through 3D modeling. To that end, we focused our endeavors on improving the user experience, and as a result, this project was able to substantially enhance the experience of 3D space and objects compared to the previous third-year project. We believe that advancement of the depth camera technology and subsequently higher resolution will allow users to experience 3D contents and services with better quality as soon as the images are shot.

References:

[1] Wei Wang, Longshe Huo, Wei Zeng, Qingming Huang, Wen Gao, "Depth Image Segmentation For Improved Virtual View Image Quality In 3-DTV," Proceedings of 2007 International Symposium on Intelligent Signal Processing and Communication Systems, Dec. 2007.

[2] Liang Zhang and Wa James Tam "Stereoscopic Image Generation Based on Depth Image for 3D TV," IEEE Transactions on Broadcasting vol. 51, no. 2, Jun. 2005.

[3] Wenxiu Sun, Lingfeng Xu, Oscar C. AU, Sung Him Chui, Chun Wing Kwok," An overview of free viewpoint Depth-Image-Based Rendering

(DIBR)," The Hong Kong University of Science and Technology.

[4] Lee Sang-beom, Ho Yo-seong, “Discontinuity-adaptive Depth Filtering for 3D View Generation,” The Korean Institute of Communications and Information Sciences, 2008 Collection of Conference Dissertations (Autumn), Nov. 2008, pp. 358-361.

[5] Hyeon Ji-ho, Han Jae-yeong, Won Jong-pil, U Ji-sang, “Generation of an Eye-contacted View Using Color and Depth Cameras,” Journal of the Korea Institute of Information and Communication Engineering, vol. 16, no. 8, 2012, pp. 1642-1652.

[6] Ryusuke Sagawa, Katsushi Ikeuchi, "Hole filling of a 3D model by Flipping Signs of a Signed Distance Field in Adaptive Resolution," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 30, no. 4, Apr. 2008.

[7] David Doria, Richard j. Radke, "Filling Large Holes in LIDAR Data By Inpainting Depth Gradients," Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on, Jun. 2012.