KINECT PROJECT

Eitan Babcock

Report to receive final EE credit

Fall 2013

CONTENTS

INTRODUCTION

The Xbox Kinect is a remarkable device that has the ability to capture 3D images. It consists of two main parts – a regular camera to capture image information, and sensor that can capture depth. This allows for each pixel to have four parameters, rather than the typical three. Normally, pixel information contains the levels of red, green, and blue (RGB). The pixel information from the Kinect has a fourth parameter of distance.

OBJECTIVE

The goal of this project is to allow for full 3D viewing of images without image shadowing. This will be done by using multiple Kinects viewing an object from multiple positions.

Image shadowing occurs when there is an obstruction in the line of site of a camera. It is analogous to a regular shadow, caused by an object obstructing the line of site of a light source such as the sun. With only one Kinect viewing an object from one position, only the one surface is able to be captured. This is shown in Figure 1. However, if you add additional Kinects around an object, all sides can be captured and combined to form one image. At this point, one can see a full 3D representation of an object viewable from all sides.

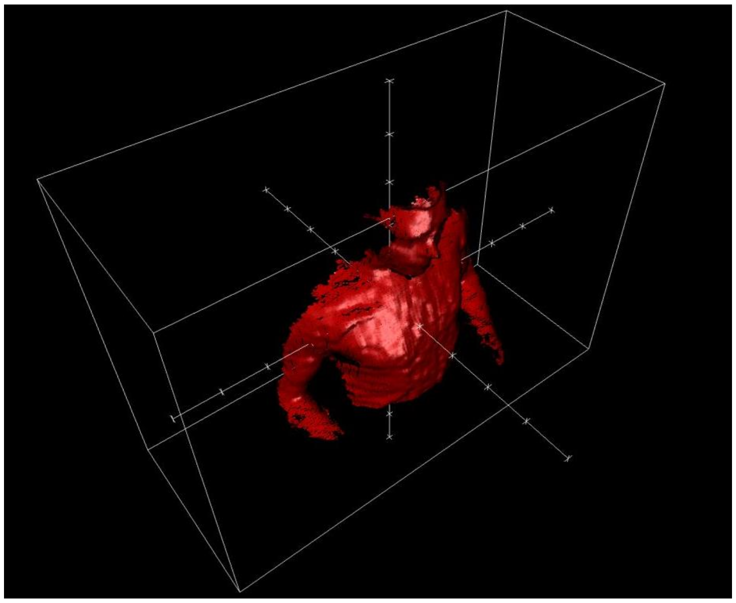


FIGURE 1: (SOURCE: HTTP://DRDAVE.CO.UK/IMAGE/10288BE3-A33A-44FA-9A64-CFC6E9DD80D7)

PROCEDURE

CONVERTING DISTANCE ARRAY TO 3D ARRAY

The Xbox Kinect stores distance information as a 2D array of distances. In order to manipulate the data, it must be converted to a 3D array in space, where the array elements correspond to X, Y, and Z spatial coordinates, and the elements of the array correspond to the pixel color information. This will yield the start of a 3D representation of the environment that can be viewed from multiple angles.

In order to do this, it is important to note that the Kinect is getting this information from a single point in space. Figure 1 shows how both the angle and depth information must be taken into account when creating the 3D array.

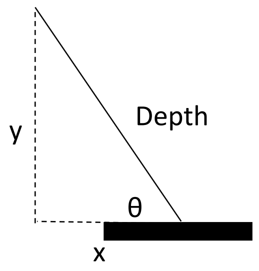


FIGURE 2

I will define the horizontal angle from the source to each pixel to be θ, and the vertical angle to be ϕ. These angles can be calculated using the Kinect system specifications. The specifications provide a total viewing angle of 57° horizontal and 43° vertical, and a resolution of 1280x960 pixels (http://msdn.microsoft.com/en-us/library/jj131033.aspx). From this, we can calculate the angle per pixel.

$$d\theta = \frac{57°}{1280\, pixels} = 0.04453\ °/pixel\ horizontal$$

$$d\varphi = \frac{43°}{960\, pixels} = 0.04479\ °/pixel\ vertical$$

Additionally, the Kinect is able to tilt up and down. For this project, I will assume that the Kinect is facing straight forward, and thus the 0° is in the exact center of the pixel array. It would not be difficult in the future to incorporate this information in the calculations.

The range of the Kinect is .8 m to 4 m. This is needed to determine how large the spatial array must be. Using the maximum distance and the maximum viewing angle, the distance between pixels can be determined.

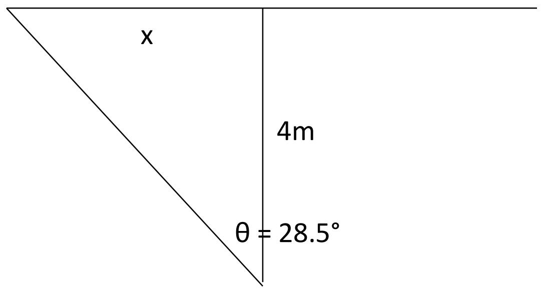
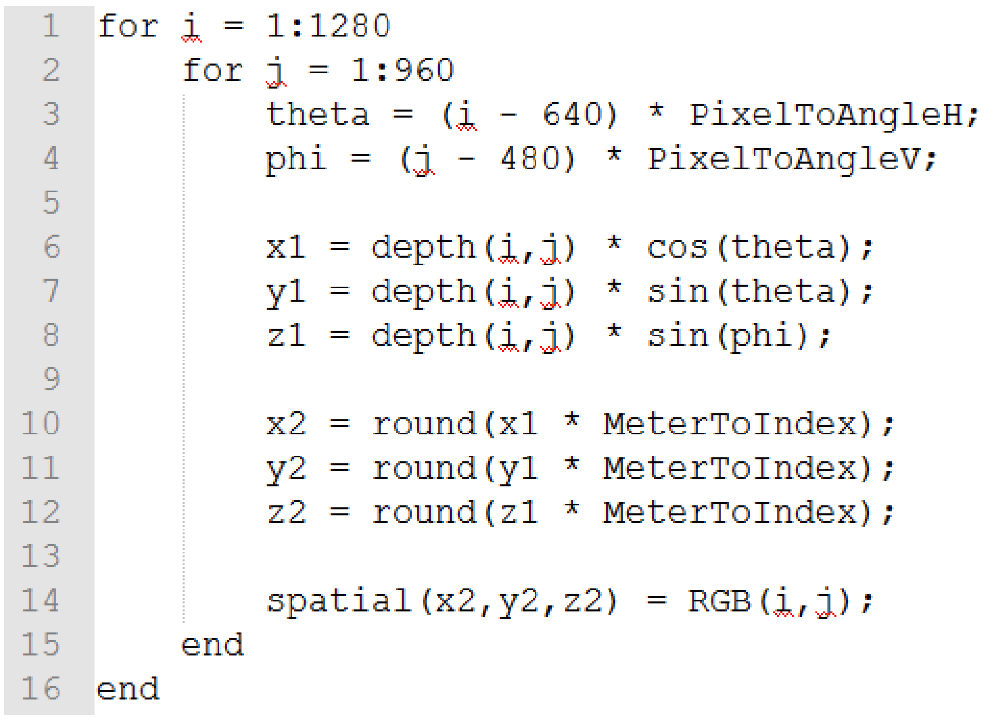


FIGURE 3

$$x = 4\,m * \tan(28.5°) = 2.17\,m$$

$$\frac{2.17\,m}{640\,pixels} = .00339\,m/pixel$$

The following MATLAB code in Figure 3 shows how to calculate the 3D depth matrix. There is also an explanation of variables in Table 1.



```
for i = 1:1280
    for j = 1:960
        theta = (i - 640) * PixelToAngleH;
        phi = (j - 480) * PixelToAngleV;

        x1 = depth(i,j) * cos(theta);
        y1 = depth(i,j) * sin(theta);
        z1 = depth(i,j) * sin(phi);

        x2 = round(x1 * MeterToIndex);
        y2 = round(y1 * MeterToIndex);
        z2 = round(z1 * MeterToIndex);

        spatial(x2,y2,z2) = RGB(i,j);
    end
end
```

FIGURE 4

TABLE 1:

Variable	Description
i, j	Iterating variables going through every pixel in the 2D array
theta, phi	Angles between the Kinect and the pixel
PixelToAngleH, PixelToAngleV	Constants used to convert from pixels to angles for horizontal and vertical, respectively
x1, y1, z1	Distances in meters
x2, y2, z2	Distances as array index integers
depth(i,j)	The 2D depth array
MeterToPixel	Constant used to convert form a distance in meters to the index of the 3D array
spatial(x,y,z)	The 3D spatial array
RGB(i,j)	The 2D pixel image array

TRANSFORMATION MATRICES

Transformation matrices are used to move from one reference frame to another. In this project, they are used to convert the reference frames from multiple Kinects to the base reference frame of the user. In this project, I am assuming all of the Kinects are pointing straight forward, so all rotations will be around the z-axis. The rotation matrix for a rotation around the z axis is as follows.

$$R_z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The transformation matrix contains the rotation matrix, plus a transposition term. The transformation matrix is as follows:

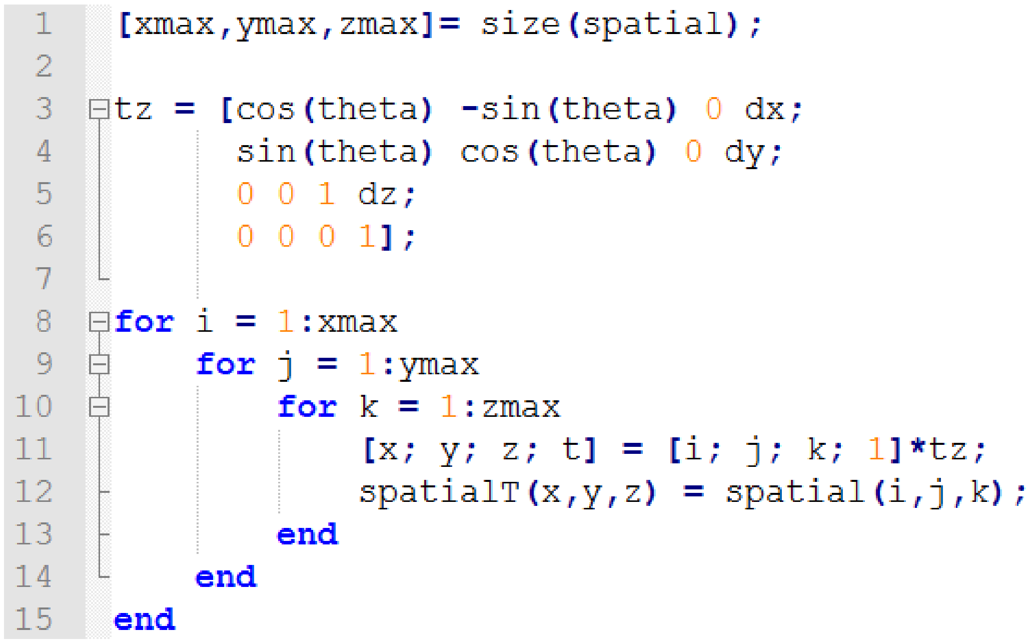
$$T_z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & dx \\ \sin(\theta) & \cos(\theta) & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In this matrix, θ is the angle the new reference frame is rotated around the z axis with respect to the base frame, dx is the displacement in the x direction, dy is the displacement in the y direction, and dz is the displacement in the z direction. For each point in space, the coordinates must be multiplied by this transformation matrix as follows:

$$\boldsymbol{X}'(x, y, z) = T_z(\theta)\, \boldsymbol{X}(x, y, z)$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & dx \\ \sin(\theta) & \cos(\theta) & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

The following MATLAB code in Figure 4 shows how this translation could be done. Table 2 contains a description of variables for this code.



```matlab
[xmax,ymax,zmax]= size(spatial);

tz = [cos(theta) -sin(theta) 0 dx;
          sin(theta) cos(theta) 0 dy;
          0 0 1 dz;
          0 0 0 1];

for i = 1:xmax
    for j = 1:ymax
        for k = 1:zmax
            [x; y; z; t] = [i; j; k; 1]*tz;
            spatialT(x,y,z) = spatial(i,j,k);
        end
    end
end
```

FIGURE 5

TABLE 2

Variable	Description
i, j, k	Iterating variables going through every pixel in the 2D array
theta	Rotational angle between this Kinect reference frame and the original reference frame
xmax, ymax, zmax	The maximum indices of the 3D spatial array.
x, y, z	New indices for transformed array
t	Place holder for 4th matrix element
dx, dy, dz	Coordinates of this Kinect
spatialT(x,y,z)	Transposed 3D spatial array
spatial(x,y,z)	The 3D spatial array

This can be done for as many Kinects as needed or wanted. The more Kinects used, the more points there can be.

COMBINING IMAGES

In order to complete the full 3D image viewable from all sides, the images from each of the Kinect sources must be combined into one image. This is a relatively easy feat in the programming. The 3D array is simply edited each time the transformation calculations are done. The first Kinect will be treated as the base frame, and thus no transformation will be done. After the code from Figure 3 is executed to calculate the spatial array, this becomes the start of the final 3D image. I will refer to this array as “Final3D.” Then, for each subsequent Kinect used, line 12 from the code in Figure 4 would be modified so that Final3D is updated as follows:

$$Final3D(x,y,z) = spatial(i,j,k);$$

In this coding method, any pixel of data in the 3D space that had information from multiple Kinects, would only show the information from the last Kinect that was entered into the program. Theoretically, this should not matter since it should be the same data from multiple sources. However, this would only be the case if all distances and angles between Kinect sources was measured exactly down to the pixel (which was calculated previously in this paper to be approximately 3 mm and 1 hundredth of a degree). This is not feasible. However, the measurements should be close enough such that a human looking at the final 3D image should not be able to notice.

CONCLUSION

I was not able to complete the coding of this project, due to losing access to the software needed. While I am sure there are a few bugs to be worked out, I am confident that I have put enough thought and mathematics into this project for it to be a place to build from for people who do have access to MATLAB or another similar coding environment. I hope that someone at Washington University in St. Louis is able to take this idea and advance it to a working prototype.