Report Computer Vision Project 1

Single Michael
08-917-445

1 Photometric Stereo (Due on 28/10/2014)

1. Calibration (35 points) In this section you should:

- Describe the algorithm you used for calculating the light directions given the images of the chrome sphere for different lighting conditions. You need to provide the formula you used to calculate such directions given: 1) The radius of the sphere; 2) The 2D coordinates of the light source highlights on the sphere; 3) The 2D coordinates of the centre of the sphere; 4) The unit vector $(0, 0, -1)$ that points towards the camera.
- The calculated light vector in the following format:

$$\mathbf{L} = \begin{bmatrix} L_{1x} & L_{2x} & L_{3x} & L_{4x} & L_{5x} & L_{6x} & L_{7x} & L_{8x} & L_{9x} & L_{10x} & L_{11x} & L_{12z} \\ L_{1y} & L_{2y} & L_{3y} & L_{4y} & L_{5y} & L_{6y} & L_{7y} & L_{8y} & L_{9y} & L_{10y} & L_{11y} & L_{12y} \\ L_{1z} & L_{2z} & L_{3z} & L_{4z} & L_{5z} & L_{6z} & L_{7z} & L_{8z} & L_{9z} & L_{10z} & L_{11z} & L_{12z} \end{bmatrix}$$

In this section we discuss and derive all essential formulas in order to compute light direction vectors using the technique of photometric stereo. Afterward, we discuss in detail how some quantities, such as the sphere radius, are computed.

We start by describing how normals on a spherical surface can be computed. Next, we discuss how we can compute the surface normals of the images of a chrome sphere. Last, by using Snell's reflection law, we tell the reader how the light directions can be estimated.

Every point (x, y, z) on the surface of a sphere in $\mathbb{R}^3$ with a radius r centered at $\mathbf{c} = (c_x, c_y, c_y)$ fulfills the following implicit function:

$$r^2 = (x - c_x)^2 + (y - c_y)^2 + (z - c_z)^2 \tag{1}$$

Equation 1 has the following vectorized representation:

$$f(\mathbf{p}) = (\mathbf{p} - \mathbf{c}) \cdot (\mathbf{p} - \mathbf{c}) - r^2 = 0 \tag{2}$$

Where $\mathbf{p}$ denotes a point (x, y, z) on the sphere's surface. Please note that f denotes the implicit function (for our case the implicit function of a sphere). An implicit function describes a set of surface points whenever we set this function equal to zero, i.e. $f(p) = 0$.

Next we describe how we can compute the normal on a sphere using our implicit representation defined in equation 2. From Mathematics we know that the normal on a surface point $\mathbf{p}$ is simply the gradient of the implicit function describing the surface. Thus, the equation for computing a normal from f looks like:

$$\mathbf{n}(\mathbf{p}) = \nabla f(\mathbf{p}) \tag{3}$$

For simplification purposes we omit the argument $\mathbf{p}$ in $\mathbf{n}(\mathbf{p})$ in the following and just write $\mathbf{n}$ instead.

Applying the identity from equation 3 to equation 2 we can compute the normal on the surface of a sphere.

$$\begin{aligned} \mathbf{n} &= \mathbf{n}(\mathbf{p}) \\ &= \nabla f(\mathbf{p}) \\ &= \nabla \left((\mathbf{p} - \mathbf{c}) \cdot (\mathbf{p} - \mathbf{c}) - r^2 \right) \\ &= 2(\mathbf{p} - \mathbf{c}) \end{aligned} \tag{4}$$

Please note that the normal $\mathbf{n}$ from equation 4 is not normalized. The normalized normal $\hat{\mathbf{n}}$ of the normal from equation 4:

$$\begin{aligned}\hat{\mathbf{n}} &= \frac{\mathbf{n}}{||\mathbf{n}||} \\ &= \frac{(\mathbf{p}-\mathbf{c})}{r}\end{aligned} \tag{5}$$

Therefore, equation 5 tells us how we can compute normalized normals at any point lying on the surface of a implicit surface.

Next, we describe how we can make use of equation 5 when using our twelve images of the chrome sphere. Since the given images are in 2d, i.e. only exhibit two dimensions, the x-and y-coordinate, we cannot make directly use of the z component in order to define the surface of a sphere using its implicit representation from equation 1.

Thus, we express the z component as a parametrization of (x, y, r):

$$\begin{aligned}z &= z(x, y, r) \\ &= \sqrt{r^2 - (x - c_x)^2 - (y - c_y)^2}\end{aligned} \tag{6}$$

In order to derive this identiy we only solved for z using equation 5. By plugging equation 6 into equation 5 we get the final representation for computign the normals:

$$\hat{\mathbf{n}} = \frac{1}{r}\begin{pmatrix} (x - c_x) \\ (y - c_y) \\ -\sqrt{r^2 - (x - c_x)^2 - (y - c_y)^2} \end{pmatrix} \tag{7}$$

Where (x, y) are coordinates on the chrome sphere exhibiting a (specular) highlight caused by light sources. For any such (x, y) we compute its normal using equation 1. Please note that we are using the negative of z as the third component in the normal vector in 1 since the z-axis points towards the camera eye (our convention).

Last, using all these computed normals, we can estimate the light directions using Snell's reflection law:

$$L_k = \mathbf{d} - 2(\mathbf{d} \cdot \hat{\mathbf{n}}_k)\hat{\mathbf{n}}_k \tag{8}$$

Where L_k denotes the k-th light source resulting from the k-th normal vector $\hat{\mathbf{n}}_k$. $\mathbf{d} = (0, 0, -1)$ denotes the direction of the camera from any point.

All retrieved light directions L_k can be assembled into a matrix $\mathbf{L}$.

$$\mathbf{L}^T = \begin{bmatrix} 0.4979 & -0.4672 & -0.7306 \\ 0.2441 & -0.1376 & -0.9599 \\ -0.0386 & -0.1768 & -0.9835 \\ -0.0953 & -0.4443 & -0.8908 \\ -0.3214 & -0.5095 & -0.7982 \\ -0.1114 & -0.5652 & -0.8174 \\ 0.2828 & -0.4257 & -0.8595 \\ 0.1013 & -0.4335 & -0.8954 \\ 0.2073 & -0.3367 & -0.9185 \\ 0.0889 & -0.3344 & -0.9382 \\ 0.1298 & -0.0465 & -0.9904 \\ -0.1446 & -0.3644 & -0.9199 \end{bmatrix}$$

$\mathbf{L}^T$ denotes the transposed of the matrix $\mathbf{L}$.

I computed (x, y) coordinates the sphere center from the given mask matrix. The same holds true for the radius. For the radius I estimated the diameter of the ones-sphere in the mask. basically, I searched for the minimum and maximumum row and column index in the matrix, that was equal to one. Then I computed the difference between the maximum and minimum of - both, clumn, and row indices extrema indices pair - divided them by two (since the diameter of a circle is twice the radius) and took the mean of both resulting radii. Similarly I retrived the center of the sphere.

For every image, I computed the center of the specular highlight - i.e. the center of the spot. I shifted the center of the spot by the center of the chorme sphere

and retrieved the normal - using this shifted center - applying equation .

2. Computing Surface Normals and Grey Albedo (30 points) In this section you should:

- Describe the algorithm you used for calculating the albedo and normals given the light directions you estimated (or the approximated one which is provided in case you did not complete the task 1). You need to provide the formula you used to calculate the normals.
- Display the image of the recovered grayscale albedo map for each dataset.
- Display the images of the three normal components (x,y and z directions) or a single colour image with the x,y and z components instead of the R,G, and B components rispectively.
- Display the image of the RGB albedo map for each dataset.

From given estimated light directions (resulting after solving task 1) I computed the albedo and normals in the following way using the $N = 12$ provided color images:

For each pixel p in a given color Image, I computed its luminance p_l:

$$p_l = p_{red} \cdot 0.299 + p_{green} \cdot 0.587 + p_{blue} \cdot 0.114 \tag{9}$$

Where $(p_{red}, p_{green}, p_{blue})$ denote the RGB color of pixel p. I processed the pixel of an image row-wise and stored all the computed p_l pixel luminance values in a column vector denoted by l_k. The index k refers to the k-th color images (assuming they are labeled by an index). I repeated this procedure for every image of the N given color image. I then assambled all the N luminance vectors $l_1, ..., l_N$ to a Matrix I, i.e.

$$I = [l_1, ..., l_N] \tag{10}$$

Next I used the system of equations of Photometrix Stereo that we have discussed during the Computer Vision class. This system - shown in equation 11 - gives us a relation between the luminance, the light directions, the albedo and the surface normals.

$$I = S\tilde{\mathbf{n}} \tag{11}$$

- I is the matrix from equation 10, containing all the luminance vectors (its columns) of every provided image. This it is a N_p (number of pixels in a image) by N (number of images).
- S is a matrix containing all (estimated) light directions. Each of its rows contains one particular (estimated) light direction (from L). Thus, in our case it is a N_p by 3 matrix (three components per vector since we are in $\mathbb{R}^3$).
- $\tilde{\mathbf{n}}$ denotes the normal direction (not normalized, i.e. not unit length).

We can reformulate equation 11 the following way:

$$\begin{aligned} I &= S\tilde{\mathbf{n}} \\ S^T I &= S^T(S\tilde{\mathbf{n}}) \\ S^T I &= (S^T S)\tilde{\mathbf{n}} \\ (S^T S)^{-1} S^T I &= \tilde{\mathbf{n}} \end{aligned}$$

From the first equation to the second, we multiplied S^T from the left hand side. From the second to the third, we used the law of associativity and from the third to the fourth equation we simply multiplied $(S^T S)^{-1}$ from the left hand side. This gives us the final identity for the normal:

$$\tilde{\mathbf{n}} = (S^T S)^{-1} S^T I \tag{12}$$

Using equation 12 we can reconstruct all the normal directions $\tilde{\mathbf{n}}$. Last, we have to normalize this normal direction vectors. We only have to divide each vector $\tilde{\mathbf{n}}$ by the grayscale albedo ρ.

$$\mathbf{n} = \frac{\tilde{\mathbf{n}}}{\rho} \tag{13}$$

In order to do so, we first need to compute the grayscale albedo values ρ for every pixel of every Image. The grayscale albedo of a pixel p is simply the norm of the normal direction of pixel p, i.e.

$$\begin{aligned} \rho &= ||\tilde{\mathbf{n}}|| \\ &= \sqrt{\tilde{\mathbf{n}}_x^2 + \tilde{\mathbf{n}}_y^2 + \tilde{\mathbf{n}}_z^2} \end{aligned} \tag{14}$$

Therefore, we first have to compute the normal directions, using equation 12. Then we have to compute the grayscale albedo values using equation 14 and lastly, we use these grayscale albedos in order to compute the normals (i.e. normalize the normal direction vectors) using equation 13.

Once we have compute all normal vectors $\mathbf{n}$ we use them for solving the following system of equations:

$$\begin{aligned} I_{red} &= \rho_{red} S\mathbf{n} \\ I_{green} &= \rho_{green} S\mathbf{n} \\ I_{blue} &= \rho_{blue} S\mathbf{n} \end{aligned} \tag{15}$$

Where S is the matrix containing all light directions as before, I_{red}, I_{green}, I_{blue}, are the red, green and blue color channels of a corresponding color image and $\mathbf{a} = (\rho_{red}, \rho_{green}, \rho_{blue})$ denotes the color albedo.

We then have to solve for every pixel

$$\begin{aligned} I &= \mathbf{a}(S \cdot \mathbf{n}) \\ (S \cdot \mathbf{n})^{-1} I &= \mathbf{a} \end{aligned} \tag{16}$$

Where I is a tensor containing all color channels for every pixel in the image. Note that equation 16 corresponds to

$$\begin{aligned} I &= \mathbf{a}(\mathbf{n}) \cdot \mathbf{L} \\ (\mathbf{n} \cdot \mathbf{L})^{-1} I &= \mathbf{a} \end{aligned} \tag{17}$$

Results In the following I show my results applied to each provided data. Figure 1 shows the recovered albedo when applying Mathlab's *imagesc* function to our grayscale albedo. Figure 2 visualizes the luminance - i.e. the grayscale albedo. In addition I have produced results of the recovered RGB albedo as shown in figure 5. Last I offer the reader two different visualizations depicting the surface normals. Figure 3 shows the absolute values of the normal vectors, whereat figure 4 shows each normal direction separatly.

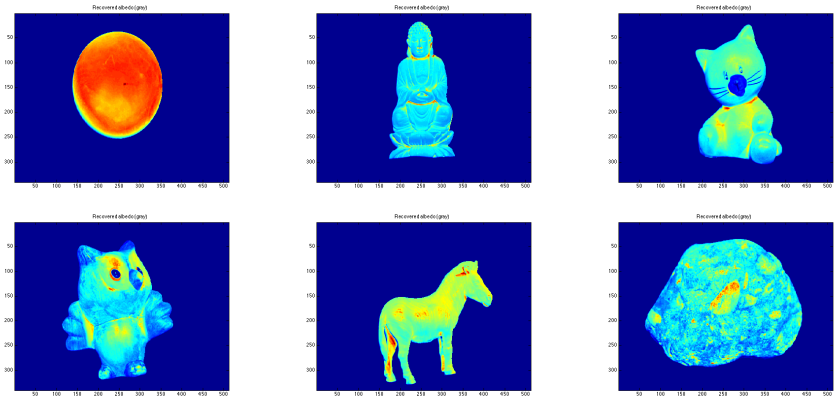

Fig. 1: Visualization of the recovered grayscale albedo (luminance) for each dataset

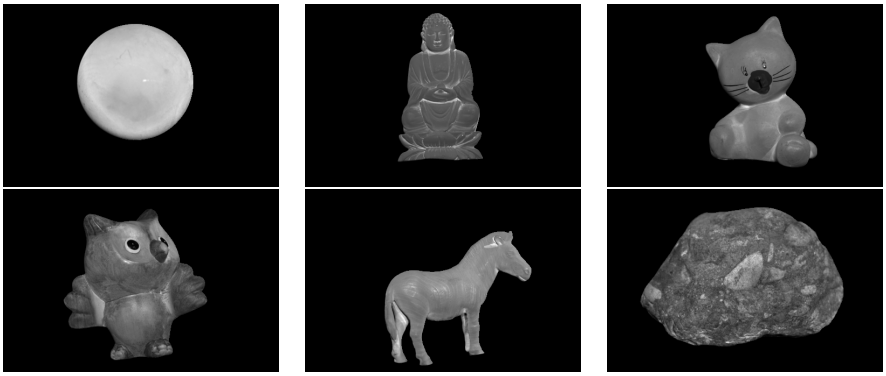

Fig. 2: Visualization of the grayscale albedo (luminance) for each dataset

3. Surface Fitting (35 points) In this section you should:

- Describe the algorithm you used for calculating the depth map given the normals you calculated before.

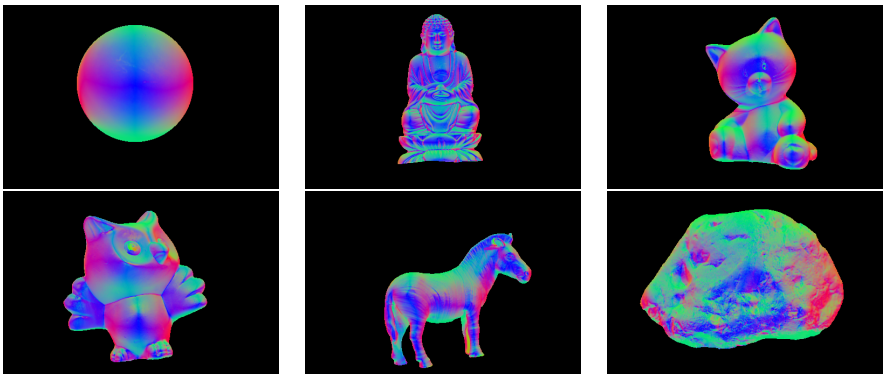

Fig. 3: Visualization of the normals for each dataset. Note that each image shows the absolute value applied to each component of each normal vector. The resulting three components are then interpreted as RGB color values. Hence, the x.-axis is pointing to the right, the y-acis is pointing upwards and the z-axis is pointing towards the viewer.

- Display the image of the depth map (in colour or grayscale) for each dataset, where higher intensity values indicate points closer to the camera.
- Describe, in no more than a few paragraphs, your assessment of when the technique works well, and when there are failures. When the technique fails to produce nice results, please explain as best as you can what the likely causes of the problems are.

Goal: Express the depth of a surface S as a function of a surface normal.
A surface S in $\mathbb{R}^3$ can be defined by a perametrization

$$S(x, y) = (x, y, z(x, y)) \tag{18}$$

Where $z(x, y)$ depicts the height field value at the point (x, y). For any point on such a paramterized surface S we can span a tangent plane using its tangent vectors at that point.

$$\begin{aligned} t_x &= \frac{\partial S}{\partial x}(x, y) = (1, 0, z_x)^T \\ t_y &= \frac{\partial S}{\partial y}(x, y) = (0, 1, z_y)^T \end{aligned} \tag{19}$$

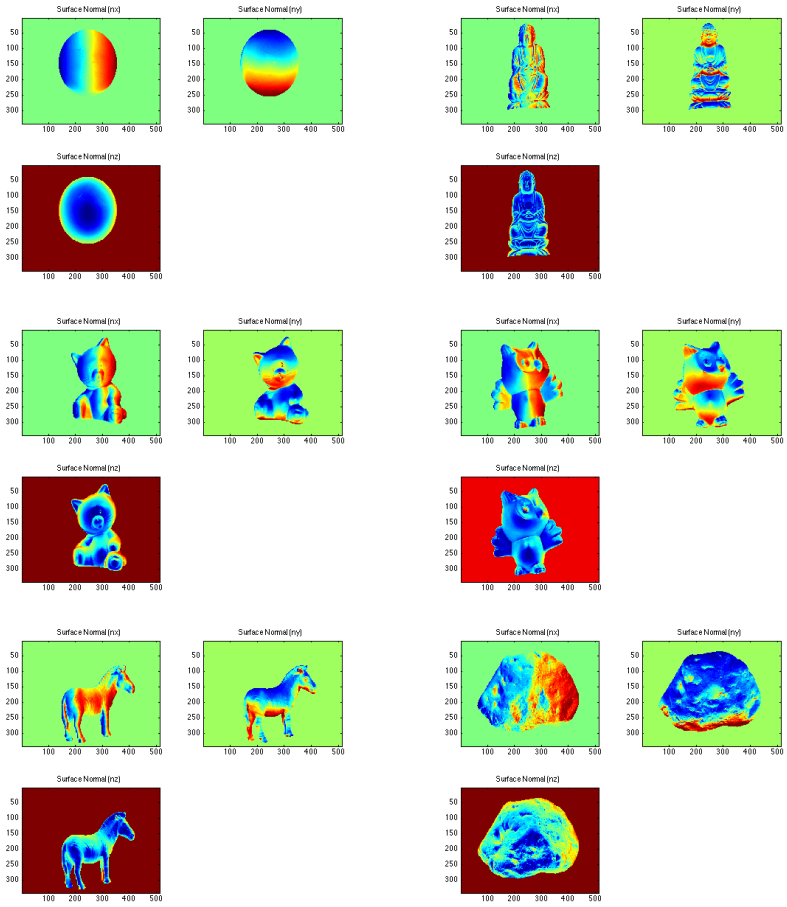

Fig. 4: Visualization of the of the three normal components (x,y,z direction) for each dataset.

Where z_x denotes the partial derivate of z along the x-axis and z_y the partial derivative along the y-axis respectively. The tangent plane spanned by those tangent vectors at a point (x, y) from equation 19 have the following normal vector:

$$\begin{aligned}\mathbf{n}(x,y) &= \frac{t_x(x,y) \times t_y(x,y)}{||t_x(x,y) \times t_y(x,y)||} \\ &= \frac{(-z_x, -z_y, 1)^T}{||(-z_x, -z_y, 1)^T||}\end{aligned} \tag{20}$$

Fig. 5: Visualization of the RGB albedo map for each dataset.

Where we simply have used the definition of the cross product applied to the two tangent vectors t_x and t_y.

Using a finite difference scheme we can compute numerically the derivatives of the z component used in equation 19. Finite differences origin from a taylor series approximation by using the first order terms. We get the following approximations for the derivatives of z along the x-and y-axis:

$$\begin{aligned} z_x(x,y) &= z(x+1,y) - z(x,y) \\ z_y(x,y) &= z(x,y+1) - z(x,y) \end{aligned} \tag{21}$$

Equation 21 allows us to solve for the tangent- and normal vectors defined as in equation 19 and equation 20 respectively.

Since every normal vector $\mathbf{n}$ is perpendicular to its tangent vectors t_x and t_y, i.e.

$$\begin{aligned} \mathbf{n} \cdot t_x &= 0 \\ \mathbf{n} \cdot t_y &= 0 \end{aligned} \tag{22}$$

we can use this equation 22 as constraint for our tangent vectors:

$$\begin{aligned} \mathbf{n}(x,y) \cdot t_x(x,y=0 \\ \mathbf{n}(x,y) \cdot t_y(x,y) = 0 \end{aligned} \tag{23}$$

For every point (x, y) on the surface S. Multiplying the equations in equation 23 out and Applying equation 21 to it, we get:

$$\begin{aligned} \begin{pmatrix} \mathbf{n}_x(x,y) \\ \mathbf{n}_y(x,y) \\ \mathbf{n}_z(x,y) \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ z(x+1,y) - z(x,y) \end{pmatrix} = 0 \\ \begin{pmatrix} \mathbf{n}_x(x,y) \\ \mathbf{n}_y(x,y) \\ \mathbf{n}_z(x,y) \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ z(x,y+1) - z(x,y) \end{pmatrix} = 0 \end{aligned} \tag{24}$$

We used $\mathbf{n} = (\mathbf{n}_x, \mathbf{n}_y, \mathbf{n}_z)$ where $\mathbf{n}$ is the vector from equation 20. Please note that to keep things simple I omitted to mention the normalization factor of the normalvector. Suppose this is encoded into its components $(\mathbf{n}_x, \mathbf{n}_y, \mathbf{n}_z)$.

The resulting two constraints from equation 24 are equivalent to:

$$\begin{aligned} -\mathbf{n}_x(x,y) = (z(x+1,y) - z(x,y))\, \mathbf{n}_z(x,y) \\ -\mathbf{n}_y(x,y) = (z(x,y+1) - z(x,y))\, \mathbf{n}_z(x,y) \end{aligned} \tag{25}$$

Thus, equation 25 gives us a linear system of equation which we can solve for the height value $z(x, y)$ on the surface S.

Note that if $z(x, y)$ sovles the system from equation 25 then also $z(x, y) + c$ $\forall c \in \mathbb{R}$ sovles it. However, we can enforce the existence of a unique solution by introducing an additional constraint

$$z(x_0, y_0) = 0 \tag{26}$$

for just one particular point (x_0, y_0) on S. In my inplementation is defined the top-left pixel to be the (x_0, y_0), i.e. set the pixel on the top-left corner equal to $(0, 0)$.

I applied equation 25 usign the constraint from equation 26 in order to retrieve the heights $z(x, y)$, i.e. the depth map, for every pixel on the surface S. I applied my algorithm to every dataset. My results are shown in figure 6.

Keep in mind to use the definition from equation 20 in order to compute the actual normal values.

Last, let us discuss what are issue-cases for the method discussed in order to retrieve the depth maps. Note that a successful reconstruction is never guaranteed. However, the more images with diverse light directions we use the better the results will get. Also, using more images will reduce the effect of self occlusions and also noise can be reduced.

In the following a list of so called (potential) failure-cases:

Failure cases

- If the material does not exhibit specular hightlights, the method will fail. Actually, materials are assumed to reflect light as a lambertain material does.
- surface must be sufficiently smooth. Objects with many sharp edges will produce bad results.
- Inter reflection is not taken care of.
- Shadows are an issue case, since this method does not take care of self-occlusion.

Therefore, our method will work rather well if the material is sufficently smooth and it has specular highlights. Furthermore, we also should not be interested in shadowing effects, due to the fact mentioned in the failure-case list.

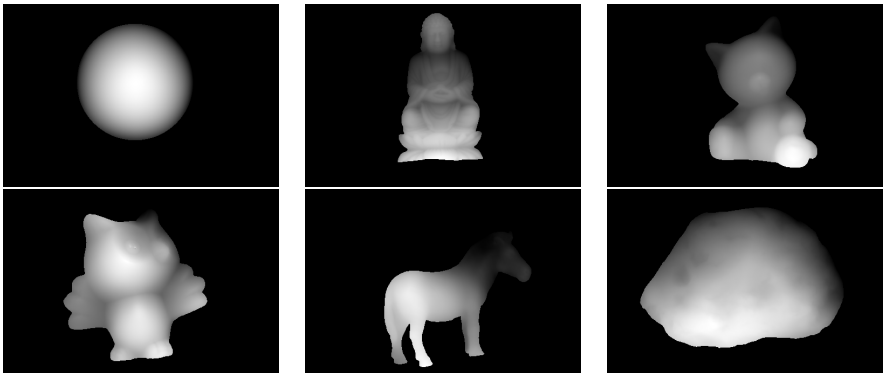

Fig. 6: Visualization of the depth map for each dataset. The brighter a pixel looks, the closer it is located to the camera.