multimedia-hw4

by 108062138 Po-Yu, Wu

Readme.md (http://Readme.md) contains only the *execution* part in `report.pdf`

report link (https://hackmd.io/@sBeNJ4fqRNqa67PhyWWV4A/rkK1Qljrn)

1. Bézier curve (50%) You are given 1 raster image (bg.png) and 1 path object (Bézier curve,extracted as points.txt) and a sample script to draw the point and line(**problem1.py** (http://problem1.py)).bg.png

a. Plot curve and point on the bg.png by using 2 different settings.

I. Low detail curve: t = {0, 0.5, ..., 1.0} ; High detail curve: t = {0, 0.01, 0.02,..., 1.0} ,and save the result as 1a.png with 2 curves on bg.png.

- 

II. Discuss how you implement the Bézier curve and the differences between high and low detail curves.

- The the Bézier curve, by the professor's slide, is composed by basis matrix M, geometry matrix G(arbitary four point), and T, which is in descending

power order. With any given 4 point, I can utilize the equation $P(t) = (T * M) * G$ in the slide and get their Bézier curve. The following lines.

- basis matrix: line 2-5 in the following code
- geometry matrix: `np.array([p0, p1, p2, p3])` in line 20
- T: t_matrix = `np.array([[my_t**3, my_t**2, my_t, 1]])` in line 19
- $P(t) = (T * M) * G$: implemented in line 17-21

- According to the discussion in eeclass, TA requires us to generate a Bézier curve every four point and each set of points is overlapped by 1. Thus, I write `for i in range(0, len(points)-3, 3):` to fit this requirement.
  - For example, given 10 points, three set: `{0,1,2,3}`, `{3,4,5,6}`, `{6,7,8,9}` correspond to 3 Bézier curves, respectively.

- Difference: High resolution curve(red one) locates closer to the dots as compared to the low resolution curve. Their difference becomes significant when we focus on Mr. Beast's peaked cap.
  - The following image is a screen shoot from `1a.png`. Each side of the hat contains a dot that seems astray.

  - I think that turns out to be a nice instance to demonstrate the difference in-between low resolution and high resolution curve. The high resolution curve(red) has much small distance with the astray dot while the low resolution curve looks faraway from the dot.

- Such difference comes from the sampling rate of the curve. Given arbitary 4 dots, the low resolution curve depictes their curve by only 3 dot, while the other depictes their curve by 101 dots. The amount of dots to drawing the curve directly makes the high resolution one fit more to the dots.

```python
def bezier_curve(points,setting):
    basis_matrix = np.array([[-1, 3, -3, 1],
                             [3, -6, 3, 0],
                             [-3, 3, 0, 0],
                             [1, 0, 0, 0]])
    result = np.empty((0, 2), float)
    if setting=='low_detail':
        step = 0.5
    else:
        step = 0.01
    for i in range(0, len(points)-3, 3):
        p0 = points[i]
        p1 = points[i+1]
        p2 = points[i+2]
        p3 = points[i+3]
        int_step = int(step*100)
        for t in range(0, 100+int_step, int_step):
            my_t = t/100
            t_matrix = np.array([[my_t**3, my_t**2, my_t, 1]])
            p = np.dot(np.dot(t_matrix, basis_matrix), np.array([p0, p1, p2, p3]))
            result = np.append(result, p, axis=0)
    print(result.shape)
    return result
```

b. Scale up the img by 4 and plot it.

I. BG.PNG IS SCALED USING NEAREST-NEIGHBOR INTERPOLATION.

> `tqdm` is used to relive my stress when executing this homework

- I reuse the NN interpolation implementation from hw1.

```python
def bilinear_interpolation(img,n):
    height, width, channel = img.shape
    new_height = height * n
    new_width = width * n
    res = np.zeros((new_height, new_width, channel), dtype=np.uint8)

    def get_coef(lu,ld,ru,rd,i,j,n):
        orig_i = i/n
        orig_j = j/n
        h1 = orig_i - lu[0]
        h2 = ld[0] - orig_i
        w1 = orig_j - lu[1]
        w2 = ru[1] - orig_j
        return h1*w1, h1*w2, h2*w1, h2*w2

    for i in tqdm(range(new_height)):
        for j in range(new_width):
            lu = np.floor([i/n,j/n]).astype(int)
            ru = np.floor([i/n,j/n+1]).astype(int)
            ld = np.floor([i/n+1,j/n]).astype(int)
            rd = np.floor([i/n+1,j/n+1]).astype(int)
            if ru[1] >= width:
                ru[1] = width - 1
            if ld[0] >= height:
                ld[0] = height - 1
            if rd[0] >= height:
                rd[0] = height - 1
            if rd[1] >= width:
                rd[1] = width - 1
            if lu[0] <0 or lu[1] <0 or ru[0] <0 or ru[1] >=width or ld[0] >= height or ld[1] <0 or rd[0] >= height or rd[1] >= width:
                raise Exception("fuck up")
            coe_rd, coe_ld, coe_ru, coe_lu = get_coef(lu,ld,ru,rd,i,j,n)
            res[i,j] = coe_rd*img[rd[0],rd[1]] + coe_ld*img[ld[0],ld[1]] + coe_ru*img[ru[0],ru[1]] + coe_lu*img[lu[0],lu[1]]
    return res
```

II. THE COORDINATES OF THE CURVE IS SCALED BEFORE THE CURVE IS DRAWN, ALSO YOU SHOULD USE THE SETTING OF HIGH DETAIL CURVE.

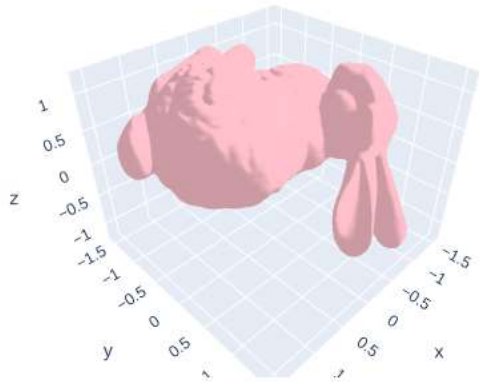
- The size of the image is (1400~ x 2500~), equivalent to four times of the original image(350~ x 600~)

- 

2. 3D Models (50%)Launch the script (problem2.ipynb) to implement the translation and illumination of bunny.obj and a 3D surface

respectively. Here, we use plotly, a python graphing library, to help us render 3D graphics/bunny.obj 3D surface

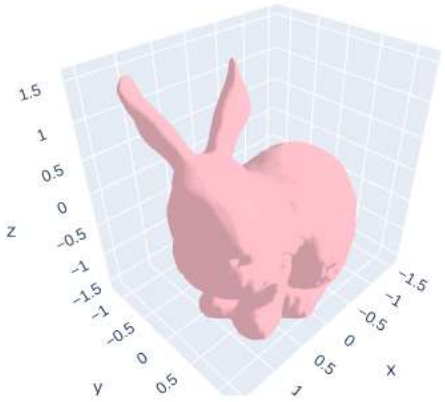
a. Shift the center of the bunny to (0, 0, 0) and save the figure as 2a.png. Center is defined as [(max(x) + min(x)) / 2, (max(y) + min(y)) / 2,(max(z) + min(z)) / 2] where x, y, z are the vertices of the object.

- 

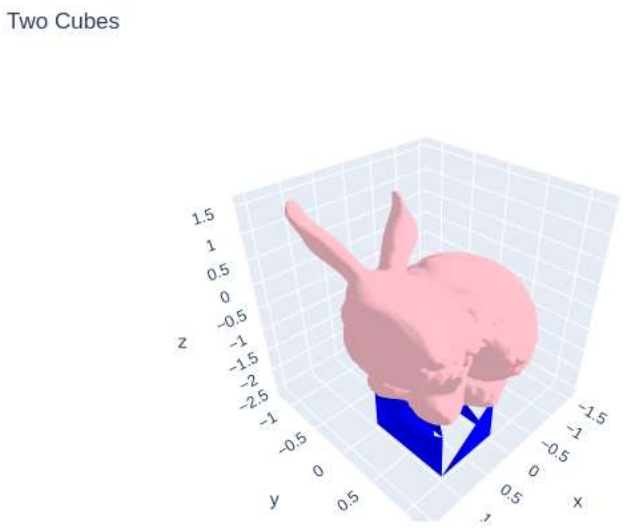
b. Based on (a), implement the rotation matrix to rotate the bunny to face the screen(like the following example). Discuss how you implement the rotation matrix and save the 3D figure as 2b.png.

- The rotatinon matrix is defined as follow(line2-13), which is similar to the slide from the professor. Since I just center(?) the rabbit, I can directly apply these rotation matrix. I utilize `rotate_x` to let the bunny's foot touch the ground and `rotate_z` to make the bunny's face toward the screen(line 24-25).

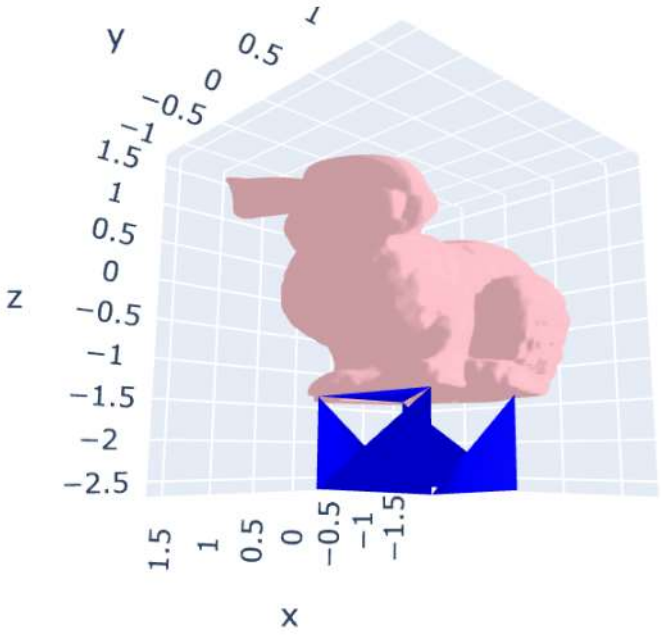
```
#rotate matrix
def rotate_x(theta):
    return np.array([[1, 0, 0],
                     [0, np.cos(theta), np.sin(theta)],
                     [0, -np.sin(theta), np.cos(theta)]])
def rotate_y(theta):
    return np.array([[np.cos(theta), 0, -np.sin(theta)],
                     [0, 1, 0],
                     [np.sin(theta), 0, np.cos(theta)]])
def rotate_z(theta):
    return np.array([[np.cos(theta), np.sin(theta), 0],
                        [-np.sin(theta), np.cos(theta), 0],
                        [0, 0, 1]])

turned_x = shifted_x.copy()
turned_y = shifted_y.copy()
turned_z = shifted_z.copy()

turn_x_90 = rotate_x(-np.pi/2)
turn_y_90 = rotate_y(-np.pi/2)
turn_z_90 = rotate_z(-np.pi)
for i in range(0,len(shifted_x)):
    vec = np.array([shifted_x[i],shifted_y[i],shifted_z[i]])
    vec = np.dot(turn_x_90, vec)
    vec = np.dot(turn_z_90, vec)
    turned_x[i] = vec[0]
    turned_y[i] = vec[1]
    turned_z[i] = vec[2]
```

- 

c. Based on (b), try to put a cubic under the foot of the bunny and save the result as 2c.png.

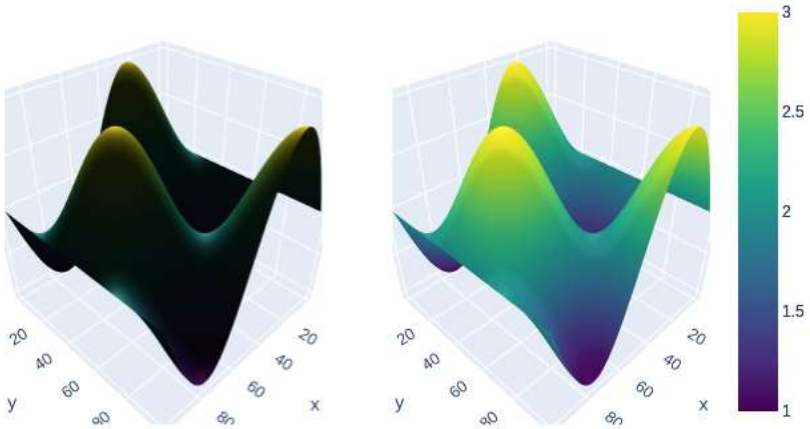
- 

- a closer peek:



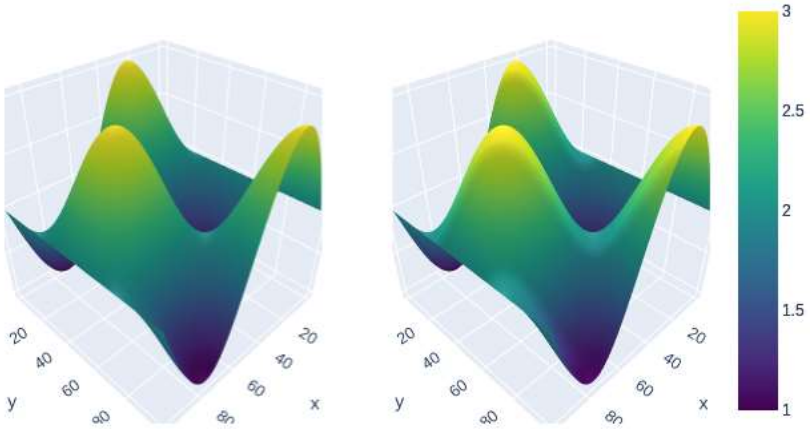
d. Given a surface, try different ambient strength ka, diffuse strength kd, specularstrength ks in the following settings:

I. (KA , KD , KS) = (0.1, 0.8, 0.05) AND (KA , KD , KS) = (0.9, 0.8, 0.05)

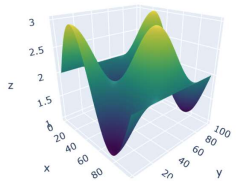
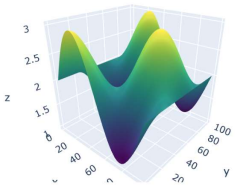
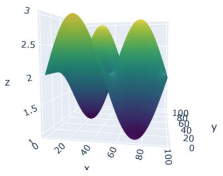
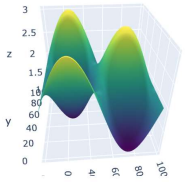
- 

- The difference is `ka`, which is ambient value. The ambient light projects on the surface seems darker when `ka` is small.

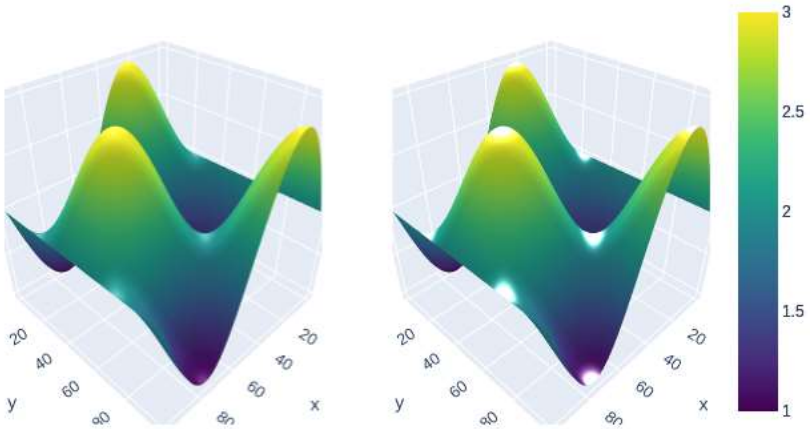
II. (KA , KD , KS) = (0.8, 0.1, 0.05) AND (KA , KD , KS) = (0.8, 0.9, 0.05)

- 

- The difference is `kd`, which is diffuse value. The surface's color varies a lot when `kd` is large. Aside from that, the fade of the surface becomes more significant when `kd` is large. I offer an extra screen shoot for I think `fig.write` 's saving result fail to show their difference.

III. (KA , KD , KS) = (0.8, 0.8, 0.2) AND (KA , KD , KS) = (0.8, 0.8, 2.0)

- 

- The differenceis `ks` , which is specular value. The one with large `ks` looks shiner than that with smaller `ks` . Large `ks` 's peaks are too shine such that they look pale in my screen.

PLOT THE RESULT FOR EACH CONDITION WITH 2 SUBPLOTS AND DISCUSS THE DIFFERENCE BETWEEN EACH SETTING. RESULTS ARE SAVED AS 2D_1.PNG, 2D_2.PNG, 2D_3.PNG RESPECTIVELY. YOU CAN REFER TO THE FOLLOWING LINK FOR MORE INFORMATION

- result and their discussion is shown in `2-a` , `2-b` , `2-c`