Computer Graphics
Jacobs University Bremen
Dragi Kamov

Course: CO19-320322
June 2nd, 2019

Homework 3

This assignment was solved together with Fezile Manana and Brian Sherif.

Exercise 3.1

Solution:

1. Mid-points of pixels on the screen:
   $(1, 1, 0), (1, -1, 0), (-1, -1, 0), (-1, 1, 0)$
   Ray vectors from camera (0, 0, -10):
   $\vec{r_1} = \begin{pmatrix} 1 \\ 1 \\ 10 \end{pmatrix} \therefore r_1 = x = y = \frac{z+10}{10}$
   $\vec{r_2} = \begin{pmatrix} 1 \\ -1 \\ 10 \end{pmatrix} \therefore r_2 = x = -y = \frac{z+10}{10}$
   $\vec{r_3} = \begin{pmatrix} -1 \\ -1 \\ 10 \end{pmatrix} \therefore r_3 = -x = -y = \frac{z+10}{10}$
   $\vec{r_4} = \begin{pmatrix} -1 \\ 1 \\ 10 \end{pmatrix} \therefore r_4 = -x = y = \frac{z+10}{10}$
   The only intersecting ray is r_1 intersecting the red triangle at point $(1.2, 1.2, 2)$

2. Phong illumination model formula:

$$I_p = k_a i_a + \Sigma_{m \in lights} (k_d (\hat{L}_m \cdot \hat{N}) i_{m,d} + k_s (\hat{R}_m \cdot \hat{V})^\alpha i_{m,s}$$

   where $\hat{N}$ is the normal, $\hat{V}$ is the direction pointing towards the viewer, α is the shininess constant and R_m is the direction of the reflected ray of light given by

$$R_m = 2(\hat{L}_m \cdot \hat{N})\hat{N} - \hat{L}_m$$

   Given:

   Vertices: $v_1 = (0, 4, 2)$; $v_2 = (0, 0, 2)$; $v_3 = (4, 0, 2)$;$v_4 = (5, 5, 2)$
   Triangle 1 ⇒ vertices = (v_1, v_2, v_3); RGB = $(1, 0, 0)$
   Triangle 2 ⇒ vertices = (v_1, v_2, v_4); RGB = $(1, 1, 0)$
   $k_a = (\frac{2}{3}, \frac{2}{3}, \frac{2}{3})$
   $k_d = (\frac{1}{3}, 0, \frac{2}{3})$
   $k_s = (\frac{1}{9}, \frac{1}{9}, \frac{1}{9})$
   $\alpha = 5$
   $V = (0, 0, -10)$
   $L = (5, 5, 1)$
   $N = (0, 0, -2)$
   With this we get intensities for ambient, diffuse and specular reflections:
   $I_{v_1} = \begin{pmatrix} 0.730912 \\ 0.666762 \\ 0.795062 \end{pmatrix} \quad I_{v_2} = \begin{pmatrix} 0.713349 \\ 0.666673 \\ 760025 \end{pmatrix} \quad I_{v_3} = \begin{pmatrix} 0.730912 \\ 0.666762 \\ 0.795062 \end{pmatrix} \quad I_{v_4} = \begin{pmatrix} 1 \\ 0.719409 \\ 1 \end{pmatrix}$
   Computed colours:
   $RGB_{v_1} = (0.730912, 0, 0)$
   $RGB_{v_2} = (0.713349, 0, 0)$
   $RGB_{v_3} = (0.730912, 0, 0)$
   $RGB_{v_4} = (1, 1, 0)$

3. As only one sample ray intersects with the triangle, this is the only ray we need to evaluate. Therefore we need to find the colour at point $p = (1.2, 1.2, 2)$

   Distance from light to camera via vertices:
   $d_{v_1} = 17.8453$
   $d_{v_2} = 19.1414$
   $d_{v_3} = 17.8453$

   Colour at vertices with attenuation $f(r) = \frac{1}{r}$
   $RGB_{v_1} = (0.0409583, 0, 0)$
   $RGB_{v_2} = (0.0372673, 0, 0)$
   $RGB_{v_3} = (0.0409583, 0, 0)$

   Therefore, colour at sample point is:
   $RGB_p = u \cdot RGB_{v_1} + v \cdot RGB_{v_2} + w \cdot RGB_{v_3} \Rightarrow (0.0350526, 0, 0)$
   where u, v, and w are attained from Barycentric coordinates about $p = (1.2, 1.2, 2)$ giving $u = -0.3$, $v = 1.6$, and $w = -0.3$.

Exercise 3.2

Solution:

	10	11	12	13	14	15	16	17	18
120	■	■							
119			■	■					
118					■	■	■		
117								■	■

Brensenham's Algorithm [1]

```
function line(x0, y0, x1, y1)
    real deltax := x1 - x0
    real deltay := y1 - y0
    real deltaerr := abs(deltay / deltax) // We assume that the line is not vertical
    real error := 0.0 // No error at start
    int y := y0
    for x from x0 to x1
        plot(x,y)
        error := error + deltaerr
        if error >= 0.5 then
            y := y + sign(deltay) * 1
            error := error - 1.0
```

$$d_x = 18 - 10 = 8$$
$$d_y = 117 - 120 = -3$$
$$d_{err} = |\frac{d_y}{d_x}| = \frac{3}{8} = 0.375$$

Step	x	y	error
1	10	120	0.375
2	11	120	-0.25
3	12	119	0.125
4	13	119	-0.5
5	14	118	-0.125
6	15	118	0.25
7	16	118	-0.375
8	17	117	0
9	18	117	0.375

Xiaolin Wu's Algorithm [2]
Please run 'python3 xiaolin.py img.jpg' and see the results.

Step	x	y	alpha
1	11	119	0.375
2	11	120	0.625
3	12	119	0.75
4	12	120	0.25
5	13	118	0.125
6	13	119	0.875
7	14	118	0.5
8	14	119	0.5
9	15	118	0.875
10	15	119	0.125
11	16	117	0.25
12	16	118	0.75
13	17	117	0.625
14	17	118	0.375

Exercise 3.3

Solution:
Please check 'bouncingballs.cpp'.
To compile and execute enter the following lines in the terminal:

```
$ g++ -Wall -o test bouncingballs.cpp -lglut -lGLU -lGL
$ ./test
```

References

[1] Bresenham's line algorithm
[2] Xiaolin Wu's line algorithm