Homework 3, Computer Graphics 2019

Dr. Szymon Krupiński, Jacobs University Bremen

Handed out 16.05.2019, due 28.05.2019 at 23:55

This is a mixed assignment. Please upload the written answers in pdf (generated in word, latex or simply scanned) using the assignment submission on moodle. You can program in C, C++ or python. Please only submit the necessary files, i.e. documents and source code files (.c, .cpp or .py) compressed together in a *.zip file. If you need library functions, use the libraries mentioned in lecture slides. To qualify for full points, your code should

- compile (if necessary) and execute cleanly
- have readable, clean code with no unnecessary operations
- produce results which are evident (= set the initial position/view/projection right, add lighting, etc)
- comply with the corresponding problem requirements defined below.

Please sign each file with an appropriate comment block in the beginning:

```
/*
Computer Graphics 2019
Problem X.Y
<First_name> <Last_name>
<your_email>@jacobs-university.de
*/
```

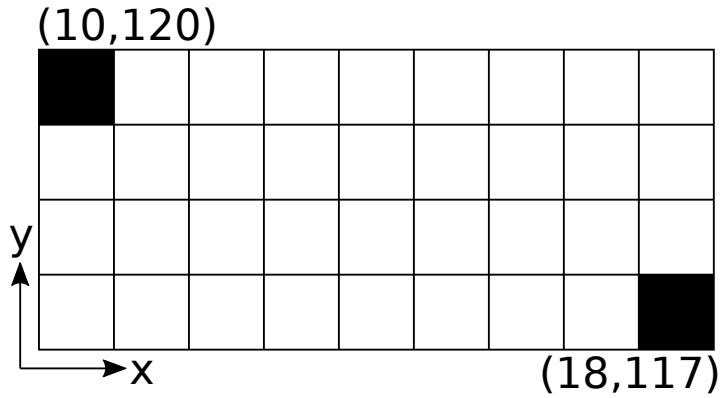
Please upload the assignment submission on moodle (`http://moodle.jacobs-university.de`) of our course. In case of technical problem with moodle submission, notify us and send it to the TAs (m.thanasi@jacobs-university.de, muh.hassan@jacobs-university.de) **before** the deadline.

Problem 3.1 *Shading and Illumination* (12 points)

Given a triangle fan with vertices $v_1 = (0, 4, 2)$, $v_2 = (0, 0, 2)$, $v_3 = (4, 0, 2)$, and $v_4 = (5, 5, 2)$ in 3D Cartesian coordinates. The first triangle is red (don't forget to write down the RGB values you are using), and the second is yellow while the background shall be black. In the same coordinate system, we define a view point $p = (0, 0, -10)$. The screen is a quadrilateral defined by the following vertices: $s_1 = (-2, -2, 0)$, $s_2 = (2, -2, 0)$, $s_3 = (2, 2, 0)$, and $s_4 = (-2, 2, 0)$. The resolution of the screen shall be 2×2 pixels. The triangles have ambient reflectance coefficient $(\frac{2}{3}, \frac{2}{3}, \frac{2}{3})$, diffuse reflectance coefficient $(\frac{1}{3}, 0, \frac{2}{3})$, specular reflectance coefficient $(\frac{1}{9}, \frac{1}{9}, \frac{1}{9})$, and specular exponent 5. Finally, we have the light source with full-intensity white light (R = G = B = 1.0) at location $l = (5, 5, 1)$. Light attenuation shall follow the function $f(r) = \frac{1}{r}$.

1. Using the view point (camera's origin), the screen and the pixel's position in the screen, set up the rays going through the sampling points in the middle of pixels and calculate the points of their intersection with the geometry.
2. Apply the Phong illumination model to the vertices of the triangles.
3. Apply shading. Assuming a ray casting approach, compute the color for a sample point in the middle of each pixel by applying Gouraud shading to the RGB colors computed in the previous point (as explained in class 12 "Textures and interpolation").

Problem 3.2 *Bresenham's algorithm* (5 points)



The drawing above shows two points (pixels colored in black) which need to be connected by an edge using Bresenham's algorithm.

- Provide the drawing of the pixels of the finished line.
- Fill in the table below with the consecutive point coordinates to be filled in. Provide the current value of the error variable. Add as many rows as necessary.

If you are using a version of the algorithm different than the one provided in the class slides, write it down (or provide the URL) for reference.

Step	`x`	`y`	error (D)
1			
2			
3			

Bonus [5 points]
Repeat the same exercise on the provided points using Xiaolin Wu's line algorithm and augment the table with a column for the `alpha` value for each step.
You can find the algorithm on Wikipedia: `https://en.wikipedia.org/wiki/Xiaolin_Wu's_line_algorithm`

Problem 3.3 *Animation: bouncing balls* (8 points)

Get the source file `bouncingballs.cpp` from the following URL: `https://cs.lmu.edu/~ray/notes/openglexamples/`

Your task is to fix the movement of the objects to follow the usual behavior of falling/bouncing bodies: the Newtonian gravitation. The bodies undergo constant downward acceleration of $g = 9.81\frac{m}{s^2}$, i.e. every second the value of $9.81\ \frac{m}{s}$ is added to their downwards velocity or subtracted from their upwards velocity. It also means that the distance that a body initially falling at speed v_i covers in time t is governed by the formula $d = v_i t + \frac{1}{2}gt^2$. You can refresh your high-school physics knowledge here:
`https://www.open.edu/openlearn/science-maths-technology/describing-motion-along-line/content-section-5.1`.
You don't have to modify/re-model the collision and bouncing itself, just the position update.

Bonus [15 points]
Modify the code to use modern style OpenGL data pipeline, i.e. a pair of GLSL shaders (vertex and fragment). You are allowed to simplify the bodies (e.g. make them cubes) and use the simplest reflection and shading model available more complex than simply attributing a constant color to all the faces.

You will get...

- 9 bonus points for correctly setting up even simple shaders (they must compile and display geometry!) and the VBO/VAO. The motion/bouncing of the objects should work (you don't have to be physically realistic);
- 4 bonus points for successfully applying any reflection shading model better than constant color;
- 2 bonus points for keeping the checkered texture on the bottom plane using GLSL.