Computer Graphics - 2016/2017

Exercise 1 - Part B

In this part of the exercise you will practice:
1. Homogenous coordinates and basic geometric transformations.
2. Animation.
3. Non-physically correct (“Faked”) lighting effect.
4. Simple collision detection.

Note: You MUST submit this exercise in pairs.

General guidelines:

(You might lose points for not following these guidelines)

- Make sure that you understand the effect of every character that you write in your code.
- Make sure that your code does what it’s supposed to do.
- Do not unnecessarily change existing code.
- Keep your code readable and clean!
    - Avoid code duplication.
    - Comment non-trivial code regions.
    - Block together (using ‘{‘ and ‘}’) logically related lines of code.
    - Preserve coding conventions when changing existing code.
- Keep your code efficient. In particular:
    - Do not unnecessarily transfer data to OpenGL (e.g. share vertices between triangles and similar shapes).
    - Do not unnecessarily allocate memory.
    - Efficiently divide the work between OpenGL and the hosting program.
    - Minimize the number of vertices being processed by OpenGL while preserving the visual quality your drawing.
- Add to your Readme.txt a list of all web-pages URLs that you used in order to complete this exercise.
- Add to your Readme.txt a list of all students’ usernames that you discussed this exercise with.

0. Download and play with the school solution

Download the school solution from here. Unzip it. Run `ex1` to run it. Click inside the window to generate additional balls. Plan to clone the school solution behaviour when you fulfill the following tasks.

1. Animate your single ball

Start with your Ex1-partA solution, and repeatedly update the position of the ball (i.e., the circle that your solution draws) as time progresses. Specifically, at each time step, advance the ball according to its current direction. Whenever the ball collides with the window boundaries, change its direction as if it was an ideal elastic bouncing off a wall (in a 2D world).

2. Create additional balls in response to user clicks

Handle the mouse click event to create an additional ball centered at the click position. Initialize the new ball with a random color and random direction, you should make sure that it's not too dark or too bright.

A simple implementation may loop over the balls with of glDrawArray.

More sophisticated implementations (**not required**) may take advantage of glDrawArraysInstanced. If you use glDrawArraysInstanced, use gl_instanceid and gldrawarraysinstanced (you should have an instance per ball). Dynamic arrays can't be passed as uniforms, so you can assume a limited number (m > 2) of balls per call to gldrawarraysinstanced and call it ceil(n / m) times to draw all balls. You may also have to set a z index (in the vertex buffer) per ball, so the drawing result would be the same order as calling glDrawArray multiple times.

3. Avoid ball-to-ball collisions by temporary shrinking

Avoid ball-to-ball position by temporarily shrinking the balls radii. Unshrink up to the original ball size when possible.

Note 1: If the user creates a new ball too close to a wall or another ball, you should create it with a smaller radius to avoid collisions. This will be the ball maximum radius for its lifetime (see the school solution).

Note 2: Preserving the aspect ratio (as you should) does not mean that you have to treat the balls as ellipses.

4. Fake light-sources / highlights with radial gradients

Fill each ball with a gradient, centered according to the ball's and a global (invisible) light source's position. The light source position should be fixed (choose a position outside the window boundaries that generates plausible results). The gradient should start with white color and ends with the ball’s fill color. Please refer to the slides on the course website and the school solution as an example.

5. Bonus (up to 2 points for perfect results)

5.A. Create an additional lights-source (in another location) and draw two highlights on each ball.
5.B. Animate the (invisible) light-source(s) position(s) in circles around the window.

6. Bonus (up to additional 7 points)

Implement an anti-aliasing algorithm for the balls in your fragment shader. You may decrease the visual radius of the ball in 1 pixel. Remember that no overlap is possible between balls (this assumption makes this task not as hard as it might be in the general case).
Set the alpha (transparency) channel of the ball according to your calculations. You may need to use OpenGL blend function glblendequation and glblendfunc to tell OpenGL how to draw non-opaque pixels.

Submission

Include the following in your submission:

1. A Readme.txt file, that includes:
   - your id and login
   - your partner’s id and login
   - A brief description of your implementation and the changes you made between ex1-part1 and ex1-part2
   - A brief description of each file of the other files that you submit
2. All files that are required for compilation of your solution with a single ‘make’ command, and the shaders necessary to run it.
3. After running ‘make’, it should be possible to run your solution with ‘ex1b’ command.

Pack all files as a single zip file named by the following pattern: ex1_<your 9 digits id>_<your_username>_<your partner’s 9 digits id>_<your partner’s 9 digis login>.zip (e.g. ‘`ex1_123456789_mylogin_987654321_myfriendlogin.zip`’).

Deadline:
You have to submit your solution (via the course’s moodle webpage) no later than Tuesday 29/11 at 23:55.
Late submission will result in 2^(N+1) points deduction where N is the number of days between the deadline and your submission (rounded up, the minimum grade is 0, friday and saturday are excluded).