E-fólio A | Instruções para a realização do E-fólio

ABERTA

COMPUTER GRAPHICS | 21020

Realisation period

It runs from 28 October to **20** November 2023

Delivery Deadline

20th November 2023, until 23:55 in mainland Portugal

Work to be done:

This work consists of two parts: first you have to implement the midpoint algorithm for drawing line segments, as a JavaScript module. Next, you will have to build a three.js graphical interface that uses this module to draw segments on the screen, in the form of boxes (tiles) on a plane in a 3D environment. The plane will represent a virtual raster display, where you can select pairs of "pixels" with the mouse and keyboard, and draw segments between these points using the midpoint algorithm.

PART I- Implementation of the midpoint algorithm (2 values):

You must implement the general case of the midpoint algorithm in the form of a JavaScript module.

The following implementation parameters are mandatory (and eliminatory):

- You should represent points by JavaScript object literals in the form {x: int, y: int}. For example {x:2, y:4} represents the point (2,4). The values of x and y are always integers.

- The module must be implemented in a file called lineMP.mjs (note the "mjs" extension) and this module has to export a function called lineMP.

- This lineMP function takes as input a pair of points and as output an array of points that corresponds to the result of the midpoint algorithm.

- The algorithm has to work for any pair of integer coordinate points (whether positive or negative), giving the correct representation of the segment according to the midpoint algorithm.

Example:

let P = {x: 0, y: 0}; let Q = {x: 3, y: 1};

let R= lineMP(P,Q);

console.log(R); // print [{x: 0, y: 0}, {x: 1, y: 0}, {x: 2, y: 1}, {x: 3, y: 1}]

Important note: it is essential that the algorithm **runs** and that you check the interface **specifications**! Follow the nomenclature and specifications presented above for functions and objects to the letter. Your algorithm will be tested by a programme that assumes these names before the code is even inspected.
Modules that do not pass this test (that do not run, or that do not respect the specified input and output parameters or that do not export correctly from the module) will not be evaluated.

For example, a file named lineMP.js instead of lineMP.mjs will not be evaluated. A lineMP function that accepts points in the form of arrays instead of objects ([0,1] instead of {x:0, y:1}) will not be evaluated.

Programmes that run and are properly specified will be judged on the correctness, efficiency and elegance of the code. They will also be judged on their documentation.

PART II- Graphic Interface (2 values)

You need to make a web page in three.js that works as follows:

1.

The page should run from an index.html file located in the home directory of your work.

The code should be imported from index.html, in the form of modules. The JavaScript code should **preferably** be contained entirely in modules and not inline in the html. The html should only import and run an initial module, which calls the others.

One of these modules will in turn have to import three.js and OrbitControls. These imports will have to be made from the CDN, as follows:

import * as THREE from 'https://unpkg.com/three@0.124.0/build/three.module.js';

import { OrbitControls } from
'https://unpkg.com/three@0.124.0/examples/jsm/controls/OrbitControls.js'

It is essential that all these dependencies work (with the exception, of course, of cases where the server is unavailable). **Only code that runs will be evaluated**.

You will also need to import the lineMP.mjs module that you designed in the section above to implement the midpoint algorithm. This module should be in the base directory. The other modules should be in the "./src" directory.

2.

The page should show a plane divided into coloured squares (with alternating colours, to help visualise the divisions between squares). This plane represents a *raster* display, with each square representing a pixel. In addition to the "pixels", the positive x and y axes should be visible, represented by thin lines running through the centre of the pixel (0,0); the positive x axis is represented by a blue line and the positive y axis by a red line.

The display must have at least 10 squares in each direction (both for the positive and negative sides of the centre pixel, so it will have at least 21 squares on a side).

The user must be able to control the view using the OrbitControls in three.js, with the view centred on the centre of the reference (centre of pixel (0,0)).

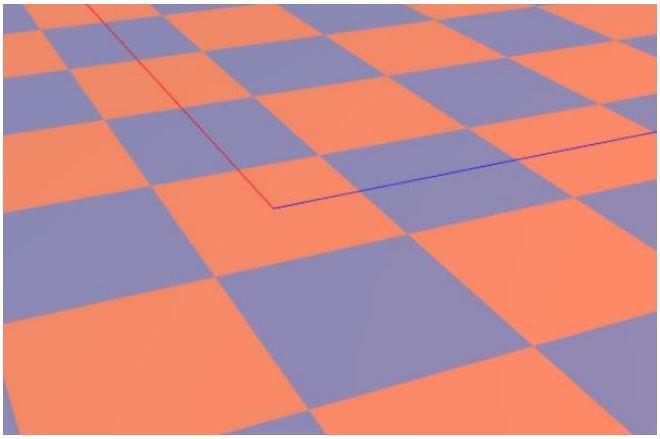
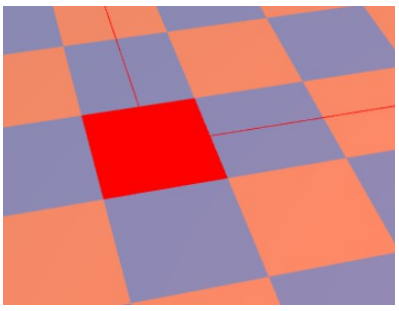


Fig 1: The virtual raster "screen" in its initial state, with all the pixels erased.

Using the three.js *raycaster,* the programme should be able to detect the square the mouse is on at that moment. The programme should indicate to the console the coordinate of the pixel it is over at each instant (it should only log the moment it changes square). Note that the
the issue is the indexing of the square on the grid, and not necessarily its "world coordinates".

When the user presses the "X" key on the keyboard, the pixel in the grid where the mouse pointer is located should change colour to red and the pixel's coordinates in the grid should be stored. Example: In the image below we pressed the X key when the mouse was over the centre pixel. The point {x:0, y:0} was saved in this case and the pixel was marked in red.



When the user again presses the "X" key on the keyboard with the mouse over a second pixel, the following happens:

- the second pixel turns red; its grid coordinate is stored.

- the programme calls the lineMP function and calculates it for the points corresponding to the two selected points.

- The resulting segment is rendered with tiles on the grid squares, representing the lit pixels. These tiles will be semi-transparent, yellow boxes with a height equal to 1/4 of the length of the side of the squares.

- The **exact** line is also rendered in black, making it visible to the transparency below the tiles.

- You can repeat the process to draw more lines;

- By pressing the Backspace key we delete all the tiles, restoring the initial grid.

EXAMPLE: In the example in the figure below, the user has selected the pixels with coordinates (0,0) and (3,1). Using lineMP, we light the four yellow pixels with coordinates (0,0), (1,0), (2,1), (3,1) which correspond to the midpoint algorithm; note that we can see both the axes and the ideal line joining the centre of the initial pixel (0,0) to the centre of the final pixel (3,1) in transparency.
The transparency also shows the red colour of the selected start and end pixels, below the tiles representing the lit pixels.

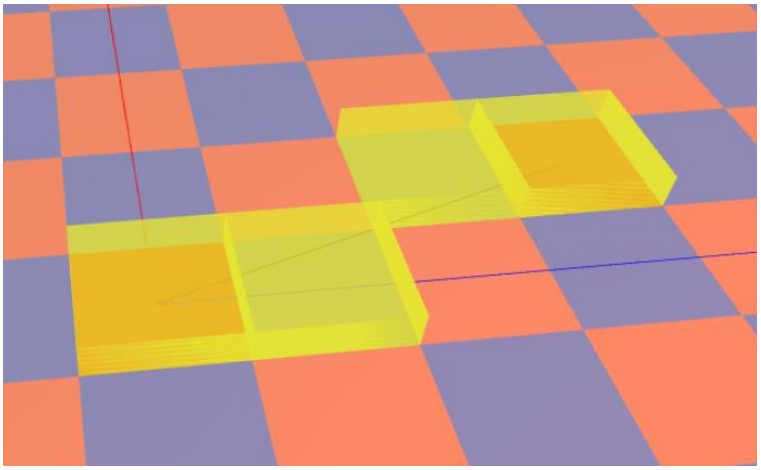


Fig. 2: Four lit pixels. Note the exact line, visible in transparency.

Below are other views of the same scene, using the OrbitControls to rotate/zoom around the centre pixel.

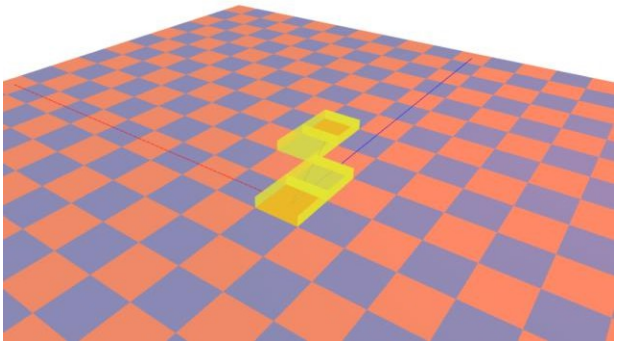


Fig. 3.

When the user presses the "C" key, the camera should move smoothly (but quickly) from its current position until it is hovering over the board to show it completely in a top view (see Figure 4). Once in this position, the user should once again have control over the camera's movement.

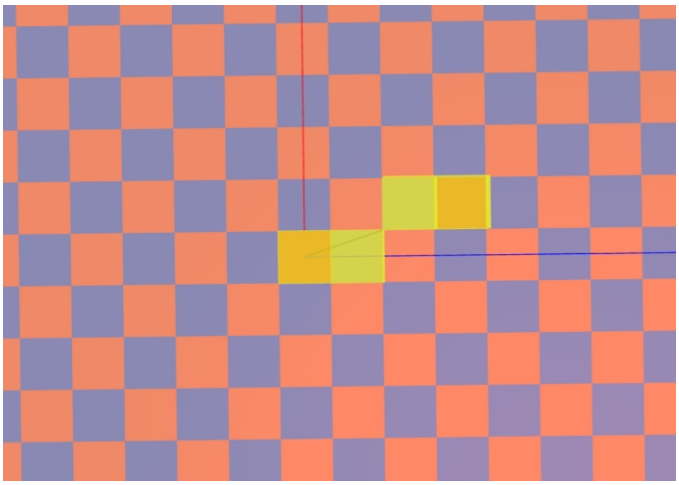


Fig. 4.

Additional standards and indications

Programme the algorithms and methods using JavaScript, HTML, CSS and Three js and comment the source code appropriately.

It must include a report explaining all the decisions taken. This report will be a descriptive account of the work carried out and should be succinct (maximum 2 A4 pages, font 10, single spaced) and complement the comments inserted in the source code;

Submit all the files by compressing them into a single zip file, using the following format: <student's last name><student number>.zip.

You must upload this file to the platform on the E-Folio A device by the deadline date and time. Avoid submitting it close to the deadline to avoid any problems.

Please indicate if you notice any problems with the statement. As this is an individual assignment, please use my email address instead of the forums for any questions that might give your colleagues an inappropriate clue (if in doubt, assume this is the case): antonio.araujo@uab.pt

The file to be sent must not exceed 8 MB.

Again, your programme must be **ready to run**, with **no broken dependencies**. Include in the base directory the **lineMP.mjs** file, the **index.html** file that runs the second part of the job, and the job report (called ReadMe.pdf); any other files, including your other modules, any css files, etc, should be placed in folders, with the JavaScript modules in a "./src" folder.

Your work will be judged on several factors:

- the implementation of the requested features.

- the correctness, elegance and efficiency of the code and its documentation.

- **essential requirement (eliminatory):** the programme must be ready to run, with no changes or broken dependencies, following the rules established above for interfaces and objects.

Best wishes for a job well done!