CSc433-Fall20-HW5. Projection Transformation, Depth Buffer and Texture.

Submissions in pairs of the theoretical part is **NOT** allowed
Submission in pairs of the implementation is allowed

Theoretical Part due: Nov 25.
Graded: ASAP.
Implementation due: Dec 2
Graded: Dec 9.

In this homework, you are expected to solve a similar problems to the ones addressed in hw3 and hw4. However, we approach this problem in a way that highlights other approaches. In particular, we will practice

1. Compute and apply a series of transformations, that describes how billboards (and the images they carry) are projected to the image plane.
2. Apply linear interpolation property to the vectices of each billboard, to obtain the coordinates of each interial points (non-vertices point) of projected corners.
3. Re-compute the interpolation coefficient for each internal billboard-pixel, when accessing the pixels in a raster-scan approach (one row after the other).
4. Practice using depth buffer for hidden surfaces removal.
5. Apply a simple version of clipping.

Hi-level description: We project each billboard is projected (via perspective projection) into the image plane. The locations of the interior points of the billboard are interpolated based on the corners' locations. Each pixel of in output image should decide weather its color should stay unchanged, or weather its colors are modified to the colors of the billboard at the very same point.

1 Further Instructions

1. recall that the input paprametyers in the the json file contains the camera parameters: The points **eye** and **LookAt**, the vectors $\vec{up}, fov\_angle$, the number of rows $height$, and the number of columns $width$.
2. Prepare two buffers (namely 2-dimensional arrays). One called $ZBuffer[height][width]$. The other is the $ColorBuffer[height][width]$. Each cell of the latter buffer could contains an RGB value, so you might perfer to implement this buffer using three uint8$[height][width]$ arrays. You can even implement it using a single array $unit8[3 \times height \times width]$. The image that the algorithm creates is stored in the $ColorBuffer$.

3. Your program should create an array We will refer to the output image as the *ColorBuffer*. It plays the same role that the image plane played at hw3. In addition, we will use the "depth buffer" or z-buffer *ZBuffer*. Both are of size $width \times height$, where $width$ and $height$ are given in the json file. Each cell in the color buffer could contain an RGB value, while each cell in the z-buffer contains a float in the range $[0, maxfloat)$. The defualt color of the pixels of the *ColorBuffer* is black $(0, 0, 0)$, and the default depth of the cells of *ZBuffer* is $-\infty$.

4. For each billboard b and each pixel (i, j) of the image that is assign to this billboard, we think about this pixel as a point in $\mathbb{R}^3$. The location of this point can be easily calculated from the locations of the corners, and the values of i, j and the number of rows and columns in this billboard. There are several ways to compute this location. In this homework we will use **linear interpolation**, which is similar (but simpler) than the linear interpolation obtained for triangles, that uses barycentric coordinates. See more details in Section 1.1.

5. Once the algorithm terminates, the Colorbuffer should contain the image as created by the camera following the input properties. In particular, this should be identical to the output of hw3.

6. As part of the input, you should print each one of the matrices used for this homework, and specify what this matrix is performing,

   You could use outside libraries, or snippets of code, to perform operations as matrix multiplication by a vector, matrix multiplication by another vector, and transposing a matrix.

7. Note that the color of a pixel (i, j) of the color-buffer, is the color of the billboard, that after all the transformations, has the smallest z-value.

8. If a billboard is not fully in from of the camera, then you do not have to render this billboard at all. This is a version of **clipping** that we will cover in the course, but this approach more loose. A stricter approach would require rendering the portion of the billboard that the camera could see.

9. In class, we describe to viewing frustum viewing frustum by specifying two vertices of the frustum, the near-left-bottom one, and the top-right-upper one. They are not specified explicitly. You need to calculate them, (or find a circumventing way) using the camera parameters.

10. If you are watching a billboard from the back, you could assign to it any color you wish.

11. billboard are always rectangular. So the vectors $\vec{U} = P_{UR} - P_{UL}$ is orthogonal to $\vec{V} = P_{LL} - P_{UL}$.

12. If you are watching a billboard from the back, you could assign to it any color you wish.

13. billboard are always rectangular. So the vectors $\vec{V} = P_{UR} - P_{UL}$ is orthogonal to $\vec{V} = P_{LL} - P_{UL}$.

14. Billboards might intersect, and might occlude each other.

In this homework you are expected to repeat most of the functionalities of hw3 and hw4, but using a very different method. This way also emphasise your understanding of transformations, and depth buffer.

The input to this problem is a json file as in hw4, (where 'IsAmbient=0'). We will stick to the same file format, so we could compare the results of hw4 and hw5. However, we will not use the location of the sun, and will not include spheres.

Algorithm 1: Generating the projected image using a Z-buffering Approach.

Input: A set of billbords $\{b_1 \dots b_{n_{boards}}\}$. The points **eye**, **LookAt**, $\vec{up}, fov\_angle$, the number of rows and columns, $width, height$.
Init: Each cell of *ZBuffer* is initialized to $-\infty$, and initialized each cell of color buffer to black.
Prepare $\vec{\mathbf{u}}, \vec{\mathbf{v}}, \vec{\mathbf{w}}$ as in hw4
Express all vectors to homogeneous coordinates
Prepare the matrices $M_{vp}, M_{perspective}, M_{cam}$, as indicated in Section 7.2 of the textbook.
Compute $M = M_{vp} \cdot M_{perspective} \cdot M_{cam}$ Using these matrices, computer M which is their product
for *each billiboard* b_j **do**
    Let $P_{UL}, P_{LL}, P_{UR}, P_{LR}$ denote the corners of this billboard. (UL==UpperLeft etc)
    Apply the transformation matrix M on each of the corners of this billboard.
        Let $Q_{UL} = M \cdot P_{UL}$, $Q_{LL} = M \cdot P_{LL}$, $Q_{UR} = M \cdot P_{UR}$, $Q_{LR} = M \cdot P_{LR}$
    for *every pixel* (i, j) *in this billboard* **do**
        /* Traditionally GPU executes this inner loop in parallel. */
        Use interpolation of $Q_{UL}, Q_{LL}, Q_{UR}, Q_{LR}$ to compute a point Q in the image buffer that matches the (i, j) pixel. Follow Section 1.1 to compute the weight of each corner.
        Perform orthographic projection of Q on the image plane, and calculate the nearest pixel (i', j').
        /* dont confuse (i, j) that are the indices of a pixel *in a billboard*, vs. (i', j') which are indices of a pixel in the image plane. */
        /* Next compare the depth $Q.Z$ of the new point Q, with the depth of a previously-stored billboard that effected the (i', j') pixel */
        if $Q.Z <$*ZBuffer*$[i'][j']$ **then**
            ZBuffer$[i'][j'] = Q.Z$
            ColorBuffer$[i'][j'] =$ *color*(billboard at pixel (i, j))
        end
    end
end

you are requested to maintain, in addition to the created image, another array called '*ZBuffer*'. Initially every cell of this array contains $-\infty$. The main look of your program will iterate over each billboard (spheres are not used in this assignment).

In this assignment, both undergraduates and graduates will develop the basic scaffolding code for implementing a ray tracer (to be built upon in the next assignment).

All students will implement the same basic features in this assignment, while the functionality for the ray tracer will be split for each group in the next assignment. Please use the link associated with your section:

1.1 Interpolation

Here is a simple linear interpolation algorithm, that will find the 3D location of (the center of) the (i, j) pixel of a billboard. We assume that the image placed on the billboard consists of n rows and m columns: Calculate $\alpha = j/m, \beta = i/n$ (similar to hw3). You could visualized what are the roles of α, β in geogabra. In general α denote how much weight are we giving to the right pair of corners, vs. the weight that we give to the left pair of corners. Analogously β splits the weights between the lower pair vs. upper pair. Define $\bar{\alpha} = (1 - \alpha)$, and $\bar{\beta} = 1 - \beta$. Then the interpolated point (at 3D) is

$$Q = \bar{\alpha}\bar{\beta} \cdot \mathbf{Q_{UL}} + \bar{\alpha}\beta\mathbf{Q_{LL}} \; + \; \alpha\bar{\beta}\, \mathbf{Q_{UR}} + \alpha\beta\, \mathbf{Q_{LR}}$$

2 Written Questions

The written question **cannot** be submitted in pairs

1. Given the input to this homework, explain how to construct a matrix R that rotates the billboard by $30°$, around the line ℓ that is passing through the eye, and is in the direction of the vector $\vec{\mathbf{v}}$.

2. Assume that one of the billboard is a mirror. Explain algorithmically how to use the Algorithm 1 above to create an image, which is identical to what the mirror displayed. Note that this image depends on the original camera parameters.

   For simplicity, assume that $\vec{w}$ intersects the mirror at its center, and is orthogonal to the mirror.

3. Is it possible that there are 3 billboards b_1, b_2, b_3 such that

   - A point of b_1 occludes from the camera one (or more) points of b_2, and at the same time...
   - a point of b_2 occludes from the camera one (or more) points of b_3, and at the same time...
   - A point of b_3 occludes from the camera one (or more) points of b_1.

   Billboards in this question are rectangular and flat. You cannot twist them. Build this example, or prove that it could not occur.

4. You are given some set of vectors as in the implementation homework. Explain how to compute a matrix T that, if applied on all vertices of the billboards, will rotate their location by 45 degrees around the line that contains **eye** and in the direction $\vec{w}$.

5. Consider a billboard with corners P_{UL}, P_{UR}, P_{LL}. Create a matrix M such that (in homogeneous coordinates) $M \cdot P_{LL}$ is on the origin $(0,0,0)$, $M \cdot P_{LL}$ is on the positive x-axis, and $M \cdot P_{UR}$ is on the positive y-axis.

   Check you answer on the geogabra app (link). You will find in this application a 4×4 matrix a, that you could change. Now, if P is a point (must start with a capital letter) in Cartesian coordinates, $P = (x, y, z)$ and we write in geogabra $a \cdot P$, geogabra will convert P to homogenious coordinates, execute the multiplication, and convert the result vector back into Cartesian. This way enables you to check yourself easily.

   To insert a matrix from scretch, write

$$a = \{\{1,2,3,4\},\{2,4,6,8\},\{12,14,16,18\},\{5,6,7,8\}\}$$

   Geogabra will understand that a is the matrix

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 6 & 8 \\ 12 & 14 & 16 & 18 \\ 5 & 6 & 7 & 8 \end{pmatrix}$$

6. As you know, if a is a matrix, then $a^2 = a \cdot a$. Let g be the matrix

$$g = \begin{pmatrix} \cos(5^\circ) & \sin(5^\circ) \\ -\sin(5^\circ) & \cos(5^\circ) \end{pmatrix}$$

(a) Using geometric arguments, and without using computers nor calculators, compute g^{72}. Hint - think about a vector $\vec{v} = (x, y)$, and understand what is the geometric meaning of the product

$$\begin{pmatrix} \cos(5^\circ) & \sin(5^\circ) \\ -\sin(5^\circ) & \cos(5^\circ) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}.$$

That is, what geometrically happened to this vector.

(b) Next, consider the matrix b

$$b = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Using geometric arguments, and without using computers nor calculators, compute b^{2020}. hint - understand geometrically what happened to a vector $\vec{v} = (x, y, 1)$ (two dimensional vector, expressed in homogeneous coordinates).

7. Algorithm 1 introduced very fast and efficient implementations, both in software and especially in hardware (GPU). Therefore, it is also used to solve many related problems.

   When we compute the illumination at the room is obtained by multiple lamps $\{\ell_1 \dots \ell_K\}$. Each lamp ℓ_i is a point in $\mathbb{R}^3$. To obtain proper shading, it is important to find for each lamp ℓ_i and each billboard b_j (or triangle) whether this billboard is *illuminated* by lamp ℓ_i, or is it completely occluded from b_j by other billboards. Suggest an algorithm (let's call it *Algorithm 2*) that uses the billboard locations and the location of a single lamp ℓ_j, call Algorithm 1 after scanning each cell of *ColorBuffer* and *ZBuffer*, Algorithm 2 produces the list of all the billboards that are illuminated by ℓ_i. In terms of running time, both the number of billboards and the number of pixels in *ColorBuffer* and *ZBuffer* could be quite large that we don't accept a running time proportional to $(number\_of\_triangles) * (number\_of\_pixels)$.

   Note that we dont distinguish between partially illuminated and fully illuminated. We say that ℓ_j illuminates a billboard if at least one point of billboard is illuminated by ℓ_j. However, if the illuminated area is very small, we allow Algorithm 2 to ignore it.

3 Submission

You should use git to submit all source code files. The expectation is that your code will be graded by cloning your repo and then executing it within a modern browser (Chrome, Firefox, etc.)

Please provide a README.md file that provides a text description of how to run your program and any parameters that you used. Also document any idiosyncrasies, behaviors, or bugs of note that you want us to be aware of.

To summarize, my expectation is that your repo will contain:

1. A README.md file
2. Answers to the written questions in a separate directory named "Theoretical" (should be submitted to gradescope)
3. A index.html file
4. A a05.js file
5. Any other .js files that you authored. You sample scene file, myscene.js, as well as the ppm file your code generated from it, myscene.ppm. (Optionally) any other test images that you want.