TECHNOLOGY ARENA - DAY 2

Welcome to STEM Games Technology Arena, round two! In the second part of the competition, you will face a single very challenging project task where you will have complete freedom. You can choose whichever path you want in approaching this problem. You are both allowed and encouraged to use all the available tools in your attempt to advance as much as possible on this problem. This especially includes various AI tools and models which can speed you up significantly. The mentor will be present in the arena and you are also encouraged to seek advice from him if you get stuck. He will also be roaming around the arena, actively giving advice to you based on which approach you decide to take.

There will be a lot of stuff here which you may have not encountered before, but do not be afraid. This is why the arena mentor is there to help and you have no restrictions on the tools you use to help yourself. Whether you decide to focus on the already existing approaches to the problem or you wish to try to implement your own ideas, you will be able to make some progress as long as you keep trying.

You are allowed to work both onsite and offsite and the only limitation is that you must submit your work no later than **May 19th 21:59**. The top four teams who are considered to have made the most progress will present their work publicly on May 20th afternoon and the winners will be announced after the presentations.

The Problem:

You are given four sets of several images of some objects. The images within the same set will be showing the same scene. Each image within the same set was taken from slightly different position and angle. Your goal is to estimate the position in 3D space (X, Y and Z coordinates) of as many pixels on those images as possible. In industry, this is called the **point cloud**. In two of the sets, the coordinates of the camera taking the images will be known, making these cases easier to solve.

Without additional guidelines, this is a very difficult problem. In the following pages we give out some **crucial guidelines** which will help you approach this problem. You can try different approaches, but these suggestions should help considerably.

IMPORTANT: Keep in mind that there are around 4 of you in a team. Use that to distribute the work efficiently.

Guidelines for tests where camera coordinates are known:

The folders **Box** and **Entrance** correspond to these tests. In addition to the images, you will be given camera coordinates, but also the camera's watching directions which determine its rotation. They will all be in a text file with the corresponding images, enumerated in the same way. Camera's watcing direction is determined by its ForwardVector, RightVector and UpVector. You can easily get good understanding of those vectors by simple Google Search, although you won't strictly need it since we will provide the necessary code samples.

In the tests **Box** and **Entrance**, all images have 1080 rows and 1920 columns. Given the camera coordinates and watching direction, it is possible to determine the equation of the line containing the camera's location point and the point where that pixel is located in 3D space. This is very useful since it gives you a better idea where certain pixel is located in 3D space. The equation of a line in 3D space is commonly given by specifying one point on that line and the 3D vector of the direction of the line.

On the next page is a code piece which calculates that line equation in the form specified above (point + direction).

The GetLineEquation() function is what you will likely want to use.
PixelRow and PixelCol are row and column coordinates of the pixel you are interested in. For example, the pixels in top row should have PixelRow = 0 and those in the bottom row should have PixelRow = 1079. Similarly, the pixels in letfmost column should have PixelCol = 0 and those in the rightmost column should have PixelCol = 1919.
ResX and ResY are the number of columns and rows of the input image. In cases **Box** and **Entrance** will always be ResX = 1920 and ResY = 1080.
The remaining inputs are the values you will be given in the text files, the position of the camera and its watching directions.

The function returns a LineEquation element, which holds the information of a line containing the pixel you are interested in. That line is determined by a point on it (OriginPoint) and its directional vector (Direction).

```
struct XYZ{
    float X;
    float Y;
    float Z;

    XYZ(){};

    XYZ(float _X, float _Y, float _Z){
        X = _X;
        Y = _Y;
        Z = _Z;
    }

    XYZ operator+ (XYZ p){
        return XYZ(X + p.X, Y + p.Y, Z + p.Z);
    }
};

inline XYZ operator* (float c, const XYZ& p){
    return XYZ(c * p.X, c * p.Y, c * p.Z);
}

struct LineEquation{
    XYZ OriginPoint;
    XYZ Direction;
};

// Assuming Field of View is 90 deg and no distortion, which is true in our cases

LineEquation GetLineEquation(int PixelRow, int PixelCol, int ResX, int ResY,
                             XYZ CamForward, XYZ CamRight, XYZ CamUp,
                             XYZ CamPosition){

    LineEquation Eq;
    Eq.OriginPoint = CamPosition;

    float coeffRight = (2.0 * (PixelCol - ResX / 2.0 + 0.5) / ResX);
    float coeffUp = (-2.0 * (PixelRow - ResY / 2.0 + 0.5) / ResY);
    coeffUp = coeffUp * (float)(ResY)/ResX;

    Eq.Direction = CamForward + coeffRight * CamRight + coeffUp * CamUp;

    return Eq;
}
```

While useful, a single line in 3D by itself does not uniquely determine the pixel 3D coordinates since the pixel can be anywhere on it. Think about how to get more information about the pixel, not just one line on which it is. You will have to use more than one image of your dataset. Remember that you don't have to resolve all pixels, only as many as you can. You can be creative and make your own algorithms or build upon existing solutions online. You can even try to do it by hand (in rare cases this is the best approach).

Do not be afraid to ask for more guidelines.

Guidelines for tests where camera coordinates are unknown:

The folders **Statue** and **Fountain** correspond to these tests. Notice that the **Fountain** images exceptionally have 2048 rows and 3072 columns. This problem is now much more difficult since there are more unknowns. We don't know the camera's position and watching directions. In the **Statue**, the camera Field of View is again 90 degrees and in **Fountain** it is around 84 degrees.

You can start by playing a guessing game, trying to guess the camera position and watching directions and then apply your solution of the previous cases. Also explore potential other options, using modern tools to speed you up.

Remember, there are around 4 of you in the team, try to distribute the work efficiently to ensure maximal efficency.

What to submit:

The most important part of your submission is the text file (one for each image set) which lists the 3D coordinates of the points in 3D space that appear in the images. Try to estimate as many of them as correctly as possible. Bonus points if you submit the visualization of those points (easy with various Python tools). That will speed up the grading process massively.

Also submit all your relevant work: code, documentation with explanation of your code, ideas and how you approached the problem.

Do not feel demoralized if you didn't progress on some (or all) of the cases, simply try to do your best.

Good luck! Borna Vukorepa, Stype CS