Computer Graphics (L.EIC)

Project 2022/2023

preliminary notes

In relation to the initial version, this document presents the following changes:

2023/04/05:

- Additional note in class ***MyPanorama*** contained in the last paragraph of section 2.
- Addition of new sections 5, 6 and 7.
- Addition of quotations inherent to the evaluation of the work.
- Addition of a Work Plan recommendation.

Computer Graphics (L.EIC)
Project 2022/2023

Version v1.0 - 2023/04/05

Goals

- Apply the knowledge and techniques acquired to date
- Use elements of interaction with the scene, through the keyboard and graphical interface elements
- Use *Shaders* in modeling/visualizing objects
- Use component animation

Description

The aim of this project is to create a scene that combines the different elements explored in previous classes. For this work, you should use the code that is provided in Moodle as a base, which corresponds to a scene with a plan of 400x400 units. You will later have to add some of the objects created in previous works.

The scene, at the end of the project, must be generically constituted (at least) by:

- A terrain with elevations, created by a shader;
- A forest, composed of trees using billboards;
- A bird, animated and controlled by the user, as well as its nest.
- A cluster of eggs scattered across the land

The following points describe the main characteristics of the different intended elements. Some freedom is given as to their composition in the scene, so that each group can create their own scene.

Environment Preparation

They must download the code made available for this work from Moodle and place the folder ***project*** contained in the .zip file, at the same level as previous works.

The code provided for the project has a scene consisting only of the axes of the coordinate system, an object ***Plane***, and a light source. You should include later in this project some elements present in previous works, namely cylinders, cones, cubes, etc., as needed.

1. Sphere Creation

Create a new class ***MySphere*** that creates a sphere with the center at the origin, with the central axis coincident with the Y axis and unit radius.

The sphere must have a variable number of "sides" around the Y axis (*slices*), and “stacks” (*stacks*), the latter corresponding to the number of divisions along the Y axis, from the center to the "poles" (i.e., number of "stacks" of each half-sphere). Figure **1** has a visual representation of the sphere.

The algorithm used must be efficient and must generate the texture coordinates for the respective application to the surface of the sphere, as shown in **Figure 2**.

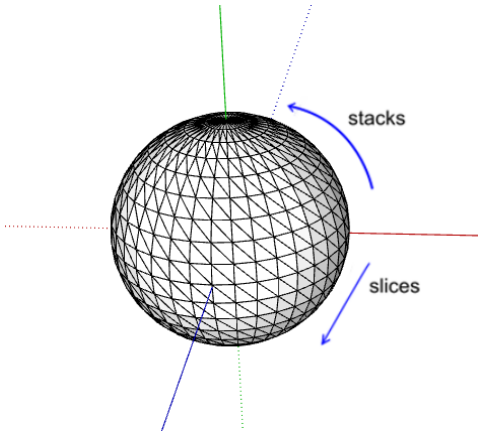

Figure 1: Example image of a sphere centered at the origin.

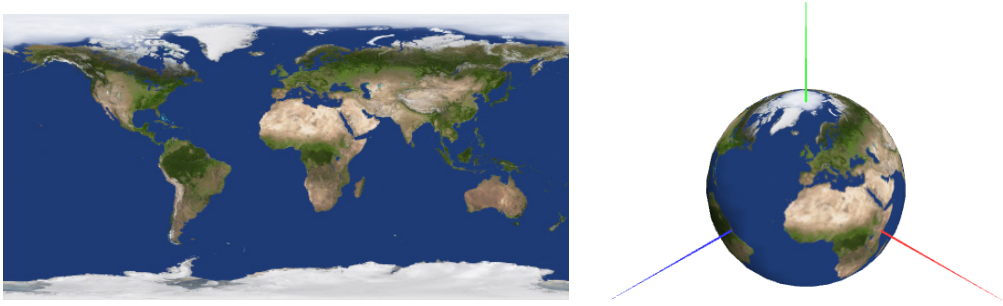

Figure 2: *Texture example (available on Moodle) and its application to a sphere*

Place a test instance of the sphere in the scene and create a tag in the repository at this point.

(1)

2. Creation of Panoramas

Parameterize the MySphere class to be able to invert its faces, so that it is visible from the inside and not from the outside (pay attention to the order of vertices and normals). The idea is to be able to use a large sphere around the scene with a panoramic image, to create a landscape, and the observer to move within that sphere. You can find panoramic / 360º images in equirectangular

format, like those in Figure 3 e.g. me
https://www.flickr.com/search/?text=equirectangular%20landscape

(on the links in Fig. 3 you can view their use interactively)

Figure 3: Two examples of equirectangular images

create the class ***MyPanorama*** what:

- in its constructor receive a ***CGFtexture***, and be responsible for creating a ***MySphere*** inverted, and a material with only an emissive component and the associated texture,
- in your method ***display***, activate the texture and draw the sphere with a radius of 200 units

Include one of these panoramas in the scene. It is suggested to experiment with different values for the FoV (field of view) parameter of the scene camera, so that the perspective distortion is satisfactory.

At this point, take screenshots that allow you to see one or two perspectives of the scene and panorama.Create a tag in the repository at this point.

(1) (2)

Additional note:

Change the class ***MyPanorama*** so that it becomes centered on the position of the camera, moving with it and giving the illusion that the spherical surface is always positioned at infinity. ***NOTE***: you can use the member *position* of the camera stored in the **scene** (*this.camera.position)*. It is a vec3, so the various coordinates can be obtained by accessing the positions*[0], [1] and [2]*.

3. Inclusion of a Bird

It is intended that a bird be included in the scene. It must be animated and provided with controllable movement from the keyboard in accordance with the following subsections.

3.1. Bird modeling

Create a new class ***MyBird***, to represent a bird. To model the bird, you can use a combination of the different objects used previously (also in other works), so that the bird consists of a body, head (with eyes and beak), and wings. Each wing shall consist of two parts. Figure 4 illustrates the intended structure, but with a simplified body and head.

The bird should use slightly more complex geometry than the one shown in the image for the body and head (but it shouldn't have an excessive number of polygons). We suggest using cones or cylinders with a reduced number of sides (3 to 5), spheres or other objects you consider relevant.

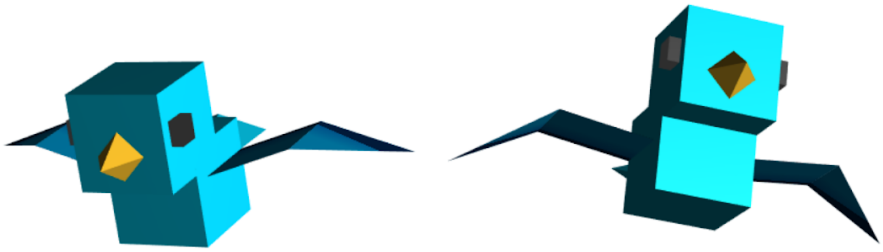

Figure 4: Exemplification of the bird model, simplified with cubes for the body and head.

The bird must be clearly visible when the scene starts (using a good definition of the camera's initial parameters) in order to facilitate its evaluation. It can be up to 2 units long, and with open wings it can be up to 3 units wide.

The different objects used to create the bird should have materials and textures applied to them, suitable for the parts of the bird they represent. Show a picture of what the bird looks like (close enough to see it in detail).

(2)

3.2. Bird animation

At this point, animation and control mechanisms must be created for an object of the class ***MyBird***.

Place the Bird in the scene about 3 units above the ground. The bird should have two animations applied steadily over time, simulating flight movement:

1. An animation to oscillate slightly up and down, with each complete oscillation (up and down) taking 1 second.
2. A second animation for wing flapping; the flapping speed should depend on the current speed of the bird (see next section).

3.3. Bird Control

To be able to control the movement of the bird in the scene, it will be necessary to press one or more keys simultaneously.

1. Change the class ***MyInterface***, adding the following methods to process multiple keystrokes at once (**NOTE**: do not copy-paste this document, as some characters may be converted incorrectly):

```
initKeys() {
    // create reference from the scene to the GUI
```

```
        this.scene.gui=this;

        // disable the processKeyboard function
        this.processKeyboard=function(){};

        // create a named array to store which keys are being pressed
        this.activeKeys={};
    }
    processKeyDown(event) {
        // called when a key is pressed down
        // mark it as active in the array
        this.activeKeys[event.code]=true;
    };

    processKeyUp(event) {
        // called when a key is released, mark it as inactive in the array
        this.activeKeys[event.code]=false;
    };

    isKeyPressed(keyCode) {
        // returns true if a key is marked as pressed, false otherwise
        return this.activeKeys[keyCode] || false;
    }
```

At the end of the function ***init()*** of ***MyInterface***, call the function ***initKeys()***.

2. in class ***MyScene*** add the following method ***checkKeys()*** and add a call to it in the method ***update()***.

```
    checkKeys()  {
        var text="Keys pressed: ";
        var keysPressed=false;

        // Check for key codes e.g. in https://keycode.info/
        if (this.gui.isKeyPressed("KeyW")) {
              text+=" W ";
              keysPressed=true;
        }

        if (this.gui.isKeyPressed("KeyS"))  {
              text+=" S ";
              keysPressed=true;
        }
        if (keysPressed)
              console.log(text);
    }
```

Run the code and check the messages on the console when “W” and “S” are pressed simultaneously.

3. Change to class ***MyBird*** according to the following:
    - Add to the constructor variables that define:
        - the orientation of the bird (hint: angle around the YY axis)

        - its speed (initially at zero)
        - Your position (x, y, z)
    - Change the function ***update*** to update the position variable depending on the orientation and velocity values
    - Use the position and orientation variables in the function ***display()*** to orient and position the bird.
    - Create the methods ***turn(v)*** and ***accelerate(v)*** to rotate and to increase/decrease the bird's speed.

4. Predict that keys can invoke methods *turn* and *accelerate* of the bird, in order to implement the following behavior:

    - Speed up or break as you press "**W**" or "**S**", respectively.
    - Rotate the bird to the left or right by pressing the "**A**" or "**D**";
    - To do "*reset*" to the position and speed of the bird using the "**R**" ; that is, the bird must be placed in the initial position, with zero rotation and zero speed.

4. Create a *slider* in the GUI called *speedFactor* (between 0.1 and 3) that accelerates and decelerates the speed of movement, rotation and wing flapping of the bird.

5. Create another *slider* in the GUI called *scaleFactor* (between 0.5 and 3) that allows you to scale the bird so that it is easier to observe its animations.

Create a tag in the repository at this point.

 (3)

4. Terrain

It is intended to improve the terrain, namely by introducing altimetry, using *shaders*. Create a new class ***MyTerrain***, which will consist of a ***Plane*** (practical class of *shaders*), to represent terrain with elevations. These must be obtained based on a texture of gray levels that works as a height map. The creation of shaders, associated textures and their application must be encapsulated within the class ***MyTerrain***.

Figure 5 contains a possible example of ***MyTerrain***. You must replace the initial ***Plane*** 400x400 units by one ***MyTerrain*** with the same dimension.

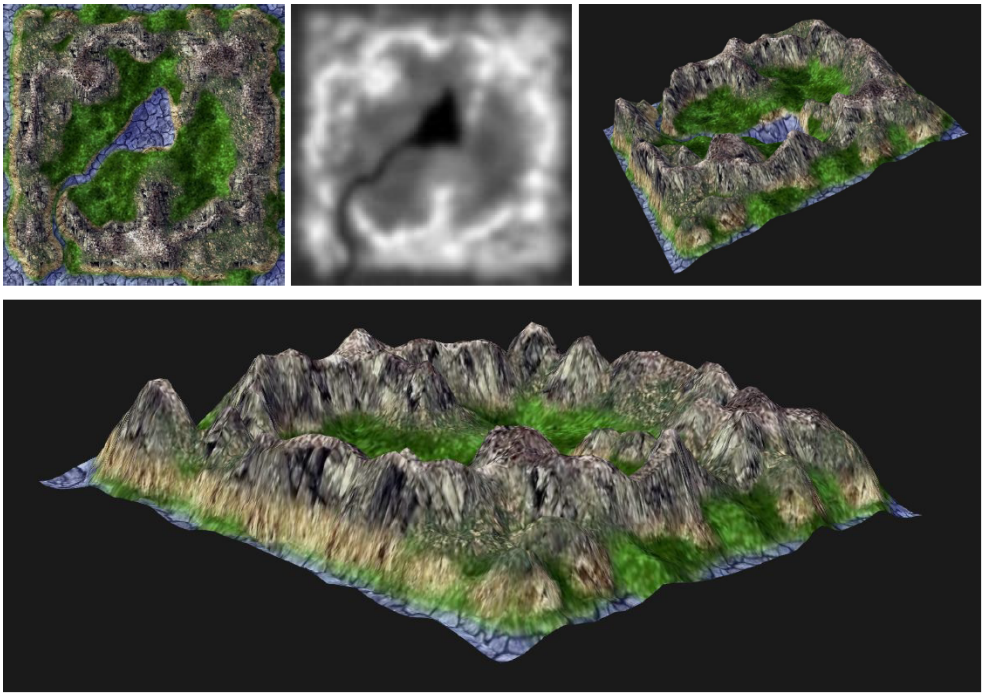

Figure 5: Color texture, heightmap and generated geometry.
(*origin:* Outside of Society https://oosmoxiecode.com/archive/js_webgl/terrain/index.html)

1. Test an initial version with the two textures, color and height, provided.
2. Modify the heights texture (using an image editor) so that there is a flat zone with approximate dimensions of 20x20 units (⅓ of the width/length of the terrain).

   NOTE: you may **replace this pair of textures with another one,** as long as you respect the rules set out.

3. change the ***shader*** so that:
   a. Get a third texture representing a color gradient - **altimetry** (see figure 6a, image provided with the code).
   b. On each fragment, instead of just applying the original terrain texture color, combine this color with the value of the gradient color corresponding to the altimetry: at higher altitudes the original color should be more “white” and at lower altitudes it will be more “blue” (see example in figures 6c and 6d). This combination should correspond to 70% of the original color and 30% of the color of the corresponding zone of the height texture (Note: some adjustments to scales or offsets may be made to adapt the gradient to the range of heights; you may put the weight of the components as a parameter).

At this point, take screenshots that allow seeing one or two perspectives of the terrain and include both, the bird and the background.Create a tag in the repository at this point.

(3) (4)

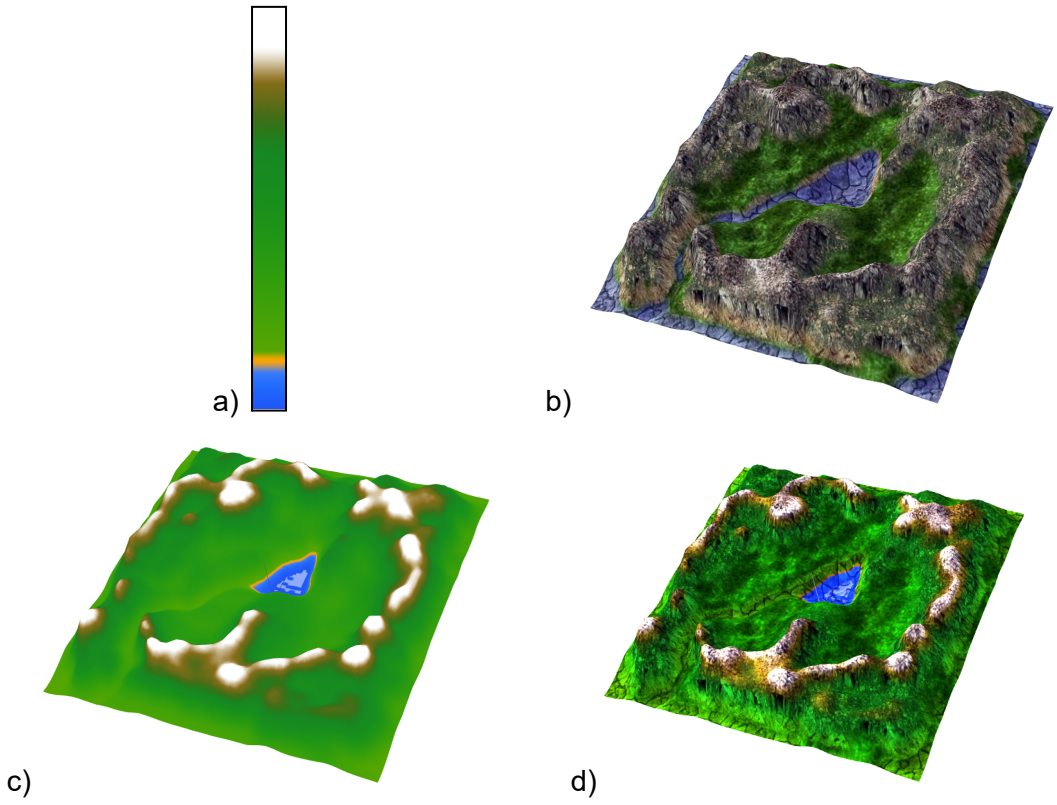

Figure 6: Use of a texture to color terrain zones depending on height
a) gradient;**b)** terrain only with original color texture
c) terrain with gradient color only;**d)** gradient and original color combination

5. Eggs and Nest

It is intended to add a new functionality to ***MyBird***, so that the bird picks up the eggs one by one (***MyBirdEgg****)* scattered across the land and drop them in the nest (***MyNest***).

1. Creation of objects
    a. Create a new class ***MyBirdEgg***, based on ***MySphere*** that apply different scale factors to *YY* to the two hemispheres, allowing one of them to be more "elongated". A suitable material and texture should be applied (they should have "spots").
    b. Place several objects of this class in the scene (four at least), with chosen/random positions and rotations, resting on the flat area of the terrain. Use a vector of eggs, so it can be generic/configurable.
    c. Create a new class ***MyNest***, which will represent a bird's nest, made up of objects of your choice. Apply suitable materials and textures to resemble a nest.
    d. Place an object ***MyNest*** in the flat area of the scene, ensuring that both the scene and the objects in ***MyBirdEgg*** are visible at the beginning of the scene, for evaluation.
2. Add to ***MyBird*** the functionality of picking up and dropping eggs as follows:
    a. When pressing the key **"P"** the bird must descend to ground level (in the flat area) and rise again to the initial height within a period of 2 seconds (keeping its speed at **XZ**).

b. If in this process, when the bird touches the ground, there is an egg at that point (a tolerance margin must be implemented), this egg must be "picked up" by the bird. **Suggestion**: add this egg's reference to ***MyBird***, so that it is drawn in the ***display*** of the bird, and remove the reference to the egg from the scene, so that it is no longer drawn on the terrain.
c. If the bird is carrying an egg and the key "***O***" is pressed when passing near the nest**,** the bird must drop the egg so that it falls and is deposited in the nest (ie its reference must be added to the nest and removed from the bird). Some possible positions for the eggs to be deposited individually must be defined within the nest.

Record this point of development:  (4)  (5)

6. Integration of Trees

It is intended to create trees based on *billboards* to enrich the environment.

6.1. Billboard creation

Create a new class***MyBillboard***, which should have an instance of ***MyQuad***. You can parameterize the constructor of ***MyBillboard*** as you see fit. Posteriorly:

1. Change the display function of the ***MyBillboard*** to receive as parameters the coordinates x, y, z where the quad should be drawn.
2. In the display function of ***MyBillboard***, rotate the quad so that it is always facing the camera. This means that the normal of the quad must have the same orientation as the vector that goes from its origin to the camera (see section 1). **Note**: the information needed to rotate a vector so that it has the orientation of another, both **normalized**, can be obtained from:
    a. Angle: $\boldsymbol{cos^{-1}}$ (***Math.acos*** function) of the value obtained from the inner product between the two vectors (***vec3.dot*** function).
    b. Axis of Rotation: the outer product (***vec3.cross*** function).
3. Modify the previous transformation so that the quad is always drawn vertically. For this, you should consider only the horizontal components (*XX* and *ZZ*) of the two referred vectors.
4. Apply a material with the texture *billboardtree.png* provided (or another one you deem appropriate) before the billboard display in the scene. To obtain the desired transparency effect (fig. 5), you should avoid drawing the quad fragments whose alpha component of the corresponding texel is not completely opaque. **Note**: you may change the *default fragment shader* from the WebCGF (lib\CGF\shaders\Gouraud\textured\fragment.glsl) in order to discard fragments that do not meet the requirement (i.e., fragments with alpha less than 1) using the keyword ***discard*** (ex.: if(...) discard;).

Figure 7: Billboard of the tree facing the camera and with transparency.

6.2. Grove of trees

Create two classes that represent two types of groups of trees, so that they can later be spread across the scene. The class ***MyTreeGroupPatch*** defines a set of 9 trees distributed in a 3x3 grid, but not perfectly aligned, and with some variety of dimensions (fig. 8) and textures. The class ***MyTreeRowPatch*** defines a group of 6 trees in a row, arbitrarily slightly misaligned (fig. 9). In both cases, the texture applied to each tree must be random from a set of 3 elements (you can use textures of your choice, as long as they are duly credited in the code/README).

Figure 8: Group of 9 trees distributed in 3x3 (***MyTreeGroupPatch***).

Figure 9: Group of 6 trees distributed in line (***MyTreeRowPatch***).

Show an aerial image of the forest and terrain. (5) (6)

7. Additional developments

From the following three alternatives, choose one (and only one) to implement.

A. Parabola trajectory of the egg when it is dropped by the bird to be deposited in the nest.
B. Deformation of trees with the wind, based on the displacement of vertices of the respective quads.
C. Placement of trees in non-flat areas. To ensure that the trees are visually aligned with the ground (neither buried nor in the air), this option must use the terrain height map.

Submit the final code.  (6)  (7)

Notes on job evaluation

The maximum classification to be attributed to each item corresponds to its optimal development, in absolute compliance with all the listed functionalities, in accordance with the values below. Without losing the creativity desired in a work of this type, any developments other than those requested will not be taken into account for evaluation purposes, as a way of compensating for other missing components.

1. **Sphere creation (1.5 points)**
2. **Creation of Panoramas (1 point)**
3. **Modeling of a Bird (2 points)**
4. **Bird animation (1.5 points)**
5. **Bird Control (2 point)**
6. **Land (1.5 points)**
7. **Eggs and Nest**
    7.1. Creation of the egg and nest objects (**1 point**)
    7.2. Implementation of the method of picking up/dropping eggs (**2 point**)
8. **trees**
    8.1.Modeling a tree**(1.5 points)**
    8.2.Grove or groups of trees **(1.5 points)**
9. **Appreciation (1.5 points)**
10. **General aspect of the scene and creativity (1.5 points)**
11. **Software structure and quality (1.5 points)**

Work plan proposal

The following work plan is recommended for the remainder of the semester:

Finalization before:	Points of the statement
April 14th	Points 1, 2 and 3 (Sphere, Panorama, Bird Inclusion)
April 21st	Point 3 - conclusion; Point 4 (Land)
April 28th	Point 5 (Eggs and Nest)
May 12nd	Points 6 and 7 (Integration of Trees, Further Development)
May 19th	Refinements

During the classes of the week of May 22nd, the assignments and respective evaluations will be made.

Checklist

By the end of the work you must submit the following images and code versions via Moodle,**strictly respecting the rule of names**:

 **Images**(6): **(names like "project-t<turma>g<group>-n.png"**)

 **GIT Commits/Tags** (7): (**format "proj-1", "proj-2", ...)**

Video

You should also prepare a **1 minute video in mp4**, to be submitted in an area to be announced in Moodle. The video must be captured from the screen, demonstrating all the implemented functionalities (type names **"project-t<name>g<group>.mp4**).