Computer Graphics (L.EIC) Project 2024/2025

Version v1.0 - 2025/03/30

Goals

- Apply the knowledge and techniques acquired to date
- Use elements to interact with the scene, using 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 3D scene that combines the learning carried out in previous classes and that explores some new Computer Graphics techniques. You must use as a base the code that is provided in Moodle, which corresponds to a scene with a shot 400x400 units. You will later have to add several new objects.

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

- A representation of the sky with moving clouds
- A flat piece of land covered with grass
- A building of firemen equipped with a "Heliport" on the roof
- A helicopter, animated and controllable by the user
- A lake
- A forest with trees of varying dimensions, colors, and positions

The following points describe the main characteristics of the different elements intended. Some freedom is given regarding their composition in the scene, so that each group can use creativity and originality in creating their own scene.

***project*, at the same level as previous works.**

The code provided for the project corresponds to a scene constituted only along the axes of the coordinate system, an object ***MyPlane***, and a light source.

For now, apply a grass texture to the plane. During the course of the project, the remaining objects must be included, as defined.

1. Sky-Sphere

One *sky-sphere* is a sphere visible from the inside and that simulates the visualization of the terrain up to the horizon line and, above this, the visualization of the sky. It is assumed that the active camera, at a given moment, is inside it.

1.1 Creating a Sphere

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

The sphere must have a variable number of "sides" around the Y axis (*slices*), and “layers” (*stacks*). Layers must correspond to divisions angular equal and their number is counted from the "equator line" to the "poles" (that is, it is the number of "layers" of each semi-sphere). THE **Figure 1** has a visual representation of the sphere.

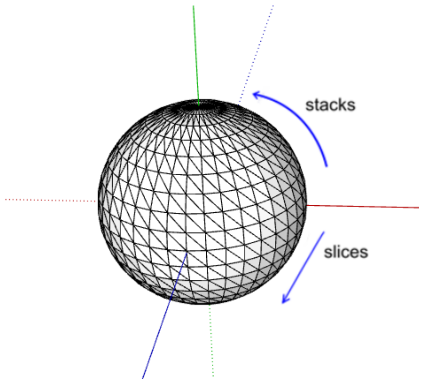
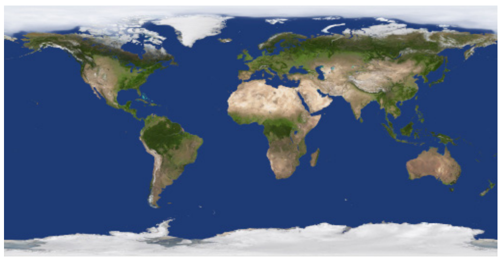


Figure 1: Example image of a sphere centered at the origin.

Notes:

1. The layers (*stacks*) **should not** be defined based on a division in height according to regular delta_y spaces; alternatively, **must be obtained by angular division** of the angle formed with the YY axis, in delta_alpha increments.
2. The layer around the North pole should be formed by triangles instead of quadrilaterals;



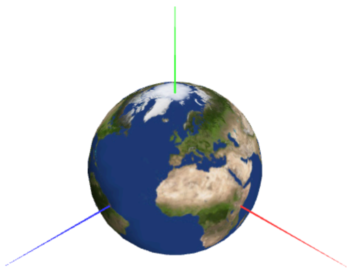


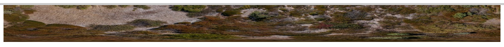
Figure 2: Example of a texture (available on Moodle) and its application to a sphere

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

(1)

1.2 Adding 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 the vertices and their respective normals). The idea is to use a large sphere around the scene with a panoramic image, to create the illusion of a landscape, and allow the viewer to move around. within of this sphere. You can find panoramic / 360º images in equirectangular format, like those in Figure 3 e.g. in https://www.flickr.com/search/?text=equirectangular%20landscape



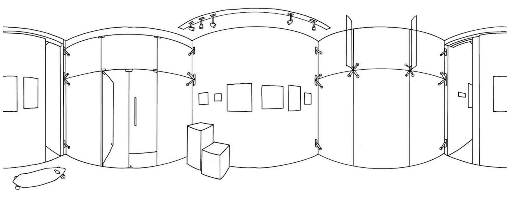


Figure 3: Two example images equirectangular

Create the class ***MyPanorama*** that:

- in its constructor receives a ***CGFtexture***, and be responsible for creating a ***MySphere*** inverted, equipped with a material with only an emissive component and covered by the aforementioned texture,
- in its 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 that you try different values for the parameter ***FoV*** (field of view) of the scene camera, so that the perspective distortion is satisfactory.

Final situation: Modify the class ***MyPanorama*** so that it becomes centered in the position of the camera, moving with it, and giving the illusion that the spherical surface is always positioned at infinity. You can use, for this purpose, the member ***position*** of the camera used in the **scene** (*this.camera.position)*. This is a vec3, so the various coordinates can be obtained by accessing the positions *[0], [1] and [2]*.

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

(1) (2)

2. Fire-fighters Building

It will be necessary to create, next to a corner of the scene plane, a fire building made up of three connected modules (see Figure 4). It must have windows and a main door, over which there is a "FIRE FIGHTERS" sign. The number of floors is configurable and the central module has more one floor than the other two; the ceiling must be flat, so as to include a heliport (circumference with a letter "H" circumscribed). Each floor has a layer with a configurable number of windows, and the ground floor of the central module must not have windows, just the door and the sign.

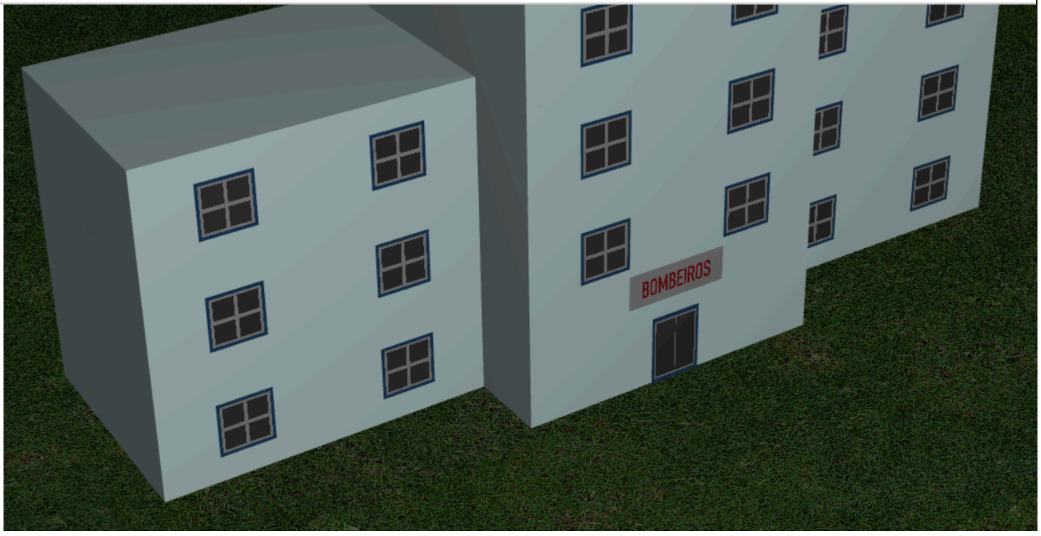


Figure 4: Simple example of the fire department building

To create the building, at least two new classes must be defined: ***MyBuilding*** and ***MyWindow***, in accordance with the following subsections.

2.1 Representation of windows

Implement a class ***MyWindow*** to represent a texture-based window. Building modules can have windows with different appearances based on class ***MyWindow***.

2.2 Representation of the building

A class ***MyBuilding***, which represents the building in the scene, must have the following parameters:

MyBuilding:

- Total width of the building (set of the three modules)
- Number of floors of side modules
- Number of windows per floor
- Window type (texture)
- Color of the building

To build the side modules, you must use these parameters relating to the central module as a base. For example: its width and/or depth can be 75% of the width of the central module.

Record this point in the development: (2) (3)

The aim is to generate a simplified model of a tree, composed only of a trunk, represented by a cone trunk made of brown material, and the crown, formed by a variable number of pyramids with green materials. The tree will be represented by a class **MyTree**.

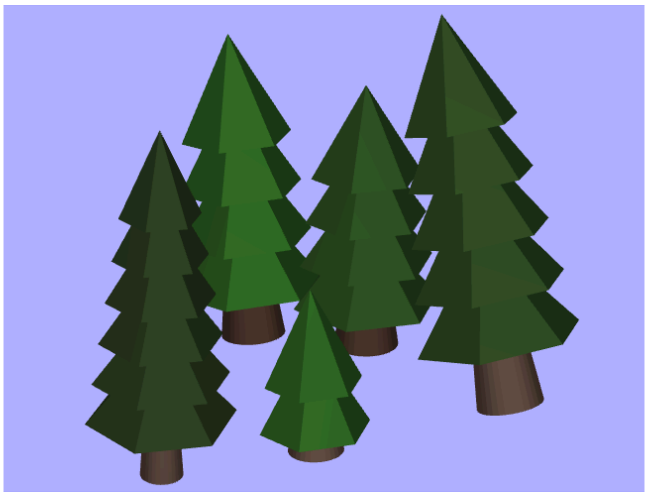


Figure 5: Examples of the intended trees

3.2 Parameterization of the tree

The previous object creates trees with the same basic structure; however, the appearance can be varied. Add to class constructor **MyTree** parameters that allow you to control its appearance:

MyTree

- Tree tilt:
    - Rotation in degrees (0º corresponds to the vertical position)
    - Axis of rotation, ***X*** or ***Z***
- Trunk base radius
- Tree height
- The color of the crown in RGB (trees can therefore appear in different shades of green)

To build the crown, you must use these parameters to generate others that may be relevant. For example:

- The height of the crown can be 80% of the height of the tree;
- The number of pyramids to use can be defined according to the height of the crown;
- The radii of the bases of the pyramids can be defined according to the trunk radius.

3.3. Forest creation

Using the function random() of Javascript, some randomness must be added to the parameter values in section 2.2: trees will be built with random parameters, within predefined intervals, and must have a small random offset in their position within the matrix.

3.4 Textures on trees

The use of textures allows you to obtain a realistic “look and feel” without increasing computational complexity.

To this end, choose and assign textures to the trunks and to the crowns.

Record this point in the development: 📷 (3) JS (4)

4. Helicopter

A helicopter is intended to be included in the scene. Its 3D shape can be realistic and based on the usual fire fighting equipment or defined in the "animation film" style, modeled in a simplified and creative way. When in flight, it hangs a bucket of water to put out fires. It must be animated and have controllable movement from the keyboard in accordance with the following subsections.

4.1 Helicopter modeling

Create a new class ***MyHeli***. To model the helicopter, you can use a combination of the different objects used in previous works, so that the object consists of a head with cabin, tail, upper propeller and rear propeller. It must have landing gear and be able to carry a bucket of water.

It is suggested to use spheres, ellipsoids, cylinders or other objects that you consider relevant (however, there should not be an excessive number of polygons so as not to affect the application's frame rate too much).

At the beginning of the application, the helicopter must be located on the building's "heliport". It should be clearly visible when the scene starts (using a good definition of the camera's initial parameters) in order to facilitate the evaluation of its modeling. Its length should be approximately 10 units.

The different objects used to create the helicopter must have materials and textures applied to them, appropriate to the parts they represent.

Show an image of what the helicopter looks like, taken from a close enough distance to see it in detail).

At this point, the animation mechanisms and control to an object of the class ***MyHeli***. Be the helicopter somewhere in the scene, hovering in the air. The helicopter must have the following behavior:

1. It is user controllable (next section) and can move forward or backwards; it can also rotate around a vertical axis (change of direction);
2. When in flight, it moves at a fixed, parameterizable altitude; go down from there to land at the heliport and, over the lake, to carry the bucket with water;
3. Leans forward a little when using the forward key (next section);
4. Leans back a little when using the backspace key (next section);
5. When lifting from the heliport, it rises to cruising altitude and automatically releases the bucket of water that remains hanging;
6. When descending to the heliport, it automatically collects the bucket of water.

4.3 Helicopter control

To be able to control the movement of the helicopter in the scene, it will be necessary to provide, in the interface, the possibility of pressing one or more keys simultaneously.

Analyze the functions related to keyboard control in the classes:

MyInterface:

- initKeys()
- processKeyDown() / processKeyUp()
- isKeyPressed()

MyScene:

- update()
- checkKeys()

Run the code and check the messages on the console when the "***W***" and "***S***" keys are pressed simultaneously. Other keys may be included according to the following text.

1. Modify the class ***MyHeli*** according to the following:
   - Add variables to the constructor that define:
     - The position of the helicopter (x, y, z);
     - Its orientation (suggestion: angle around the YY axis);
     - The velocity vector (initially zero) updated with the

position the helicopter;

- Create the methods ***turn(v)*** and ***accelerate(v)*** to rotate and to increase/decrease the speed of the helicopter:
  - turn(v): directly affects the orientation variable (angle); it must also cause an update of the velocity vector (only in direction, maintaining the norm);
  - accelerate(v): directly affects the norm of the velocity vector, and its direction must be maintained.

2. Plan that keys can invoke the methods *turn* and *accelerate* of the helicopter, in order to implement the following behavior:

   - Speed up or break with the "**W**" or "**S**", respectively;
   - Rotate the helicopter left or right respectively with the “**A**” or “**D**”;
   - "*Reset*" the position and speed of the helicopter using “**R**”: the helicopter must be placed on the heliport, in a resting position.
   - Raise the helicopter: key "**P**"
     - If it is at rest on the heliport, the propellers begin to move and take off vertically, until reaching cruising altitude; from then on it is under the user's control;
     - If you're filling the bucket with water in the lake, it rises vertically to cruising altitude; from then on it is under the user's control (section 5).
   - Lowering the helicopter: key "**L**":
     - If you are on the lake and have an empty water bucket, go down until the bucket touches the water (section 5);
     - If you are in any other position in the scene and have an empty water bucket, starts an automatic movement to position itself over the heliport, in landing position, and descends until it lands; stops the movement of the propellers and comes to rest.

3. Create one *slider* in the GUI called ***speedFactor*** (between 0.1 and 3) that changes the sensitivity of the previous acceleration and rotation, that is, "accelerates and decelerates" the displacement speed and rotation.

5. Water and fire

It is intended to add a new functionality to ***MyHeli***, so that the helicopter collects water from the lake and drops it on the fire that is burning in the forest. Use whatever representation you consider appropriate for the water as it falls from the bucket onto the fire.

1. Creation of objects
   a. Create a new class ***MyFire***, which will represent an ongoing fire in the forest. Flames can be simulated by triangles of variable size; apply appropriate materials and textures to resemble a fire (the animation of the flames will be done later, using shaders);
   b. Place a lake ***MyLake*** in the scene; it can be represented by a simple 2D object, with a suitable material/texture;
   c. Ensure that the helicopter, building, lake and forest (with objects of ***MyFire***) are visible at the beginning of the scene for evaluation.
2. Add to ***MyHeli*** the functionality of picking up and dropping water as follows:
   a. When pressing the key **"L"**, and under the condition that the helicopter is *stopped* over the lake, the helicopter must descend vertically until the bucket touches the water;
   b. Pressing "**P**", the helicopter goes up again, now carrying water in the bucket; when reaching cruising altitude, it maintains its orientation and speed at zero, as well as reacting again to the W, S, A and D keys;
   c. By pressing "**O**", and under the condition that the helicopter is over the fire, the helicopter must open the bucket of water; this falls on the fire, turning it off.

Record this point in the development: 📷 (6) JS (6)

6. *Shaders* and animation

This section aims to give more realism to the scene, giving it some movement, so that some objects will be endowed with an elementary animation:

- The triangles that represent the flames of the fire now have animated geometry;
- The helicopter's lifting and landing maneuvers are now marked with texture animation in the heliport runway.

The aim of this section is to animate the polygons that make up the flames of the fire. Each one must curve cyclically but with some randomness, giving the illusion, as a whole, of the undulation visible in a fire.

This effect can be achieved through the displacement/oscillation of the vertices of the triangles that make up the flames. The displacement must, for this purpose, occur in all triangles, albeit in a different way (again, some randomness).

Record this point in the development: 📷 (7) JS (7)

6.2 Heliport in maneuvers

It is now intended to implement innovative signage on the heliport, indicating the occurrence of helicopter maneuvers: varying the texture of the heliport and using pulsating lights:

- When maneuvering to take off, the texture used on the heliport must alternate ("blink") between the texture used until now (letter H") and a new texture, with the letters "UP".).
- When landing, you must proceed in a similar way but with a texture with the letters "DOWN".
- Four small objects must be added to the four corners of the heliport. When maneuvering, these objects change their material, starting to have emissive properties; emissivity must be sinusoidally variable in time, in order to create the sensation of pulsating lights.

7. Additional developments

These developments are for appreciation. Therefore, they are of less priority and have less weight in the evaluation of the work.

From the following alternatives, choose one (and only one) for implementation:

A. Animated Sky-Sphere, simulating the sky in a more realistic way; In addition to the previously applied texture, a second texture, with clouds, must be applied to the sphere created in ***MyPanorama***.
The texture chosen for the clouds should be in shades of gray, with a channel *alpha* in such a way that it is possible to “compose” the texture overlaying the previously applied texture (see, for example, the texture ***waterMap*** from TP5).
To apply both textures simultaneously, you will need to use *shaders* that manipulate the coordinates of the new texture so that the clouds move in the sky. The *fragment*

of the sphere's stacks so as not to overlap the "ground".

B. Depending on the type of maneuver, the heliport must alternate between two textures with the letter "H" and the letters "DOWN" or "UP", but mixing the textures in the transitions. Mixing must be carried out based on *shaders*.

C. Replace the polygonal representation of the lake: make a black and white mask of the entire terrain (400x400); paint the mask black in the area you want to be the lake (free format); using shaders, each pixel of the terrain is “painted”:
   a. In the white area of the mask: with a representation of “grass” (texture), as before;
   b. In the black area of the mask: with a representation of “water”, representing the lake; the surface of the water must have undulations (geometry) and animated textures in order to become more realistic.

Submit the final code. 📷 (8) JS (8)

Notes on work evaluation

The maximum rating to be given to each paragraph corresponds to an optimal development of it, in absolute compliance with all the features listed. Without losing the creativity desired in a work of this type, **will not be counted**, for evaluation purposes, **any other developments** in addition to those requested, **namely as a way of compensating for missing components**.

In the item **General appearance and creativity of the scene**, it will be assessed how the **use of techniques** defined **in this work and previous works** allow a **greater degree of innovation, originality, as well as the complexity and visual care of the application**.

In the item **Software quality** consideration will be given to **code efficiency, its structure, readability and the use of comments** for documentation purposes.

The criteria to be used in the evaluation and respective weights are the following:

1. **Sky-sphere [1.0]**
   1.1. Creating a sphere [0.5]
   1.2. Adding Panoramas [0.5]
2. **Fire-fighters Building [2.5]**
   2.1. Representation of window [1.0]
   2.2. Representation of the building [1.5]

4. **Helicopter [4.0]**
    - 4.1.Helicopter modeling [1.0]
    - 4.2.Helicopter animation [1.5]
    - 4.3.Helicopter control [1.5]
5. **Water and fire [2.0]**
6. **Shaders and animation [1.5]**
    - 6.1.Flames ripple [1.0]
    - 6.2.Heliport in maneuvers [0.5]
7. **Additional developments [1.5]**
8. **General appearance and creativity of the scene [2.0]**
9. **Software quality [2.5]**

Work plan proposal

The following work plan is recommended over the remaining time of the semester (assignment start and end dates):

Pontos do enunciado	Turns Mond.	Turns Tuesd.	Turns Wedn.	Turns Thurs.
Sky-Sphere + Building	31/3 -> 6/4	1/4 -> 7/4	2/4 -> 8/4	3/4 -> 9/4
Building (concl.)+Tree	7/4 -> 13/4	8/4 -> 14/4	9/4 -> 15/4	10/4 -> 16/4
---	14/4 -> 20/4	15/4 -> 21/4	16/4 -> 22/4	17/4 -> 23/4
Forest+ Helicopter	21/4 -> 27/4	22/4 -> 28/4	23/4 -> 29/4	24/4 -> 30/4
Helicopter (concl.)	28/4 -> 4/5	29/4 -> 5/5	30/4 -> 6/5	1/5 -> 7/5
---	5/5 -> 11/5	6/5 -> 12/5	7/5 -> 13/5	8/5 -> 14/5
Water and Fire	12/5 -> 18/5	13/5 -> 19/5	14/5 -> 20/5	15/5 -> 21/5
Shaders and Animation	19/5 -> 25/5	20/5 -> 26/5	21/5 -> 27/5	22/5 -> 28/5
Aditional developments	26/5 -> 29/5	27/5 -> 30/5	28/5 -> 31/5	29/5 -> 1/6

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

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

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

Video

You should also prepare a **1 minute video in mp4**, to be submitted in a specific area to be indicated in Moodle. The video must be captured from the screen demonstrating all implemented features (type names **"projeto-t<turma>g<group>.mp4"**).