DINASOURS

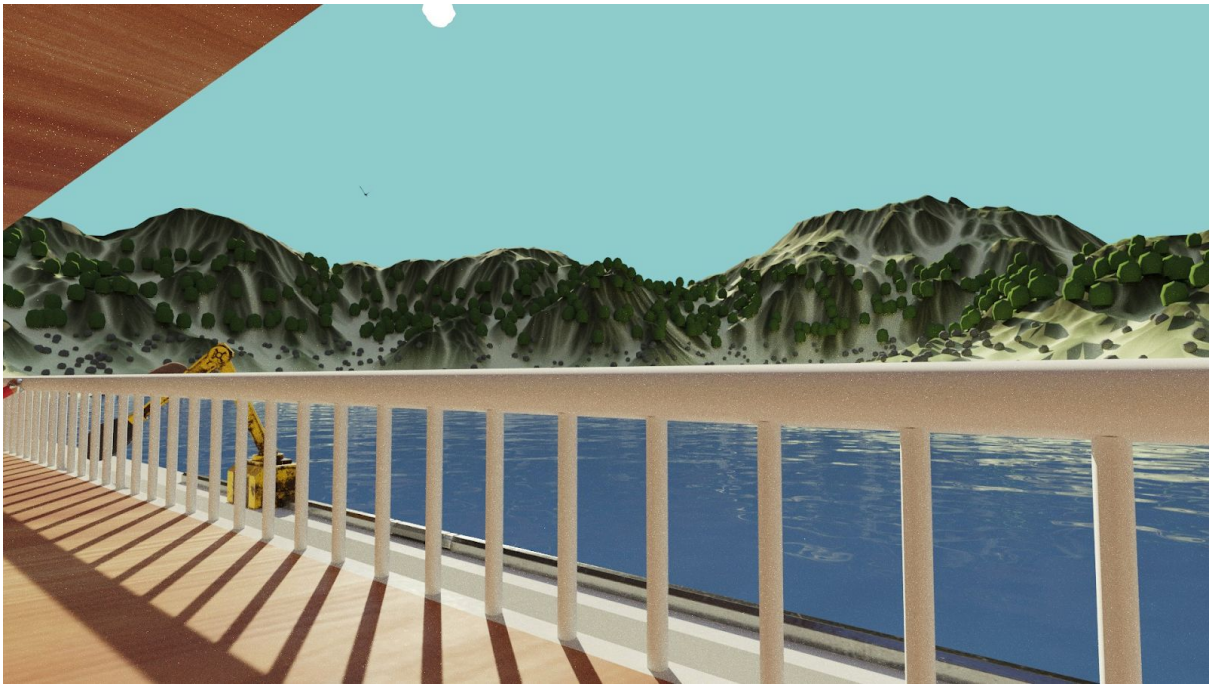
Antoine Colombier - Sayak Samadar - Anna Gomez

For this project we've decided to implement the code in C++ given that our team has more experience with this programming language and it's one of the most supported programming languages to work on OpenGL.

Notice

This engine has been designed to work with the OpenGL Z-up standard models (like Blender). It is thus not able to load properly models made with the Y-UP standard provided in lab 7.

Models and textures (and skinning animation) have been provided by the 3D artist student Emilien Colombier.



Composition and render made by the Emilien with al the item he has provided

Resources

For the design of our scene, we've asked a 3d artist to help us with the majority of the resources. He provided us with blend and dae files, along with textures for all the meshes. All of the resources are in the res file. The cubemaps for the skybox are the only resources downloaded from the internet.

Scene

You can run the final scene after making the project, by using the default executable without any argument. By default, you will end up on the ship from where we set our point of view. As the camera movement are tracked as soon as the software starts, you might start with a weirs camera position. Use ‘P’ to reset.

The important thing to see our the two animation (mechanical arm + pterodactyl in the air), vulcano particules, and material shading. Note that the point of light has been animated as well so we can see the lightning behavior changing.

A list of the arguments that you can use is provided at the end of this report.

Structure of the project

The project folder is divided into subparts:

- **assets** contains all the scene specific code that are use especially to instantiate the scene, build the hierarchy (especially the arm), declare animations.
- **objects** & **textures** contain all the 3D object loaded into our scene, and differents textures applied on these objects.
- **shaders** contains all the vertex and fragment shaders used in the project.
- **core** contains all the implementation of our 3D engine.

For this project we focused on creating an organized engine for our scene. The folder core contains all of our main code to be able to recreate a scene with OpenGL. The folder res contains all of our resources. The folder shaders we used and in assets we placed some files that recreate the scene we designed for this project.

To recreate a scene we created a tree like structure that consists in one root node Scene which contains all the nodes to be included. (see *--display-tree* parameter)

Loading the resources

To load the resources we used the library Assimp that helped us to get all the information of any assets file. In here, .dae files were provided by the 3D artists. For every element in the scene we loaded a .dae file which contained the vertex, normal, UV, material information of all the meshes included in the file, and node structure that we were extracting and re assembling into our final tree. We assigned a Material object to every mesh, which contained all the diffuse, specular and shininess information and all the texture maps provided.

Hierarchical drawing

With the structure of our scene, every node contained it's own transformation information, which allowed us to recreate this feature in an organized way.

Skybox

For the skybox we downloaded a free resource from the internet and created a cube texture with the 6 textures provided.

Camera

We recreated a dynamic camera controlled by the mouse and keyboard. You can control the view direction with the mouse and the movement with the arrows. For the design of our scene, we still have movement of the camera but it is bounded by the limits of the ship. You can discard these limits using the *--free-camera* argument

Lighting and shading

We created one basic directional light, and for the shading effects we chose the phong illumination model and we implemented bump mapping. To be able to to do this, when we loaded the meshes, we used the library assimp to generate the tangents and bitangents from the normals.

Animation

Keyframe animation

We recreated a mechanical arm attached to the ship that shows the keyframe transformations for translations and rotations. Furthermore, the pterodactyl is animated using to keyframes as well.

Particles

For the physic based animation, we recreated some particles that are coming out of the volcano of our scene.

Skinning

Unfortunately, despite many hours spent on it, we could not fix our system to get them working and had discard them completely in the end. You can still see it “not-working-implementation” in the code.

Compilation and running

To compile the project do:

make

To run the program:

./petit_pied

You can control the camera using the mouse, and the four directional arrows. You can also use ‘P’ to reset the camera, and SHIFT to move faster

Here is the list of available positional arguments:

- --attach-marker <node_name>
  This argument enable the XZY marker, that can be attached to a local transformation using the node name that can be gathered using the --display-tree. For example, use *--attach-marker Scene* to set the marker at the root or *--attach-marker Plane* to set the marker on the Pterodactyl
- --free-camera
  This argument turns off bounding off the ship and gives you the ability to move everywhere you want.
- --show-fps
  Shows a FPS counter in the CLI.
- --display-tree
  Dump the complete tree (more than 2000 items)
- --disable-skybox
  Turn off the skybox