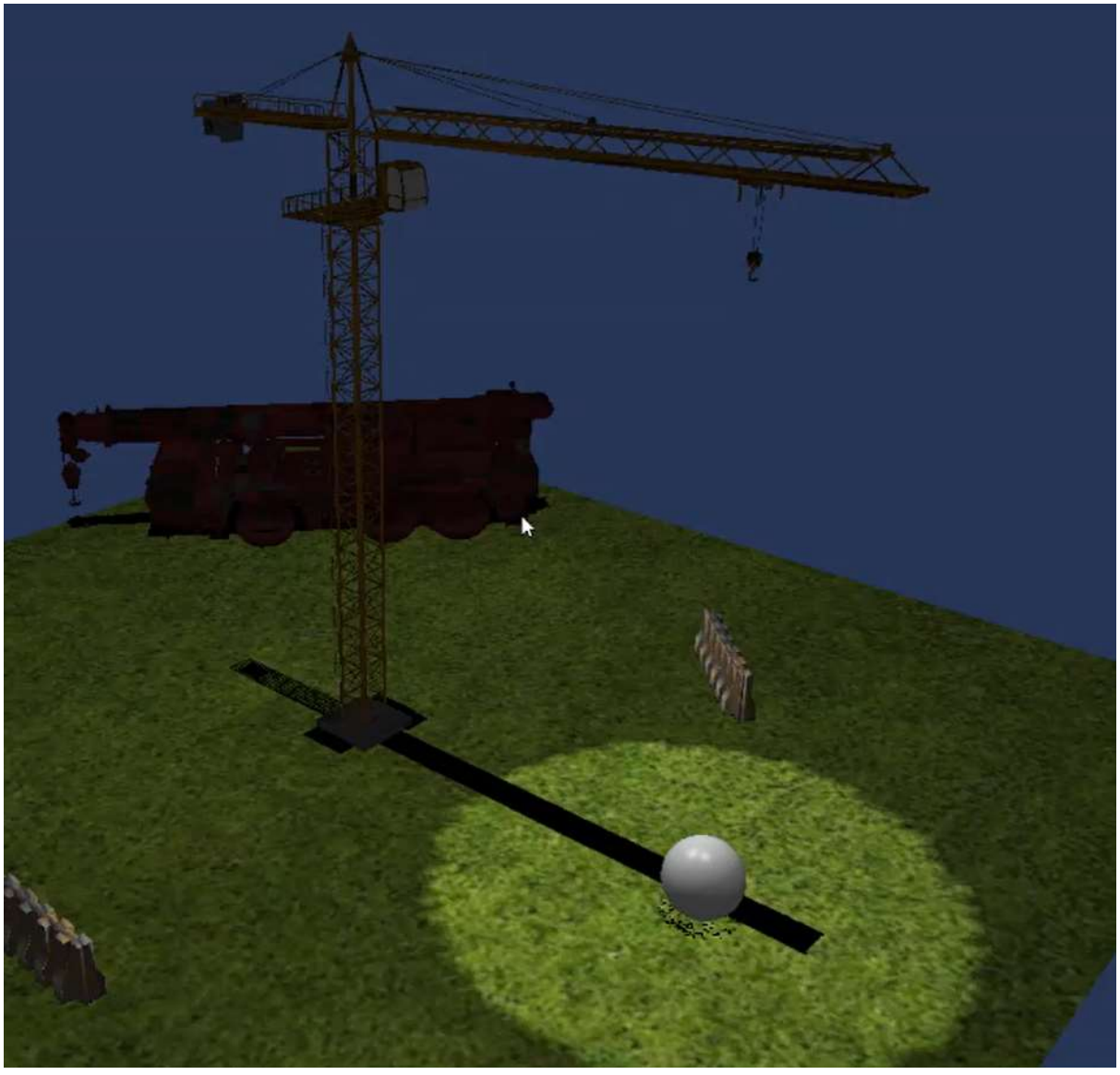
Technical report

THE CRANE

Study cycle:	**Computer engineering**
Course:	**M-I5070 – Graphic**
Students:	**Gabriel Mendonça Gomes, Leonardo Babbucci, Alessia Sarak**
Teacher:	**Achille Peternier**



Index

1 Abstract

In this project, a 3D modeling of a mechanical crane was carried out using the OpenGL and 3DS Max computer graphics tools. The modeling was created from reference images and includes all the main components of the crane, such as the boom, the trolley and the rolling bridge. In addition, an animation feature has been implemented that allows you to simulate the movement of the truck along the crane boom and the lifting of the load using the hook. The end result is a detailed and realistic modeling of the mechanical crane, which can be used for design or training purposes.

2 Introduction

3D modeling is a computer graphics technique that allows you to create three-dimensional representations of objects or environments. In many fields, such as engineering, architecture and animation, 3D modeling has become an indispensable tool for the design and visualization of complex objects or environments.

In the present project, we wanted to carry out a 3D modeling of a mechanical crane using the OpenGL and 3DS Max computer graphics tools. The mechanical crane is a machine used to lift and move large loads in industrial or construction site environments. It consists of a movable arm, a trolley that slides along the boom and a hook for lifting the load.

Our goal is to create a detailed and realistic modeling of the mechanical crane, which can be used for design or training purposes. To carry out this project, we used reference images to reconstruct the main components of the crane, such as the boom, the landing gear and the rolling bridge. In addition, we have implemented an animation feature that allows you to simulate the movement of the truck along the crane boom and the lifting of the load using the hook.

The project was developed in three phases:

- Requirements analysis: We have collected information on the structure and operation of the mechanical crane, using sources such as technical manuals, reference websites and reference images.
- Modelling design: We used OpenGL and 3DS Max computer graphics tools to create the mechanical crane modeling, using the information gathered during the requirements analysis phase.
- Implementation and testing: we tested the modeling to verify its correctness and effectiveness, making any necessary changes and improvements.

The result is a 3D modeling of the mechanical crane that meets the objectives set and that can be used for design and study purposes.

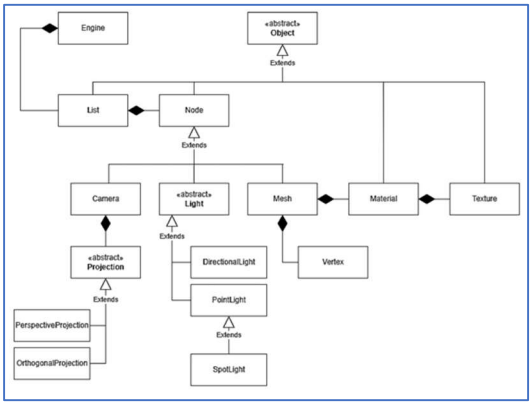
3 Implementation

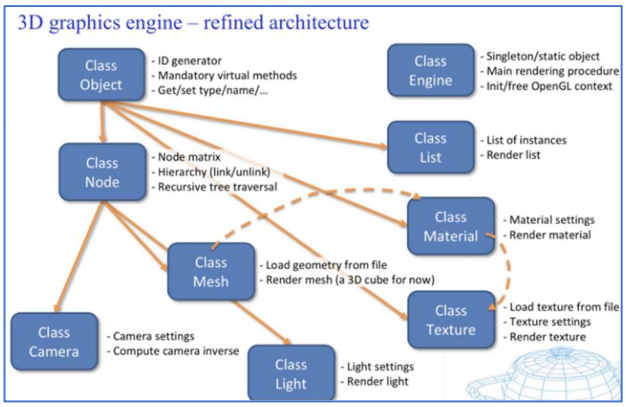
In this chapter, you will describe all the classes within your project, how they work, and how they were implemented.

3.1 Engine

The first component to describe is the engine. The engine is the fundamental part of the project as it deals with the actual operation of the crane. Before development, a UML diagram and a general scheme of the engine operation were drawn up, also with the help of the slides seen in class.

3.1.1 Structure





From the diagrams you can see how the engine is separated from the rest, so if you were to change the graphic part of the project you would not change the engine.

The use of the engine involves the insertion of information relating to the structure and operation of the mechanical crane, such as the arrangement of the vertices, the textures to be applied to the surfaces and the light sources present in the scene. Once this information is entered, the engine uses OpenGL and 3DS Max computer graphics tools to create 3D modeling of the mechanical crane and to implement features such as animating the movement of the truck along the crane boom and lifting the load with the hook.

In the next points the important classes of the project will be described.

3.1.2 Camera

The "camera" class was used to define the position and orientation of the camera displaying the scene.

The class extends the Node class and contains a constructor that initializes the Camera object with an ID, name, and object of type Projection. The class also includes an empty destroyer and two getter methods. The getProjection() method returns the Projection object associated with the camera, while the getInverse() method returns the inverse array of the final array of the chamber, obtained from the transformations of the node hierarchy. The class also includes an instance variable to store the Projection object associated with the camera.

3.1.3 Texture

The "texture" class was used to define the textures to be applied to the crane surfaces.

The class contains a constructor that initializes the Texture object and checks whether anisotropic filtering support is available on your graphics card. The class also contains a render() method that configures texture wrapping and filtering behavior, and a setTexId() method that loads an image from disk and uses it to create an OpenGL texture. In addition, there is a setPath() method that sets the search path of the images to be loaded as textures and a destructor that deletes the texture from memory. The class also includes some class and instance variables to store texture information, such as texture ID, texture name, and supported anisotropic filtering level.

3.1.4 Vertices

The "vertices" class was used to define the geometries of the mechanical crane, i.e. the arrangement of the vertex points that make up the crane surfaces.

The class contains a constructor that initializes the Vertex object with the position and normal vector of the vertex, as well as with predefined texture coordinates (0,0). The class also includes an empty destructor and three getter methods to obtain the position, normal vector, and texture coordinates of the vertex. In addition, there is a setTextureCoordinates() method that allows you to set vertex texture coordinates. The class also includes three instance variables to store this information: a position vector, a normal vector, and a texture coordinate vector.

3.1.5 Spotlight

The "spotlight" class was used to implement spotlights, i.e. light sources with a limited light cone.

The class extends the "PointLight" class and includes a direction vector that specifies the direction of the light beam emitted by the light source. The class also includes a "render" method that is used to set the direction of the light beam using OpenGL functions and then to call the "render" method of the base class "PointLight" to set all other properties of the light.

3.1.6 Mesh

The "mesh" class was used to represent a mesh 3. The class contains information such as vertex position, texture mapping coordinates, vertex normals, and a reference to an object of type "Material". The class also includes a number of methods for handling object transformations, lighting, and material application, as well as adding new vertices to the mesh and retrieving existing vertices. In addition, the class includes a "render" method that is used to draw the mesh on the screen using OpenGL.

3.1.7 FileReader Property

This class is a file reader for a specific file format used to load 3D objects into a graphics project that uses OpenGL. The class includes a "readFile" method that takes as an argument the path to the file to be read and returns a "Node" object that represents the root of the tree of 3D objects loaded from the file.

The class also includes a recursiveLoad recursive loading method that is used to scan the file and load the 3D objects contained in the file. In particular, the "recursiveLoad" method reads the information in the file and uses it to instantiate different classes, such as "OvObject", "OvMesh", "OvLight", "Mesh", "Light", "PointLight", "DirectionalLight" and "SpotLight". The "recursiveLoad" method also handles the loading of materials, textures, and lighting information.

3.1.8 Material

The class contains two constructors that initialize the Material object with an ID, name, and material parameters, such as emission vector, environment vector, diffusion vector, specularity vector, and gloss coefficient. One of the constructors also accepts a Texture object as an optional parameter. The class also includes an empty destructor and a render() method that configures material parameters on the GPU and, if present, also makes the texture associated with the material. The class also

includes five getter methods to get material parameters and a setTexture() method to set the texture associated with the material. In addition, the class includes six instance variables to store this information: four vectors for material parameters and a gloss coefficient, as well as a pointer to the texture associated with the material.

3.1.9 Node

The Node class represents a node in a tree. A Node object can have a relative and can have a series of children. The class contains functions to add and remove children, to find a child node based on its ID or name, to remove a node and its children, and to get the number of children of a node. The class also contains a rendering function, which can be overwritten into subclasses to provide custom rendering functionality. The Node class is a subclass of Object, which has an ID and a name. The Node class is marked as LIB_API, which means it can be used in an external library.

3.1.10 Projection

The Projection class represents a projection for an OpenGL window. Provides functionality to set the OpenGL projection matrix, get and set the projection matrix, get and set the window width and height, and update the projection. The Projection class can be used to create different types of projections, such as orthogonal or perspective projections, depending on the projection matrix set. The Projection class is marked as LIB_API, which means it can be used in an external library.

3.1.11 Lights

The "lights" class was used to define the light sources present in the scene. It was possible to define different types of lights, such as point lights, directional lights or spot lights, and set their parameters such as intensity, color and position. In addition, it was possible to use the "highlights" class to implement shading effects, such as dynamic shadows or fixed shadows, which allow for a more realistic scene.

The class system used to manage lighting is a hierarchical system that starts from the abstract light class and reaches the spotlight. Therefore, when the render() method of the List class is called, the light runs through the entire hierarchy automatically rendering the lights according to their type.

3.1.12 Shadow

The FakeShadow class is a subclass of Mesh that provides functionality to create a shadow of an OpenGL model. An object of type Shadow can be set as a child of another Mesh node, so that the shadow of the model is cast on the surface of that node. The Shadow class contains functions for setting and offsetting the shadow from the model, for obtaining the shadow scaling matrix, and for obtaining the shadow rotation matrix. The Shadow class also implements the rendering function to draw the shadow on the scene. The Shadow class is marked as LIB_API, which means it can be used in an external library.

4 Test

Several tests were carried out on the operation of the product, it was not considered necessary to report them on the following report, since the product is functional. As a result, all tests have passed and there is no particular test to report.

5 Conclusion

In conclusion, the computer graphics project that uses OpenGL to represent a crane, has proven to be a successful project. The use of OpenGL allowed to create a realistic representation of the crane, including different parts of the mechanism such as the boom, the hook and the trolley. The crane was able to move smoothly and perform several actions, such as raising and lowering the load and rotating the boom. In addition, shadows could be implemented to improve the appearance of the scene. However, there are some areas where the project could be improved. For example, it might be useful to implement a physics system to make the crane's movements even more realistic and add more detail to the scene, such as a terrain or buildings in the background. In addition, it would be useful to implement an input system to allow the user to interact with the crane and control its movements. In general, the crane computer graphics project was a success and demonstrated the effectiveness of OpenGL as a tool for creating realistic 3D scenarios.