**Progetto RTPG Carugno**

Soft bodies simulation using OpenGL and Bullet Physics

**Classes overview:**

* **Main**: Main file of the project
  + Sets up OpenGL
  + Starts the rendering loop
  + Collects users inputs through the GUI
  + Steps forward the simulation
* **Model**: Uses Assimp to import a model from an external file
  + Given the file location this class retrieves the vector containing all the meshes composing the model by iterating over its hierarchy
* **Mesh**: Defines a mesh by storing all the vertices and indices
  + Uses the OpenGL graphics APIs and the shaders to render itself
* **Shader**: Shader loading class
  + Receives vertex and fragment shader
  + Creates and handles the shader program
* **Camera**: Flying omni-directional camera
  + Sets up a camera that can move
  + The mouse is used to look around
  + Different keys allow movement along different directions
* **VAO**: Vertex Array Object class
  + Sets up the VAO so that it can be linked to VBOs and EBOs
* **VBO**: Vertex Buffer Object class
  + Given a vector of vertices it generates the corresponding VBO buffer and passes all the vertices data to it
  + The vertices are defined as a structure whose attributes are the vertex’s position, normal and colour
* **EBO**: Index Buffer Object class
  + Given a vector of indices it generates the corresponding EBO buffer and passes all the indices data to it
  + The indices are defined as a vector of OpenGL unsigned integers
* **Physics**: Bullet physics simulation
  + Creates the simulation
  + Generates the soft bodies

**Execution flow:**

1. **Initialization**
   1. OpenGL window and context setup
   2. Load vertex and fragment shader
   3. Initialize camera
   4. Load external models
   5. Bullet physics simulation setup
   6. Generate world plane
   7. Simulation’s frame rate setup
   8. GUI initialization
2. **Rendering loop**
   1. React to keyboard input
   2. Calculate current frame
   3. Move forward physics simulation
   4. Activate shader program
   5. Update camera position
   6. Draw world plane
   7. Collect GUI input from user
   8. If “Generate” button is clicked then create a soft body using the GUI collected input
   9. Draw the present soft bodies
   10. Swap window’s front and back buffers
   11. Process events in the queue
3. **Free resources**
   1. Delete GUI
   2. Delete shader program
   3. Delete physics simulation
   4. Delete OpenGL window and context

**Relevant design choices:**

* **Camera**: A flying omni-directional camera has been used. It uses WASD, Ctrl and Shift to move along all the possible directions. The left mouse button is used to look around. While being more complex to implement it allows more freedom to move around and it’s possible to use it also as a visual debugger to make sure that the soft bodies in the simulation are behaving correctly both during the generation and the interactions.
* **Soft bodies generation**: A soft body is Bullet is defined as a set of nodes, faces and links. Given a mesh the nodes are the vertices while faces and links are generated using the mesh indices. The faces are the mesh triangles while the links are used to hold nearby faces together. In Bullet there are two main ways to generate a generic soft body. The first one uses a function from an helper class that generates a soft body from a list of vertices and indices. This helper function automatically generates all the vertices, faces and links that are necessary to render the soft body. The second method instead is to create the soft body directly from a list of vertices. However, once the list of vertices is passed to the constructor, only the nodes are generated. Since faces and links are still missing they have to be added manually through an iterative procedure. To decide which vertices are part of a face or a link the list of indices is used. Indices are taken in sequential triples. For each triple the corresponding face and its three links are generated and appended to the soft body until the list of indices is finished. During development the first method was yielding wrong soft bodies because their meshes were unexpectedly unfolding. The reason is that such method generates the faces correctly but not the links. To generate the links between nearby vertices an array of Boolean values is used. Such array is actually a 2D matrix turned mono-dimensional using row-major ordering. By looping over such array links between nearby nodes are created and appended to the soft body. However, since such Boolean array is not a circular one, links between indices at the end and at the start are missing. This causes generic soft bodies meshes to unfold along shared borders of nearby triangles which were generated at the beginning and at the end. This is the reason why the second method was used. Creating directly a soft body using its constructor rather than an helper method provides way more control over it. The problem becomes ensuring that all the faces and links are created correctly since now it has do be done manually. To simplify and ensure the correctness of such procedure the next design choice was introduced.
* **Merging of meshes**: Previously a model was imported by iterating over his hierarchy and extracting all its meshes to put them in vector. Each meshes contains its corresponding vertices and indices. By using Assimp such meshes were pre-processed so that once imported they were made of triangles and identical vertices were merged with their indices updated accordingly. Later a soft body was generated. As previously described generating soft bodies from models imported this way was causing their meshes to unfold during the simulations. So soft bodies had to be generated using the second method described in the previous design choice. Such necessity resulted in this design choice. The idea behind this one is to invest more computational time at the start of the program by adding a function that, right after loading a model, generates a unique big mesh instead of keeping multiple ones for each model. Such function merges duplicate vertices present in different meshes and updates all the indices accordingly. It returns a unique big mesh with all the unique vertices and all the indices of all the meshes but updated accordingly to match the new unique vertices. It benefits also the rendering of the soft body since instead of using a VBO and an EBO for each mesh it just uses one VBO and one EBO with also all the vertices and indices optimized. Such decision fixed the soft body generation, its rendering and also its physical behaviour.
* **GUI**: A GUI allows to choose the parameters to use to generate a soft body. The soft bodies in Bullet have multiple configuration parameters. The problem is that some are used to optimize the behaviour of the soft body during the simulation (ex. collision margin, soft body solver iterations, internal pressure, faces and links stiffness and so on). To provide enough freedom while also avoiding breaking the simulation only some configurations parameters are available to choose. Hidden ones are either set automatically or have been optimized manually through trial and error to better fit the scope of the project.
* **Soft bodies drawing**: Soft bodies in Bullet are different from rigid bodies since they’re not defined by a unique world transform but all the nodes have their own unique world coordinates. To render them correctly all the nodes positions have to be retrieved during each frame and used to update the vertices positions to later pass them to the VBO. Since soft bodies also do not have their corresponding indices list such list has to be retrieved to pass it to the EBO. Once vertices and indices of a soft body have been collected such soft body can be finally rendered correctly.

**Results**:

* The Bullet physics simulation is set up and working correctly
* A soft body can be generated from any imported external model
* Soft bodies to generate can be configured through the use of the GUI
* Each soft body can interact and collide both with other rigid and soft bodies
* Soft bodies physics is rendered in real-time with OpenGL
* Simulation and soft bodies parameters can be fine tuned to decide whether to focus on achieving physical accuracy or rendering speed