Operational Report

Incarico di lavoro autonomo n. 27/2024

Matteo Scuderi - 1937090

Purpose of the Collaboration

The main purpose of this assignment is creating a "digital twin" for a surgery phantom, with the aim of integrating it in a digital VR simulation.
The said phantom is a 1:1 reproduction of the anatomical section bounded by the vertebrae C1 and C7, including all surrounding organs and tissues, such as muscles, blood vessels, esophagus and medulla.

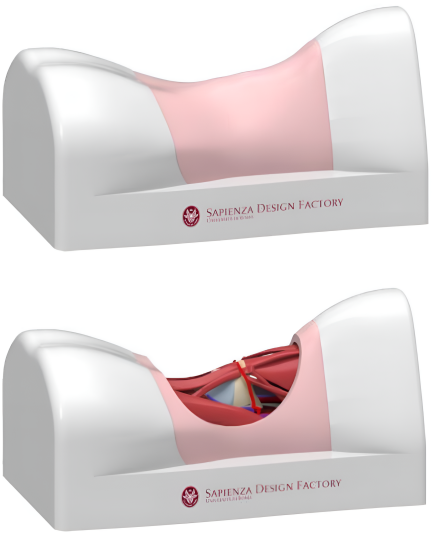


Figure I: Spine Surgical Phantom in use

This spine phantom is realized via additive manufacturing and therefore, a 3D model of each component was provided. In the table below, a list of all components is reported, as well as some additional information (such as rigidity and color).

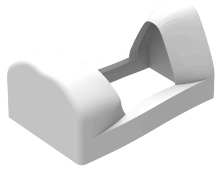
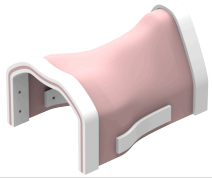
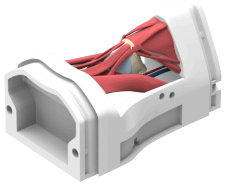
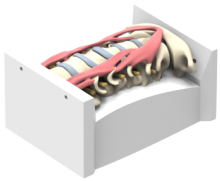
Immagine	Elemento	Caratteristiche Per la durezza delle parti stampate abbiamo usato la scala da Shore 30 (morbido) a 95 (duro)	Colore
	Scocca Esterna	Rigido	C:0 M:0 Y:0 K:0
	Pacchetto Tessuti		
	Scocca Staffe	Rigido	C:0 M:0 Y:0 K:0
	Esofago	Shore 30	C:23 M:89 Y:92 K:15 R:170 G:57 B:43
	Pacchetto Laringe-Trachea	Shore 60	C:29 M:14 Y:10 K:0 R:179 G:199 B:214
	Legamenti	Shore 60	C:13 M:16 Y:31 K:0 R:221 G:206 B:178
		Shore 60	C:13 M:16 Y:31 K:0 R:221 G:206 B:178
	Pacchetto Muscoli_DX	Shore 30	C:15 M:86 Y:69 K:3 R:202 G:71 B:76
	Pacchetto Muscoli_SX		
	Muscolo Miloioideo_DX	Shore 30	C:15 M:86 Y:69 K:3 R:202 G:71 B:76
	Muscolo Miloioideo_SX		
	Osso Ioide	Shore 95	C:14 M:19 Y:43 K:0 R:221 G:199 B:155
	Tiroide	Shore 30	C:23 M:89 Y:92 K:15 R:170 G:57 B:43
	Muscolo Sternocleidomastoideo_DX	Shore 30	C:15 M:86 Y:69 K:3 R:202 G:71 B:76
	Muscolo Sternocleidomastoideo_SX		
	Arteria Carotide_DX	Shore 30	C:0 M:100 Y:99 K:0 R:255 G:0 B:19
	Arteria Carotide_SX		
	Nervo Vago_DX	Shore 30	C:13 M:20 Y:72 K:0 R:224 G:195 B:101
	Nervo Vago_SX		
	Vena Giugolare_DX	Shore 30	C:15 M:86 Y:69 K:3 R:35 G:46 B:145
	Vena Giugolare_SX		
	Membrana Giugulare-Carotide_DX	Elastica	Semitrasparente
	Membrana Giugulare-Carotide_SX		
	Scocca Supporto Muscoli	Rigido	C:0 M:0 Y:0 K:0
	Scocca Interna	Rigido	C:0 M:0 Y:0 K:0
	Viti	Rigido	
	Vene-Arterie Vertebrale_DX	Shore 30	Arterie: C:0 M:100 Y:99 K:0 Vene: C:15 M:86 Y:69 K:3
	Vene-Arterie Vertebrale_SX		
	Disco Intervertebrale C2-3	Superficie Esterna: Shore 60 Parte Interna: Shore 30	Parte Esterna C:46 M:33 Y:14 K:0 R:144 G:157 B:186 Parte Interna: C:91 M:66 Y:13 K:1 R:37 G:95 B:156
	Disco Intervertebrale C3-4		
	Disco Intervertebrale C4-5		
	Disco Intervertebrale C5-6		
	Disco Intervertebrale C6-7		
	Flessori Spinali	Shore 30	C:15 M:86 Y:69 K:3 R:202 G:71 B:76
	Midollo Parte Interna	Consistenza gelatinosa	C:13 M:20 Y:72 K:0 R:224 G:195 B:101
	Midollo Parte Esterna	Shore 30	
	Vertebra C1	Shore 95	C:19 M:13 Y:44 K:0 R:209 G:204 B:157
	Vertebra C2		
	Vertebra C3		
	Vertebra C4		
	Vertebra C5		
	Vertebra C6		
	Vertebra C7		
	Scocca Supporto Posteriore	Rigido	C:0 M:0 Y:0 K:0
	Vertebre Supporti	Shore 30	C:0 M:0 Y:0 K:0

Figure II: Table of Phantom Components

The simulation software chosen for the phantom's digital simulation is MuJoCo. This choice was agreed on with Prof. Vendittelli for three main reasons:

- Realistic Simulation of contact forces (essential for surgical applications)
- Possibility of simulating deformable tissues/objects
- Experience of the writer in the use of the Software in medical applications.

The followed workflow is shown below.

Contents

1 Importing the 3D Model in MuJoCo

The preliminary step for simulation is importing the 3D model in the MuJoCo "scene" (i.e. the *.xml* file).
In order to better understand the issues with this procedure, a few MuJoCo requirements are stated below:

1. In MuJoCo, 3D models of each component must be imported as **single files** in a standard format (in this case, *.stl*).

2. Single files, or better *meshes*, must have a vertex count below 100,000.

3. For a smooth, bug-less simulation of contact forces, all meshes must be closed surfaces (in 3D-modeling lingo, they must be *manifold*).

As for point 1, the provided files were already separated component-wise. Unfortunately, some of them violate both Point 3 and/or Point 2.

For this reason, some modifications to the "incompatible" files were necessary. The 3D-modeling software **Blender** was used for this task.

1.1 Blender Modifications

Before discussing the chosen problem-solving strategy, a premise is necessary. The writer has no expertise in 3D-modeling and faced the said problem as a neophyte, having used Blender for the very first time.
Therefore, it is possible that the chosen strategy is not the most efficient, even though the result is still satisfactory, in the writer's opinion.

The main tool used for making the *.stl* files MuJoCo-compatible is Blender's built-in ***remesh*** function. This simple algorithm was used:

Algorithm 1 Remeshing for MuJoCo compatibility

```
Import .stl files in Blender
for each imported mesh do
  if mesh is not "manifold" then
    Apply: Voxel Remesh
    Apply: Quad Remesh (4.000 Vertices)
  else if #Vertices > 100.000 then
    Apply: Quad Remesh (4.000 Vertices)
  end if
end for
Export new meshes as single .stl files with scale 0.001 (for realistic dimensions)
```

Applying this algorithm, all "non-manifold" meshes (i.e. not convex and closed) are firstly "decomposed" in voxels (3D pixels). After that, the object's surface is *remeshed* with a Vertex count of 4.000, which reduces the file dimensionality (and

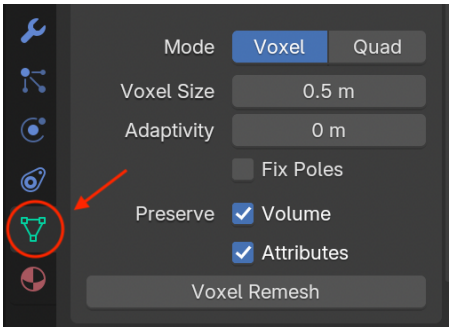


Figure III: Blender Command for Remeshing

consequently, the mesh quality) in order to make it MuJoCo-compatible. This last step is necessary for all *.stl* files that violate Point 2 (vertex count above 100,000). It is possible to choose a different balance between mesh quality and dimensionality, remeshing the files with an higher vertex count (as long as Point 2 is not violated).

After this process, all files can be imported in the MuJoCo scene (practically, in the *.xml* file) using a command of this type:

$$< geom\ name = "name\_of\_component"\ mesh = "path\_to\_mesh\_stl\_file" / >$$

1.2 Creation of the .xml file

As previously mentioned, the scene's "topology" is created via an *.xml* file. For generating this file, some additional python scripts were created.

Starting from a very "basic scene" (in which only some parameters are specified, but no objects are present), the script creates a new file and adds all the phantom's element with a hierarchical structure (parent-children).

The *.xml* file can be manually modified after its creation, adding or modifying both objects and parameters. However, the python script automates an otherwise tedious operation which is calculating the geometries' position in the scene.

2 Building MuJoCo Scene

In the implementation of the MuJoCo scene, three main key-points were considered: **computational demand, "user-friendliness" and realism**. All implementative choices were made to find a balance between the three. In particular sacrificing the latter often translates in an improvement of the other two key-points. Let's analyze these choices step by step, so to reconstruct the "backbone" of the MuJoCo scene.

2.1 CHOICE 1: Interactive Elements in the scene

As the real surgical phantom includes a lot of flexible elements - such as muscles - its digital counterpart has to deal with some computational limitations. The hardware used for simulation struggles to run smoothly in presence of too many deformable objects, as their computational demand is elevated. For this reason, it was chosen to limit the interactibility of the scene to the spine, as it will be the interested area in surgery. On the contrary, every other organ and tissue present is only a "background element" when the surgery is performed and their simulation is not as essential.

For this reason, **the main focus of this work was to replicate realistic spine movement** that matched the physical phantom's counterpart.

2.2 CHOICE 2: Python Simulation vs Built-in MuJoCo Simulation

MuJoCo's physics engine has some "primitives" that are common to most simulation Software (such as joints, actuators exc.). In particular, **joints** in MuJoCo assign to each object a certain number of **Degrees of Freedom** (from now on, **DOFs**). These DOFs define how the body can move in the scene - for instance, hinge joints define rotational motio around the joint's axis. A body with no DOFs is otherwise fixed in space and can't move even if forced.

With this in mind, the challenge was **creating a "joint pattern" for the spine that would replicate the motion of interest in surgery**, while keeping in mind the three key-points previously defined.

A first simplification was made and the spine was considered as a planar vibrating string. Of course, we would need a "discrete" version of the otherwise continuous and infinite-dimensional model, whereas we only have a certain number of verte-

brae and discs that move according to this motion. In this scenario, the "nodes" are represented by the vertebrae C1 and C7 respectively, which in the phantom are attached to the structural frame and therefore fixed.

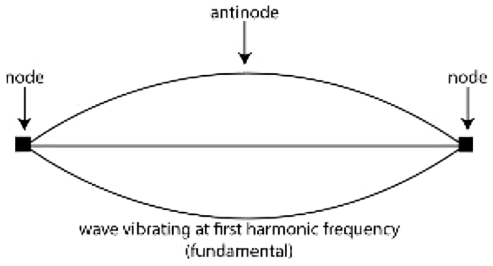


Figure IV: Vibrating String Simplification

With this premise we can face the main choice made so far: choosing between a Python-driven simulation or a built-in simulation.
Let's suppose we've placed our joints in a certain way (i.e. assigned the DOFs that define the spine movement). We can now proceed in two ways.

1. **External (Python) Script:**
   Place actuators (for instance velocity/position controlled motors) and assign the joints' velocities/positions calculated as a solution of the vibrating string equation. This action must be performed in real-time with an external script (e.g. Python script) and is computationally demanding, especially if the phantom is inserted in a larger scene with other interactions to be simulated. Moreover, this choice hides some compatibility issues from the MuJoCo-Python integration. For example, on MacOS the Python script must be run through a plug-in called *mjpython*, which however often crashed due to faulty implementation. As a result, any Mac owner would not be able to start the simulation due to compatibility issues.

2. **MuJoCo built-in Simulation:**
   In this case, the simulation runs natively in the MuJoCo application. This means that it can be run virtually on any computer, as the computational demand is much lower and there are no compatibility issues. Moreover, it can be easily integrated in a bigger scene, as there's no external code that might generate conflicts. On the down side, the built-in MuJoCo engine only allows direct assignment of **constant** actuator's references. Therefore, the "vibrating string" dynamics can't be assigned. Instead, one must build, using the engine's primitives, an interconnection between the spine elements that behaves as close as possible to the said string.

In this way, the spine will deform when interacted with, but all the computations will be held in real-time by the physics engine only - again, reducing computational expense drastically.

The best option with respect to the three key-points was found to be the latter.

2.3 Choice 3: Joints' Configuration and Interconnections

To recap, the problem at this point was to **find a configuration of MuJoCo primitives that approximate the string-like behavior of the spine.**
At first, it was considered to use prismatic joints -one for each vertebra and each disc- to emulate the sting-like behavior. However, this choice produced a "staircase-like" motion that wasn't satisfactory. A snapshot of such movement is shown below.

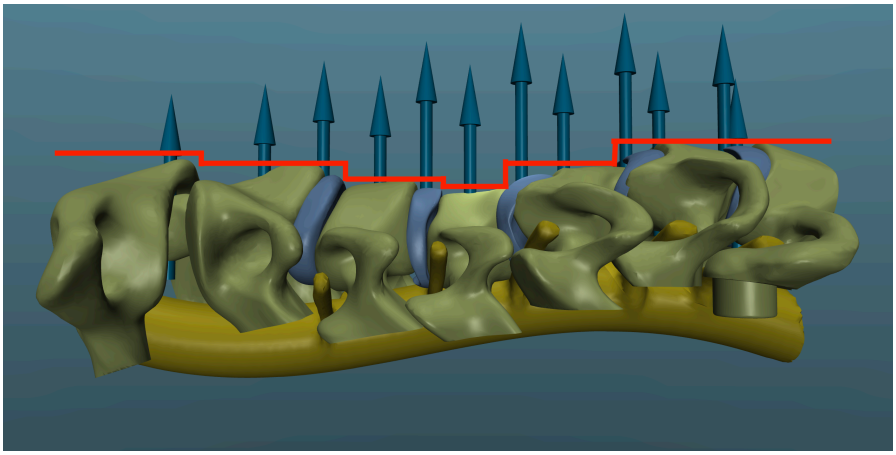


Figure V: Staircase-like movement

In red, we can see the approximated shape that is created by interacting with the spine, while the blue arrows represent the prismatic joints' direction of motion. This staircase-like movement is even more evident in simulation, as the spine dynamically rearranges but maintains the same kind of squared pattern.

To avoid this non realistic behavior, it was decided to use **a combination of hinge joints and tendons**. In fact, hinge joints are used to model each vertebra/disc rotation around the "string" normal axis. Of course, such joints are limited to allow only small-angle rotations, as for a real spine wouldn't allow great deformations without breaking. The tendons are added to "link" all the joints and merge all the single rotations into a smooth movement. In fact, tendons in MuJoCo are nothing but **global mobility constraints** imposed between the joints

of interest. This translates into a much smoother and more realistic behavior, as shown below. The movement is much more arch-shaped and the joints' axes are shown as blue spikes.

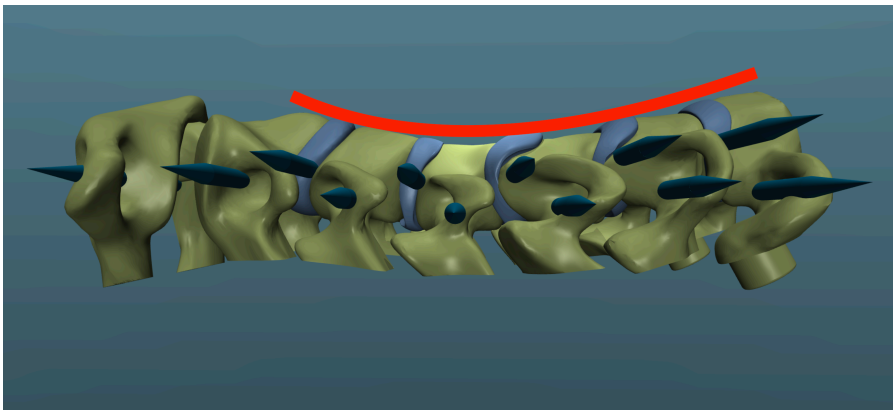


Figure VI: Arch-shaped movements

2.4 Choice 4: Parameters

At this point, the scene has a well defined structure that allows quite realistic simulation of the spine's string-like behavior. What can be done to improve the final version of the scene is to **tune the physical parameters** of the spine elements.

This can be done directly on the *.xml* file using MuJoCo's built-in physics engine's parameters. In particular, we can adjust (to make a few examples), the components' masses and inertia, as well as joints' elasticity, damping and limits.

However, notice that **several tuning paradigms can be made depending on the application of use for this scene**. For purely didactical purposes, one could accentuate the spine's mobility in order to allow larger curvature (which wouldn't be allowed, for instance, in a surgery). This can be easily obtained by imposing larger joint limits and longer tendons.
On the contrary, one could reduce spine mobility and adjust joint damping/elasticity to try to match the physical phantom's properties. This would instead be the right choice for creating a digital twin.

The "parameter tuning" done by the writer was in the direction of **compromise between the two possible applications**. On one hand, accentuate a bit the spine's mobility so to show the correct behavior and make the simulation enjoyable by non-experts. On the other, not exaggerate such mobility and add

elasticity/damping to make it more similar to the physical phantom.

Notice that moving the tuning paradigm towards the didactical purpose can be easily obtained via some in-simulation trial and error, which can be done using the MuJoCo application only.
Vice-versa, moving towards the digital twin requires to work in a virtual reality environment (possibly, with some haptic glove) and in close contact with the physical phantom. In fact, for this purpose one must try to match the real-life feeling of "deformation" and "hardness" of the physical spine with the feeling rendered by MuJoCo and the haptic glove. To have a realistic interaction between the spine and the user's hand, it's necessary to track its movements in the scene, thus one must use a VR environment to simulate the scene.

Before discussing the VR integration of the scene, let's briefly discuss the final version of the simulation (which again can run on the MuJoCo application with no external script).

3 Final MuJoCo Scene

The final scene involves all the elements that compose the physical phantom. In particular, the writer decided to split the components into five groups [1,2,3,4,5], since each group can be enabled/disabled by just pressing one button. As a result, one can remove from the scene some "layers" to better visualize the inner ones.

Figure VI: Group enable/disable

3.0.1 Group 1: Frame

This group only contains the structural elements of the physical phantom. Therefore, these components would not be present in a real human patient, but are

strictly necessary to hold the physical phantom in place.

3.0.2 Group 2: Various

This group is the most variegated one as it includes organs (such as the esophagus and thyroid), membranes, muscles and nerves. As for the muscles, it includes all those that "surround" the human throat and support the spine. In the future, it would be interesting to make them deformable (once again, impossible as for now due to hardware limitations and computational overload). They're grouped together with other "background" organs as they're not of interest for spinal surgery and MuJoCo only allows 6 total groups for its scenes.

3.0.3 Group 3: Veins

This group is made of arteries and veins. They all could be elements of interest in a surgery, as damaging a blood vessel is something to avoid at all costs. However, at the moment they're all left as non-interactible objects. A relatively easy task to add would be to implement some kind of contact detection, whereas "damage" and perforation simulation would be a much more complicated problem.

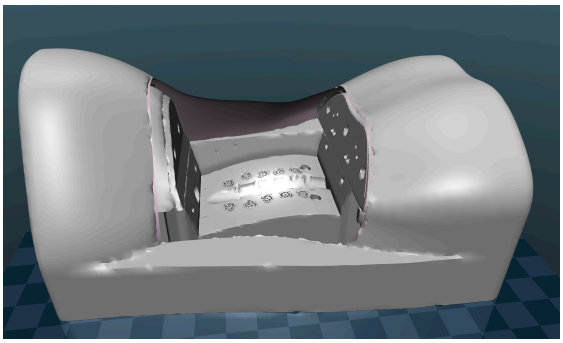
3.0.4 Group 4: Medulla

This group is only made by two objects: inner and outer medulla. Since it's an element of interest for spinal surgery, it can be made interactible in a similar manner as for the blood vessels. However, for complicated problems such as needle insertion, a lot more of study on the Software must be done. Notice also that **medulla is not deformable**, hence it doesn't match the spine deformation. For small spine "arches" this is barely an issue, but for greater mobility it can disrupt the authenticity of the simulation.

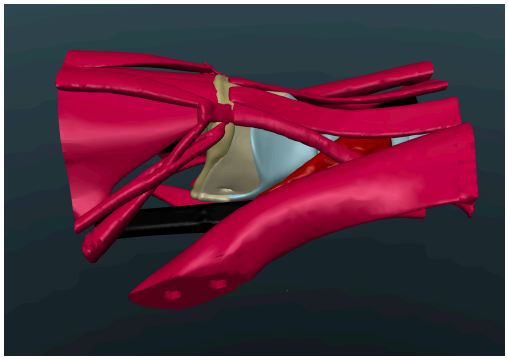
3.0.5 Group 5: Spine

Finally, the last group is made by vertebrae and discs. As previously discussed, each element possesses a rotational DOF and is linked to the adjacent vertebra/disc by a tendon. The resulting structure behaves like a vibrating string fixed on both ends. In particular, the first and last vertebrae (C1, C7) are fixed in place using a **welding constraint** (MuJoCo built in functionality).

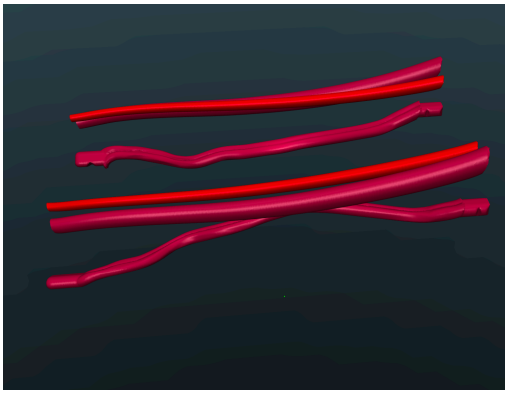
In the next page, images of these groups are reported.



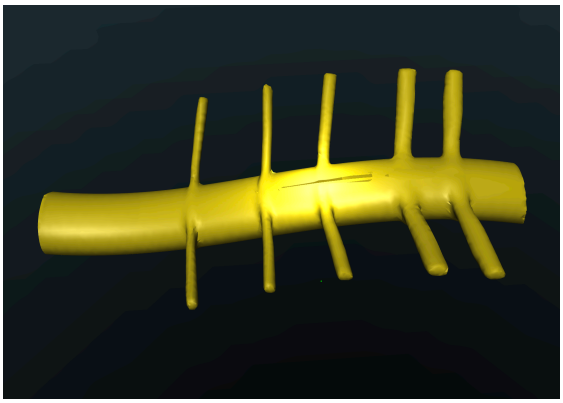
(a) **Group 1:** Frame



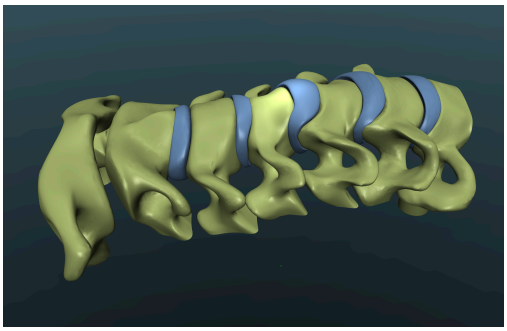
(b) **Group 2:** Various



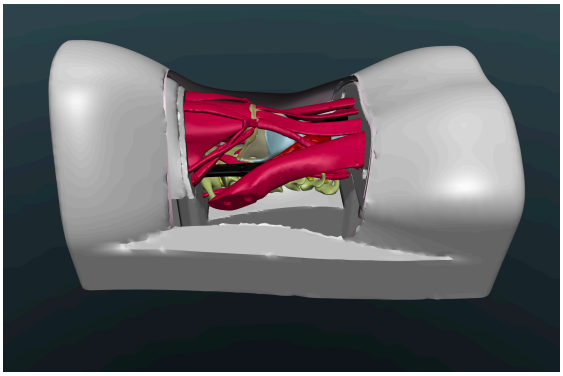
(c) **Group 3:** Veins



(d) **Group 4:** Medulla



(e) **Group 5:** Spine



(f) **Full Digital Phantom**

Figure VIII: Visualization in MuJoCo of the Digital Phanom

4 VR and Haptic Glove Integration

Taking advantage of the writer's previous work, it was relatively easy to add the digital phantom to a bigger scene. In particular, the phantom was inserted in a scene that integrated Virtual Reality (thanks to Oculus Rift S and openxr) and the WeArt Haptic glove. This allows to replicate the user's hand and fingers' movement on a virtual hand that's inserted in the VR simulation. Moreover, thanks to the haptic glove, the user perceives the contact forces with virtual objects on his fingertips.

Of course this kind of simulation can only run thanks to an external (Python) script that also handles all communications between the various devices and environments. It's clear that the computational power required is substantial and not all hardware can perform such operation. That's why the work done in simulating the spine using only MuJoCo primitives is important in order to not weigh down even more the computational load. A snapshot of the integrated VR environment is shown below.

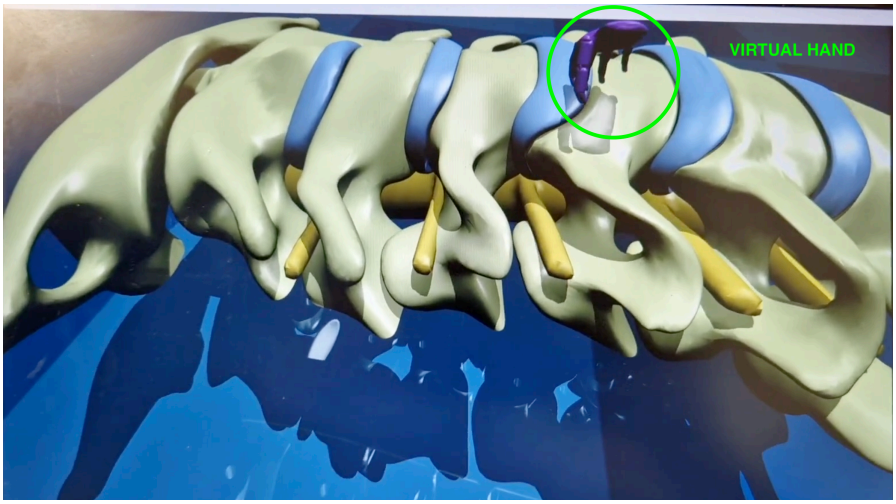


Figure IX: VR Integrated Simulation

In this simulation, it was chosen to rescale the spine by a factor 10x, for purely didactical purposes. The virtual hand can be seen near the top left corner and caused the deformation (arch) visible in the figure.

5 Personal Considerations and Possible Further Developments

The main challenge with this kind of technology is finding a balance between the three key-points (computational demand, "user-friendliness" and realism). The first one represents the biggest factor to take into account, as a non-smooth simulation disrupts the whole experience, especially for VR integrated experiences. I've tried to work with this idea in mind, often making compromises to keep the scene "light".
That's one of the reasons I discarded the idea of simulating the spine by solving differential equations and assigning joint references with an external Python script. However, this could still be a possibility, as long as one can make a good enough mathematical model.

The MuJoCo model can surely be improved depending on the use, but I believe it's a solid base for further developments.

If the main goal is to create a digital twin for the physical phantom, then it is necessary to start working side by side with it and tune all the spine's parameters trying to match the phantom's behavior. This process will surely take time and will be mostly a matter of trial and error. With enough attempts, I'm confident that good results can be obtained.

If the final goal is creating a didactical tool instead, one can work with scaling the model (as in the VR experience created) and "exaggerate" spine mobility, as well as making more elements interactible.

An interesting idea could be to integrate neural networks in the simulation of deformable objects (as shown by Doctor De Santis). One could start for instance from the medulla, which at the moment doesn't deform with the spine but is instead static. It's surely an easier task than simulating deformable closed surfaces (such as kidneys) as medulla could be approximated to a vibrating string as well. I'm not sure how much work is necessary to do such thing, but it could be worth investigating.

In the hypothesis of having a surgeon trainee operating on the physical phantom, instead, one could use a camera and computer vision techniques to reconstruct the phantom spine pose, and then assign directly the joint references from that information. This would eliminate the need for simulating the "vibrating cord" dynamics and allow for a 1 on 1 matching of the exact pose of each vertebra/disc.

However, this is a case of application for which the simulation is used as an additional tool to the "in-real-world" training and wouldn't work for "virtual" training.