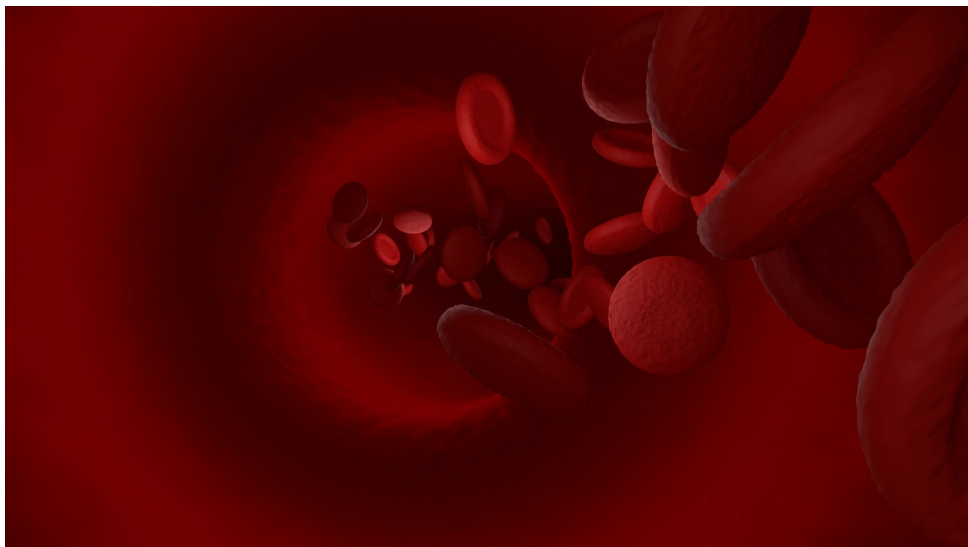


PART 1 - INTRODUCTION

Hello everyone!

In this tutorial, created using Blender version 3.3, we will see how to create the animation of some red blood cells moving inside a blood vessel.



In the tutorial, we will learn **how to model** the blood vessel and red blood cells, **how to generate** the red blood cells with an **Emitter-type particle system**, and how to make them **follow the path** of the blood vessel using a **Force Field**.

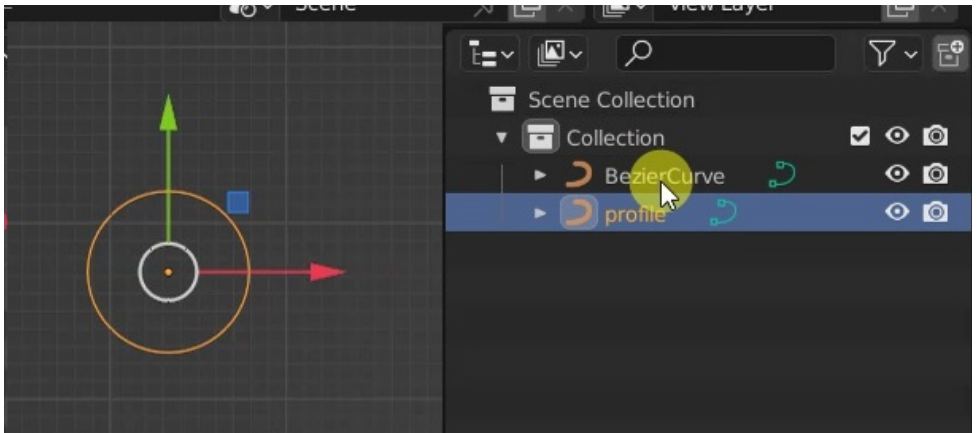
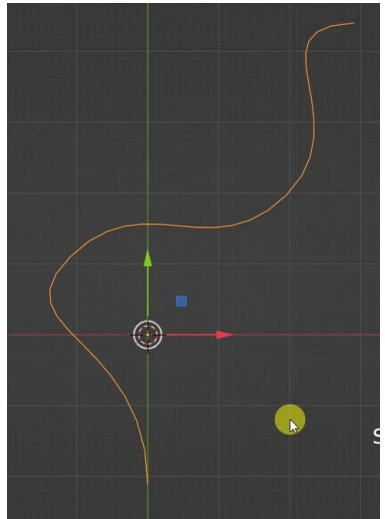
PART 2 - BLOOD VESSEL MODELING

Let's start with the modeling of the blood vessel in a completely empty scene.

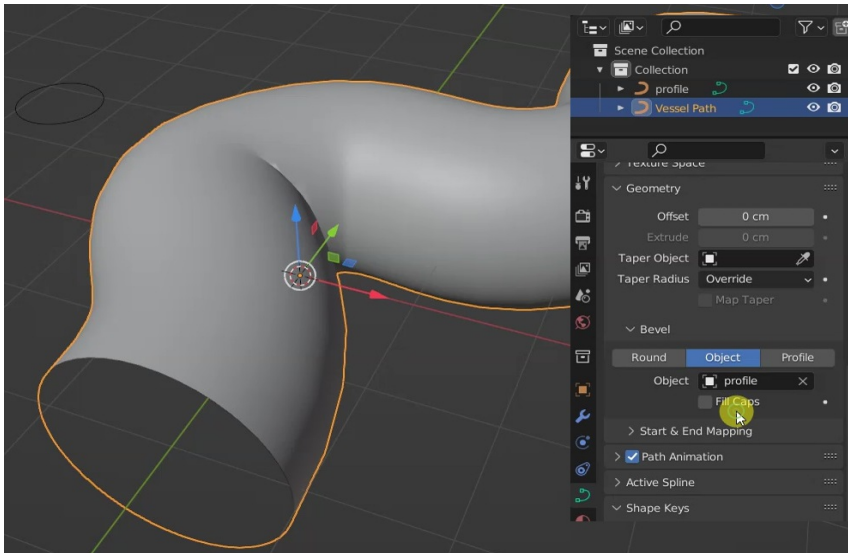
First, in a Top view, preferably orthographic, we **insert a Bezier curve**, which we model by adding and moving a couple of **control points** in Edit Mode.

This curve represents the path of the blood vessel, and for our purposes, we can keep it in the 2D plane (that's why I'm framing the scene from above, with an orthographic view).

After modeling the path, we go back to Object Mode and **insert a Bezier Circle** into the scene: this will represent **the profile of the path**, which we will keep circular; we will disturb the inner walls of the path later with the **Displace modifier**.



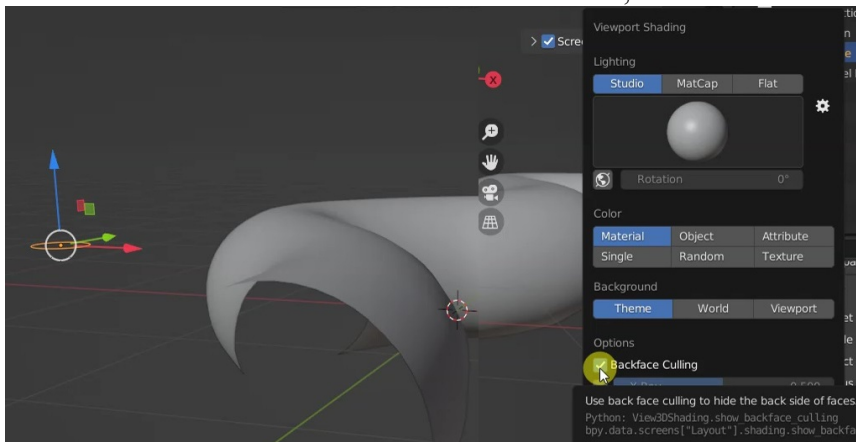
To set the **Bezier Circle** as the **profile** of the **Bezier Curve** (which, by the way, we rename to **Vessel Path** for clarity), we have to select **Vessel Path**, open the **Object Data** tab within the Properties editor, then access the **Geometry - Bevel - Object** section and specify the **Bezier Circle** we just created as the **Bevel object**.



If the profile appears too large or too small, we can resize it simply by scaling the Bezier Circle; as mentioned, there is no need to modify the Bezier Circle's profile in Edit Mode because the circular profile works just fine.

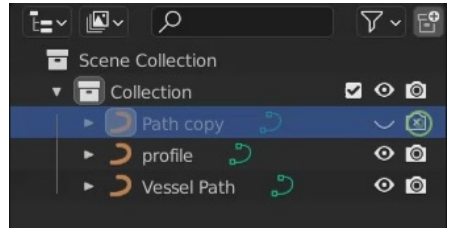
In my case, **the blood vessel appears invisible when viewed from the inside**: this is because the ***Background Culling*** option is enabled in the ***Viewport Shading*** menu.

This option hides faces that do not have their normals facing the viewer; it is an option that can be useful in some cases but not in this one, so I disable it.



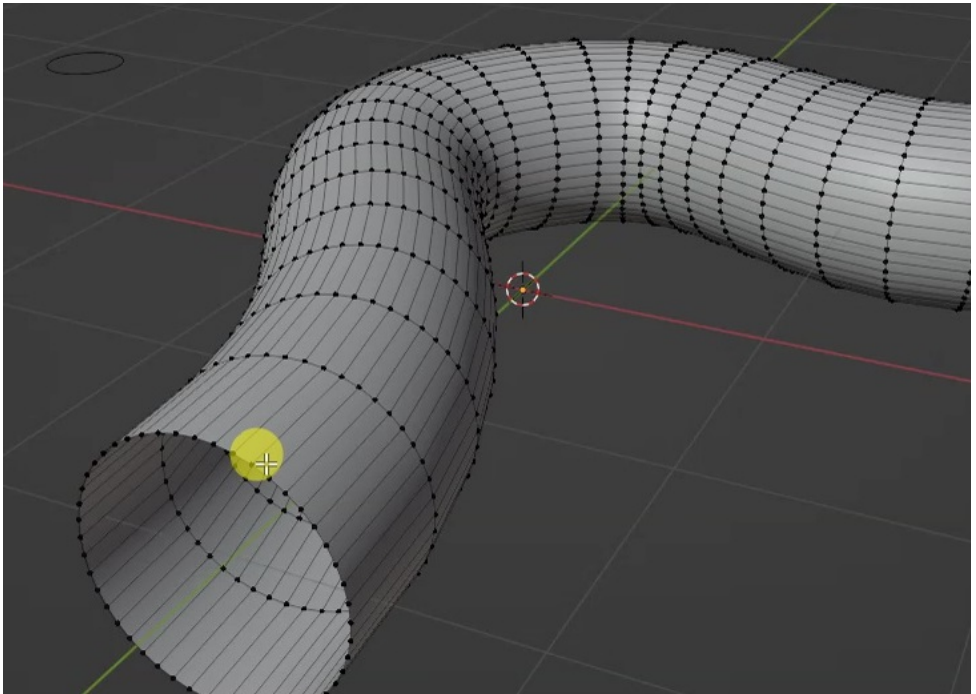
Let's make a copy of the path (with **SHIFT+D** and **Enter**), calling it, for example, **Path Copy**, and temporarily hide it; this copy will be useful for two reasons:

- for safety, as a copy of the path, because we will soon transform it from a Curve to a Mesh;
- and as a copy of the original curve so that we can use it later as a path for the red blood cells to follow.

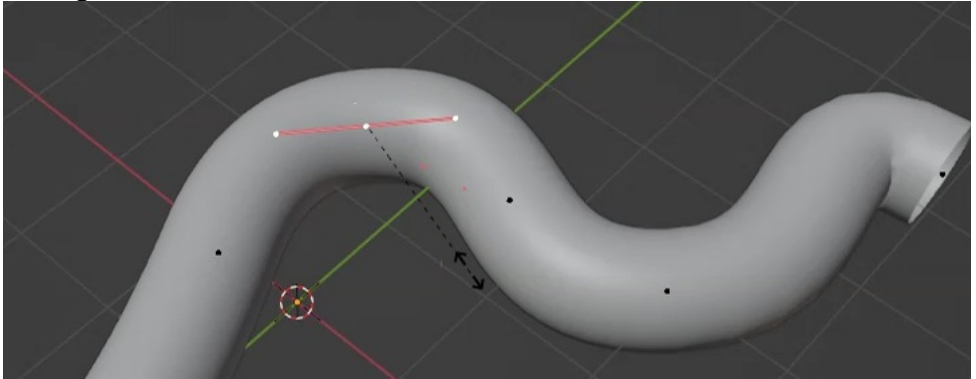


The path needs to be converted to Mesh because this way we can apply a Displace modifier to it, which will disturb the blood vessel walls, making them visually more interesting.

The conversion can be done with a **right-click and Convert To Mesh**, but I want to point out something: in my case, **the vertices of the circular profile are much denser** than the segments that make up the path.

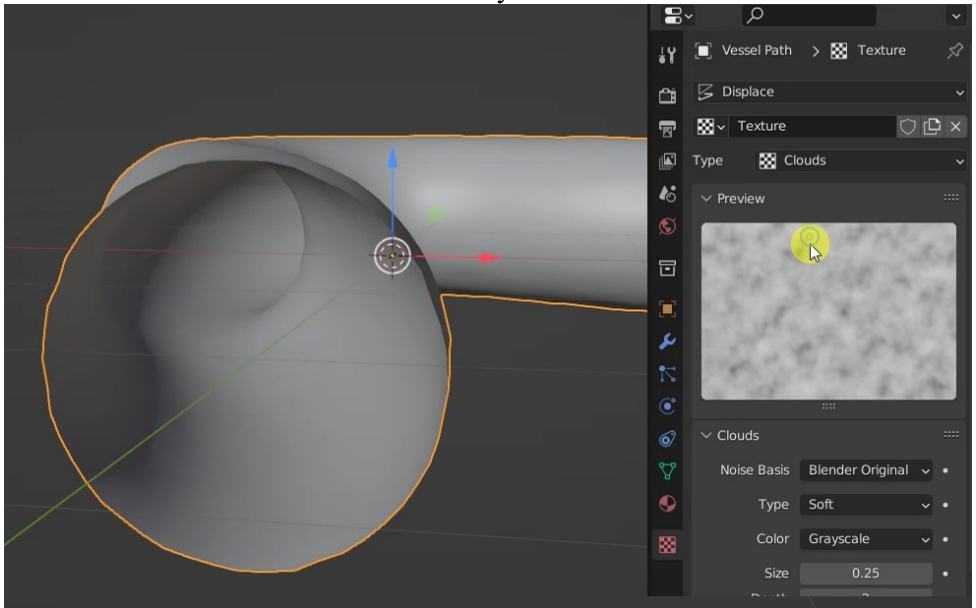


Since I prefer to have faces as square as possible, I undo the conversion operation (**CTRL+Z** in Object Mode), then **insert intermediate control points** where the original control points are too far apart and have too long handles, which produce rectangular faces.



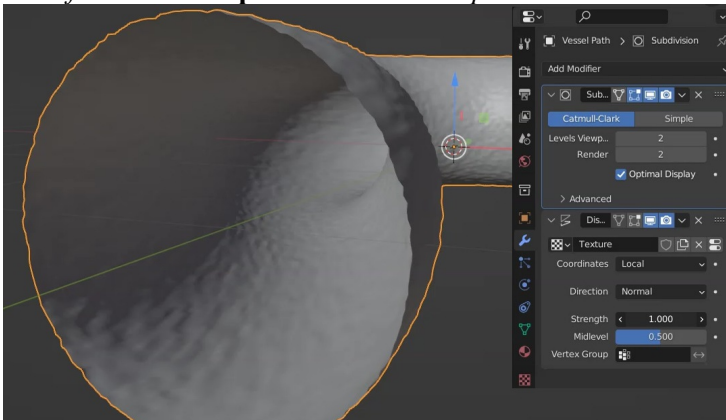
I then perform the **conversion to Mesh**, as previously mentioned, with a **right-click** and *Convert To Mesh*.

Next, I add a **Displace** modifier to the object, specifically setting a **Clouds** texture with a low scale to introduce many distortions.

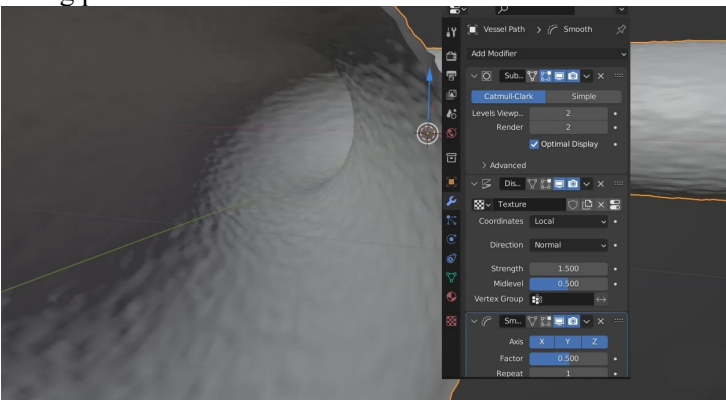


However, the effect is disappointing: this is due to the fact that **the mesh obtained from the conversion is not "dense" enough**, i.e., it has few vertices and faces to deform; to solve this problem, I add a ***Subdivision Surface*** modifier to the object, with at least 2 subdivisions.

The ***Subdivision Surface*** modifier needs to be moved above the ***Displace*** modifier: the order is important because the mesh is **first subdivided** with ***Subdivision Surface*** and **then perturbed** with ***Displace***.

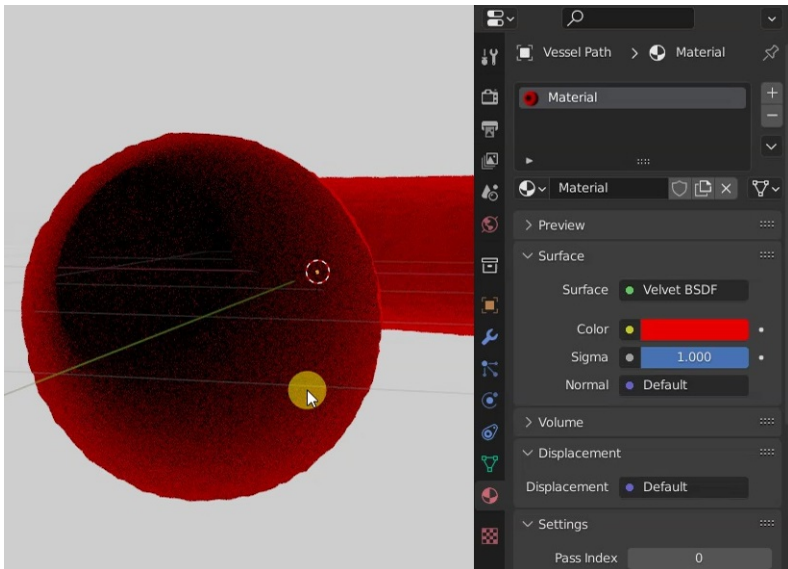


So, let's modify the ***Size*** values of the Texture and the ***Strength*** of the ***Displace*** modifier until we achieve a result that pleases us; if necessary, we can also **try different Textures for *Displace***, or we can add a ***Smooth*** modifier as the last modifier in the stack, to soften the perturbations a bit and complete the blood vessel modeling phase.



PART 3 - MATERIALS, LIGHTS, AND INITIAL CAMERA PLACEMENT

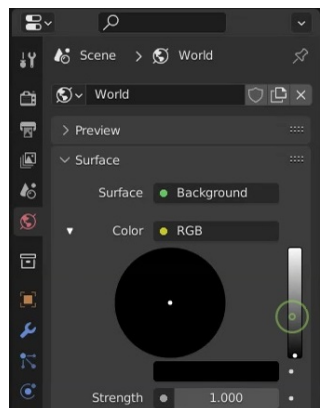
Once this work is completed, **let's assign a material to the blood vessel**; in my case, to achieve the look you saw in the tutorial preview, I use a **red Velvet material** because this material reflects light in a particular way: the result, in this case, doesn't need to be photorealistic, so a softer, more blurred look is fine.



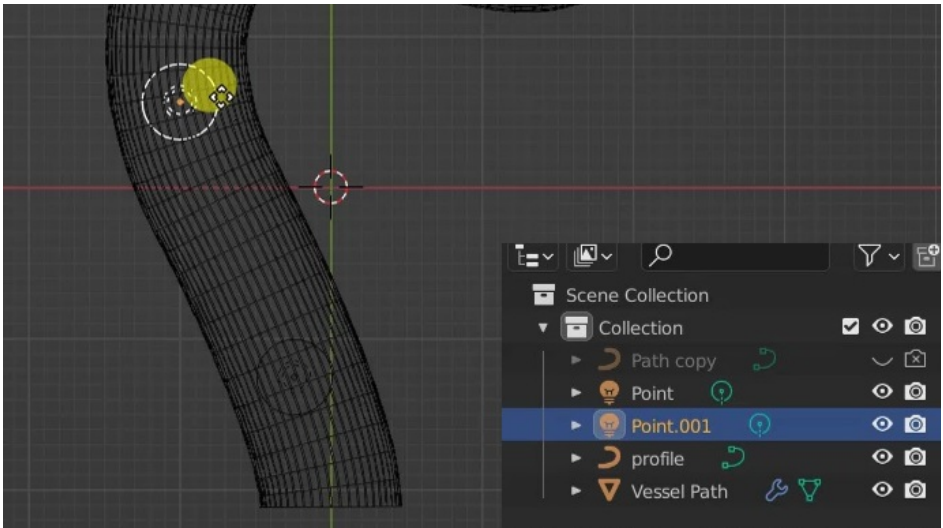
However, to have a preview of the result, we must first perform two operations:

- set the **virtual universe's background to black** (from *World - Background - RGB Color*)
- and insert at least a couple of **Point Light** sources inside the blood vessel.

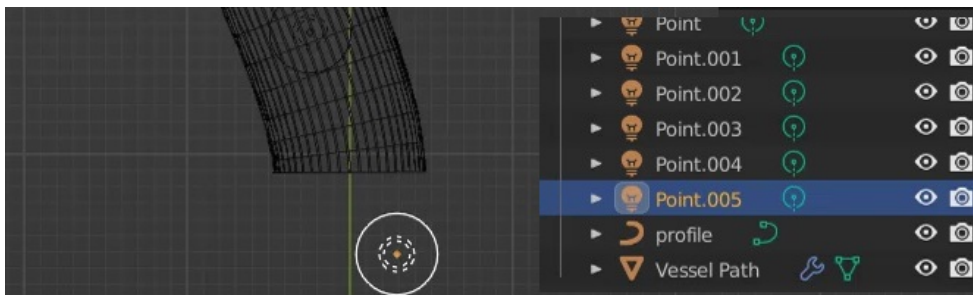
Inserting the Point Lights can be easily done from a Top Ortho view; I suggest placing the light sources near the curves and initially creating only one, which can then be **duplicated using ALT+D** to



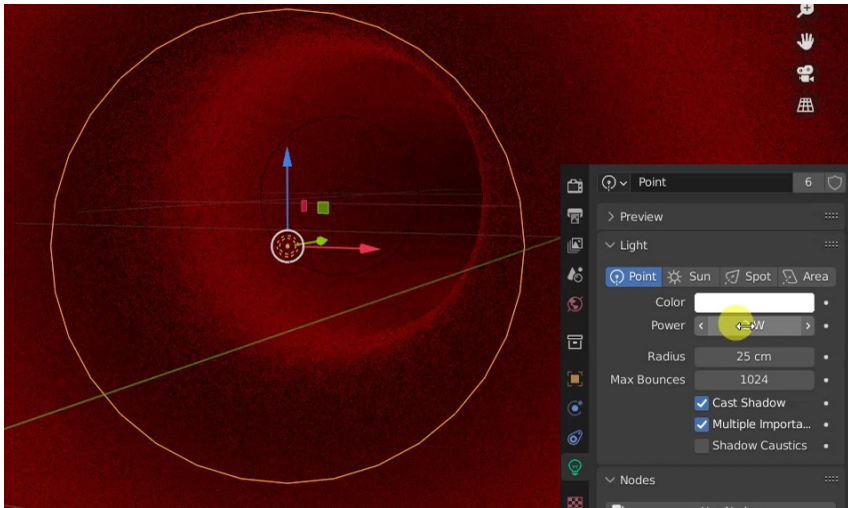
obtain **linked copies** that will copy the first one's parameters: this can be useful because initially, we won't know the intensity to give to the light sources and will have to experiment, but with the linked copy, we can modify the intensity and color of just one light source and find the same settings applied to the linked copies as well.



Place a linked copy of the light source at the entrance of the blood vessel, otherwise, our red blood cells will be too dark at the beginning.



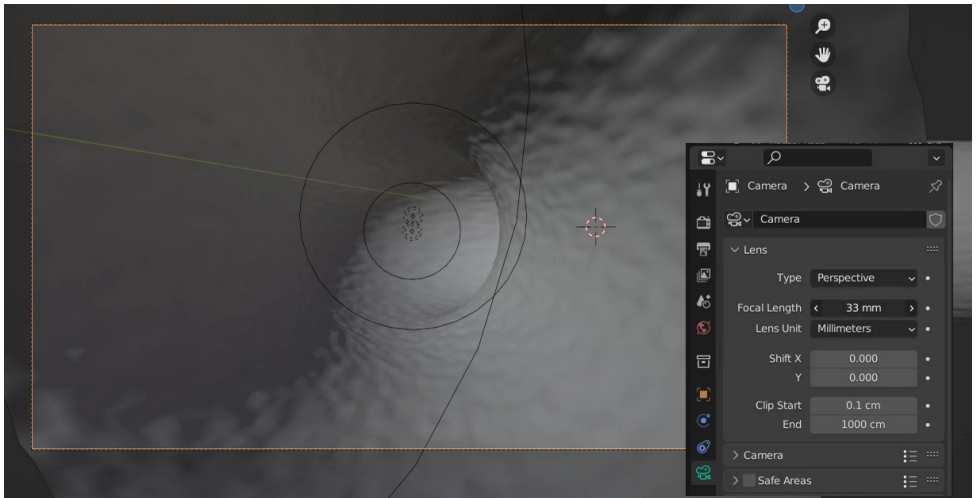
We can view the rendering preview with **Z - Rendered** in the **3D Viewport**, allowing us to set the light parameters until we achieve the desired result.



At this stage, we can also insert and position the camera that we will later use for rendering; specifically, in Object Mode, insert a **Camera** object into the scene with **SHIFT+A** and **Camera**, then position ourselves in the 3D Viewport **to define the initial framing** of the sequence, and open Blender's search box (which, in my case, can be accessed with the spacebar) and look for the **Align Camera to View** option (the shortcut is CTRL ALT NUMPAD 0).



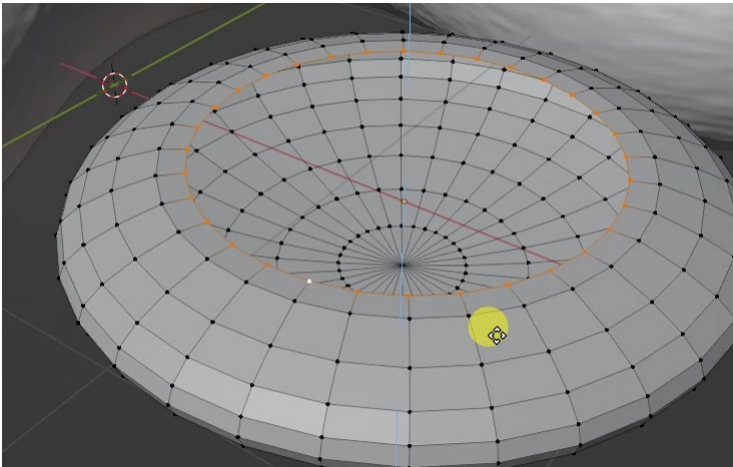
Adjust the camera's position and, if necessary, change its **focal length** in the **Object Data** tab, for example by lowering its value to achieve a wider, wide-angle framing.



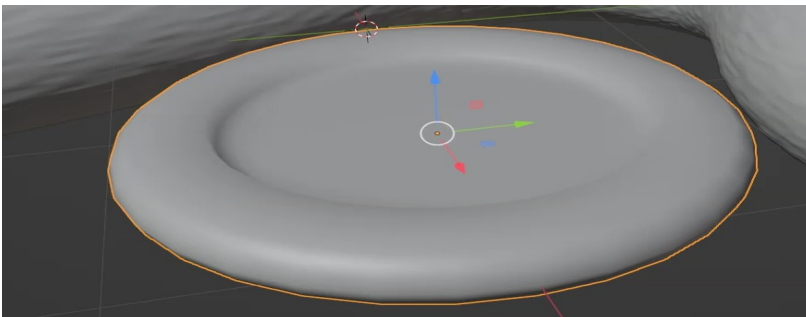
PART 4 - RED BLOOD CELL MODELING

Once the materials, lights, and camera setup is completed, switch back to the **Solid** view mode of the 3D scene and move to an empty area **to model the red blood cell prototype**.

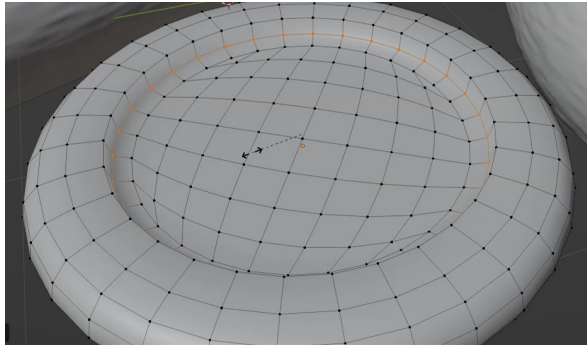
There are several ways to model this object; the choice is vast; in my case, I am creating a **UV Sphere**, flattening it at the poles; plus, I am **deleting the pole vertices** and filling the empty rings created with **GRID FILL** in Edit Mode.



I am performing these operations because I intend to apply a **Displace modifier** to the red blood cell as well... however, before that, I change the shading to **Shaded** in Object Mode because the default **Flat shading** is too faceted!

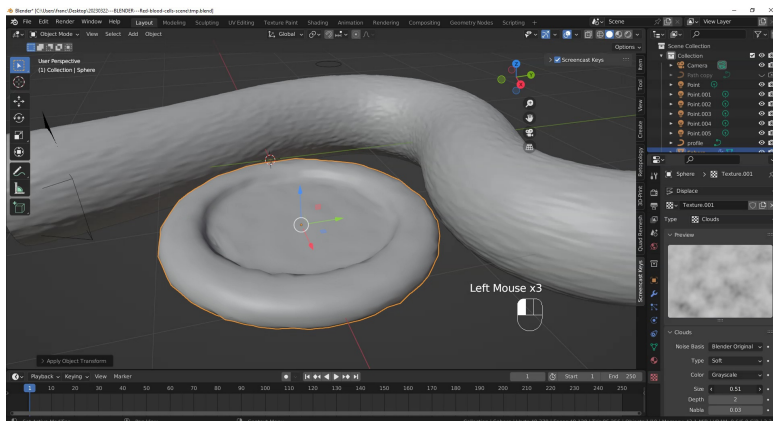


I further model the red blood cell, flattening it near the poles; I perform these operations by selecting the vertices I want to model and then moving or scaling them along the **Z**-axis (vertical) until I achieve the desired result.



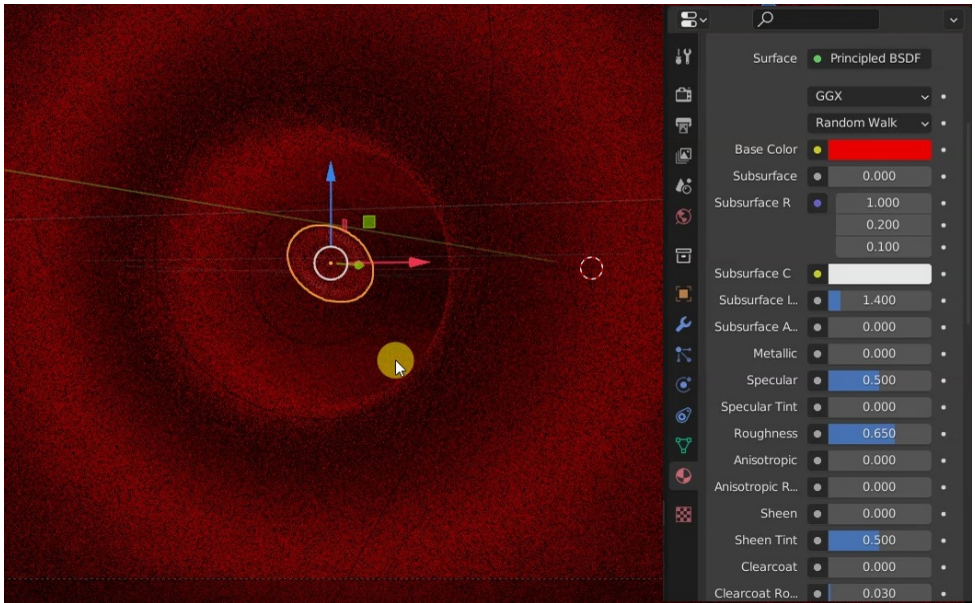
Regarding geometry distortion, perform the same operations as with the blood vessel, i.e., add and set the **Subdivision Surface** and **Displace** modifiers, with the difference that this time, probably only one subdivision for **Subdivision Surface** will be enough.

If the deformations introduced by **Displace** are not well visible (or, on the contrary, are too strong), even when significantly varying the Texture's Size parameter, try applying the object's scale transformations with **CTRL+A** and "**Apply Scale**", especially if you resize the object. This consideration also applies to the blood vessel, indeed.



Once the modeling phase is complete, assign a **Material** to this red blood cell prototype; at this stage, temporarily move the red blood cell prototype inside the blood vessel to a well-lit point to preview the final effect in Rendered view mode.

This time, I choose a **Principled BSDF Material** with a slightly dark red color and a **Roughness** of around 0.6 to provide a subtle specular reflection on the object.

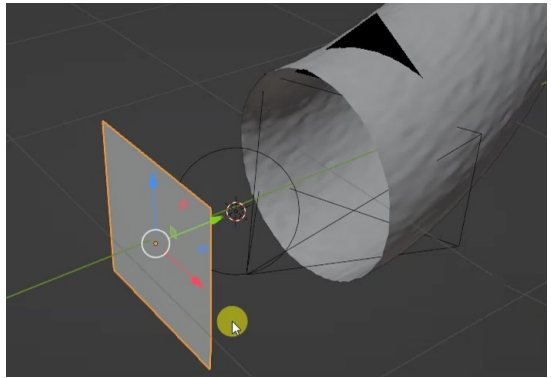


After this step, move the red blood cell back outside the blood vessel and return to **Solid** view mode to focus on creating the particle system that will emit the red blood cells.

PART 5 - PARTICLE SYSTEM CREATION AND FORCE FIELD

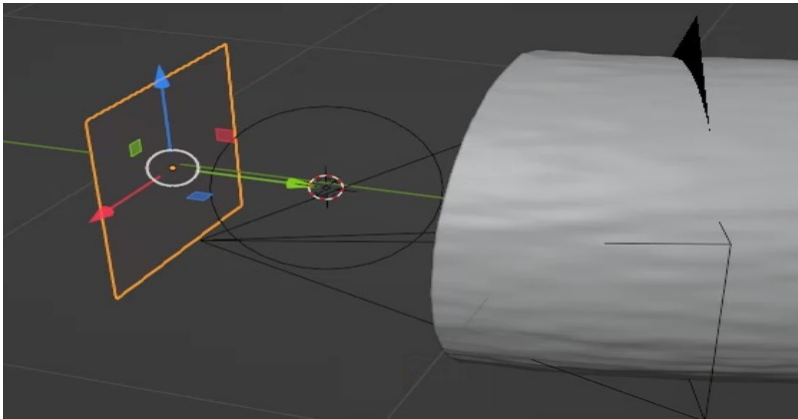
To emit red blood cells as particles of a particle system, we **first need an emitter object**.

In our case, a *Plane* will work just fine, so insert one into the scene and, most importantly, orient its upper face to point towards the entrance of the blood vessel, as that's the direction we want to emit the red blood cells.



To associate a particle system with the newly created *Plane*, **open the *Particle Properties* tab** of the object and click on the "+" button.

By default, the newly created particle system will be of the *Emitter* type, which is what we need.



We don't want to create too many particles, but we want to make sure they are copies of the red blood cell prototype, have random rotation, are emitted throughout the animation's duration, and are present until the animation's end. The latter point needs clarification because, as we will see, **particles have a lifespan** that we will set accordingly.

As for **the number of particles**, we will test as we evaluate the animation, but for now, let's set them to **100** in the *Number* field of the particle system tab.

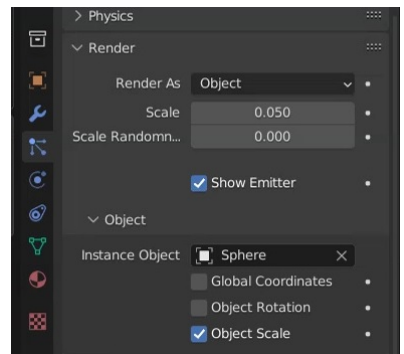
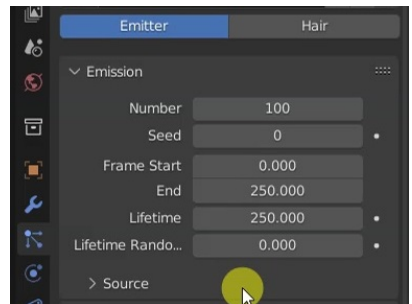
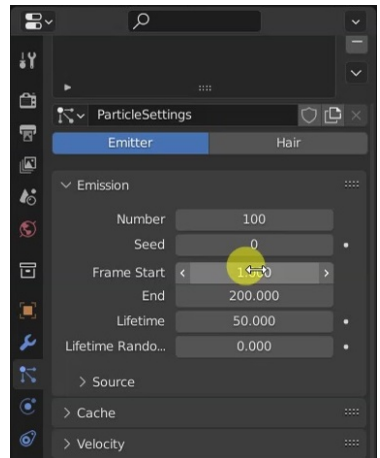
Regarding the animation's duration, I am leaving Blender's default 250 frames (equivalent to 10 seconds of footage at 25 frames per second), so we can set:

- Frame Start 0
- End Frame 250
- Lifetime 250

To make the new particles **instances of the red blood cell prototype**, open the *Render* tab of the particle system and change the *Render As* field from *Halo* to *Object*, then specify the red blood cell as the *Instance Object* further down.

Now, when starting the animation, we will notice some things to fix in the generated particles:

- they have an inadequate size;
- they fall into the void;
- they all have the same orientation;
- they are not projected towards the blood vessel.

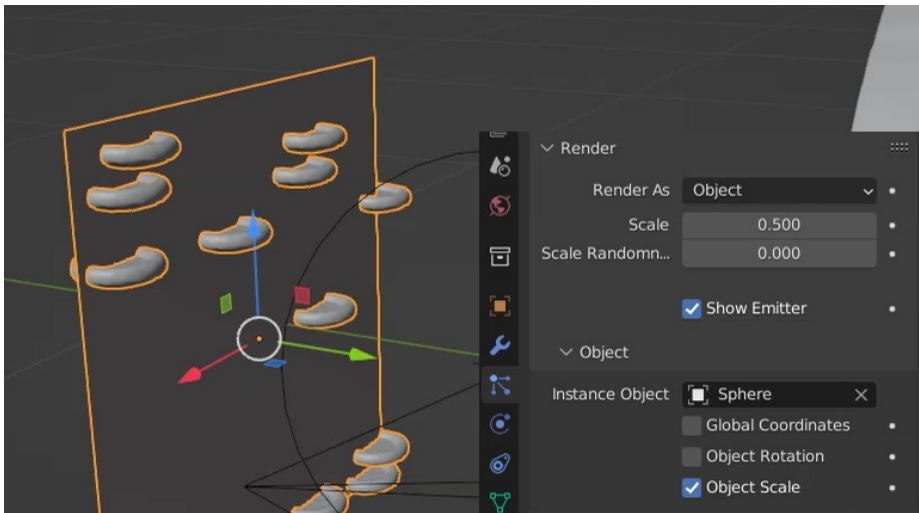


We will solve all these problems one by one, starting with the simplest: **removing gravity**.

Open the **Field Weights** tab of the particle system and set the **Gravity** value to **0.0**. Now the particles will no longer fall into the void, as we can see by positioning ourselves at frame 1 of the **Timeline** and starting the animation again.

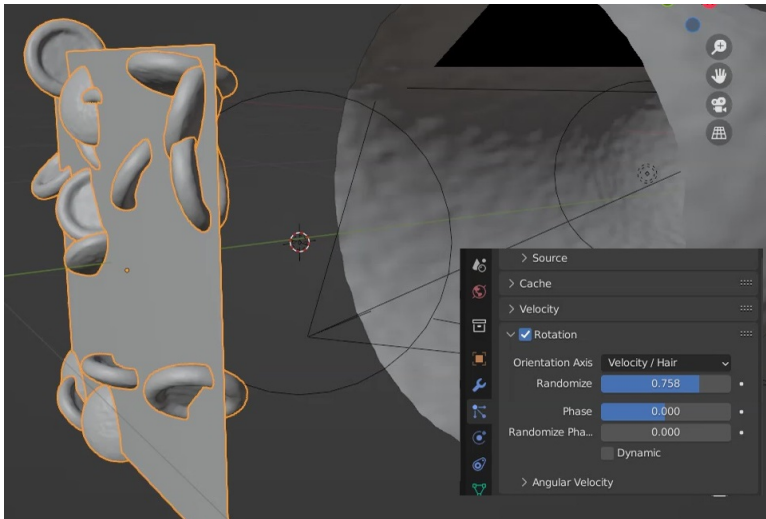


The size of the particles can be modified from the **Render** tab of the particle system by varying the **Scale** parameter. I am not sure if red blood cells can have different sizes, so I leave the **Scale Random** value, which introduces random size variations, at 0.0.



Note that we can still modify the red blood cell prototype, both in Object and Edit mode, observing the changes also in the particles, as these are instances of that prototype.

For the **particles' rotation**, we can set an initial random rotation by first activating the **Rotation** section of the particle system and then setting a **value greater than 0.0** in the **Randomize** field within this tab.

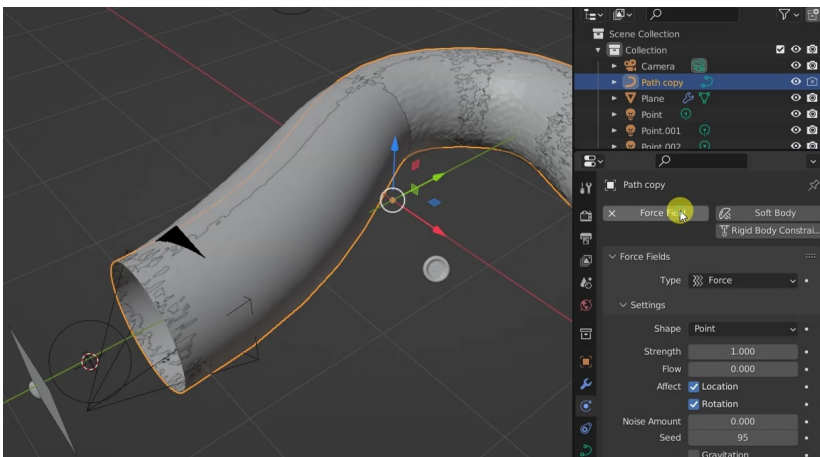


We can observe the generation of randomly oriented particles by returning to the first frame of the Timeline and starting the animation.

The last issue to solve is **moving the particles along the blood vessel**.

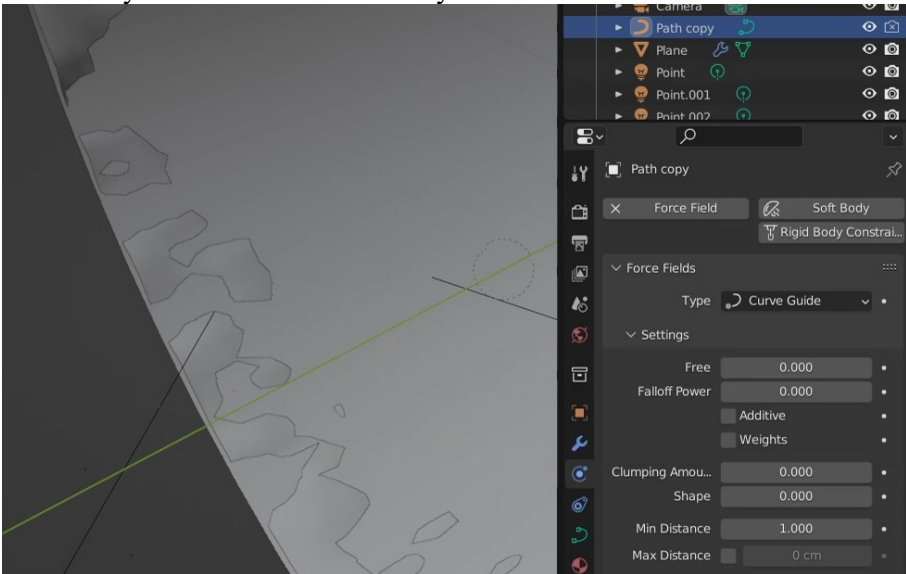
Do you remember the copy of the blood vessel, called ***Path Copy***, of the **Bezier Curve** type, hidden and superimposed on the blood vessel (which has been transformed into a Mesh)? Well, it's time to use it as a path for the particles.

Select ***Path Copy***, open the ***Physics*** tab in the **Properties** editor, and add a ***Force Field***.

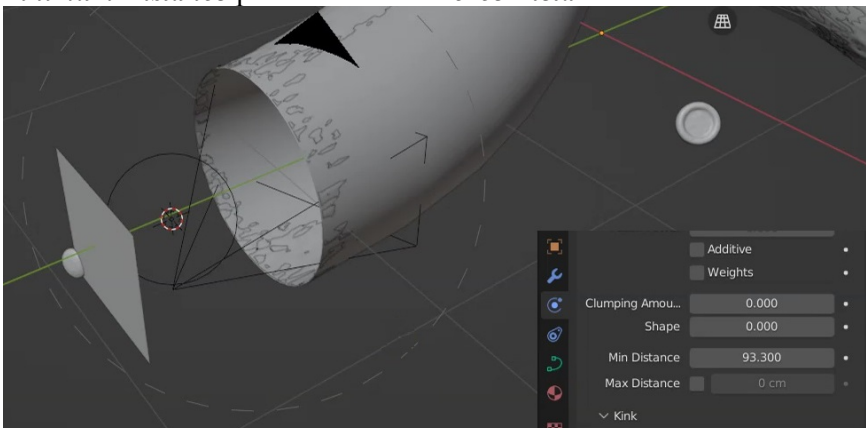


Change the **Force Field** type from **Force** to **Curve Guide**; we won't notice anything special, also because **Path Copy** is currently **invisible**, so make it visible again and **look for a dashed circle** at the entrance of the blood vessel.

This circle indicates the area of influence for the particle system: the particles will be affected by the Force Field when they are in this area.



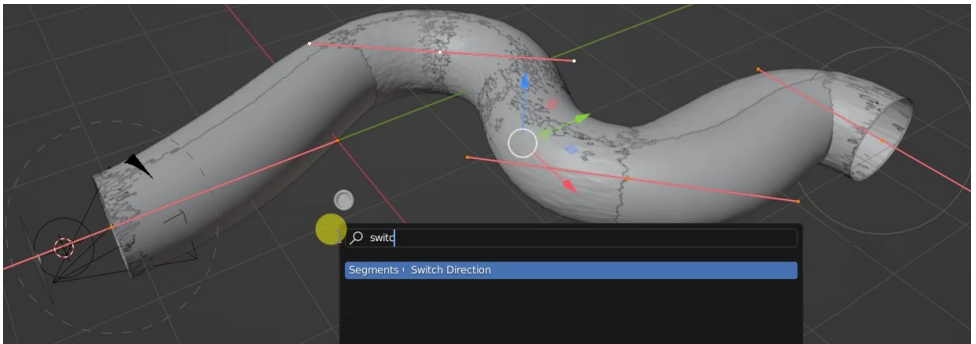
We must then move the **Plane** closer to this area and, above all, enlarge it, so as to include the generated particles; to carry out this operation, we change the value of the **Minimum Distance** parameter in the **Force Field** tab.



We return to **frame 1** of the *Timeline* and start the animation to observe the result.

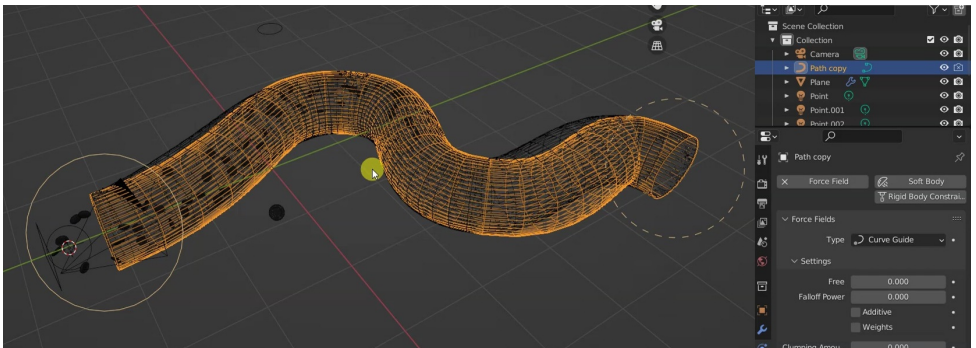
If the particles were to follow **a path different from that of the curve**, the reason could lie in the fact that **Bezier** curves have a **beginning and an end**: probably, what we consider as the entrance of the blood vessel is, in reality, **the endpoint** of the curve!

To solve this problem, we switch to Edit Mode, select all the control points of the curve, and **reverse their direction** by searching for the "*Switch Direction*" command in the Blender **search box**.



We return to Object Mode, reposition ourselves at frame 1 of the animation, and click on the *Play* button to observe the obtained result.

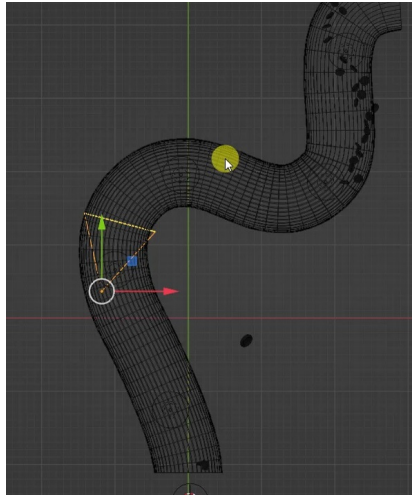
If the result is the desired one, we **hide the Path Copy** object again.



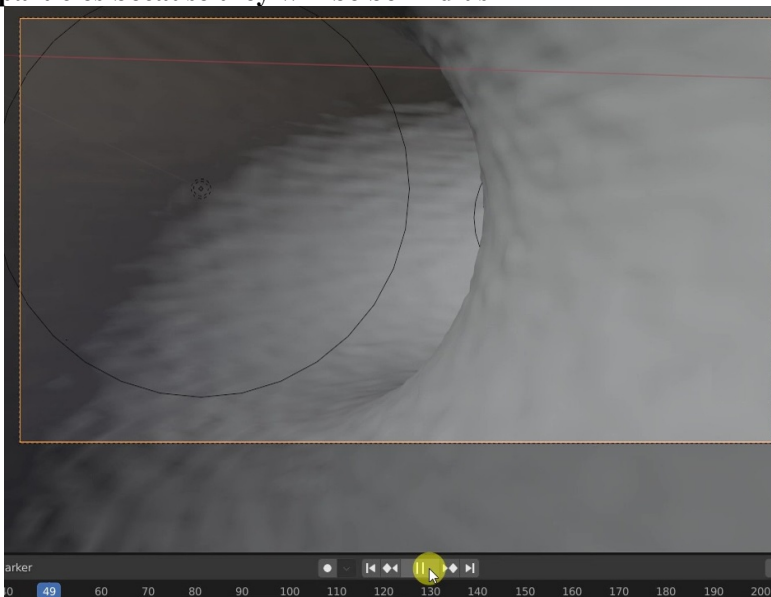
PART 6 - CAMERA ANIMATION AND FINAL RENDERING

Now we can also preview the animation from the perspective of the virtual camera and eventually change the number of red blood cells generated by our particle system or other parameters, such as the intensity of the *Displacement* effect or the dimensions, according to our needs.

Finally, we can **animate the movement of the virtual camera within the scene**; this can be done easily by observing the scene from a *Top Ortho* view and inserting keyframes for the position and rotation of the camera inside the blood vessel, depending on the animation we want to achieve, and verifying the result from the camera's point of view in the 3D View...



... and we notice that there is a problem! If, in fact, we move the camera to cover most of the blood vessel in the 250 frames of the animation, then **we will never see the particles because they will be behind us!**



There are several ways to solve this problem; one of these is **to emit the particles before the animation starts**, for example by setting **-100** in the **Frame Start** field of the particle system, but be careful: in this case, the **Lifetime** value must be **350**, otherwise the particles will disappear at frame 150 of the *Timeline*!

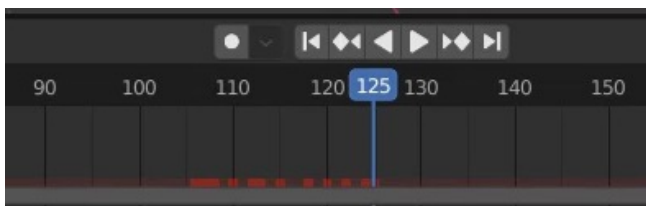
After making these changes, review the animation in the 3D Viewport to ensure that everything is OK this time; at the end, we can start rendering the animation.



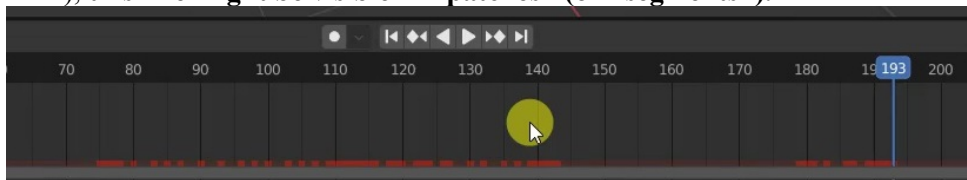
PART 7 - NOTES ON CACHE AND BAKING OF THE PARTICLE SYSTEM

Before closing this tutorial, a small observation on **the caching of the particle system** and some **issues** that may arise during rendering, especially after having done various intermediate renderings to display the state of the scene at various moments before proceeding to the final rendering of the animation.

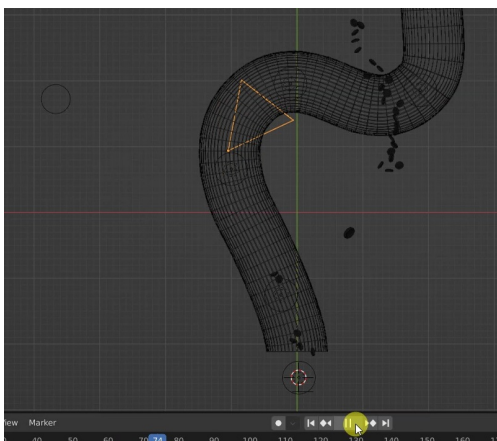
In these cases, in fact, you might see **a red line appear at the bottom of the Timeline**, indicating that Blender has **calculated and cached the animation** up to that point.



The worst thing is that sometimes, after several rendering previews (in the 3D view), **this line might be visible "in patches" (or "segments")**.

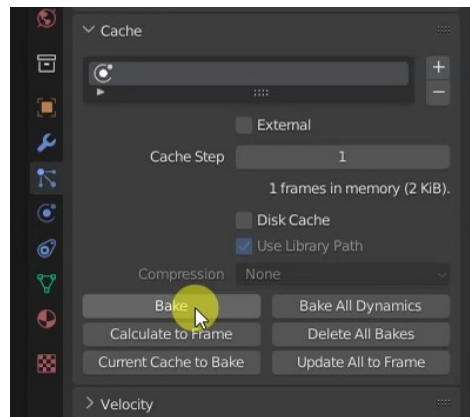
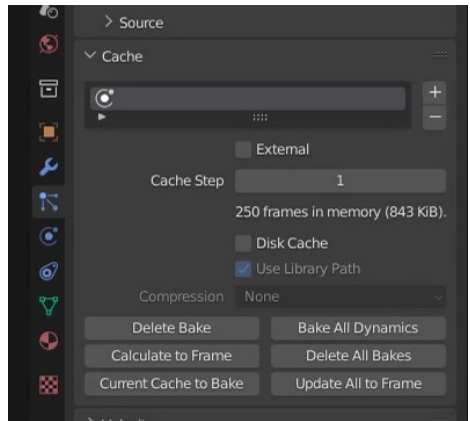


The problem is that, in this case, Blender would have some correct frames ready, as we want them, while other frames would be calculated in other ways, producing incorrect results: in certain frames, you might see the particles in the right place, while in others you might see them in incorrect configurations or not see them at all!

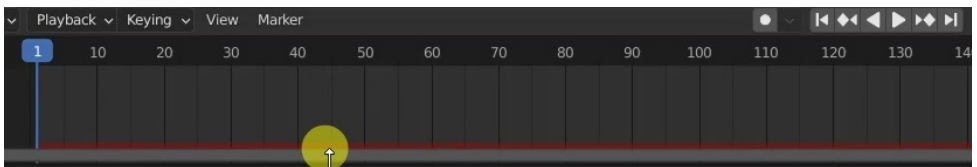


What you need to do before starting the animation rendering is:

- go back to the first frame of the animation;
- access the **Cache** tab of the emitting **Particle System**, where you will find information on the frames available in memory;
- click on the + symbol to add a new cache (which will initially be empty) and then click on the - symbol to delete the previous cache;
- finally, **click on the Bake button** to calculate and cache the entire correct particle simulation.

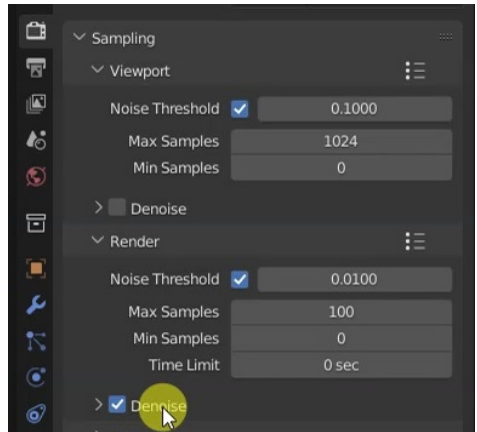


At this point, **the red bar should cover the entire Timeline**, indicating that all frames have been **calculated and correctly stored in memory**.



Now, starting the animation rendering (from **Render - Animation**), we should not have any nasty surprises in the file or files produced...

... but, as a very last note, I remind you to **activate the *Denoise* for Rendering in the Render Properties tab**: with these materials and this lighting, it is very useful and will allow you to render the animation with a reduced number of **Samples** (which, in my case, are just 100).



PART 8 - RECAP

To recap, in this tutorial:

- we have seen **how to model** the blood vessel path and red blood cells;
- we have provided a basic **Material** to the objects and simple **lighting** to the scene;
- we have seen how to generate **instances** of red blood cells using a **Particle System**;
- we have seen how to make the red blood cells **follow the path** inside the blood vessel.

I hope this tutorial has been helpful!
See you soon!
