Mireia Espuga 206319
Aina Vendrell 205975
Anna Domènech 204798

ADVANCED VISUALIZATION

LAB 3: Volume Rendering

Until now, we have focused on computing light on surfaces of meshes, but sometimes we are interested on lighting a region of the scene instead.Then, the main purpose of this lab is to implement Volume Rendering to visualize volumes as CT medical images. Since volumes does not have associated meshes, we will create a ray for each pixel of the screen and use it to evaluate the light contribution from the volume to that pixel:

Ray marching

To evaluate the light contribution of each ray, we had implemented the ray marching algorithm, in a few words we will sample the volume value of points along the ray and add their contribution to the final color. This algorithm has different parts.

- **Ray setup:**
  First we have created a rayProperties struct, which contains the ray direction, ray step, step vector and the Gradient(in case it is used). In this step we had to define the rayStep using an uniform, then we have defined the ray direction by computing the vector that relates the voxel position and the camera position. Notice that we had converted the camera coordinates to local coordinates. Finally we have defined the step vector, used to compute the next sample position in the ray loop.

- **Ray Loop:**
  We have restricted the maximum number of steps to 1000 iterations. The start position is ---. We have named current_position to each ray sample.

  - Volume sampling
    Here we will sample the volume based on the current position. Given that to fetch textures we need to use texture coordinates within [0,1] range, for each sample position we had redefined its range to [0,1] range. It should be pointed out that we are working with local coordinates then we do not need to apply any transformation to the current position. We have created a function sample_volume() to implement this step.
  - Classification
    The volume does not contain information about the color, then for the moment we have defined the color_sample as (d,d,d,d) thus we will obtain a grayscale image.
  - Composition
    In this step the color of the sample is accounted for the final color. For the moment, we have implemented an iterative computation of the discretized volume-rendering integral where:
    ```
    finalColor += stepLength * (1.0 - finalColor.a) * sampleColor
    ```
    Some instances of composition like this one requires that the sampleColor RGB values are pre-multiplied by the alpha before the composition.

  Finally, the next sample position is computed using the current one and the step vector

Jittering

After implementing ray-marching we can see that since all samples start in parallel and have the same step length a wood pattern appears. To avoid that we have used a noise texture to get a random offset and we have added it in the first position of each ray in its direction

```
initialSample = initialSample + randomOffset * rayDirection
```

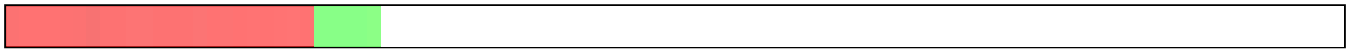
At the beginning we used the blueNoise texture, but since the results were not very accurate we have generated some other noise textures using: http://www.noisetexturegenerator.com/ to obtain better results.

Medical Path:

Transfer function

As we have learnt, transfer functions are mappings between volume properties and visual properties. Until now, we were rendering the volume considering the density value from the texture of the volume, now we will use the density value of each voxel to define its color and alpha. In order to implement that we have created a LUT texture, our goal is be able to visualize separately bones, muscles and skin. Then, we had to find the density values of each part: we have assigned the red color and an alpha of 0.5 to the density values between 0 and 0.25,i.e, to the skin. The muscles will be visualized as green color and with an alpha of 0.7, its density range is 0.25 to 0.28. All values

greater than 0.28 will be considered as bones thus white color and an alpha of 1 will be assigned. The resulting look up table have the following form:



LUT texture

Notice that the LUT texture has a 1x100 size, the height does not matter because we are only interested in the x value. If we apply the transfer function, it's done in the classification step.

Isosurface

As we know, an isosurface is a surface in the volume defined by points that have the same value in order to visualize clearer surfaces. Applied to our medical path, to get the shade on the skeleton, we'll want to compute the normal vector of the point to get the diffuse component. To do so, we consider the gradient, which since it's a derivative of the function in the point, acts as a normal vector. We use an approximation of the gradient through partial derivatives:

$$\nabla f(x,y,z) \approx \frac{1}{2h}\begin{pmatrix} f(x+h,y,z)-f(x-h,y,z) \\ f(x,y+h,z)-f(x,y-h,z) \\ f(x,y,z+h)-f(x,y,z-h) \end{pmatrix}$$

In the formula we find two main components, the constant small advancement value **h** and the **f()** function which corresponds to the sampling of the volume. To sample the volume, since we loaded the CT-abdomen volume into a texture, we use `texture3D(u_texture, vec3(x_pos, y_pos, z_pos)).x` to access the sample. In order to sample the volume at the x, y and z positions affected by **h** we only need to select the sample as we showed, but in the x_pos+h, x_pos-h, etc positions. Once we compute the difference between the samples in slightly different positions on each axis, we divide it by 2**h** and get the gradient normal vector.

Once we have the normal vector we proceed to get the diffuse component as always. First we compute the L vector between the eye and the world pixel position, and we get it to our 0-1 range. Then we compute the dot product between L and our gradientN and adapt it to our 0-1 range as well. Finally we assign to our rgb components of the final color the transmissivity and color components plus the diffuse NdotL `finalColor.rgb += (1.0 - finalColor.a) * sample_color.rgb * NdotL;` Also, we assign to our alpha component in the final color the maximum opacity, which yields 0 transmissivity thus not letting light through.

Volume clipping

Since in the volume there were parts that didn't belong to the body and we didn't want to visualize them, we have used two planes to define a region on top of the volume. At the beginning we tried to do it with one plane but since there were curved parts we ended up using tho planes to define a triangular region and skip the computations of parts that were on top of it.

ImGui

We have make the renderer dynamic to be able to modify parameters:

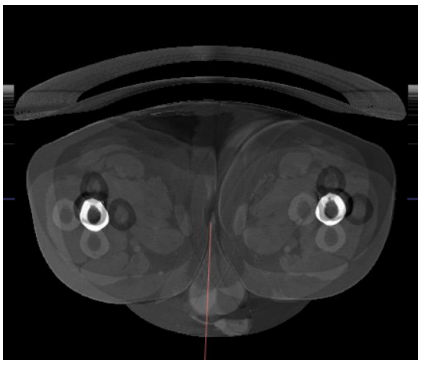
Camera: In this section you find the type of camera you want, and also the controls to modify the position, fov, near and far of the camera.

Entities: This section displays the different nodes and their characteristics. Depending on the type of entity we find unique options

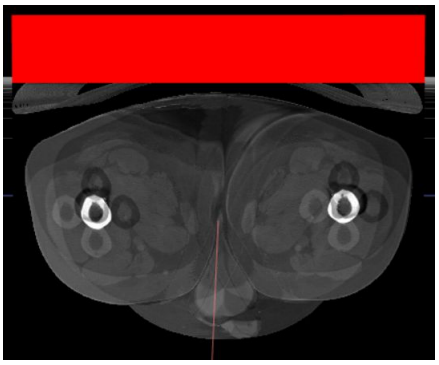
- **Scene node (Volume):**
  - Model:
  - Material:
    - **Threshold:** Enable when we want to apply gradient illumination, is a slider to modify the threshold and see the different densities.
    - **Step Length:** Step length of the ray marching algorithm.
    - **H:** Length of the displacement for gradient computing
    - **Apply Jittering:** Enable/disable to apply jittering
    - **Apply Transfer Function:** Enable/disable to apply Transfer function
    - **Apply Volume Clipping:** Enable/disable to apply Volume Clipping
      - **Volume clipping:** To edit the planes.

ANNEX

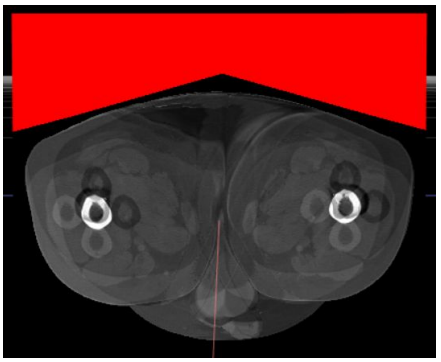
Volume Clipping



Without volume clipping

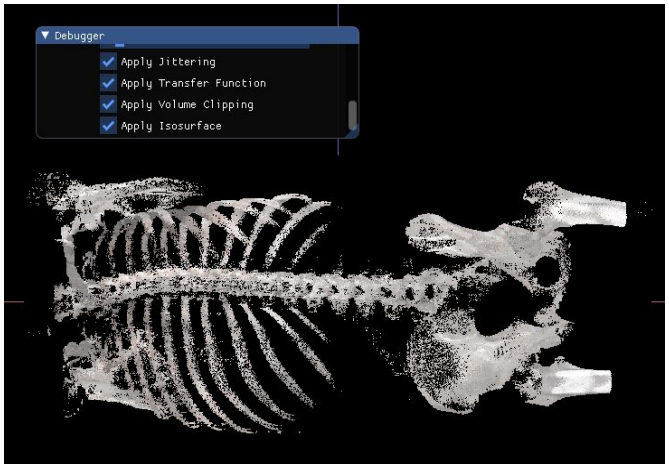


Volume clipping with one plane

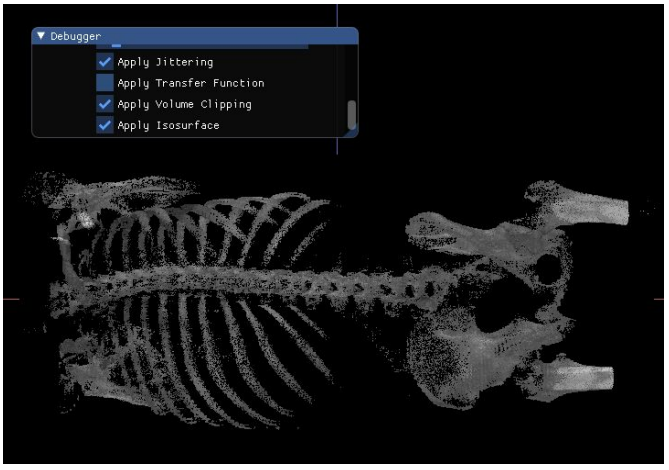


Volume clipping with two planes

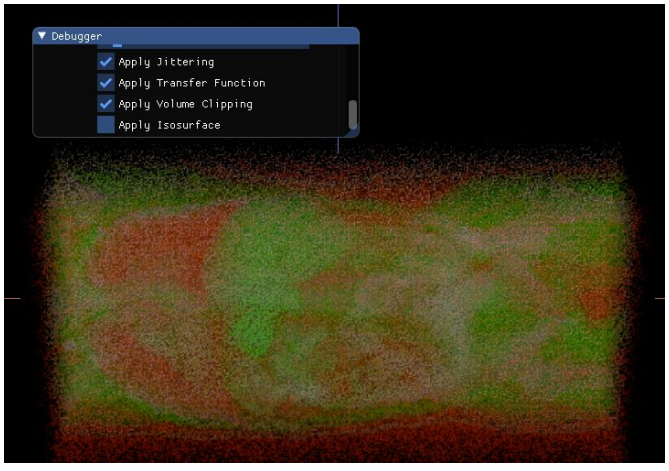
Imgui modifications



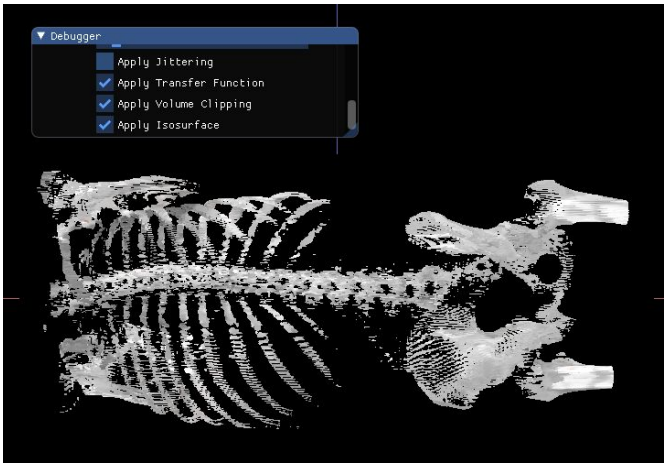
Volume with jittering, volume clipping, transfer function and isosurface visualization



Volume with jittering, volume clipping and isosurface visualization



Volume with jittering, volume clipping and transfer function



Volume with volume clipping, transfer function and isosurface visualization