2IMV20 Visualization

VOLUME RENDERING

Andrea Favia (1498010), Paolo Berizzi (1518798)

Introduction

The first assignment required the implementation of volume rendering based on raycasting. Practically, we had to implement both *Maximum Intensity Projection(MIP)* and *Compositing* raycasting in order to visualize an orange dataset.
By using volume rendering, data samples are projected onto the picture plane and this allows us to display more detailed images compared to geometric primitives approaches. The main idea with raycasting is, as the name suggests, a ray which is cast and equidistant points are samples. In order to get a better resolution, the voxel value is obtained by using trilinear interpolation.

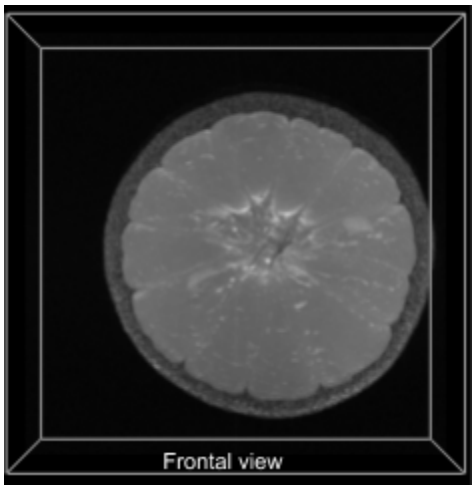
Maximum Intensity Projection (MIP)

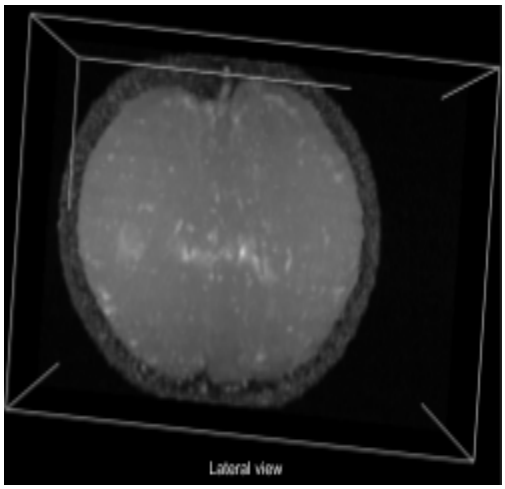
Using the MIP technique we project onto the visualization plane the voxel with maximum intensity along the ray axis. It is a fast technique but, since we suppress low-intensity voxel values, some details are hidden.

In order to implement the *render_mip* function, we first had to understand the render_slicer function. The idea is that we cast a ray along the z-axis, we select k equidistant values and then we select the maximum value among these values. Our ray must cover the whole figure, no matter the orientation and, therefore, we had to come up with a range of the for loop that made this possible. Simply, we assumed that the dataset can be approximated to a cube (w.r.t longest side) and we calculated the longest possible distance among two different points in the cube (i.e. the internal diagonal of the cube).
However, since our image can be rotated and translated along any arbitrary axis, we had to use the *view_matrix* from the ModelView OpenGL library that, basically, allows us to follow the axis and therefore make it possible to see different views of the orange. Also, in order to select the closest voxel to the ray and to have an overall better resolution, we had to use **trilinear**

interpolation. A value *val* can be thought as surrounded by a cube in which vertex has a value $Vijk$. The idea is to define the position of the value *val* with respect to the position of the cube vertices, represented by voxel values presented in the current dataset.

Once the trilinear interpolation is implemented, we can retrieve the voxel value for that sampled coordinates and keep the final **maximum** value for the ray. Then, repeating what is implemented in the *render_slicer*, we define simple colors to display the image.





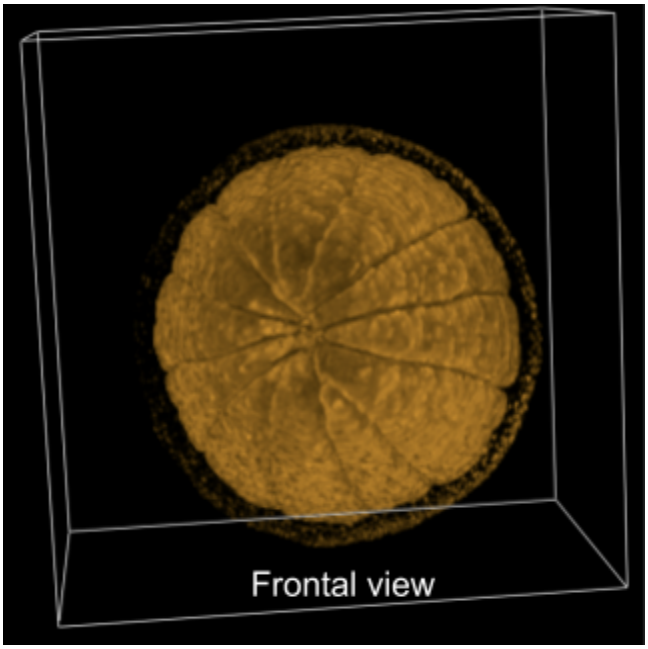
As it is possible to see in the images, the overall shape of the orange is clear. Also, it is possible to distinguish between the different parts of the orange such as the peel, slices, albedo, and pedicels.

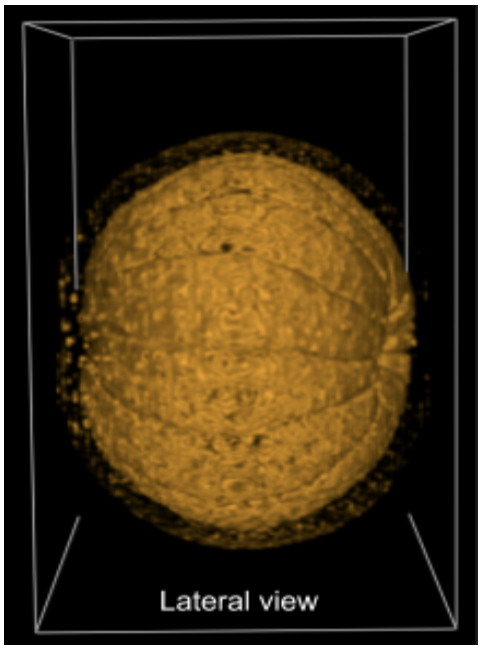
Compositing

The idea behind the **compositing** rendering is, like in the MIP, a ray cast onto the axis to capture equidistant samples. For each trilinear interpolated value we obtain a color **c** and opacity **α**. Next, we merged all colors sampled with each other by using a back-to-front technique and the opacity weighted sum to compute the composited value. $C_{final} = C_{current} * \alpha + C_{previous} * (1 - \alpha)$.

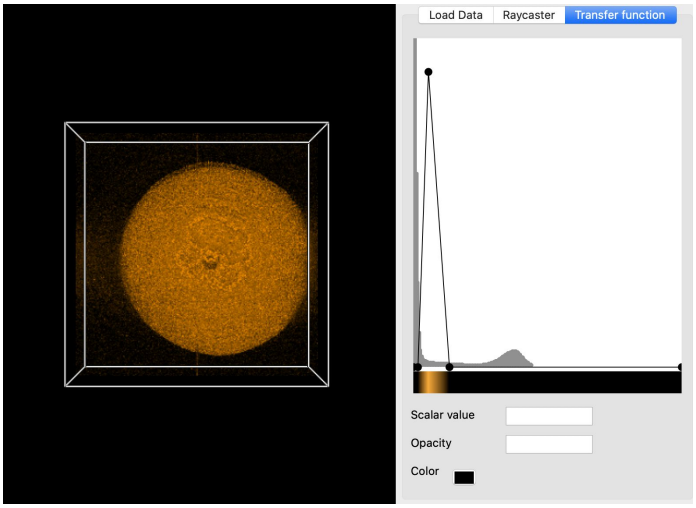
The intuition is that we weight more the current color value rather than the previous one. Also, in the end, we set a fully opaque background (**α** = 1) and then we obtain a single color for the entire ray.

Once the program finishes running, it is possible to customize the transfer function in order to highlights details in the image. The easiest method is to use intensity/opacity histogram that shows binned scalar data. By looking at the histogram, we can deduce and distinguish valuable data from noise. Hence, we can adjust the TF in order to display meaningful data. Screenshots of our implemented program are shown below:





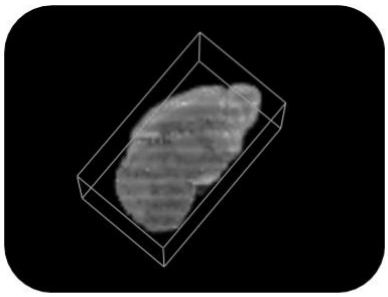
In the end, we tried to use the **transfer function** to highlight other details in the orange. We customized the TF in order to show the peel of the fruit. It is possible to see in the image, although some noise, the peel of the orange and also its pedicel (the darker spot in the middle).



Real use case - Mouse brain dataset

The 2013 IEEE Scientific Visualization Contest dataset provides gene expressions in a mouse brain throughout 7 different stages of development in which region shapes and sizes can change. Regions have been annotated with IDs and there are 11 different gene categories.

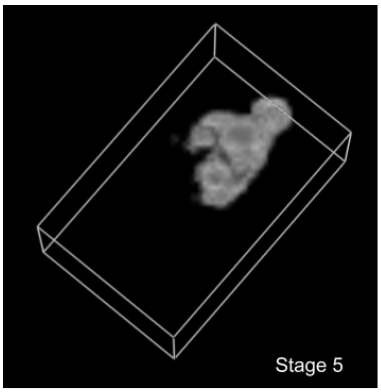
First, we wanted to get a general overview of the mouse brain and therefore we displayed the entire volume (all regions) per stage. In this way, we were able to get some ideas on the composition and shape of the brain.

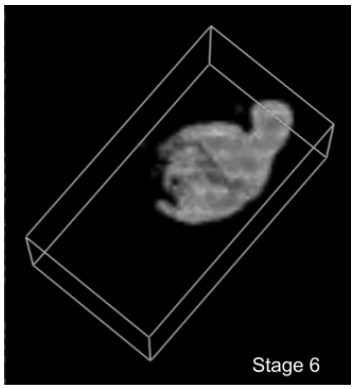


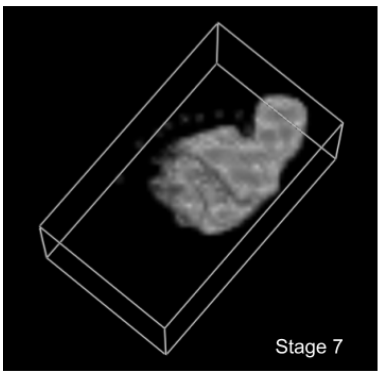
Representation of all brain regions at stage 7

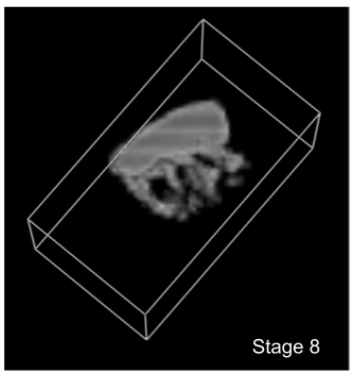
Next, we delved into the different regions present in each *annotation_volume.*

We selected the region with index 15750 (one of the largest) that represents the *Alar plate of the evaginated telencephalic vesicle (TelA).* The region is shown during the 5-th, 6-th, 7-th and 8-th stages.









Clearly, we can see the development of the region during the first 3 stages. The shape is getting bigger and crisper. When we look at the last stage, though, the shape looks slightly different. We suppose that, since it represents the adulthood stage (according to the contest description), in the completely-developed brain some parts of that region have grown differently and, likely, brought to new sub-regions labeled from experts that we cannot easily identify.

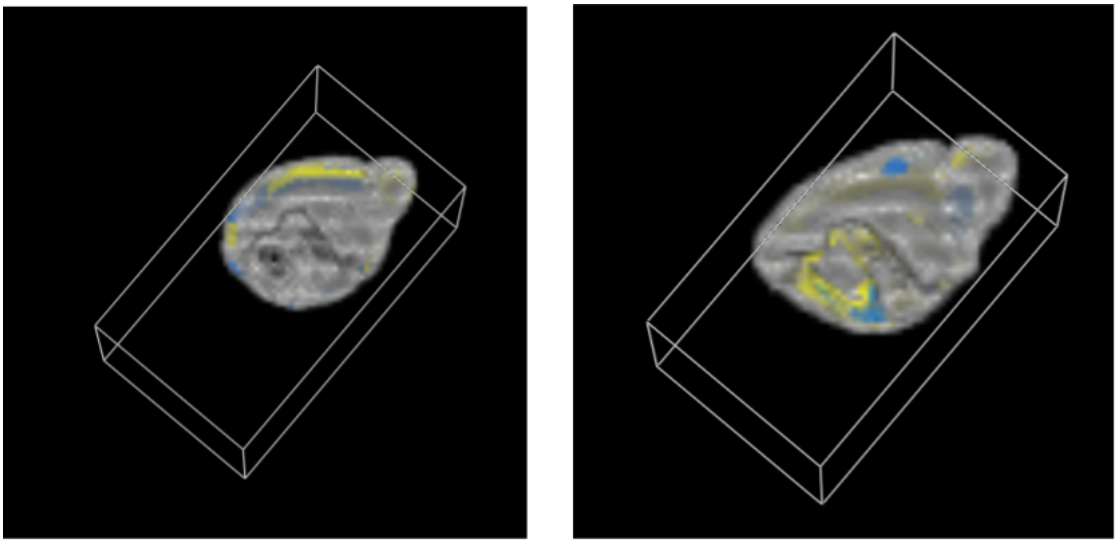
Following, we wanted to understand how energies are expressed across different regions and different stages. Since we did not have all the energy files for each *annotation_volume*, we decided first to get some insights from the *csv* files.

As explained in the contest description, each *energy_file* represents the intensity of the expression of a gene and, at the same time, we found that each gene can be expressed by multiple *energy_file* (even at the same stage). In order to visualize interesting data, we first found genes that are expressed by *energy_file* in more than three different *annotation_volume* in different stages.

The table below shows the genes that have more than three *energy_file* for different stages. Due to the lack of other files, some genes cannot be expressed for all stages (e.g. *annotation_volume* 5 does not have a connection with the Gap43 gene)

	(energy_file, annot_volume)	(energy_file, annot_volume)	(energy_file, annot_volume)	(energy_file, annot_volume)	(energy_file, annot_volume)
Actb	(100055980_energy.mhd, 5)	(100041480_energy.mhd, 7)	(100046731_energy.mhd, 2)	(100056178_energy.mhd, 6)	None
Gap43	(100034571_energy.mhd, 7)	(100054842_energy.mhd, 6)	(100051890_energy.mhd, 3)	None	None
Nnat	(100054703_energy.mhd, 5)	(100051802_energy.mhd, 5)	(100053198_energy.mhd, 3)	None	None
Ttr	(100051792_energy.mhd, 5)	(100055174_energy.mhd, 6)	(100051791_energy.mhd, 5)	(100055110_energy.mhd, 5)	(100037336_energy.mhd, 3)

For instance, we display regions *15750* (Alar plate of the evaginated telencephalic vesicle - TelA) and *16001* (Dorsal pallium/isocortex - DPal) which are the most relevant part in stages 6 and 7 to show how genes **Gap43** (Growth associated protein 43) and **Actb** (Actin, Beta) are expressed by the *energy_volume*. We must point out that in the 3-th stage, the region TelA and DPal, two of the most relevant regions in the latest development stages, are small and barely evolved and therefore the gene expression is scarce.



Expression of Gap43 (yellow) and Actb (blue) in stages 6 and 7