Assignment 7 — Direct Volume Rendering

New Attempt

- Due Nov 20 by 11:59pm
- Points 20
- Submitting a media recording or a file upload

Objective

The goal of this assignment is to help you practice and reflect on the lecture covering Direct Volume Rendering (DVR). You will learn how to load medical volume imaging data (DICOM dataset) and visualize it directly using optical properties, without reconstructing an isosurface.

DICOM (Digital Imaging and Communications in Medicine) refers to medical imaging files that follow the DICOM standard for storing and transmitting images such as X-rays, CT scans, MRIs, and ultrasounds.

- You'll **upload a screen-recorded video** that includes a walkthrough code review and a demonstration of your scene.
- Make sure to show the folder with **YOUR NAME** clearly visible.
- **Upload your scripts** along with your submission, and **walk through the code** during the screen recording.
- **I don't expect AI-generated code**—please do your best. I'm happy to support you and answer any questions along the way.

Follow the steps below for detailed instructions.

Preparation:

1. Project setup — **Skip this step, if you have already imported UnityVolumeRendering**:

- Import **UnityVolumeRendering (https://github.com/mlavik1/UnityVolumeRendering/releases/tag/2024.4)** to your Unity project
  - If you use Unity 6 editor, download **UnityVolumeRendering-2020.3+.unitypackage (https://github.com/mlavik1/UnityVolumeRendering/releases/download/2024.4/UnityVolumeRendering-2020.3+.unitypackage)** and import it to your existing project
  - Once you imported, enable SimpleITK by clicking on Volume Rendering in the menu -> Settings. A window will pop-up, click Enable SimpleITK.

- If you use Mac with Apple Silicon and face issues with SimpleITK, try the following:
  - First, you could try to Disable SimpleITK in the UnityVolumeRendering settings if you previously enabled it.
  - Delete all the files inside SimpleITK folder under Assets\UnityVolumeRendering\Assets\3rdparty\SimpleITK
  - Download the new version of SimpleITK here **https://github.com/SimpleITK/SimpleITK/releases/tag/v2.5.2 (https://github.com/SimpleITK/SimpleITK/releases/tag/v2.5.2)**
    - **(https://github.com/SimpleITK/SimpleITK/releases/tag/v2.5.2)** Download SimpleITK-2.5.2-CSharp-macosx-11.0-anycpu.zip
  - Unzip and copy those files to the SimpleITK folder in your Unity project Assets\UnityVolumeRendering\Assets\3rdparty\SimpleITK
  - Test it and let me know if you need support.

2. Medical volume DICOM dataset:

- In this assignment, you will use an open-source dataset from **Liver segmentation 3D-IRCADb-01 (https://www.ircad.fr/research/data-sets/liver-segmentation-3d-ircadb-01/)**.
  - Download the first patient case (DOB: 1944) and unzip the **3Dircadb1.1.zip**
  - Inside this folder, unzip **PATIENT_DICOM.zip** as you will work directly with the volume data without any segmentation masks.

Step 1: Load series DICOM volume data

- Similar to previous assignment, you can use SimpleITK to load the volume data. However, in this assignment, instead of loading one file of volume data, you will learn how to load a series of DICOM imaging data.
- Make sure to import SimpleITK to your script, e.g., `using itk.simple;`
- The `folderPath` can be given in the Editor mode.

```
var imageFileReader = new ImageSeriesReader();
VectorString dicom_names = ImageSeriesReader.GetGDCMSeriesFileNames(folderPath);
imageFileReader.SetFileNames(dicom_names);
var volImage = imageFileReader.Execute();
```

- Cast the image to 32-bit float to support GetBuffer array in Unity

```
var volumeImage = SimpleITK.Cast(volImage, PixelIDValueEnum.sitkFloat32);
```

- Print out the total number of pixels to make sure everything works fine when loading the volume data

`Debug.Log(volumeImage.GetNumberOfPixels());`

- You can create a new script to work independently with volume rendering. Once you created, you can reference it with your main script. In this example, I created a script called `DirectVolumeRendering.cs` and reference it in the Editor.
- I would call this function in the main script for now to make sure I won't forget it later.

`//var volRendObject = await volumeRendering.LoadVolumeDataAsRenderedObject(volumeImage);`

Step 2: Config Volume Rendering

- In the DirectVolumeRendering script, import the namespaces

```
using System.Threading.Tasks;
using UnityEngine;
using UnityVolumeRendering;
using System;
using System.Runtime.InteropServices;
```

- You can create an asynchronous function to work asynchronously.

```
public async Task<VolumeRenderedObject> LoadVolumeDataAsRenderedObject(itk.simple.Image volumeImage)
{

}
```

- Alright! Now let's get back to the `LoadVolumeDataAsRenderedObject` in the DirectVolumeRendering.cs, you can first read the image buffer and store in a float array

```
    int nx = (int)volumeImage.GetWidth();
    int ny = (int)volumeImage.GetHeight();
    int nz = (int)volumeImage.GetDepth();
    int numPixels = nx * ny * nz;
    float[] voxelData = new float[numPixels];
    Vector3Int dimension = new Vector3Int(nx, ny, nz);

    IntPtr bufferImg = volumeImage.GetBufferAsFloat();
    Marshal.Copy(bufferImg, voxelData, 0, numPixels);
    Debug.Log(voxelData.Length); //Print the length to debug
```

- Import the voxel data based on the float array to VolumeDataset to work with UnityVolumeRendering

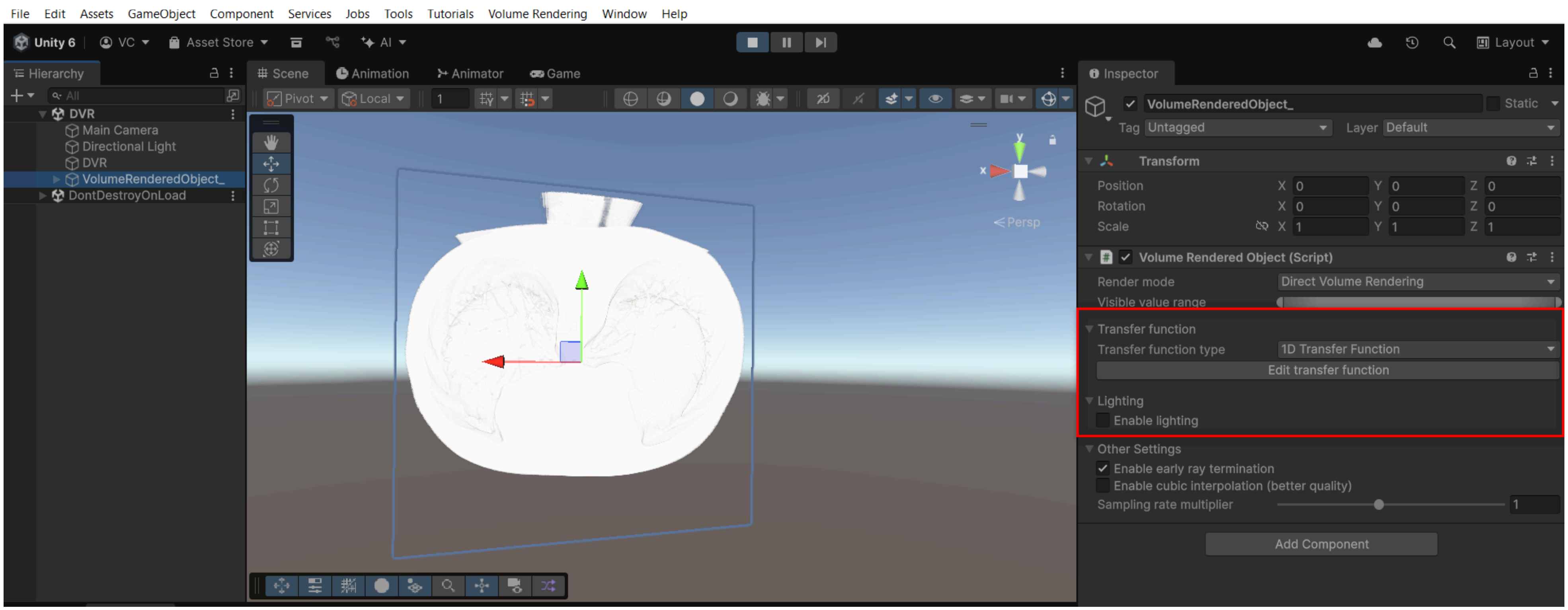
```
        VolumeDataset dataset = new VolumeDataset();
        dataset.data = voxelData;
        dataset.dimX = dimension.x;
        dataset.dimY = dimension.y;
        dataset.dimZ = dimension.z;
```

- Based on this volume dataset, you then now can load and create Unity GameObject to render it.

```
        if (dataset != null)
        {
            // Load Volume Data and Create GameObject
            Debug.Log("Loading dataset");
            var volRendObject = await VolumeObjectFactory.CreateObjectAsync(dataset);

            return volRendObject;
        }
        else
        {
            Debug.Log("Dataset not found!");
            return null;
        }
```

- I believe it should work, now. You can go back to the Editor and test it. The result should look like the following:


File Edit Assets GameObject Component Services Jobs Tools Tutorials Volume Rendering Window Help
Unity 6
Hierarchy
Scene
Animation
Animator
Game
Inspector
Pivot
Local
Persp
Add Component

Hierarchy:
- DVR
  - Main Camera
  - Directional Light
  - DVR
  - VolumeRenderedObject_
- DontDestroyOnLoad

Inspector:

VolumeRenderedObject_ Static
Tag Untagged Layer Default

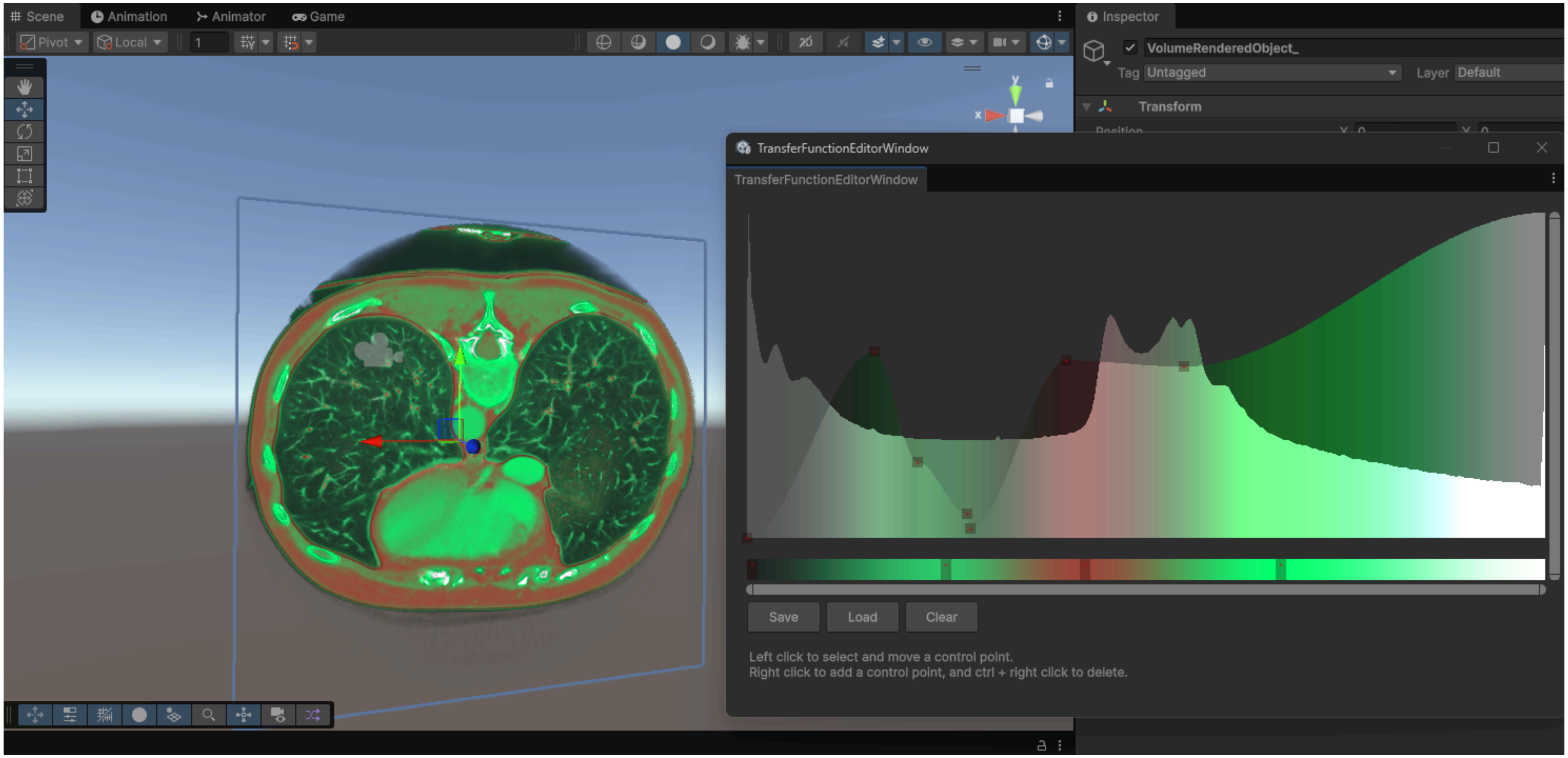
Transform

Position	X	0	Y	0	Z	0
Rotation	X	0	Y	0	Z	0
Scale	X	1	Y	1	Z	1

Volume Rendered Object (Script)
Render mode: Direct Volume Rendering
Visible value range
Transfer function
Transfer function type: 1D Transfer Function
Edit transfer function
Lighting
Enable lighting
Other Settings
✓ Enable early ray termination
Enable cubic interpolation (better quality)
Sampling rate multiplier: 1

Step 3: Lighting and Transfer Function (Manually)

- In this step, you will see the difference between volume renderings with and without lighting. Thus, during the runtime, click on VolumeRenderedObject_ gameobject, and in the inspector window, enable lighting.
- You should also try to edit transfer function by adding new value point and changing the color.
- To get the point for this step, recording your screen during runtime, enabling lighting, and editing transfer function to whatever color you prefer. There is no right/wrong color map you defined. This step is to allow you to understand the effect of adjusting transfer function resulting the final volume rendering.



Left click to select and move a control point.
Right click to add a control point, and ctrl + right click to delete.

Instructions to earn credits for the following steps:

- You should make a **new recording** and **submit a new video addition to the previous one** to earn the credits for following steps.
- Also walk me through the code for the new recording

Step 4: Transfer Function (Script and Preset)

- Instead of manually adjusting transfer function, you can load existing preset for transfer function.
- In this step, you can load the preset that I defined based on VTK colormap (**erdc_rainbow_dark.tf (https://sjsu.instructure.com/courses/1613868/files/84387126?wrap=1)** ↓ **(https://sjsu.instructure.com/courses/1613868/files/84387126/download?download_frd=1)**) or you can load the saved transfer function you did in Step 3.
- To get credit for step 4, you should enable lighting, adjust window visibility, and load transfer function preset automatically via script. I'll give a few tips in the following.
- First, you can import the following namespaces to your main script in Step 1.

```
using System.Collections.Generic;
using UnityVolumeRendering;
using System.IO;
```

- Create a struct to store the lists of color points and alpha points.

```
    private struct TF1DSerialisationData
    {
        public List<TFColourControlPoint> colourPoints;
        public List<TFAlphaControlPoint> alphaPoints;
     }
```

- Then, you can create a new function to load the transfer function from path.

```
public UnityVolumeRendering.TransferFunction LoadTransferFunction(string colorMapPath){

}
```

- Let's load the transfer function preset to a variable (string). Since the transfer function formatted as a JSON style, you can then use JsonUtility in Unity to serialize the file.

```
       string colorMap = File.ReadAllText(colorMapPath);
        if (colorMap == null)
        {
            Debug.LogError(string.Format("File does not exist: {0}", colorMapPath));
            return null;
        }

        TF1DSerialisationData data = JsonUtility.FromJson<TF1DSerialisationData>(colorMap);
```

- Let's debug it to make sure it works well after loading.

```
Debug.Log(data.colourPoints.ToString());
Debug.Log(data.alphaPoints.ToString());
```

- You now can create and assign loaded transfer function to UnityVolumeRendering.TransferFunction to assign it during runtime.

```
        var tf = new UnityVolumeRendering.TransferFunction();
        tf.colourControlPoints = data.colourPoints;
        tf.alphaControlPoints = data.alphaPoints;
        return tf; //return the newly created transfer function to load in the main script.
```

- Return to the main script in Step 1 where you created `volRendObject` to further modify the volume rendering game object.
- You can call `LoadTransferFunction` and set loaded transfer function to the `volRendObject`
- The `colorMapPath` is the path to the preset in your local drive. You can declare this variable to get the path in the editor.

```
        var newTF = LoadTransferFunction(colorMapPath);
        if (newTF != null)
           volRendObject.SetTransferFunction(newTF);
```

- In this step, you can enable lighting asynchronously via script as well.

```
       await volRendObject.SetLightingEnabledAsync(true);
```

- To adjust window visibility, you can declare a Vector2, e.g., public Vector2 visibilityWindow = new Vector2(0.6, 1);
- You can now adjust the window visibility automatically via script.

```
       volRendObject.SetVisibilityWindow(visibilityWindow);
```

- The results should look similar to the following:

0:00/0:00

Step 5: Voxel filtering

- This step is quite similar to previous assignment (Assignment 5 Step 2) where you can load volume image buffer and store it as the float array. This allow you to filter the pixel values.
- To earn credit for this step, you should modified Step 2. Instead of assigning the float array (`voxelData`) directly to `VolumeDataset`, you should filter the pixel values, e.g., **pixel > 0**
- Adjust the **visibility window to 0 and 1**
- Show the code for those loops and filtering in the screen recording
- The results should be similar to the following:

0:00/0:00

Direct Volume Rendering

Criteria	Ratings		Pts
Step 1	**3 pts** **Full Marks**	**0 pts** **No Marks**	3 pts
Step 2	**10 pts** **Full Marks**	**0 pts** **No Marks**	10 pts
Step 3	**2 pts** **Full Marks**	**0 pts** **No Marks**	2 pts
Step 4	**3 pts** **Full Marks**	**0 pts** **No Marks**	3 pts
Step 5	**2 pts** **Full Marks**	**0 pts** **No Marks**	2 pts
			Total Points: 20