Assignment 5 - Surface Rendering

New Attempt

- Due Nov 7 by 11:59pm
- Points 20
- Submitting a media recording or a file upload

Objective

The goal of this assignment is to allow you to practice and reflect on the lecture related to indirect volume rendering (surface rendering). You will need to load medical volume imaging and reconstruct 3D surface using Marching cube technique. Follow steps below for detailed instructions.

- You'll **upload a screen-recorded video** for a walk-through code review and demonstration of your scene.
- Show the folder with **YOUR NAME**
- **Upload your scripts** to the submission and **walk-through the codes** during screen recording
- **I don't expect AI-generated codes**, please try your best. I am happy to support and answer any questions.

Preparation:

1. Project setup:

- 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.
    - SimpleITK is an open-source, cross-platform image analysis library that provides a simplified interface to the Insight Segmentation and Registration Toolkit (ITK). We use it to read and analyze volume imaging.

    - 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.
- Import Geometry3Sharp library to your project.
    - I cleaned and customized the library. You can simply download and extract the **geometry3Sharp.zip (https://sjsu.instructure.com/courses/1613868/files/84330642?wrap=1) (https://sjsu.instructure.com/courses/1613868/files/84330642/download?download_frd=1)** to your Unity Asset folder
    - Geometry3Sharp is a C# library for 2D/3D geometric computing, compatible with the Unity game engine. It provides a wide range of functionalities for geometric and mesh processing, including vector math, curves, implicit surfaces, mesh simplification, and remeshing.

2. Medical volume dataset:

- You will use an open-source dataset from **MedSeg Liver segments dataset (https://figshare.com/articles/dataset/MedSeg_Liver_segments_dataset/13643252)**
    - I cleaned and made it ready for your practice.
    - Download **Liver_Mask_01.tif (https://sjsu.instructure.com/courses/1613868/files/84330643?wrap=1) (https://sjsu.instructure.com/courses/1613868/files/84330643/download?download_frd=1)** to your folder

Step 1: Loading Volume Imaging

- You can load the volume image data using SimpleITK using the simple codes below, where `datasetFile` links to your dataset file

```
var imageFileReader = new itk.simple.ImageFileReader();
imageFileReader.SetFileName(datasetFile);
var volImage = imageFileReader.Execute();
```

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

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

Step 2: Reconstruct the 3D surface mesh from the volume data

- You can create a new script or a new function to work with mesh reconstruction.
- Import itk.simple, g3, System, and System.Runtime.InteropServices to work with SimpleITK, Geometry3Sharp, and system runtime array pointer
- Since the original image data is 16-bit signed integer, we need to cast it to 32-bit float to support GetBuffer array in Unity

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

- Get volume image buffer array and assign it float array to extract the surface

```
int depth = (int)volumeImage.GetDepth();
int height = (int)volumeImage.GetHeight();
int width = (int)volumeImage.GetWidth();
int length = width * height * depth;

IntPtr bufferImg = volumeImage.GetBufferAsFloat();

float[] bufferAsArrayImg = new float[length]; // Allocates new memory the size of input
Marshal.Copy(bufferImg, bufferAsArrayImg, 0, length);
```

- Now, you can create a dense grid to work with Marching Cube. But first you can filter the isovalue through the 3D volume (width, height, depth), and only those above > 0 should be included in the grid.

```
    DenseGrid3f grid = new DenseGrid3f(width, height, depth, 1);

    for (int k = 0; k < depth; k++)
       {
           for (int j = 0; j < height; j++)
           {
               for (int i = 0; i < width; i++)
               {
                   int idx = i + width * (j + height * k);
                   float pixel = bufferAsArrayImg[idx];

                   if (pixel > 0)
                   {
                       grid[idx] = -pixel; //Assign value to liver grid
                   }
               }
```

```
    }
}
```

- You can then work with Marching Cube to reconstruct the surface mesh based on the dense grid of volume data. Finally, you can have the resulting isosurface

```
double cellsize = volumeImage.GetSpacing()[0];
double numcells = 64;
var iso = new DenseGridTrilinearImplicit(grid, Vector3f.Zero, cellsize);

MarchingCubes c = new MarchingCubes();
c.Implicit = iso;
c.RootMode = MarchingCubes.RootfindingModes.Bisection;      // cube-edge convergence method
c.RootModeSteps = 5;                                        // number of iterations
c.Bounds = iso.Bounds();
c.CubeSize = c.Bounds.MaxDim / numcells;
c.Bounds.Expand(3 * c.CubeSize);
c.Generate();
var _result = c.Mesh;
```

- It's not done yet, you would need to convert and create GameObject in Unity scene to render it. It's noteworthy to mention that Unity use left-hand coordinate system, so you need to flip the resulting mesh. It's optional, but it's good to make it consistent. You can then create new Material via script using default shader (Universal Render Pipeline/Lit).

```
MeshTransforms.FlipLeftRightCoordSystems(_result);
MeshNormals.QuickCompute(_result);

var liverObject = new GameObject("Liver");
liverObject.transform.parent = gameObject.transform;
liverObject.AddComponent<MeshFilter>();
var liverMat = liverObject.AddComponent<MeshRenderer>();
liverMat.material = new Material(Shader.Find("Universal Render Pipeline/Lit"));
g3UnityUtils.SetGOMesh(liverObject, _result);
```

- Finally, it's time to run and test it.

Step 3: Mesh refinement (mesh smoothing)

- If you look the results, there are artifacts, e.g., staircase artifacts resulting from image segmentation when reconstructing isosurface
- You don't need to use Connected Component Analysis because the data is good enough for the large surface

- You can enhance it by using mesh smoothing. In this case, you can use Remesher from Geometry3Sharp to enhance it, you can change the parameters `EdgeLengthMultiplier, _remeshPasses, and smoothSpeedT` to evaluate your desired results.

```
//Remeshing and smooth
float EdgeLengthMultiplier = 5.5f;
int _remeshPasses = 20; //20
float smoothSpeedT = 1.0f;

Remesher _remesh = new Remesher(_result);
_remesh.PreventNormalFlips = true;
_remesh.SetTargetEdgeLength(EdgeLengthMultiplier);
_remesh.SmoothSpeedT = smoothSpeedT;
_remesh.SetProjectionTarget(MeshProjectionTarget.Auto(_result));

for (int k = 0; k < _remeshPasses; ++k)
    _remesh.BasicRemeshPass();
```

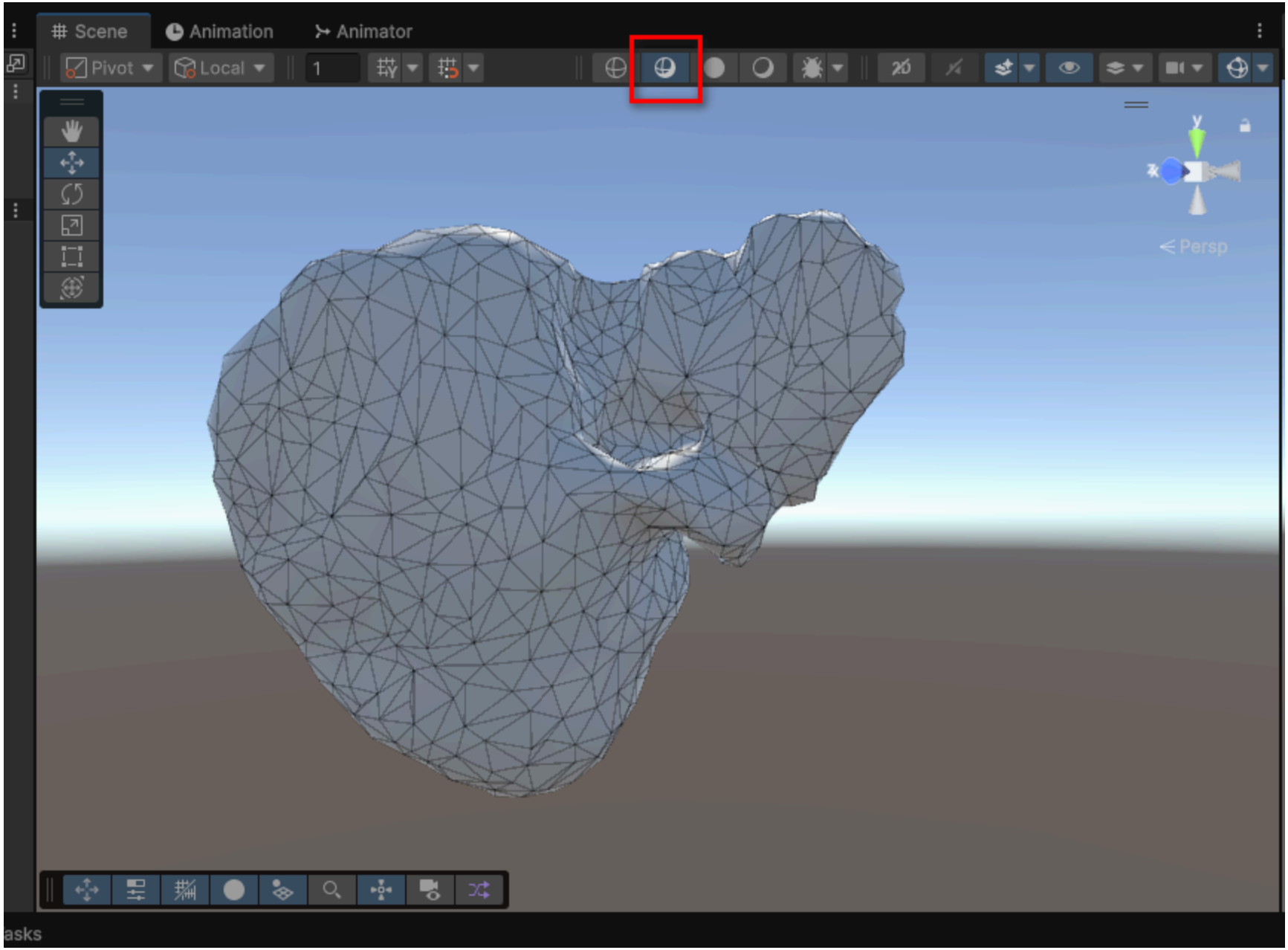
Step 4: Mesh refinement (mesh decimation)

- Since the number of triangles are huge, you may consider to reduce the number of those triangle using mesh decimation technique. In this case, you can use Reducer script to reduce it. Try to change the value for _reducer.ReduceToTriangleCount to evaluate the final result.

```
// Mesh decimation
 Reducer _reducer = new Reducer(_result);
 _reducer.ReduceToTriangleCount(2000);
```

Result:

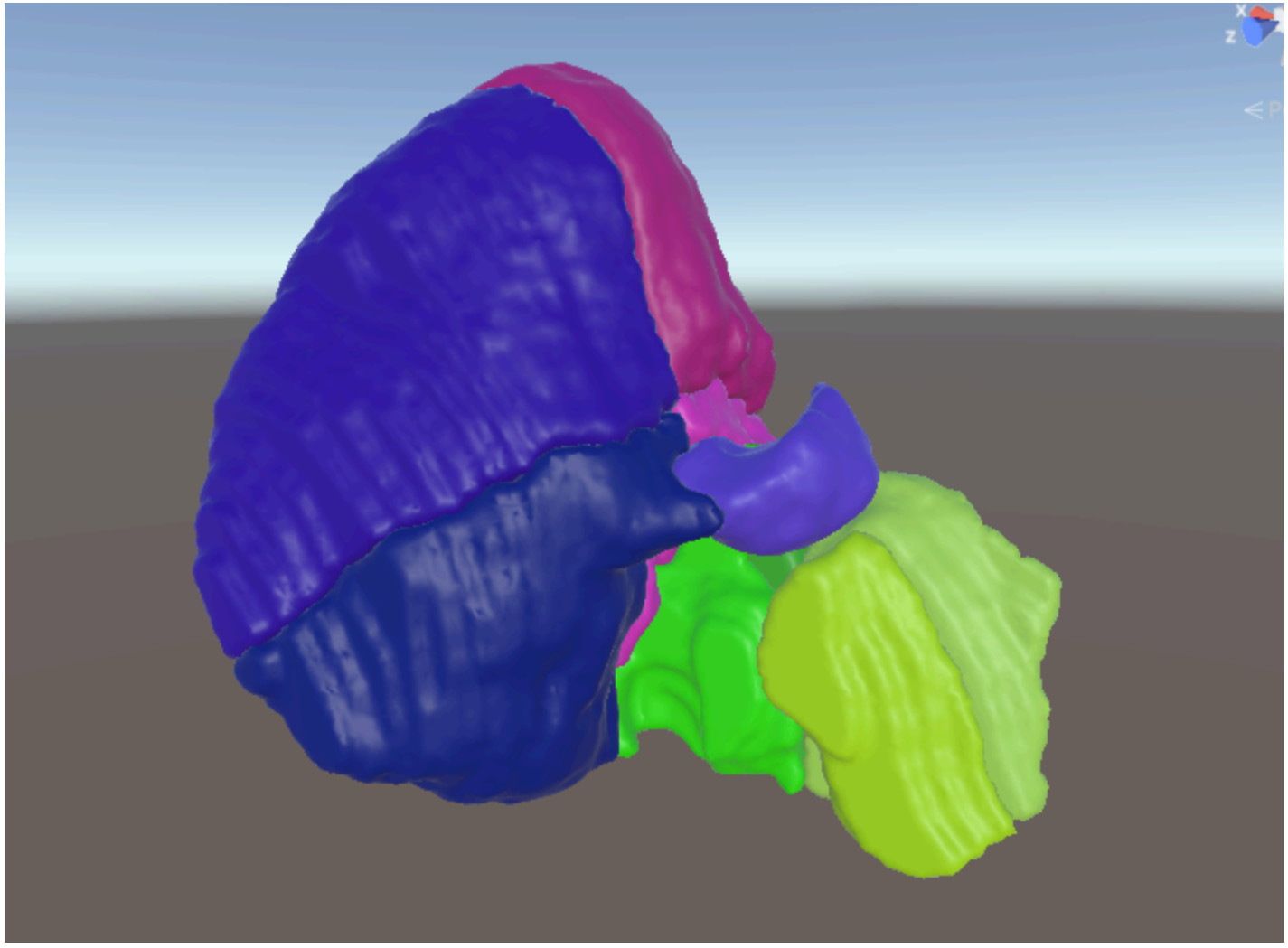
- The result should look similar to the following. You should enable the wireframe (see red square in the screenshot) to show final mesh with mesh decimation.



Step 5: Advance and Extension to Different Liver Segments

- In this dataset, there are separated liver segments (isovalue 0 is background, but **1 to 9** are correspond to those segments.
- The final result should show different meshes for those segments with different colors

- Tips: in step 2, you filter the `pixel > 0`, but you can consider to create a list of dense grid. During the filtering, you can create another loop to filter pixel for those segments, e.g., pixel == 1 is segment 1, and store in the list of dense grid you just created.
- Loop through the list of dense grid to reconstruct the isosurface for those segments
  - You can try to apply new color for the materials based on random color.
- Final results should look similar the following:



Surface Rendering

Criteria	Ratings		Pts
Step 1	**2 pts** **Full Marks**	**0 pts** **No Marks**	2 pts
Step 2	**8 pts** **Full Marks**	**0 pts** **No Marks**	8 pts
Step 3	**3 pts** **Full Marks**	**0 pts** **No Marks**	3 pts
Step 4	**2 pts** **Full Marks**	**0 pts** **No Marks**	2 pts
Step 5	**5 pts** **Full Marks**	**0 pts** **No Marks**	5 pts
			Total Points: 20