3G4 Lab Report: Data Manipulation for 3D Imaging

Shanzi(Monica) Ran
sr2021@cam.ac.uk

March 6, 2025

1 Introduction

Manipulation of three-dimensional (3D) data is a crucial step in the realm of 3D graphics, including its various applications in the context of medical imaging. In this lab exercise, the code implementation of nearest neighbour reslicing is explored, followed by subsequent investigation of surface extraction and rendering performances of the interpolated data. Different rasterisation and shading techniques are also discussed and compared in this report.

2 Results and Discussion

2.1 Datasets

In this recorded exercise, there are two datasets used:

- `ct_data.dat`: A 3D dataset of a Computed Tomography (CT) scan of a human skull
- `synthetic_data.dat`: A syntehtic 3D dataset of a sphere with an inscribed smiley face.

When visualised, the 3D structure of the dataset is depicted using extracted isosurfaces of intensity value within the volume data. Applying different thresholds to the CT data to adjust the accuracy of estimated contours, it can be inferred that the skull present is of a yound child whose teeth are still developing.

2.2 Nearest Neighbour Reslicing

Reslicing and interpolation are essential to generate a dense data matrix for more precise visualisation. The interpolation technique explored in this exercise is the nearest neighbour interpolation, where the interpolation value is directly assigned to the nearest voxel in the original data. This interpolation in the *x-z* and *y-z* planes is implemented as follows:

Listing 1: *x-z Reslice Nearest Neighbour*

```
void VulkanWindow::
    construct_xz_reslice_nearest_neighbour(
    void) {
    int x, z, nearest_slice, data_index,
        reslice_index, alpha = 0;
    reslice_index = 0;
    for (z = 0; z < VOLUME_DEPTH; z++) {
    nearest_slice = round(z / DELTA_Z);
    data_index = (int)nearest_slice *
        VOLUME_WIDTH * VOLUME_HEIGHT +
        VOLUME_WIDTH * y_reslice_position;
    for (x = 0; x < VOLUME_WIDTH; x++) {
        texture_store.xz_rgba[reslice_index++]
            = MapItoR[data[data_index]];
        texture_store.xz_rgba[reslice_index++]
            = MapItoG[data[data_index]];
        texture_store.xz_rgba[reslice_index++]
            = MapItoB[data[data_index]];
        texture_store.xz_rgba[reslice_index++]
            = alpha;
        data_index++;
    }}}
```

Listing 2: *y-z Reslice Nearest Neighbour*

```
void VulkanWindow::
    construct_yz_reslice_nearest_neighbour(
    void)
{
int z, y, nearest_slice, data_index,
    reslice_index, alpha;
alpha = 0;
reslice_index = 0;
for (z = 0; z < VOLUME_DEPTH; z++) {
    nearest_slice = round(z / DELTA_Z);
    data_index = (int)nearest_slice *
        VOLUME_WIDTH * VOLUME_HEIGHT +
        x_reslice_position;
    for (y = 0; y < VOLUME_HEIGHT; y++) {
    texture_store.yz_rgba[reslice_index++] =
        MapItoR[data[data_index]];
    texture_store.yz_rgba[reslice_index++] =
        MapItoG[data[data_index]];
    texture_store.yz_rgba[reslice_index++] =
        MapItoB[data[data_index]];
    texture_store.yz_rgba[reslice_index++] =
        alpha;
    data_index = data_index + VOLUME_WIDTH;
    }}}
```

The reslice functions for *y-z* and *x-z* planes exhibit structural similarities with two critical mapping distinctions. The starting pixel index is determined distinctly based on the respective plane coordinates, with *x-z* reslice using y position and *y-z* reslice using x position. Data index incrementation is also different: x-z reslice employs a direct increment by 1, while y-z reslice increments by the volume width, reflecting

the horizontal-first indexing structure of the underlying data representation.

As shown in Fig 1, the difference between dataset properties would result in different visual effects when resliced. For example, the curve roughness of the nearest neighbour estimate of the CT scan data that arises from the discrete nature of the algorithm is not as apparent in the synthetic data case. As natural images inherently contain less discontinuity than generated data, nearest neighbour reslicing might not be optimal for CT scans when smoothness is desired.

Many alternative interpolation techniques are capable of providing a higher level of smoothness, such as linear interpolation which achieves numerical continuity, but the computation cost is higher. Higher smoothness interpolations may also sacrifice sharpness of the extracted curves and surfaces, which should also be taken into account when details are specifically important.

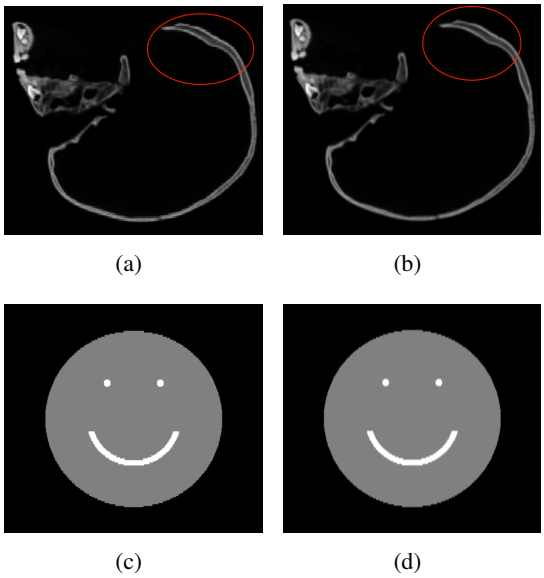


Figure 1: *Nearest Neighbour and Linear Interpolation Reslice for CT scan and synthetic data: a) CT scan with nearest neighbour reslice, b) CT scan with linear interpolation reslice, c) Synthetic data with nearest neighbour reslice, d) Synthetic data with linear interpolation reslice.*

2.3 Surface Analysis

After reslicing, surfaces are extracted via shape-based interpolation, beginning with intensity-thresholded isosurface contours. These contours are interpolated to create dense surfaces, then triangulated into closed meshes at adjustable resolutions.

Tables 1 and 2 show mesh characteristics at varying interpolation resolutions and thresholds. As intensity threshold decreases from 254 in synthetic data, vertex and triangle counts increase dramatically due to more voxels being included, significantly raising computational costs. This effect is more pronounced in CT data, where a threshold of 10 produces over a million triangles which is computationally impossible for real-time rendering. Thus, threshold selection critically balances computational efficiency against surface accuracy. Interpolation resolution has a more modest impact, with higher resolutions creating denser meshes with an approximately quadratic increase in vertex and triangle counts, as it influences surface smoothness rather than structural composition.

Setting a proper threshold and interpolation resolution also affacts area of the extracted surface and the meshed volume. While the effect from interpolation resolution is less significant, a lower threshold still results in area and volume of orders of magnitude larger. This is because a higher threshold would result in a more compact shape and segmentation region, while the resolution preserves the size of the object. When setting up the threshold and resolution, it is important to also consider physical significances of the area and volume data (e.g. reference size of body tissues) when balancing extraction accuracy and computational costs.

Alternative Polygon Structure In the code implementation of this exercise, the traingle meshes are stored using separate lists of `NUM_TRIANGLES` and `NUM_VERTICES` for each mesh. Therefore, the overall mesh takes up $\texttt{NUM\_TRIANGLES} * 12 + \texttt{NUM\_VERTICES} * 12$ bytes of storage. In comparison, if a single list of pointers is adopted, the same mesh would take up $\texttt{NUM\_TRIANGLES} * (3 * 3 * 4) = \texttt{NUM\_TRIANGLES} * 36$ bytes of storage as each pointer stores 3 `float` values.

For example, with the synthetic data at a threshold of 254 and interpolation resolution of 1, the two-list scheme would take up 237 `kB` of storage, while the single-list scheme requires 475 `kB`. In another example where the number of vertices and triangles are even greater, such as the CT data at a threshold of 10 and interpolation resolution of 1, the two-list scheme would take up 20.7 `MB` of storage, while the single-list scheme requires 41.4 `MB`. This suggests that the original data structure is more memory efficient as the vertex data

Table 1: *Synthetic data parameters and resulting mesh characteristics*

Intensity Threshold	Interpolation Resolution	Number of Vertices	Number of Triangles	Area (cm^2)	Volume (ml)
254	8	182	352	22.8	2.25
	4	413	814	27.2	3.73
	2	1,640	3,260	29.7	4.39
	1	6,580	13,200	30.4	4.49
127	8	4,510	9,020	699	1,740
	4	10,200	20,400	701	1,740
	2	40,900	81,900	706	1,740
	1	166,000	331,000	713	1,740

Table 2: *CT data parameters and resulting mesh characteristics*

Intensity Threshold	Interpolation Resolution	Number of Vertices	Number of Triangles	Area (cm^2)	Volume (ml)
30	8	22,700	47,000	2,430	505
	4	42,700	86,400	2,410	557
	2	156,000	312,000	2,480	572
	1	594,000	1,190,000	2,500	573
10	8	18,600	37,900	2,350	660
	4	37,600	75,600	2,360	697
	2	146,000	292,000	2,420	703
	1	573,000	1,150,000	2,430	703

is only stored once, which is more suitable for large datasets where memory usage is a concern.

2.4 Viewing Transformations

When visualizing triangular meshes, vertex positions are transformed from local to world coordinates via the modelview matrix, which is then aligned with the view coordinates. The implemented fly-through viewing simulates first-person camera movement around stationary objects, enabling 3D space navigation. This viewing transformation applies an offset translation followed by rotations around y and x axes, implemented using the GLM library as follows:

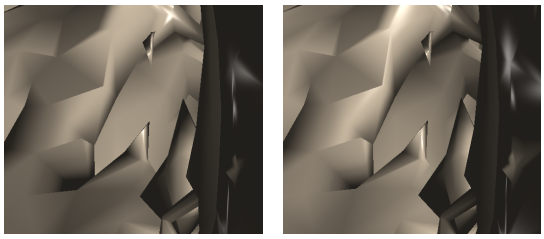
Listing 3: *y-z Reslice Nearest Neighbour*

```
glm::mat4 VulkanCanvas::viewingTransformation
    ()
{
    glm::mat4 matrix(1.0f);

    if (fly) {
    matrix = glm::translate(matrix, glm::vec3(
        offset_x, offset_y, -offset_z));
    matrix = glm::rotate(matrix, heading, glm
        ::vec3(0.0f, 1.0f, 0.0f) );
    matrix = glm::rotate(matrix, pitch, glm::
        vec3(1.0f, 0.0f, 0.0f) );
    }

    matrix = glm::translate(matrix, glm::vec3
        (0.0f, 0.0f, -depth_offset));

    return matrix;
}
```

As shown in Fig reffig:lightmove, the fly-through viewing mode can be further enhanced by adjusting the light source position setting relative to the observer. When the light source moves with the observer, the inner structures of the skull can be seen clearer as the scene appears to be illuminated from the observer's perspective, creating a more realistic visual effect. In contrast, when the light source remains stationary, the scene appears to be illuminated from a fixed direction, which may not accurately reflect the observer's natural viewpoint.

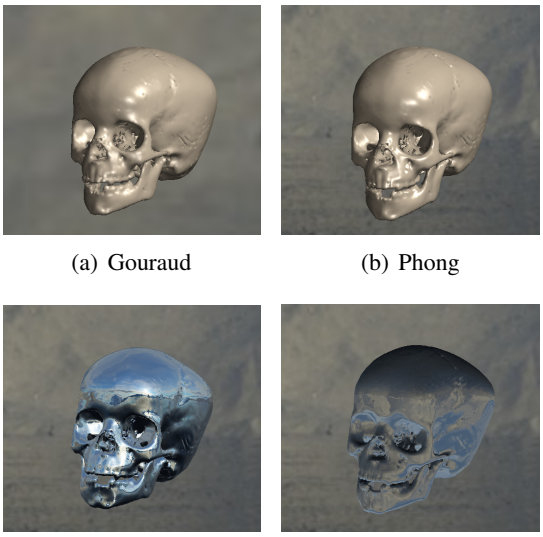
2.5 Rasterisation and Shading

Fig 3 illustrates how different shaders adopted provide different surface rendering effects by expression of the



(a) (b)

Figure 2: *Visualisations in fly-through viewing mode: a) light source remains stationary, b) light source moves with viewer.*



(a) Gouraud (b) Phong

(c) Reflective (d) Refractive

Figure 3: *Different shaders produce different rendering*

surrounding lightings. All examples shown in Fig 3 are in view angle ranging in perspective projection rendered with skybox technique, which simulates a distant background of natural scenery. It is worth noting that environment skybox is not available to orthographic projection, where projection rays are parallel, because there is no perspective distortion and hence no size difference will be observed between the skybox and the object.

In addition to visual effects, the shading mechanism also affects the frame rate (in units frames per second (fps)) of the rendering. From Table 3, it can be observed that Gouraud shading has a frame rate aroud 1.6 times higher than the Phong shading scheme. This agrees with expectations because Gouraud shading is a vertex-based calculation, which inherently reduces the computational complexity and GPU workload than the Phong shading where calculation is performed per pixel.

Another noticeable feature is the frame rate difference when the viewing distance is adjusted. As shown in the example of a reflective shader in Table 4, the frame rate gradually decreases as the viewer proceeds towards the object in fly-through mode. This suggests that the reflective shading mechanism is computationally expensive, especially when the viewing distance is close to the object, as the reflection and refraction effects are more pronounced and require more complex calculations. It is also shown as a baseline for comparison that the frame rate is significantly higher when the surface is completely ignored, as the rendering workload is reduced.

Shader Type	Average Frame Rate (fps)
Gouraud	4890
Phong	2980

Table 3: *Frame rates of Gouraud and Phong shading*

Viewing Distance	Average Frame Rate (fps)
Far (Original)	580
Close	170
Ignore Surface	8620

Table 4: *Frame rates of reflective shading at different viewing distances*

3 Conclusion

In this lab exercise, the reslicing, surface extraction, and rendering of 3D data for visualisation is explored. It is observed that the nearest neighbor interpolation is computationally efficient but may not yield optimal results for smooth surfaces in natural objects. The choice of interpolation resolution, intensity threshold, and rendering settings significantly impact surface quality and computational cost. Additionally, different shading schemes would also affect visual effects and frame rate, which are important for assessing rendering efficiency. As each of the above mentioned facets of 3D graphics is only touched upon in this exercise, further explorations remain to be conducted to compare other more complex 3D imaging techniques for better visualisation results and efficiency.