Team -20

COMPUTER GRAPHICS PROJECT

REAL TIME VOLUME RENDERING WITH TRANSFER FUNCTION AND SHADING

Presented by Deepanshu Dabas (2021249)

About Volume Rendering

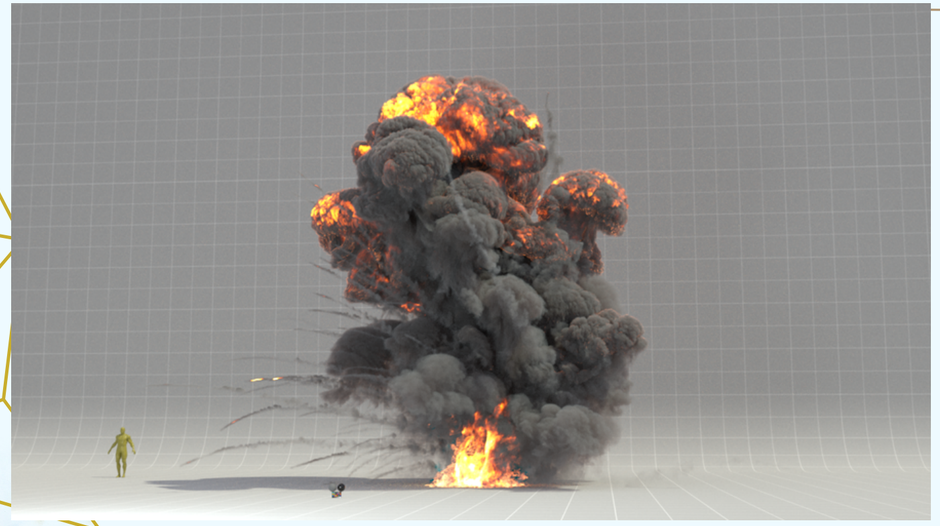


Fig 1: Explosion Effect

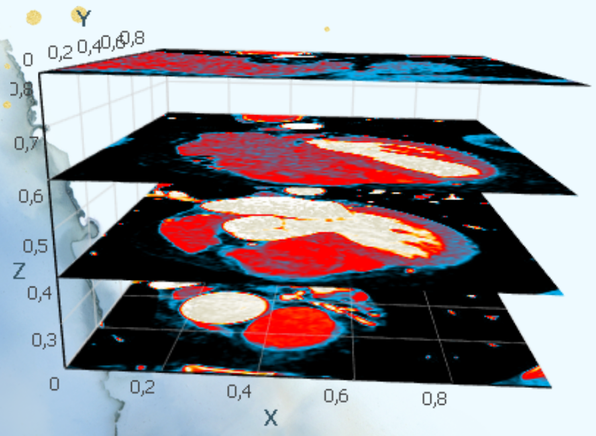
Fig 2: Heart Visualisation

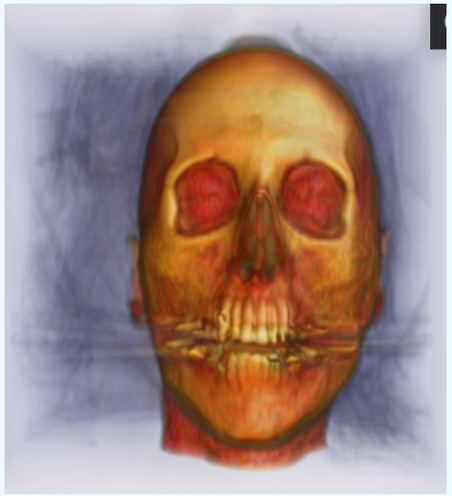
Volume rendering represents a collection of methods used in computer graphics and scientific visualization to create a 2D projection from a discretely sampled data set.

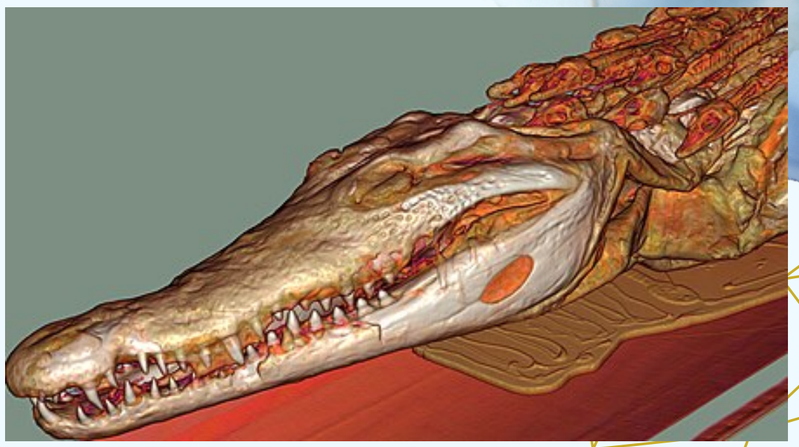
Source : Delight Cloud , Atomicus Chart

Use Case: Scientifc Visualization

Volume rendering in scientific visualization enables the analysis and exploration of complex data sets, generated by X-rays and CT-Scans,etc







Source: Wikipedia , Atomicus Chart

Use Case: Computer Graphics

Volume rendering can produce realistic and/or observable reprentations of volumetric phenomena, such as clouds, fog, smoke, fire, etc. by modeling the interaction of light with the medium inside the volume

Source: Scratch Pixel

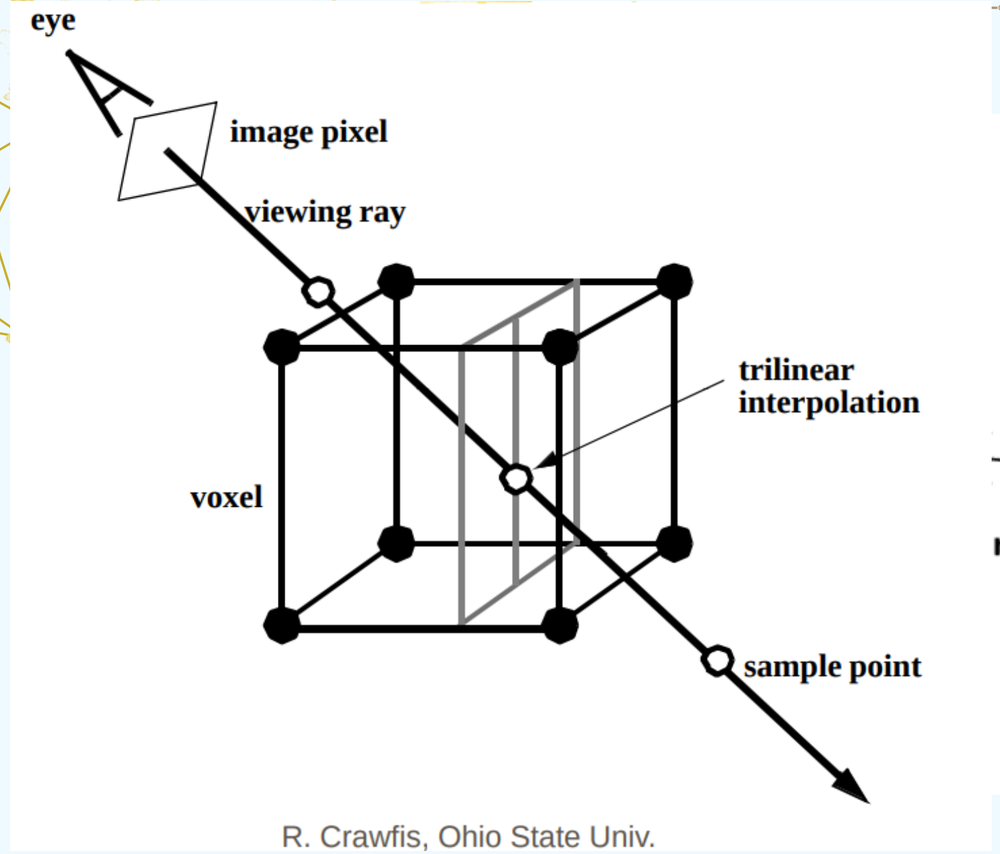
Why Volume Ray Casting?

- Image Based Volume Rendering
- Derived directly from the rendering equation.
- Produces results of very high quality rendering.
- Simulated rays are traversed iteratively,
- Often used in cases where creating explicit geometry is not a good option.. Example: Triangles

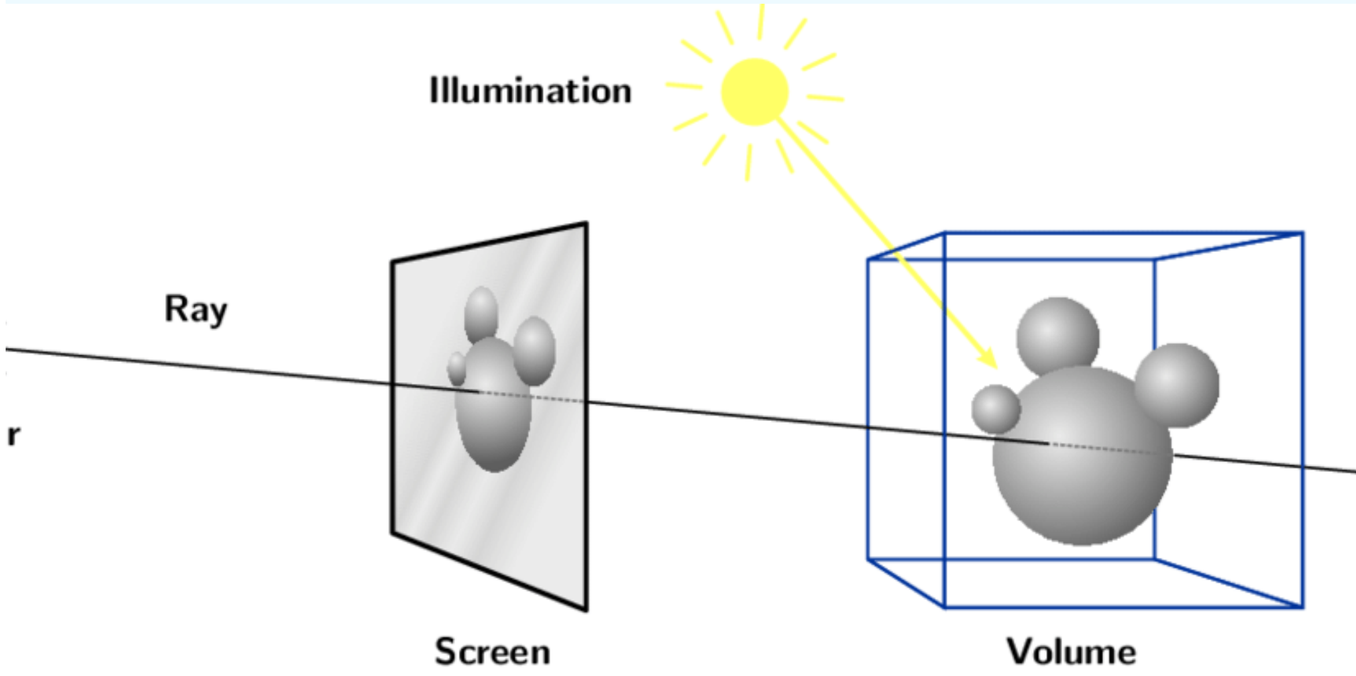
Steps For Volume Rendering

- **Ray Casting:** Shoot a ray through the volume for each pixel of using a bounding primitive to intersect the ray with the volume.
- **Sampling:** Select equidistant sampling points along the ray within the volume, interpolating values from surrounding voxels using trilinear interpolation1.
- **Shading:** Apply a transfer function to retrieve RGBA material color and compute a gradient of illumination values for each sample, shading them according to their orientation and light source location2
- **Compositing:** Composite shaded samples along the ray to determine the final pixel color, using either back-to-front or front-to-back order to ensure proper layering and efficiency.

Steps For Volume Rendering



R. Crawfis, Ohio State Univ.



Pseudo Code : Volume Ray Casting

Algorithm 1 Volume Rendering Shader Pseudocode

```
function VOLUME RAY CASTING
    // Calculate ray direction
    dirn ← CalculateRayDirection()
    startPoint ← eye
    direction ← Normalize(u · xw + v · yw − dirn · w)

    // If the ray does not hit the volume, return black
    if not LiangBarsky(startPoint, direction) then
        return vec4(0.0, 0.0, 0.0, 0.0)
    end if
    accColor ← vec4(0.0, 0.0, 0.0, 0.0)
    i ← 1
    currPoint ← start

    while True do
        // Propagate the ray
        p ← startPoint + direction · currPoint
        if currPoint > endPoint then
            break
        end if
        if accColor.a > 0.95 then
            break
        end if

        // Trilinear interpolation
        sample ← texture(volumeTexture, (p + (tMax − tMin)/2)/(tMax − tMin))

        // Transfer function color
        transferFuncColor ← texture(transferfun, sample.r)

        // Normal vector from normal texture
        normalFromTexture ← texture(normalTexture, gl_FragCoord.xyz).rgb
        normal ← Normalize(normalFromTexture · 2.0 − 1.0)

        // Bing Phong shading
        normal_mag ← Length(normal)
        if normal_mag > 0.01 and currPoint > stepSize then
            transferFuncColor ← BingPhongShading(p, transferFuncColor, −dir, Normalize(normal))
        end if
        if transferFuncColor.a > 0.0 then
            accColor ← accColor + (1.0 − accColor.a) · transferFuncColor · sample.r
        end if
        currPoint ← currPoint + stepSize
    end while

    return accColor
end function
```


Note:

- For tri-linear interpolation ,3 D texture is used along with filltering method Mipmapping
- Early Ray termination based optimization is performed
- Normals for shading are calculated using central difference method
- Front to back composition is performed for better optimization

Pseudo Code : Liang-Barsky Algo

Algorithm 2 Liang-Barsky Algorithm

```
1:  function LiangBarsky(startPoint, direction)
2:      tuMin ← 0.0, tuMax ← 0.0, tvMin ← 0.0, tvMax ← 0.0
3:      n ← 1 / direction
4:      if n.x < 0 then
5:          start ← (tMax.x − startPoint.x) × n.x
6:          endPoint ← (tMin.x − startPoint.x) × n.x
7:      else
8:          start ← (tMin.x − startPoint.x) × n.x
9:          endPoint ← (tMax.x − startPoint.x) × n.x
10:     end if
11:     if n.y < 0 then
12:         tuMin ← (tMax.y − startPoint.y) × n.y
13:         tuMax ← (tMin.y − startPoint.y) × n.y
14:     else
15:         tuMin ← (tMin.y − startPoint.y) × n.y
16:         tuMax ← (tMax.y − startPoint.y) × n.y
17:     end if
18:     if start > tuMax then
19:         return false
20:     end if
21:     if tuMin > endPoint then
22:         return false
23:     end if
24:     start ← max(start, tuMin)
25:     endPoint ← min(endPoint, tuMax)
26:     if n.z < 0 then
27:         tvMin ← (tMax.z − startPoint.z) × n.z
28:         tvMax ← (tMin.z − startPoint.z) × n.z
29:     else
30:         tvMin ← (tMin.z − startPoint.z) × n.z
31:         tvMax ← (tMax.z − startPoint.z) × n.z
32:     end if
33:     start ← max(start, tvMin)
34:     endPoint ← min(endPoint, tvMax)
35:     if start ≤ 0 or endPoint ≤ 0 or start ≥ endPoint then
36:         return false
37:     else
38:         return true
39:     end if
40: end function
```

Pseudo Code : Bing-Phong Shading

Algorithm 3 Bing Phong Shading Pseudocode

```
1:  function BingPhongShading(fPos, fColor, dir, normal)
2:      // Directional light properties
3:      lightDir ← Normalize(vec3(1.0, 1.0, 1.0))          ▷ Light direction
4:      lightColor ← vec3(1.0, 1.0, 1.0)                   ▷ Light color
5:      diff ← Max(Dot(Normalize(normal), −lightDir), 0.0)
6:
7:      // Calculate the diffuse component
8:      diffuse ← fColor.rgb × diff × lightColor
9:
10:     // Calculate the specular component
11:     reflected ← Reflect(−lightDir, normal)
12:     specularIntensity ← Max(Dot(dir, reflected), 0.0)
13:     specular ← vec3(Pow(specularIntensity, shineConst))
14:
15:     // Return the color and alpha value
16:     return vec4(diffuse + specular, fColor.a)
17: end function
```

Tech Stack Used

- OpenGL
- ImGui
- glfw
- glm
- Cmake
- Make

Outcomes

- Volumetric Rendering Techniques: Explored methods for creating 3D images from volumetric data, crucial in fields like medical imaging and special effects.
- Transfer Functions: Investigated the use of transfer functions in rendering to differentiate materials within a volume, enhancing image accuracy and utility.
- Real-Time Rendering: Implemented real-time volume rendering using graphics hardware acceleration, optimizing performance through techniques like early ray termination.
- Shading and Composition: Applied surface shading and composition algorithms to add realism to the images and accurately represent 3D volumetric data.
- Learned more about OpenGL and gl_shaders and overall improved understanding of computer graphics

Additional Features

Edit Transfer Functions

User can edit transfer function values using color paletter provided as input

Load/Save Transfer Function

User can load pre exisitng transfrer fiunctions as well as save current by specifying name of Output file

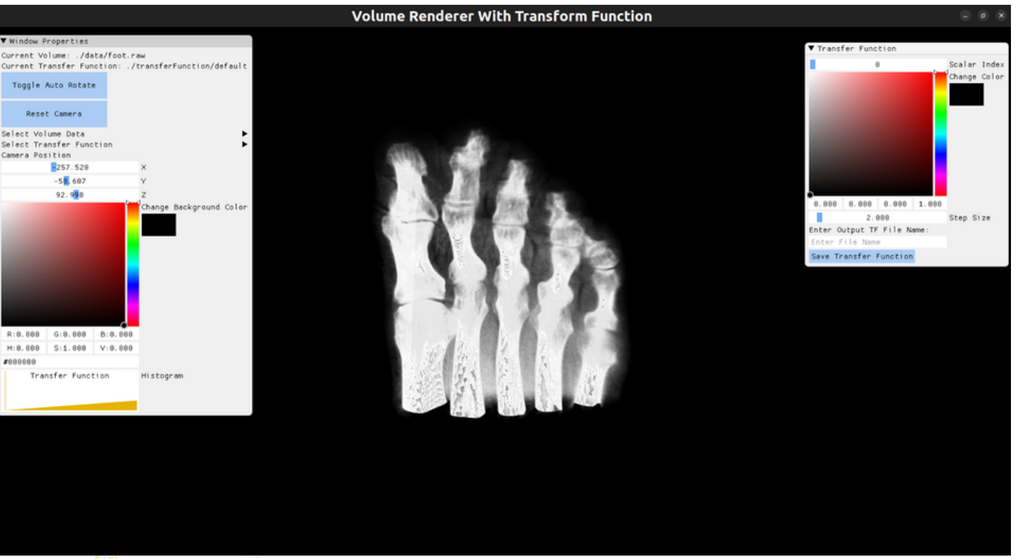
Transformations

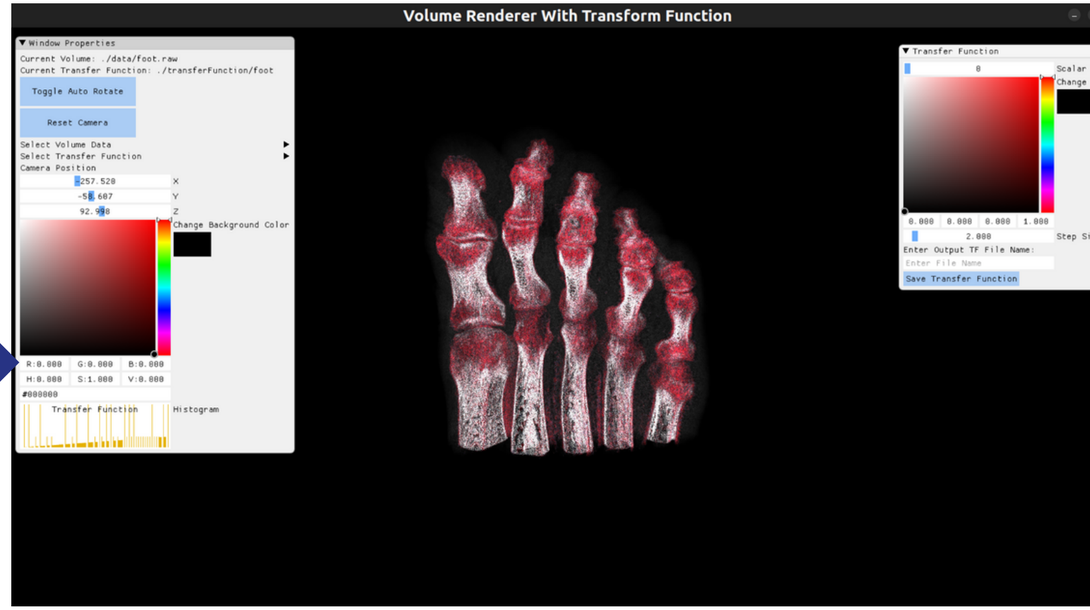
Rotation, View Transformations,,etc

Change Volume

Need to just Add dataset to visualise in Data folder

Output Images



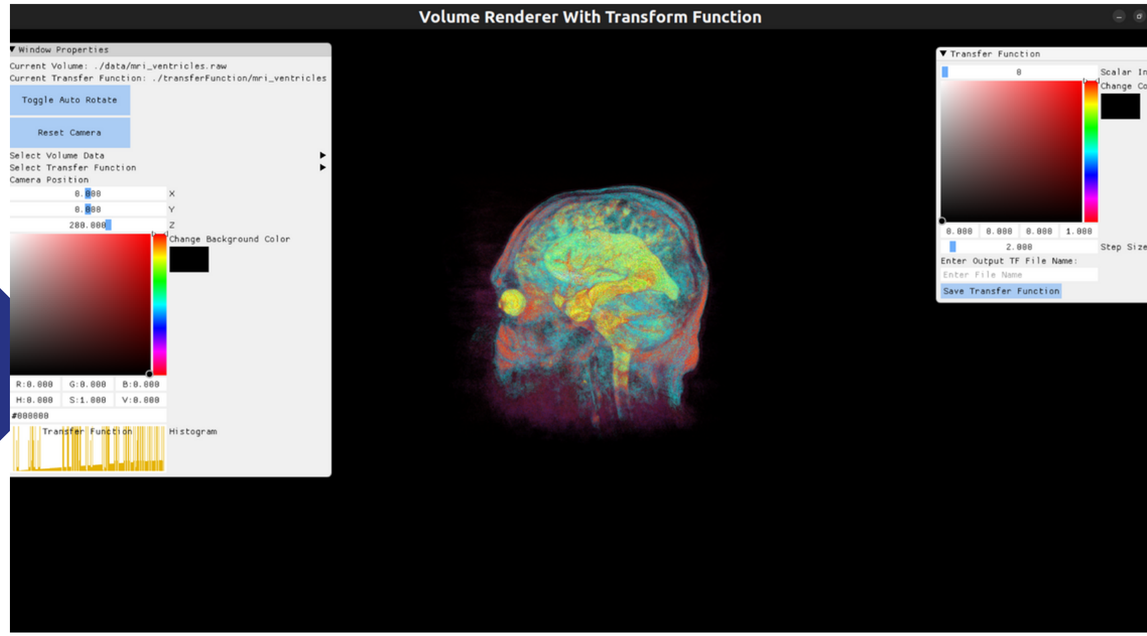


GrayScale Based Transfer Function

Foot-Custom Transfer Function

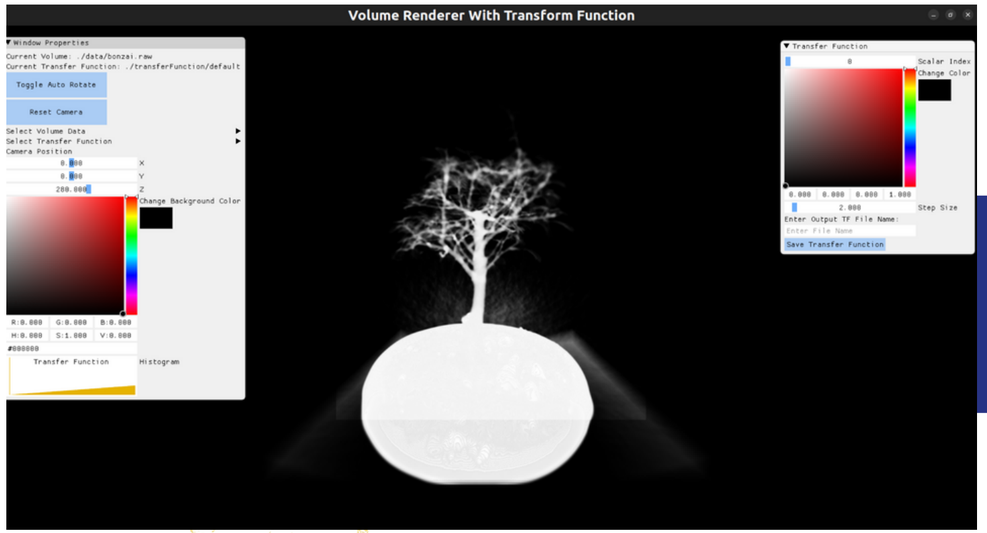
Output Images

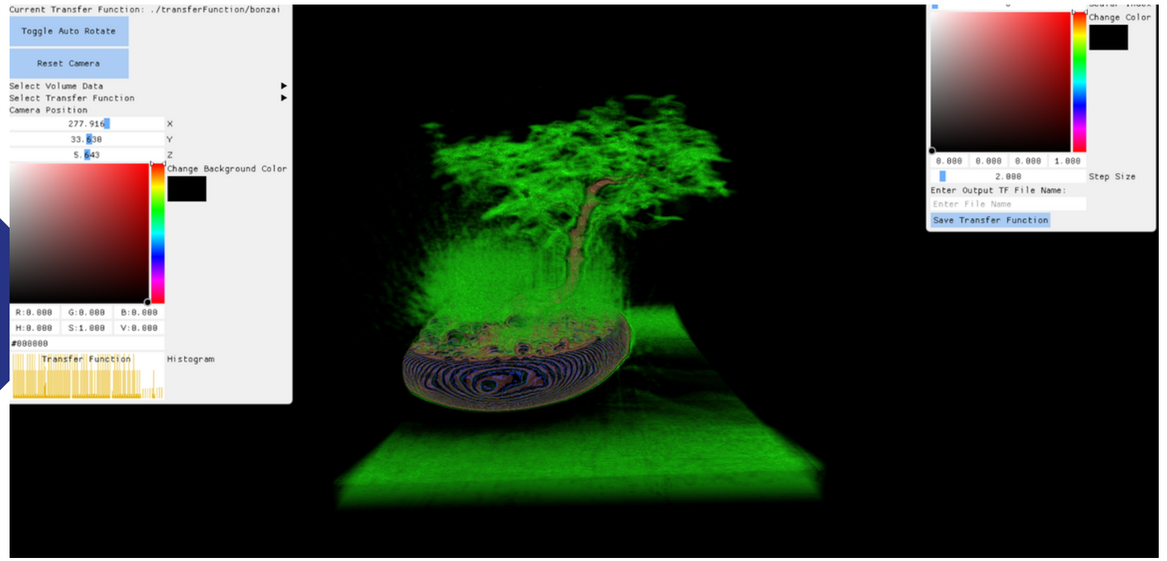
GrayScale Based Transfer Function



MRI--Custom Transfer Function

Output Images





GrayScale Based Transfer Function

Bonzai--Custom Transfer Function

THANK YOU

Team 20, Deepanshu Dabas, 2021249