

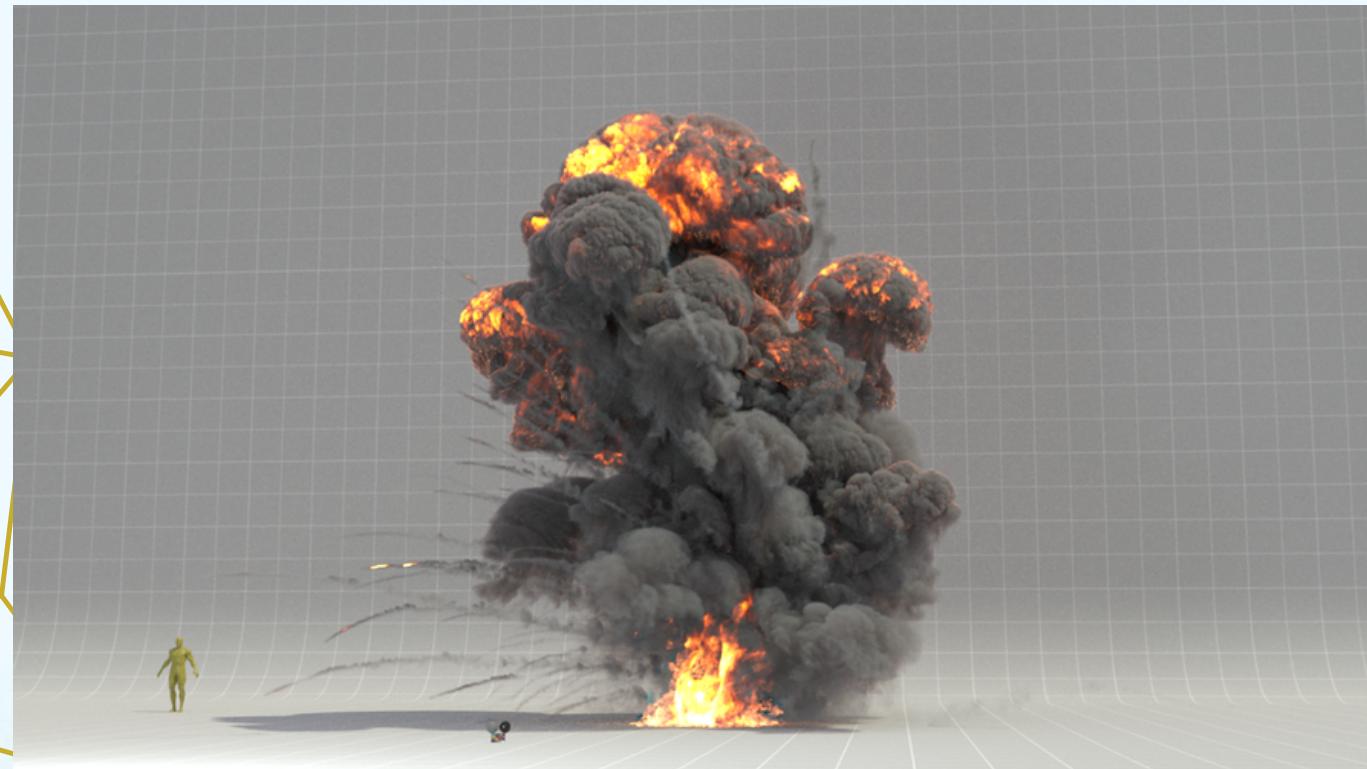


# COMPUTER GRAPHICS PROJECT

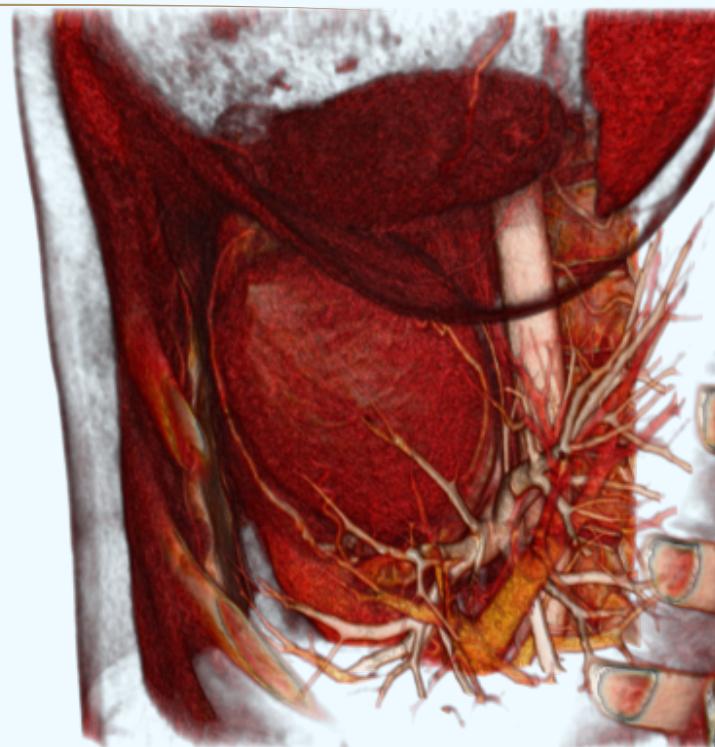
**REAL TIME VOLUME RENDERING  
WITH TRANSFER FUNCTION AND SHADING**

Presented by Deepanshu Dabas (2021249)

# About Volume Rendering



**Fig 1: Explosion Effect**

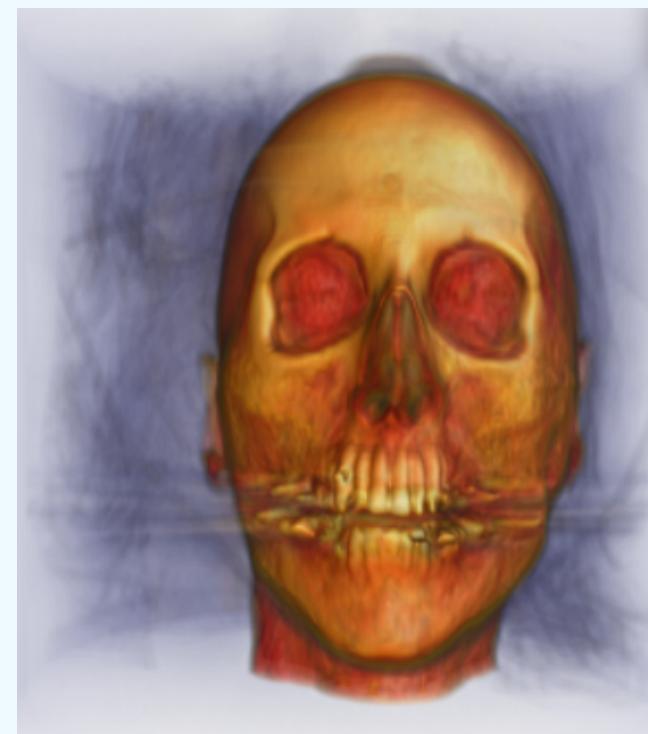
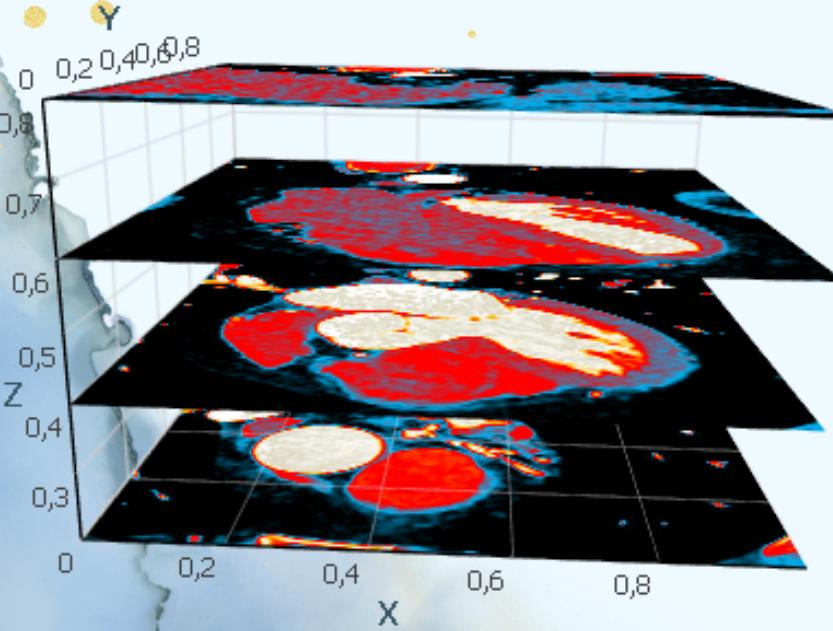


**Fig 2: Heart Visualisation**

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.

# Use Case: Scientific Visualization

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



# Use Case: Computer Graphics

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



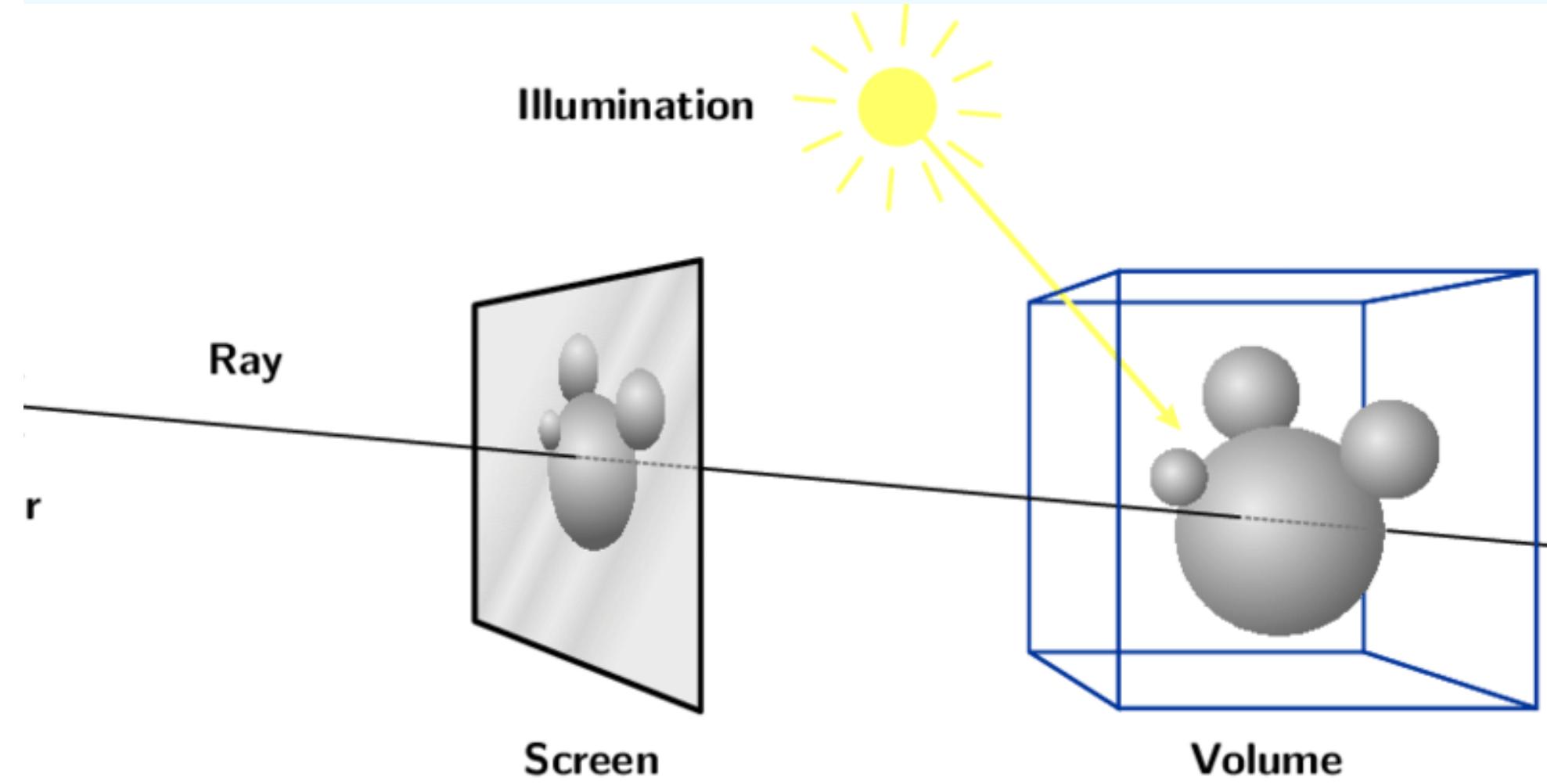
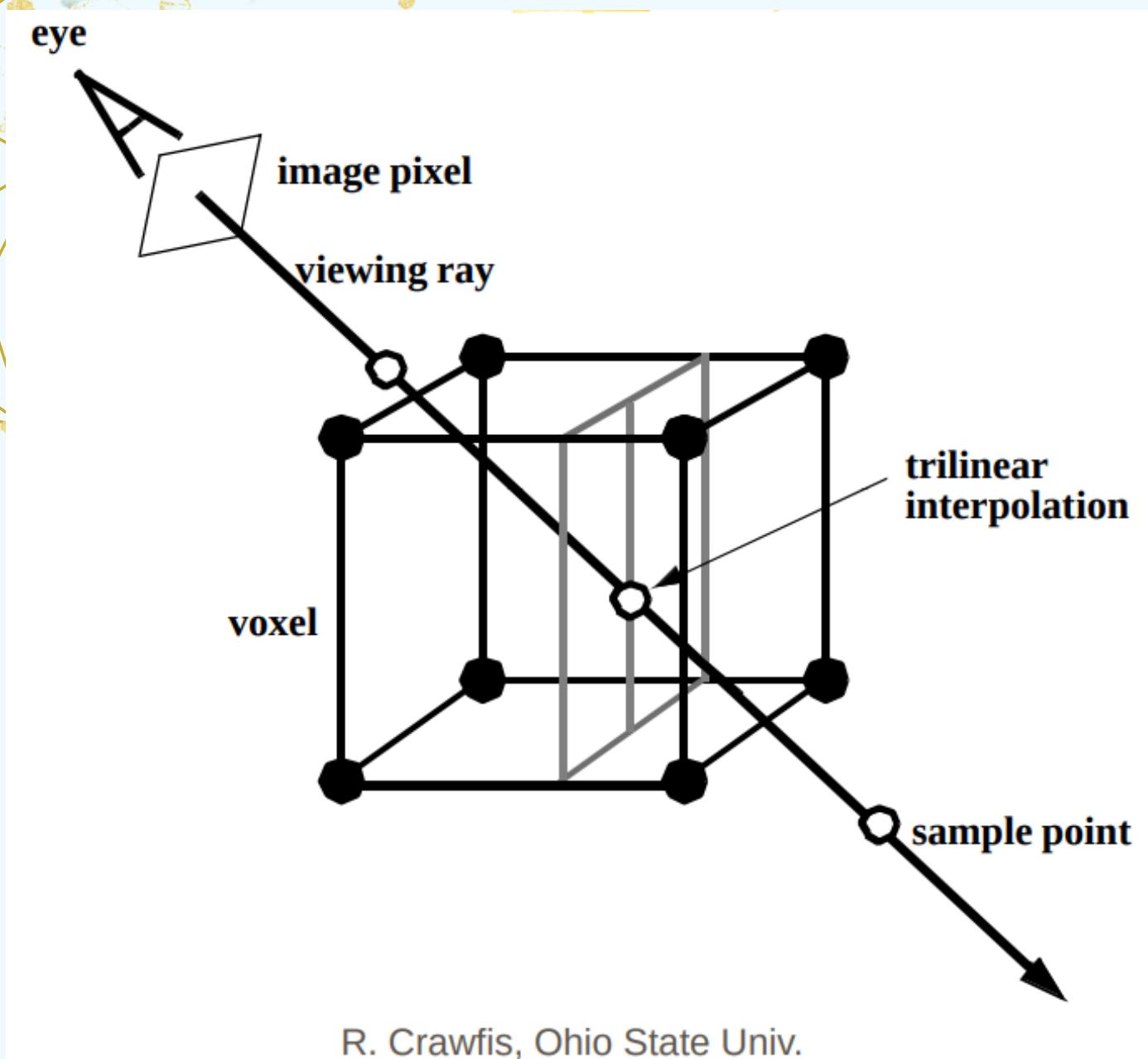
# Why Volume Ray Casting?

- 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 interpolation<sup>1</sup>.
- **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 location<sup>2</sup>
- **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



# Pseudo Code : Volume Ray Casting

## Algorithm 1 Volume Rendering Shader Pseudocode

```
1: function VOLUME RAY CASTING
2:   // Calculate ray direction
3:   dirn ← CalculateRayDirection()
4:   startPoint ← eye
5:   direction ← Normalize( $u \cdot xw + v \cdot yw - dirn \cdot w$ )
6:
7:   // If the ray does not hit the volume, return black
8:   if not LiangBarsky(startPoint, direction) then
9:     return vec4(0.0, 0.0, 0.0, 0.0)
10:  end if
11:  accColor ← vec4(0.0, 0.0, 0.0, 0.0)
12:  i ← 1
13:  currPoint ← start
14:
15:  while True do
16:    // Propagate the ray
17:    p ← startPoint + direction · currPoint
18:    if currPoint > endPoint then
19:      break
20:    end if
21:    if accColor.a > 0.95 then
22:      break
23:    end if
24:
25:    // Trilinear interpolation
26:    sample ← texture(volumeTexture, (p + (tMax - tMin)/2)/(tMax - tMin))
27:
28:    // Transfer function color
29:    transferFuncColor ← texture(transfertfun, sample.r)
30:
```

```
31:   // Normal vector from normal texture
32:   normalFromTexture ← texture(normalTexture, gl_FragCoord.xyz).rgb
33:   normal ← Normalize(normalFromTexture · 2.0 - 1.0)
34:
35:   // Bing Phong shading
36:   normal_mag ← Length(normal)
37:   if normal_mag > 0.01 and currPoint > stepSize then
38:     transferFuncColor ← BingPhongShading(p, transferFuncColor, -dir, Normalize(normal))
39:   end if
40:   if transferFuncColor.a > 0.0 then
41:     accColor ← accColor + (1.0 - accColor.a) · transferFuncColor · sample.r
42:   end if
43:   currPoint ← currPoint + stepSize
44: end while
45:
46: return accColor
47: end function
```

, Vol. 1, No. 1, Article . Publication date: December 202

## 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  $\leftarrow$  0.0, tuMax  $\leftarrow$  0.0, tvMin  $\leftarrow$  0.0, tvMax  $\leftarrow$  0.0
3:   n  $\leftarrow \frac{1}{\text{direction}}$ 
4:   if n.x < 0 then
5:     start  $\leftarrow (t_{\text{Max}}.x - \text{startPoint}.x) \times n.x$ 
6:     endPoint  $\leftarrow (t_{\text{Min}}.x - \text{startPoint}.x) \times n.x$ 
7:   else
8:     start  $\leftarrow (t_{\text{Min}}.x - \text{startPoint}.x) \times n.x$ 
9:     endPoint  $\leftarrow (t_{\text{Max}}.x - \text{startPoint}.x) \times n.x$ 
10:  end if
11:  if n.y < 0 then
12:    tuMin  $\leftarrow (t_{\text{Max}}.y - \text{startPoint}.y) \times n.y$ 
13:    tuMax  $\leftarrow (t_{\text{Min}}.y - \text{startPoint}.y) \times n.y$ 
14:  else
15:    tuMin  $\leftarrow (t_{\text{Min}}.y - \text{startPoint}.y) \times n.y$ 
16:    tuMax  $\leftarrow (t_{\text{Max}}.y - \text{startPoint}.y) \times n.y$ 
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  $\leftarrow \max(\text{start}, \text{tuMin})$ 
25:  endPoint  $\leftarrow \min(\text{endPoint}, \text{tuMax})$ 
26:  if n.z < 0 then
27:    tvMin  $\leftarrow (t_{\text{Max}}.z - \text{startPoint}.z) \times n.z$ 
28:    tvMax  $\leftarrow (t_{\text{Min}}.z - \text{startPoint}.z) \times n.z$ 
29:  else
30:    tvMin  $\leftarrow (t_{\text{Min}}.z - \text{startPoint}.z) \times n.z$ 
31:    tvMax  $\leftarrow (t_{\text{Max}}.z - \text{startPoint}.z) \times n.z$ 
32:  end if
33:  start  $\leftarrow \max(\text{start}, \text{tvMin})$ 
34:  endPoint  $\leftarrow \min(\text{endPoint}, \text{tvMax})$ 
35:  if start  $\leq 0$  or endPoint  $\leq 0$  or start  $\geq$  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))
4:   lightColor ← vec3(1.0, 1.0, 1.0)
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
```

---

► Light direction  
► Light color

# 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 palettes provided as input



## Load/Save Transfer Function

User can load pre-existing transfer functions as well as save current by specifying name of Output file



## Transformations

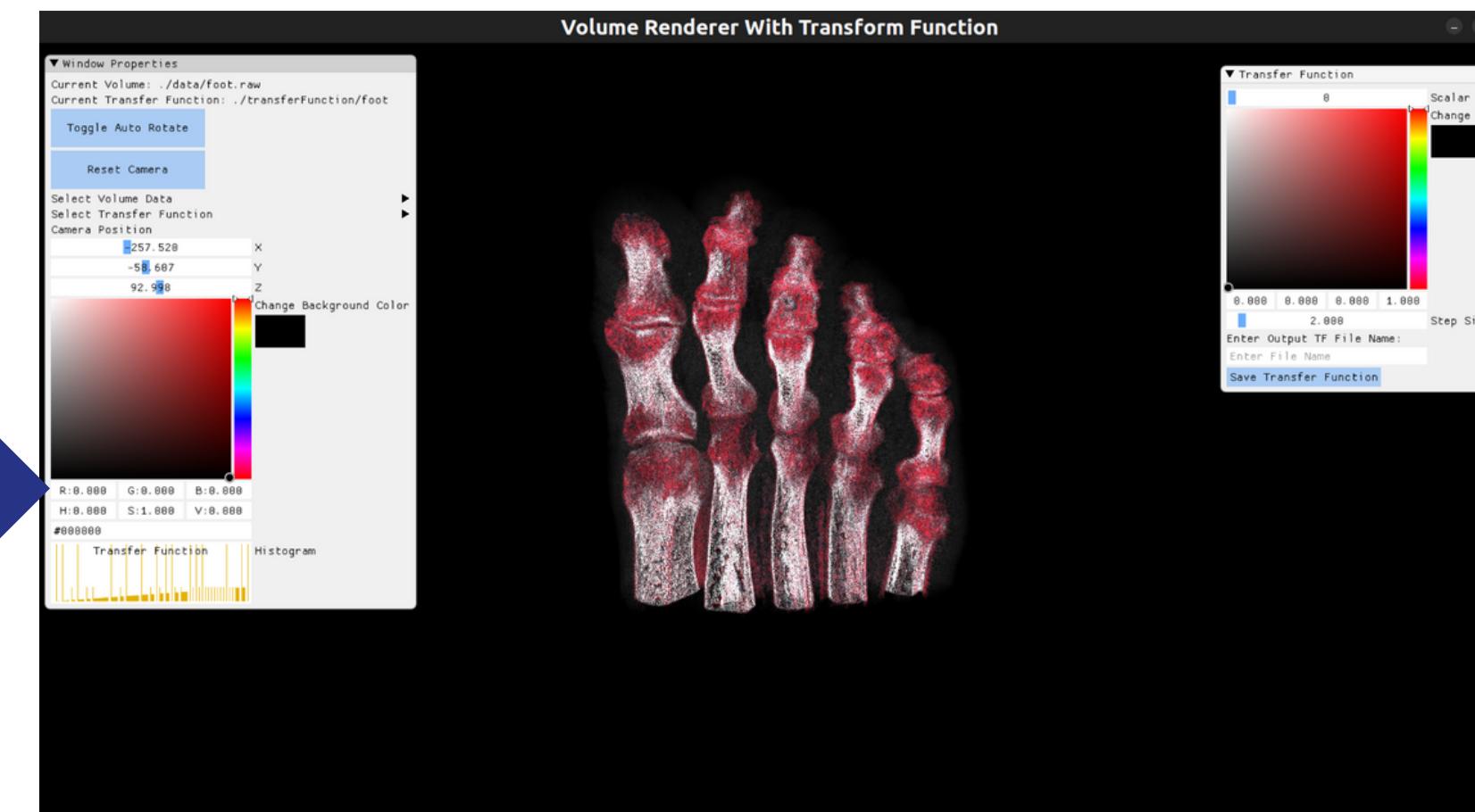
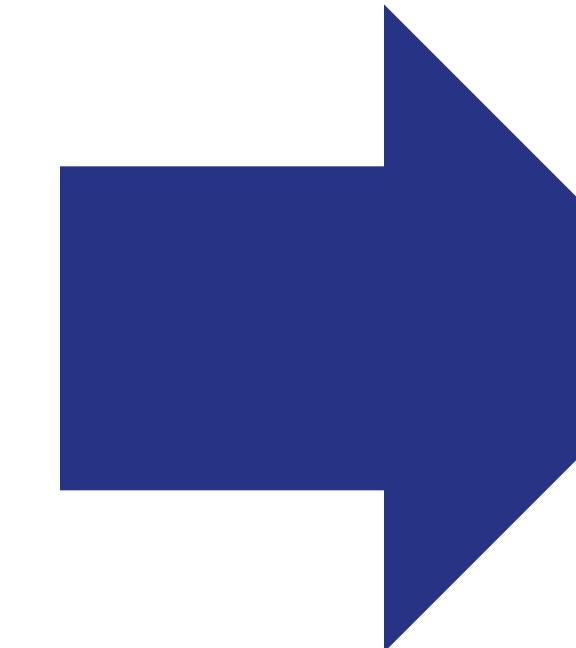
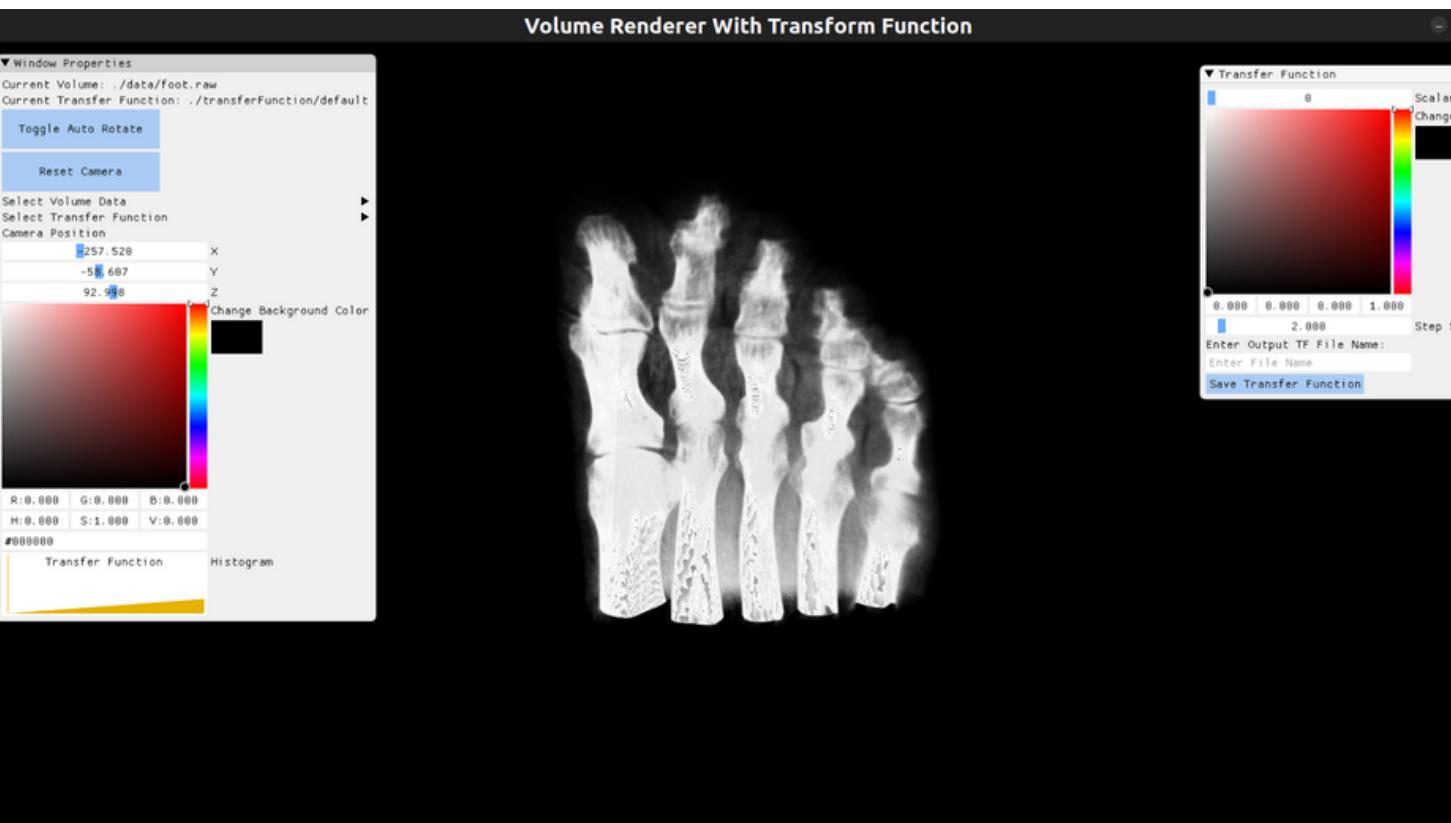
Rotation, View Transformations, etc



## Change Volume

Need to just Add dataset to visualise in Data folder

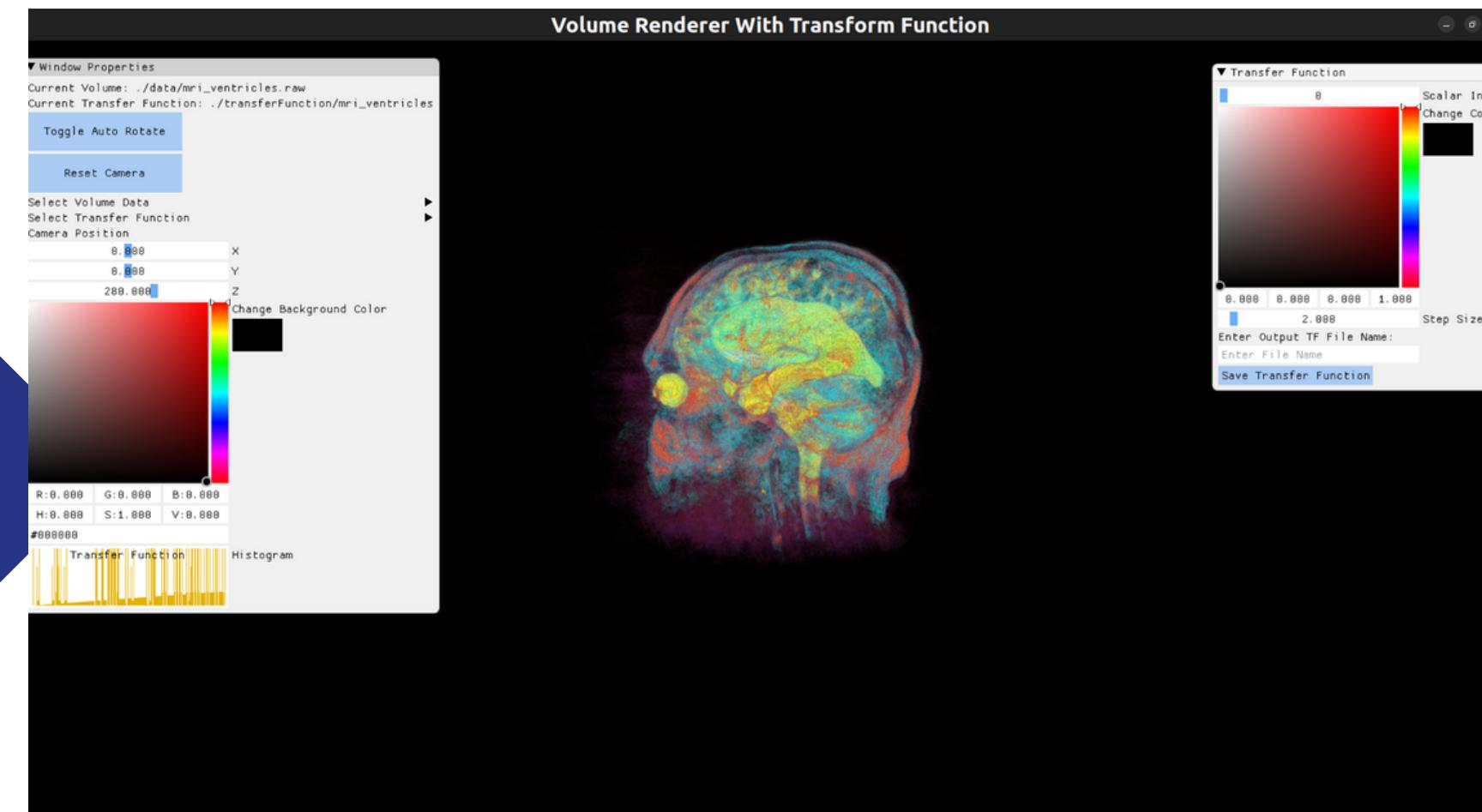
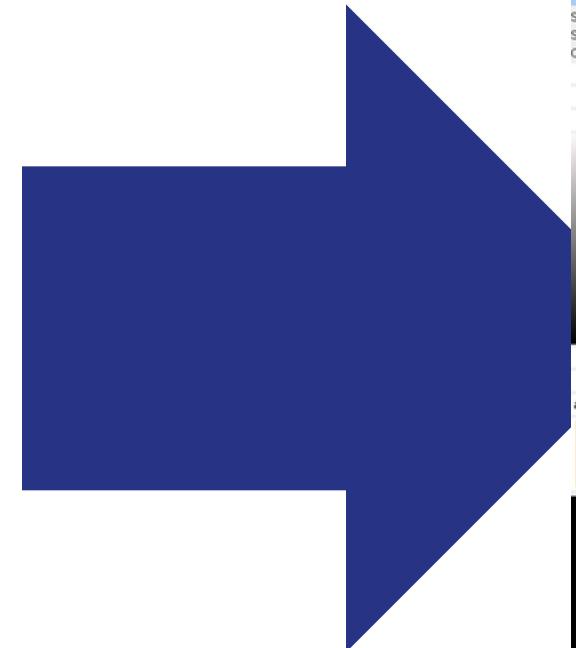
# Output Images



GrayScale Based Transfer Function

Foot-Custom Transfer Function

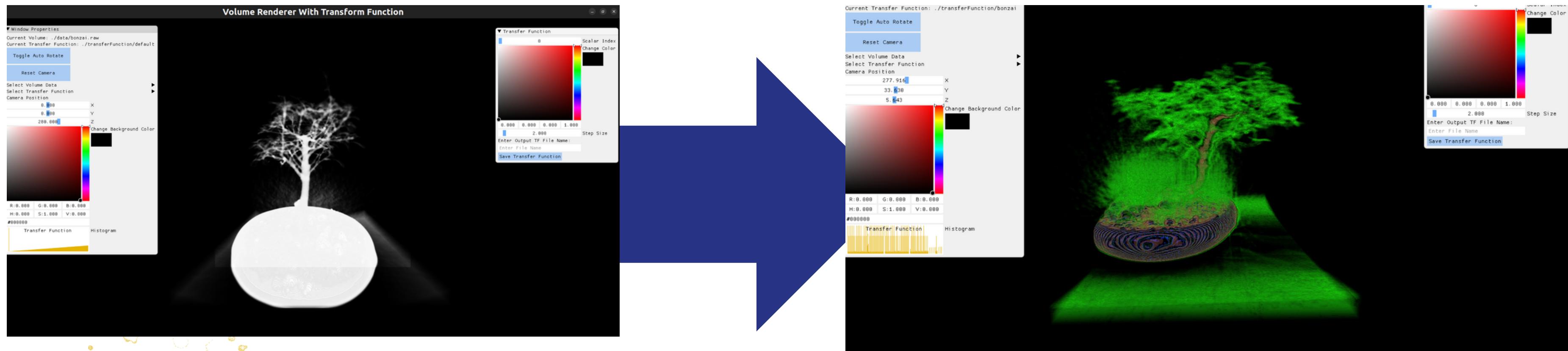
# Output Images



GrayScale Based Transfer Function

MRI--Custom Transfer Function

# Output Images



GrayScale Based Transfer Function

MRI--Custom Transfer Function

# THANK YOU

Team 20, Deepanshu Dabas, 2021249