Volume Rendering

Ray Casting
Perspective Projection

- Ray specified by
  - Eye position
  - Pixel position on image plane

- Ray dir computed as in ray-tracing
Orthogonal Projection

• All rays parallel
• Ray direction determined by view direction
Some Details

- Image order ray casting
  - Scan image row by row, column by column
  - $P_{i,j}$ is the pixel location
  - $P_{0,0}$ image origin
  - $u$, $v$ image plane coordinate vectors
  - $\Delta i$ and $\Delta j$ determined by image resolution and width, height of the image plane

$$P_{i,j} = P_{0,0} + i \cdot \vec{v} \cdot \Delta i + j \cdot \vec{u} \cdot \Delta j$$
Ray Casting

- Pixel color computed by compositing along the ray from the pixel
- X-Ray type:
  - Accumulate volumetric shading results $f(x,y)$ along the ray
- MIP type:
  - Find the maximum $f(x,y)$
Discrete Form

- Practical volume rendering:
  - Discrete data set on a grid
  - Volumetric integral implemented by compositing together at all sampling steps along ray direction
Sampling Point Interpolation

• Ray sampling points do not always fall onto the known grid points.
• Most often falls in between
• Interpolation: estimate the values of samples from the existing regular grid points
• Various interpolation methods can be applied
Interpolation Kernel

- Interpolation kernel centered at sample point
- Convolution
  - the values of grid points * Kernel value
Popular Kernels

- Nearest neighbor:
  - Simply pick the value of nearest grid point

- Linear filter:
  - Use linear combination in a small neighborhood
Popular Interpolation Kernels

- Cubic filter: \((B=1, C=0)\)

\[
h_{BC}(x) = \begin{cases} 
(12 - 9B - 6C)|x|^3 + (-18 + 12B + 6C)|x|^2 + (6 - 2B)|x| & \text{if } |x| < 1 \\
(-B - 6C)|x|^3 + (6B + 30C)|x|^2 + (-12B - 48C)|x| + (8B + 24C) & \text{if } 1 \leq |x| < 2 \\
0 & \text{else}
\end{cases}
\]
Discussions

- Nearest is the fastest with SHARP edges
- Cubic filter gives smooth results but with slow computation
- Linear filter is most popular in volume rendering
Practical 3D Linear Interpolation

• All interpolation kernels here can be separated in 3 dimensions
  \[ H(x, y, z) = H(x)H(y)H(z) \]
• Linear interpolation in 1D

\[ f(x) = f(P_u) = (1 - u) \cdot f(P_0) + u \cdot f(P_1) \]
Bilinear Interpolation

- **2D case:**

\[
\begin{align*}
    f(P_{0,u}) &= (1 - u) \cdot f(P_{0,0}) + u \cdot f(P_{0,1}) \\
    f(P_{1,u}) &= (1 - u) \cdot f(P_{1,0}) + u \cdot f(P_{1,1}) \\
    f(P_{v,u}) &= (1 - v) \cdot f(P_{0,u}) + v \cdot f(P_{1,u}) \\
    f(x, y) &= f(P_{v,u}) = (1-v) (1-u) f(P_{0,0}) + (1-v) u f(P_{0,1}) + v (1-u) f(P_{1,0}) + v u f(P_{1,1})
\end{align*}
\]
Trilinear Interpolation

- General 3D case
  - Can be decomposed into 7 linear interpolations

\[
f(P_{0,v,u}) = \text{BilinearInterpolation}(P_{0,0,0}, P_{0,0,1}, P_{0,1,0}, P_{0,1,1})
\]
\[
f(P_{1,v,u}) = \text{BilinearInterpolation}(P_{1,0,0}, P_{1,0,1}, P_{1,1,0}, P_{1,1,1})
\]
\[
f(x, y, z) = f(P_{w,v,u}) = \text{LinearInterpolation}(P_{0,v,u}, P_{1,v,u})
\]
\[
f(x, y, z) = \sum_{i=0}^{1} \sum_{j=0}^{1} \sum_{k=0}^{1} u^i (1-u)^{1-i} v^j (1-v)^{1-j} w^k (1-w)^{1-k} P_{i,j,k}
\]
Trilinear Interpolation

To compute fDens of an arbitrary position \((x,y,z)\)

\[
\text{int } i = \text{floor}(x); \text{ int } j = \text{floor}(y); \text{ int } k = \text{floor}(z);
\]
\[
\text{float } u,s,w;
\]
\[
u = x - i; s = y - j; w = z - k;
\]
\[
fMass[1]=\text{GetDensity}(i,j,k); \quad fMass[2]= \text{GetDensity}(i+1,j,k);
\]
\[
fMass[3]= \text{GetDensity}(i+1,j,k+1); \quad fMass[4]= \text{GetDensity}(i,j,k+1);
\]
\[
fMass[5]= \text{GetDensity}(i,j+1,k); \quad fMass[6]= \text{GetDensity}(i+1,j+1,k);
\]
\[
fMass[7]= \text{GetDensity}(i+1,j+1,k+1); \quad fMass[8]= \text{GetDensity}(i,j+1,k+1);
\]
\[
fDens = fMass[1]*(1 - u)*(1 - s)*(1 - w) + 
\]
\[
fMass[2]*u*(1 - s)*(1 - w) + 
\]
\[
fMass[3]*u*(1 - s)*w + 
\]
\[
fMass[4]*(1 - u)*(1 - s)*w + 
\]
\[
fMass[5]*(1 - u)*s*(1 - w) + 
\]
\[
fMass[6]*s*u*(1 - w) + 
\]
\[
fMass[7]*s*w*u + 
\]
\[
fMass[8]*(1-u)*s*w;
\]
X-Ray Ray Casting

RenderXRay(Volume V, int stepSize, Image I)

if (projectionMode == Orthographic)

    ray = v × u / |v × u| /* view direction vector is perpendicular to image plane */

for each image pixel (i, j)

    P(i, j) = P(0, 0) + i · v · Δi + j · u · Δj;

    sum = 0;

    if (projectionMode == Perspective)

        ray = (P(i, j) - eye) / |P(i, j) - eye| /* normalized view direction vector */

    IntersectRayWithVolumeBoundingBox(V, ray, t_front, t_back);

    for(t = t_front; t ≤ t_back; t += stepSize) /* traverse the volume front to back */

        sampleLoc = P(i, j) + t · ray /* step along the ray */

        intVal = Interpolate(V, sampleLoc);

        sum += intVal · stepSize; /* add interpolated value to X-ray sum */

    I(i, j) = sum;

NormalizeImage(I);
MIP Ray Casting

RenderMIP(Volume V, int stepSize, Image I)

if (projectionMode == Orthographic)

    ray = v \times u / | v \times u| /* view direction vector is perpendicular to image plane */

for each image pixel (i, j)

    P(i, j) = P(0, 0) + i \cdot v \cdot \Delta i + j \cdot u \cdot \Delta j;

if (projectionMode == Perspective)

    ray = (P(i, j) - eye) / | (P(i, j) - eye) | /* the ray direction vector, normalized */

max = 0;

IntersectRayWithVolumeBoundingBox(V, ray, t_front, t_back);

for(t = t_front; t <= t_back; t += stepSize) /* traverse the volume front to back */

    sampleLoc = P(i, j) + t \cdot ray /* step along the ray */

    intval = Interpolate(V, sampleLoc);

    if(intVal > max)

        max = intval;

I(i, j) = max;