Indirect Volume Rendering

Balázs Csébfalvi
Department of Control Engineering and Information Technology
Budapest University of Technology and Economics

Classification of visualization algorithms

- Direct Volume Rendering (DVR): The rendering engine can directly process the volumetric data.
- Indirect Volume Rendering (IVR): An intermediate representation is required by the rendering engine:
  - 3D Fourier transform – Fourier Volume Rendering (FVR)
  - Random point cloud – Monte Carlo Volume Rendering (MCVR)
  - Triangular mesh of an isosurface – Marching Cubes algorithm

Fourier Volume Rendering

- Tomographic reconstruction
  - The input data is a set of projections
  - The 3D signal is reconstructed from the projection data on the points of a sampling lattice
- Fourier Volume Rendering
  - The inverse of the tomographic reconstruction
  - The input volumetric data is the result of the tomographic reconstruction
  - A projection image for a specific viewing direction is reproduced from the volume data
  - Simulated X-ray rendering
Tomographic reconstruction

Radon transform

The Fourier projection-slice theorem

The Fourier transform of the projection

\[ P_{\theta}(\omega) = \int_{-\infty}^{\infty} p_{\theta}(t) e^{-j2\pi\omega t} dt \]

Fourier projection-slice theorem:

\[ P_{\theta}(\omega) = F(\omega \cos \theta, \omega \sin \theta) = F(u, v) \]
\[ f(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} P_{\theta}(\omega) e^{j2\pi(ux + vy)} dudv \]
The 2D Fourier transform of \( f(x,y) \):

\[
F(u,v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y) e^{-j2\pi(ux+vy)} \, dx \, dy
\]

The Fourier transform of the projection:

\[
P_\theta(\omega) = \int_{-\infty}^{\infty} p_\theta(t,s) e^{-j2\pi \omega t} \, dt
\]

Coordinate transformation

\[
\begin{bmatrix} t \\ s \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}
\]

The projection in the transformed \((t, s)\) space:

\[
p_\theta(t) = \int_{-\infty}^{\infty} f(t,s) \, ds
\]

The Fourier projection-slice theorem

\[
P_\theta(\omega) = \int_{-\infty}^{\infty} p_\theta(t,s) e^{-j2\pi \omega t} \, dt = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(t,s) ds \cdot e^{-j2\pi \omega t} \, dt
\]

In the original \((x, y)\) space:

\[
P_\theta(\omega) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y) \cdot e^{-j2\pi \omega t} \, dx \, dy
\]

\[
t = x \cos \theta + y \sin \theta
\]
The Fourier projection-slice theorem

Fourier transform of the projection

\[ P_x(\omega) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) \cdot e^{-j2\pi x \omega \cos \theta + y \omega \sin \theta} \, dx \, dy \]

2D Fourier-transform of a slice

Fourier projection-slice theorem:

\[ P_x(\omega) = F(\omega \cos \theta, \omega \sin \theta) = F(u, v) \]

\[ f(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} P_x(\omega) e^{j2\pi (ux + vy)} \, du \, dv \]

Filtered Back Projection (FBP)

Integration by substitution:

\[ u = \omega \cos \theta, \quad v = \omega \sin \theta \]

\[ f(x, y) = \int_{0}^{2\pi} \int_{0}^{\infty} F(\omega, \theta) e^{j2\pi x \omega \cos \theta + y \omega \sin \theta} \, \omega \, d\omega \, d\theta \]

\[ \omega \text{ is the Jacobian determinant of the transformation} \]

Filtered Back Projection (FBP)

\[ f(x, y) = \int_{0}^{\pi} \int_{0}^{\infty} F(\omega, \theta) e^{j2\pi x \omega \cos \theta + y \omega \sin \theta} \, \omega \, d\omega \, d\theta + \int_{0}^{\pi} \int_{0}^{\infty} F(\omega, \theta + \pi) e^{j2\pi x \omega \cos (\theta + \pi) + y \omega \sin (\theta + \pi)} \, \omega \, d\omega \, d\theta \]

From the definition of the Fourier transform we obtain:

\[ F(\omega, \theta + \pi) = F(-\omega, \theta) \]
Filtered Back Projection (FBP)

\[ f(x, y) = \int_{0}^{\pi} F(\omega, \theta) \left| \omega \right| e^{i2\pi \sqrt{x^2 + y^2} \cos \theta} \sin \theta d\omega d\theta \]

\[ = \int_{0}^{\pi} P(\omega) \left| \omega \right| e^{i2\pi \sqrt{x^2 + y^2} \cos \theta} \sin \theta d\omega d\theta \]

Convolution filtering, where the transfer function is \(|\omega|\).

The transfer function of the filter

1. For each angle \(\theta\) compute the Fourier transform \(P_\theta\) of the corresponding projection \(p_\theta\).
2. Filtering in the frequency domain \((P_\theta\) is multiplied by \(|\omega|\)) – equivalent to a convolution in the spatial domain:

\[ q_\theta(t) = \int P_\theta(\omega) |\omega| e^{i2\pi \sqrt{x^2 + y^2} \cos \theta} d\omega \]

3. Back projection:

\[ f(x, y) = \int_{0}^{\pi} q_\theta(x \cos \theta + y \sin \theta) d\theta \]
Discrete approximation

1. For each angle $\theta$ compute the discrete Fourier transform $P_\theta$ of the corresponding projection $p_\theta$.
2. Filtering in the frequency domain ($P_\theta$ is multiplied by $|\omega|$)
3. Inverse discrete Fourier transform of the result of step 2.
4. Back projection as a finite sum:

$$f(x, y) = \sum_{i=1}^{N} q_\theta(x \cos \theta + y \sin \theta)$$

Fourier Volume Rendering

- 3D Fourier transform
- Slicing
  - Along a plane perpendicular to the viewing direction the 3D Fourier transform is resampled
- 2D inverse Fourier transform
- Based on the Fourier projection slice-theorem, the obtained image is the projection of the original 3D data

Complexity of FVR

- Assume that the volume resolution is $N^3$
- The complexity of the 3D DFT is $O(N^3 \log N)$ – performed only once in a preprocessing
- The complexity of the 2D slicing is $O(N^2)$ if a compact resampling filter is used
- The complexity of the 2D inverse Fourier transform is $O(N^2 \log N)$ – performed for each frame
- The complexity of the rendering is practically proportional to the number of the pixels
- The traditional methods (ray casting, splatting) visit all the voxels; therefore, their complexity is $O(N^3)$
Practical problems

- Due to the smoothing effect of the practical resampling filters, the central part of the volume is overemphasized
- Premultiplication: The 3D DFT is premultiplied by the reciprocal inverse Fourier transform of the resampling filter

![Diagram of spatial and frequency domains]

Drawbacks

- Alpha compositing is not supported
- Only simulated X-ray images
- Only parallel projection
- Limited practical application

Depth cueing

- Those voxels that are farther from the image plane are rendered with lower intensity
- The depth perception can be improved
- Depth cueing can be implemented in the frequency domain by a 2D operation
- The 3D DFT does not have to be recalculated

\[
\begin{align*}
    & FT\left\{ f(x)d(x)p(x) \right\} = H(v) \\
    & = (F(v) * D(v) * P(v)) * H(v) \\
    & = (F(v) * P(v)) * (H(v) * D(v)) \\
    & = (F(v) * P(v)) * H'(v)
\end{align*}
\]
Hemispherical illumination

- Classical hemispherical illumination:
  \[ I = \frac{1}{2} \left( 1 + (N \cdot L) \right) \]

- Hemispherical illumination for volume data:
  \[ I = \frac{1}{2} \left| \nabla f(x) \left( 1 + \frac{\nabla f(x) \cdot L}{\left| \nabla f(x) \right|} \right) \right| = \frac{1}{2} \left| \nabla f(x) \right| + \nabla f(x) \cdot L \]

- Evaluation in the frequency domain:
  \[ \frac{1}{2} \left( \text{FT} \left[ \nabla f(x) \right] \ast \rho(x) \right) + \text{iFT} \left[ f(x) \rho(x) \right] (v \cdot L) + H(v) \]

Monte Carlo Volume Rendering

- The density function is integrated in the region that is projected onto the given pixel
- The Monte Carlo integration is applied
Monte Carlo Volume Rendering

- The continuous representation of the density function:
  \[ f(x) = \sum_{i,j \in \mathcal{V}} f(x_{i,j}) \cdot h(x - x_{i,j}) \]

- Monte Carlo integration:
  \[ I = \int f(x) dx = \int \frac{g(x)}{p(x)} p(x) dx = \frac{\int g(x)}{\int p(x)} = \frac{1}{M} \sum_{i=1}^{M} g(x_i) \]

  - \( x_i \) is the \( k \)th sample of a random variable \( x \), where
  - \( p(x) \) is the probability density function of \( x \)
  - the averaging gives an unbiased estimation for the integral
  - the more proportional \( p(x) \) to the integrand \( g(x) \), the lower is the variance of the estimation (importance sampling)

---

Monte Carlo Volume Rendering

- The calculation of pixel \( I_{i,j} \):
  \[ I_{i,j} = \int f(x) dx = \int f(x) v_{i,j}(x) dx = E \left[ \frac{f(x)v_{i,j}(x)}{p(x)} \right] \]

  - \( v_{i,j}(x) \) is the visibility function: \( v(x)_{i,j} = \begin{cases} 1 & \text{if } x \in V_{i,j} \\ 0 & \text{otherwise} \end{cases} \)

- Partial importance sampling (\( p(x) \) is proportional only to the density function \( f(x) \)):
  \[ P_{i,j} = \frac{\int f(x) dx}{\int f(x) dx} = P_{i,j} = \frac{1}{M} \sum_{i=1}^{M} v_{i,j}(x_i) = \frac{M_{i,j}}{M_{i}} \]

---

MCVR algorithm

- Random point samples are generated with a probability proportional to the continuous density function
  - first a random voxel position is selected
  - then a random translation is added, where the probability density of the translation is proportional to a reconstruction kernel
- The point samples are projected onto the screen
- The density of a pixel is proportional to the number of samples projected onto the given pixel

\[ F(v_i) = \frac{1}{\sum_{j} f(v_j)} \sum_{j} F(v_j) \]

\[ F(v_i) = \frac{1}{\sum_{j} f(v_j)} \sum_{j} F(v_j) \]

\[ F(v_i) = \frac{1}{\sum_{j} f(v_j)} \sum_{j} F(v_j) \]

\[ F(v_i) = \frac{1}{\sum_{j} f(v_j)} \sum_{j} F(v_j) \]

\[ F(v_i) = \frac{1}{\sum_{j} f(v_j)} \sum_{j} F(v_j) \]

\[ F(v_i) = \frac{1}{\sum_{j} f(v_j)} \sum_{j} F(v_j) \]
Complexity

- The variance of the estimation can be reduced by increasing the number of samples.
- The normalized intensities are quantized.
- The standard deviation can be reduced below the level of the quantization error.
- It is easy to see that the number of the necessary samples is proportional to the number of the pixels rather than the number of voxels.
- In this sense the complexity of MCVR is $O(N^2)$, which is better than the complexity $O(N^2 \log N)$ of Fourier Volume Rendering.

Convergence

Images rendered by MCVR
Shading based on gradients

- Not the original data values are used as a probability density function
- The probability is proportional to the gradient magnitude
- The well-defined isosurfaces are enhanced
- The point samples are shaded based on their gradients before the projection
- The compositing is a simple integration
- The results are similar to that of the shaded Fourier Volume Rendering

---

1M

16M

Marching Cubes

- The intermediate representation is a triangular mesh approximating a given isosurface
- For each cubic cell it is checked whether it is intersected by the isosurface
- Inside a cubic cell a local triangulation is applied depending on the binary classification of the corner points
- The obtained geometrical model is interactively rendered by the incremental image synthesis approach
The Marching Cubes algorithm

- Specification of an isosurface by a threshold \( t \)
- Binary classification of the voxels (volume element)
- Triangulation of the intersected cells:
  - an 8 bit long index is calculated based on the binary classification of the corner points
  - reading an edge list from a precalculated lookup table (LUT)
  - calculation of intersection points along the edges using a linear interpolation
  - linear interpolation of the normals estimated at the lattice points

Classification – index calculation

\[
\begin{align*}
&\text{index kiszámítása:} \\
&P_1 < t \quad \Rightarrow \text{külső pont (} b_1 = 0) \\
&P_1 \geq t \quad \Rightarrow \text{beııső pont (} b_1 = 1) \\
&\quad b_7 | b_6 | b_5 | b_4 | b_3 | b_2 | b_1 | b_0
\end{align*}
\]

Triangulation of the cells

- The 8 bit long index addresses 256 cases
- There are only 14 topologically invariant cases \( \Rightarrow \) the precalculated LUT contains only 14 edge lists
- The other cases can be retrieved by rotation or mirroring transformations
Calculation of the triangle vertices

- Vertex calculation:
  \[ V_{i,j} = \frac{(1 - f(P_i)) \cdot P_j + (f(P_j) - t) \cdot P_i}{f(P_j) - f(P_i)} \]

- Normal calculation:
  \[ N_{i,j} = \frac{(1 - f(P_i)) \cdot N_j + (f(P_j) - t) \cdot N_i}{f(P_j) - f(P_i)} \]

A reconstructed isosurface

Drawbacks

- Huge number of triangles is generated
- Triangle decimation is needed
  - Triangles representing a surface regions of low curvature are contracted
  - Octree-based hierarchical approach
- Piecewise linear approximation – the edges of the triangles are apparent from a short distance
- Iterative smoothing of the triangular mesh – curvature minimization based on a penalty function
- Today the display of a triangular mesh is already slower than a GPU-accelerated direct volume rendering
Application in CAD systems

- Acetabulum (Socket)
- Hip Ball
- Femur (Thigh Bone)

Mechanical simulation

- Finite element analysis