Volume Ray Casting

Neslisah Torosdagli
Overview

- Light Transfer Optical Models
- Math behind Direct Volume Ray Casting
- Demonstration
- Transfer Functions
- Details of our Application
- References
What is Volume Ray Casting?

- Volume Ray Casting is construction of 3D volume using stack of 2D slices.
- Since now in Ray Tracing course, when a ray hits to a surface, ray tracing stops for that ray.
- In Volume Ray Casting, ray tracing accumulates information until bounding volume is exited or certain stopping criteria holds.
Light Transfer Optical Models

Light traveling thru a medium is affected by:

- **Emission**: The volume is assumed to consist of particles that emit light.
- **Absorption**: The volume is assumed to consist of black particles that absorb light.
- **Scattering**: In-Scattering (Light reaching to a volume particle which is scattered from other particles in the volume.) and Out-Scattering (Light scattered from volume particle.)

In direct volume rendering, Emission-Absorption Optical Model is sufficient in quality, scattering is complex and neglected.
Light Transfer Optical Model - Emission & Absorption

Assume that each voxel is a light source, covered with a semi-transparent membrane. And also assume only 50% of radiant energy of each voxel can pass to the next voxel in ray direction.

1/4 green + 1/2 purple + blue

1/4 blue + 1/2 purple + green

If there is no absorption in the ray direction:

\[ I(x(0)) = I(x(n)) \]
Radiant Energy Reaching the Camera

\[ I(x(0)) = I(x(n)) e^{-\tau(x(0),x(n))} \]

Some portion of the radiant energy at a point \( x(n) \) is lost while passing thru the volume towards the camera due to absorption.
Radiant Energy Reaching the Camera

Absorption along the ray segment from $x(n)$ to $x(0)$

$$I(x(0)) = I(x(n)) e^{-\tau(x(0),x(n))}$$
Extinction (Optical Depth) \( \tau (o, t) \)

Small values of optical depth point to transparent mediums, while large values refer to opaque mediums.

A form of measure of how long light may travel before it is absorbed completely.

\[
\tau (o, t) = \int_{0}^{t} \kappa (\hat{t}) d\hat{t}
\]

Absorption Coefficients : \( \kappa \)

Absorption at certain position in the medium.

Absorption along the line segment 0 to \( t \):

\[ e^{-\tau (0,t)} \]
If there are more than one radiant energy sources, the total radiant energy arriving to the camera is the summation of radiant energies arrived to the camera.
Ray Integration

\[ C = \int_{0}^{\infty} c(t)e^{-\tau(0,t)} dt \]

Radiant Energy reaching eye from any ray direction is integration of radiant energy at each point along the ray that reached the camera.
Ray Integration Discretization

Since volume data is discrete, integrations can be approximated with Reimann Sum as follows:

\[
\tau(o, t) = \int_0^t \kappa(\hat{t}) \, d\hat{t} \quad \Rightarrow \quad \ddot{\tau}(0, t) = \sum_{i=0}^{[t/\Delta t]} \kappa(i \cdot \Delta t) \Delta t
\]
Ray Integration Discretization

Since volume data is discrete, integrations can be approximated with Reimann Sum as follows:

\[ C = \int_0^\infty c(t)e^{-\tau(0,t)}dt \rightarrow \tilde{C} = \sum_{i=0}^{\left\lfloor \frac{T}{\Delta t} \right\rfloor} C_i e^{-\tilde{\tau}(0,t)} \]
Discrete Extinction

\[ \ddot{\tau}(0, t) = \sum_{i=0}^{\left\lfloor t/\Delta t \right\rfloor} \kappa(i \cdot \Delta t) \Delta t \]

\[ e^{-\ddot{\tau}(o,t)} = e^{-\sum_{i=0}^{\left\lfloor t/\Delta t \right\rfloor} \kappa(i \cdot \Delta t) \Delta t} \]

\[ e^{-\ddot{\tau}(o,t)} = \prod_{i=0}^{\left\lfloor t/\Delta t \right\rfloor} e^{-\kappa(i \cdot \Delta t) \Delta t} \]
Opacity of each Slice

\[ A_i = 1 - e^{-\kappa(i \cdot \Delta t) \Delta t} \]

Absorption

\[ e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lceil t/\Delta t \rceil} e^{-\kappa(i \cdot \Delta t) \Delta t} \]

\[ e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lceil t/\Delta t \rceil} (1 - A_i) \]
Discrete Ray Integration

\[ C = \int_0^\infty c(t) e^{-\tau(0,t)} \]

\[ \tilde{C} = \sum_{i=0}^{[T/\Delta t]} C_i e^{-\bar{\tau}(0,t)} \]

\[ \tilde{C} = \sum_{i=0}^{n} C_i \prod_{j=0}^{i-1} (1 - A_{ij}) \]
How to apply this formula?

\[ \tilde{C} = \sum_{i=0}^{n} C_i \prod_{j=0}^{i-1} (1 - A_j) \]

\[ \Delta A = 1 - e^{-\tau(0,t)} - (1 - e^{-\tau(0,t-\Delta t)}) \Rightarrow (1 - \prod_{i=0}^{t/\Delta t} (1 - A_i)) - (1 - \prod_{i=0}^{(t-\Delta t)/\Delta t} (1 - A_i))) \Rightarrow (A_j - 1) \ast \prod_{i=0}^{(t-\Delta t)/\Delta t} (1 - A_i) + \prod_{i=0}^{(t-\Delta t)/\Delta t} (1 - A_i) \Rightarrow A_j \ast \prod_{i=0}^{(t-\Delta t)/\Delta t} (1 - A_i)) \Rightarrow \Delta A = A_j \ast e^{-\tau(t-\Delta t)} \Rightarrow \Delta A = A_j \ast (1 - A_{\text{previous}}) \]

\[ C_{dst}^{i+1} = C_{dst}^i + (1 - A_{dst}^i) \ast C_{src}^i \]
If I worked on blue slices and currently working on purple slice, purple slice’s contribution to final radiance is reduced in proportion with multiplication of opacities of blue slices.

\[
\tilde{C} = \sum_{i=0}^{n} C_i \prod_{j=0}^{i-1} (1 - A_j)
\]
Front-to-Back Volume Ray Casting

\[ C_{dst}^{i+1} = C_{dst}^i + (1 - A_{dst}^i) \cdot C_{src}^i \]
Front-to-Back Volume Ray Casting Algorithm

```cpp
vec4 frontToBackRayCast()
{
    vec4 src = vec4(0.0);
    vec4 dst = vec4(0.0);

    for (int idx=0; idx<MAX_STEPS; idx++)
    {
        vec3 ijs = currentPos * RAStoIJSRatio;
        float value = sampleAs3DTexture(uVolumeSampler, ijs).r;

        src.rgb *= src.a;
        dst += ((1.0 - dst.a) * src);

        currentPos += stepSzScaledRayDirection;

        if ((dst.a >= 0.95) || ((len-=stepSz) < stepSz))
        {
            break;
        }
    }

    return (dst);
}
```
Back-to-Front Volume Ray Casting

\[ C_{dst}^i = C_{src}^i + (1 - A_{src}^i) \times C_{dst}^{i-1} \]
Back-to-Front Volume Ray Casting Algorithm

```cpp
vec4 backToFrontRayCast()
{
    vec4 src = vec4(0.0);
    vec4 dst = vec4(0.0);

    for (int idx=0; idx<MAX_STEPS; idx++)
    {
        vec3 ijs = currentPos * RAStoIJSRatio;
        float value = sampleAs3DTexture(uVolumeSampler, ijs).r;

        src.rgb *= src.a;
        dst = src + ((1.0 - src.a) * dst);

        currentPos += stepSzScaledRayDirection;

        if ((len-=stepSz) < stepSz)
        {
            break;
        }
    }

    return (dst);
}
```
Cons of Back-to-Front Volume Ray Casting

- Information about back face of the volume is included in the rendering, which may be unrelated and may have negative impact on quality of the rendering.

- Early ray termination cannot be applied, as whatever alpha value is computed, first slice should be included in the rendering.
Cons of Back-to-Front Volume Ray Casting

• Alpha and RGB values usually exceed 1.0 when the rayTracing loop is completed, they should be normalized.

• Since relative values are not known a-priori, normalization of the whole rendering, requires an extra step to compute max value computed, and normalization with respect to this max value.
Does each voxel in each slice have the same contribution?
Emission & Absorption Coefficients

- In the first part of this class, we assumed that emission and absorption coefficients are known a-priori. However, in real life we are provided just slices of images of some gray values. There is no natural way to obtain these coefficients.

- Instead, the user should decide how different structures in the data should look by assigning emission and absorption coefficients according to data values.

- For instance, the user may decide that soft tissues such as muscles may be 50% transparent while hard tissues such as bones may be 80% opaque.
• If we know which region of the volume is soft-tissue and which region is hard-tissue, or we know how to recognize regions in a volume before the user sets their mapping, there is no issue.
• However, regions in a volume are not known either.
Transfer Functions

- Transfer function can be simply described as setting opacities for each voxel in such a way that both emission (by premultiplication of rgb values by opacity) and absorption for a voxel can be computed.

- Design of transfer function which leads to a high quality and informative rendering is a difficult process.
Demonstration
Transfer Functions

- What can be used to define the mapping:
  - Voxel Properties are as follows:
    - grayScale Value
    - histogram
    - First Derivative
    - Second Derivative
- In 1D, it is a simple lookup table mapping such as set opacity of grayScale values equal to $g$ to $o$.
- Usually just grayScale value is not sufficient to differentiate different regions of a volume with same grayScale value.
• Tooth data surrounded by a circle of dark gray.
• In the first four samples, there is a black color region inside the tooth. Then 1D mapping of black color to opacity 0, will fail for inner black regions.
• For this sample, the question is how to differentiate boundary from inner details of the tooth?
1D Transfer Functions

- Furthermore, 1D Transfer functions cause artifacts due to sharp changes between consecutive grayScale values.
Derivatives

• First and second derivatives of voxels is a good measure of edges in a volume.
• Kindlemann utilized edge information for semi-automatic generation of transfer functions.
• He assigned automatically maximum applicable opacity to edges and smoothly assign closer opacities to voxels close in spatial domain to the edge voxel.
Transfer Functions

- 2D, 3D Transfer Functions are much useful to differentiate different regions with the same grayScale value.
2D Levoy Transfer Function

In order to compute 2D Transfer Function as Levoy suggested,
- A set of \((\text{grayScale}, \alpha)\) values should be inputted.
- Thus, the user should specify approximate transparency of certain grayScale values.
- After inputting these tuples, following formula is used to compute the 2D Transfer Function:

\[
\alpha(x_i) = |\nabla f(x_i)| \begin{cases} 
\alpha_{v_{n+1}} \left( \frac{f(x_i) - f_{v_n}}{f_{v_{n+1}} - f_{v_n}} \right) + \text{ if } f_{v_n} \leq f(x_i) \leq f_{v_{n+1}} \\
\alpha_{v_{v_n}} \left( \frac{f_{v_{n+1}} - f(x_i)}{f_{v_{n+1}} - f_{v_n}} \right) \\
0 \text{ otherwise}
\end{cases}
\]

* formulas are taken from original paper.
For 8 bit data, for each grayScale value in domain 0 to 255, and for each gradient value in domain 0 to 128, alpha value is computed using inputed tuples \((\alpha_v, f_v)\) by the user.

* formulas are taken from original paper.
Using, Levoy 2D Transfer Function, a smoother rendering is obtained. Not only, specified grayScales are set to specified alpha values, but also according to gradient of the voxel in question, closer surfaces are assigned to closer alpha values. Thus, sharp changing artifacts are avoided.
For 8 bit data, for each grayScale value in domain 0 to 255, and for each gradient value in domain 0 to 128,

Compute gradientHistogram
initialize transfer function 2D to:

\[ 1 - 3 \times \text{Math.pow}(\text{histValGrad}[i][j] / \text{maxHist}, 0.22) \]

Multiply transfer function 2D with user inputted 1D transfer function.
2D Transfer Functions

2D Barthel Transfer Function

2D Levoy Transfer Function
UCF Graphics Lab Web-Based Real-Time Volume Rendering Application

• In our application, we used Levoy and Barthel 2D Transfer functions computed using grayScale values and 1st derivatives.
UCF Graphics Lab Web-Based Real-Time Volume Rendering Application

Technologies:

- Javascript
- WebGL
- Medical File Parser (XTK)
- D3
Some Sample Renderings
Some Sample Renderings
UCF Graphics Lab Web-Based Real-Time Volume Rendering Application Video

Web-based Interactive Real-Time Volume Rendering

Neslisah Torsdagli, Sumanta Pattanaik, Curtis Lisle

1 Computer Graphics Research Group • Dept. of EECS (Computer Science Division) • University of Central Florida • Orlando, FL USA
2 KnowledgeVis, LLC • Maitland, FL USA
Contact: neslisah@knights.ucf.edu

- Customize 1D Transfer Function
- Modify Palette
- Apply 1D/2D Transfer Functions
- Insert Anchor Point
- Enable Disable Gradient Shading
UCF Graphics Lab Application

Link of the application is:
http://graphics.cs.ucf.edu/tools/VOLREN/
References