kD-Trees for Volume Ray-Casting

Anita Schilling

Special Seminar for Computer Graphics
15. January 2009
Outline

1. Introduction
2. Ray-Voxel Intersection Techniques
3. kD-Trees
4. Evaluation
Introduction

- high-performance ray-tracing
- scientific visualization
- iso-surface ray-casting

Volume Raycasting with CUDA (Marsalek & Slusallek 2008)
Iso-Surface Extraction

Marching Cubes

- simple algorithm
- precomputed lookup table for iso-surface/voxel intersection
Marching Cubes

- holes possible
- only approximation of iso-surface
- generates too many triangles
- iso-surface extraction for every iso-value necessary
Iso-Surface Rendering

- ray-casting the iso-surface
- exact intersection calculation of ray with volume
- higher image quality

Volume Ray Tracing (Marmitt et. al. 2004)
Direct Ray-Casting of Iso-Surface

Problems

- efficiently finding voxels hit by ray containing iso-surface

- correct intersection point calculation of ray with interpolated implicit surface of voxel
correct intersection point calculation is expensive
Accurate Intersection Method

- Voxel data values $\rho_{ijk}$, $i, j, k \in \{0, 1\}$
- Compute density at any point $(u, v, w) \in [0, 1]^3$
- Using trilinear interpolation

$$\rho(u, v, w) = \sum_{i,j,k \in \{0,1\}} u_i v_j w_k \cdot \rho_{ijk}$$

With $u_0 = u$, $u_1 = 1 - u$, $v_0 = v$, $v_1 = 1 - v$, $w_0 = w$, $w_1 = 1 - w$
Accurate Intersection Method

- computing density $\rho(p)$ of any point $p \in V$

- spatial location of voxel cell
  $V = [x_0 \ldots x_1] \times [y_0 \ldots y_1] \times [z_0 \ldots z_1]$

- transform $p = (x, y, z)$ into voxel unit coordinate system

$$p(u^p_0, v^p_0, w^p_0) = \left( \frac{x_1 - p_x}{x_1 - x_0}, \frac{y_1 - p_y}{y_1 - y_0}, \frac{z_1 - p_z}{z_1 - z_0} \right)$$
Accurate Intersection Method

- Ray \( R(T) = O + T \cdot D \)
  - \( O \) ... ray origin and
  - \( D \) ... ray direction in world coordinates
  - passing the voxel in interval \([T_{in}, T_{out}]\)

2. transform ray \( R(T) \) to voxel unit coordinate system

\[ r(t) = a + tb \]

- \( p_{\text{entry}} = r(t_{in} = 0) \)
- \( p_{\text{exit}} = r(t_{out} = 1) \)
density at every ray point within voxel

\[
\rho(t) = \rho(r(t)) = \sum_{i,j,k \in \{0,1\}} (u_i^a + tu_i^b)(v_j^a + tv_j^b)(w_k^a + tw_k^b) \cdot \rho_{ijk}
\]
Accurate Intersection Method

Expand to \( \rho(t) = At^3 + Bt^2 + Ct + D \) with

\[
A = \sum_{ijk} u_i^b v_j^b w_k^b \cdot \rho_{ijk}
\]

\[
B = \sum_{ijk} (u_i^a v_j^b w_k^b + u_i^b v_j^a w_k^b + u_i^b v_j^b w_k^a) \cdot \rho_{ijk}
\]

\[
C = \sum_{ijk} (u_i^b v_j^a w_k^a + u_i^a v_j^b w_k^a + u_i^a v_j^a w_k^b) \cdot \rho_{ijk}
\]

\[
D = \sum_{ijk} u_i^a v_j^a w_k^a \cdot \rho_{ijk}
\]
Approximate Method

Simple Midpoint Algorithm

- trading quality for speed
- intersection set to midpoint between entry and exit point of ray
- blocky artifacts at size of voxels
- only for performance reference
Linear Interpolation Method

- linearly interpolate intersection point on the ray

\[ t_{\text{hit}} = t_{\text{in}} + (t_{\text{out}} - t_{\text{in}}) \frac{\rho_{\text{iso}} - \rho_{\text{in}}}{\rho_{\text{out}} - \rho_{\text{in}}} \]

- significantly more costly
- two tri- or bilinear interpolations
- fails in more complex cases
  - if function has 2 roots such that entry and exit densities are both larger or smaller than \( \rho_{\text{iso}} \)
Neubauer’s Method

- repeated linear interpolation

\[ t = t_0 + (t_1 - t_0) \frac{\rho_{iso} - \rho_0}{\rho_1 - \rho_0} \]

- if \( \text{sign}(\rho(r(t)) - \rho_{iso}) = \text{sign}(\rho_0 - \rho_{iso}) \)
  - then \( t_0 = t, \rho_0 = \rho(r(t)) \)
  - else \( t_1 = t, \rho_1 = \rho(r(t)) \)

- typically 2 or 3 times
- fails in more complex cases
  - falsely returns last intersection point if 3 intersections are within a voxel but 2 of them are in the first ray segment
Accurate Intersection Method

- slow and correct or fast and sometimes incorrect methods

Key Observations
- only need first intersection with implicit surface
- repeated linear interpolation does find the correct root, if the start interval contains exactly one root
find ray parameter $t$ where $\rho(t) = \rho_{iso}$

$f(t) = \rho(t) - \rho_{iso} = 0$ with $t \in [t_{in} = 0, t_{out} = 1]$, 

smallest root of $f(t) = At^3 + Bt^2 + Ct + D - \rho_{iso}$ in interval $[0, 1]$
Accurate Root Finding Method

4. Compute extrema $e_0$, $e_1$ of $f(t) = At^3 + Bt^2 + Ct + D - \rho_{iso}$ with $f'(t) = 3At^2 + 2Bt + C = 0$

5. Compute density value at start point $f(e_0)$ and end point $f(e_1)$ of interval
if \( \text{sign}(f(t_{in})) \neq \text{sign}(f(e_0)) \), then interval contains exactly one root.

else advance ray to next segment.
Accurate Root Finding Method

7. apply repeated linear interpolation (2-3 times)
   - efficient computation of any density value along the ray with $\rho(t)$

8. transform voxel unit ray parameter $t_{hit}$ back to world coordinate ray parameter $T_{hit}$

9. calculate intersection point $p_{intersect} = R(T_{hit})$
Find the right voxel for intersection

- test every voxel whether voxel contains iso-value and ray hits voxel
Uniform Grid Traversal

- split voxel grid into uniform macro-cells
- store min/max iso-values for macro-cells
- test whether macro-cell contains iso-value
  - test whether ray hits macro-cell
  - test every single voxel of macro-cell
kD-Tree

- empty space skipping in polygon ray-tracing
- scenes with varying primitive density
- kD-Trees often outperform other datastructures
kD-Tree

- apply to volume data
- split node at center of largest dimension
- parent node stores min/max iso-range of children
- easily determined recursively
- test for iso-value before following down branch
kD-Tree

- middle split of scene cuts through objects
- better split into separate objects and empty space
- build kD-Tree with surface area heuristic
- apply to volume data
- assumption
  - there is always some empty space around
- only optimized for one iso-value for initial evaluation
- precompute surface area with summed area table
- evaluate all possible split locations on building
Surface Area of Iso-Surface

- summed area tables for one iso-value

\[ sat(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y') = sat(x - 1, y) + \sum_{y' \leq y} i(x, y') \]

- Rectangle area

\[ SA_{rectangle} = \sum_{x' = A_x, y' = A_y} i(x', y') \]

\[ SA_{rectangle} = sat(A) + sat(C) - sat(B) - sat(D) \]
Surface Area of Iso-Surface

- no directly computed surface area
- 3d summed area table for volume data

\[ SA = sat(G) + sat(E) + sat(B) + sat(D) \]
\[ -sat(A) - sat(F) - sat(H) - sat(C) \]
**kDTree with Surface Area Heuristic**

- **splitCost** = \( \max \left( \frac{SA_{left} \cdot \frac{SA_{left}}{noCells_{left}} + SA_{right} \cdot \frac{SA_{right}}{noCells_{right}}}{SA_{parent}} \right) \)

- **splitCost** = \( \max \left( \frac{SA_{left}}{noCells_{left}} + \frac{SA_{right}}{noCells_{right}} \right) \)

- **splitCost** = \( \min \left( \frac{SA_{left} \cdot \frac{noCells_{left}}{noCells_{parent}} + SA_{right} \cdot \frac{noCells_{right}}{noCells_{parent}}}{SA_{parent}} \right) \)
3 possibilities

1. $SA_{parent} = 0$
   no further split

2. $SA_{parent} = noCells_{parent}$
   split in center of largest dimension

3. $0 < SA_{parent} < noCells_{parent}$
   evaluate every possible split location

if maximum node dimension $\leq$ threshold then no split
## Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Dimensions</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inner Ear</td>
<td>$128 \times 128 \times 30$</td>
<td>8 bit</td>
</tr>
<tr>
<td>Mouse Skeleton</td>
<td>$201 \times 201 \times 326$</td>
<td>8 bit</td>
</tr>
<tr>
<td>Teddy Bear</td>
<td>$128 \times 128 \times 62$</td>
<td>16 bit</td>
</tr>
<tr>
<td>Tooth</td>
<td>$128 \times 128 \times 160$</td>
<td>12 bit</td>
</tr>
<tr>
<td>Engine</td>
<td>$256 \times 256 \times 110$</td>
<td>8 bit</td>
</tr>
<tr>
<td>Head</td>
<td>$256 \times 256 \times 225$</td>
<td>8 bit</td>
</tr>
<tr>
<td>Neghip</td>
<td>$64 \times 64 \times 64$</td>
<td>12 bit</td>
</tr>
</tbody>
</table>
## Results

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Tree</th>
<th>No. of Nodes</th>
<th>Max. Depth</th>
<th>Build Time</th>
<th>Render Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>kD</td>
<td>1056183</td>
<td>21</td>
<td>1,31</td>
<td>15,27</td>
</tr>
<tr>
<td>4</td>
<td>SAH</td>
<td>3241961</td>
<td>311</td>
<td>19,34</td>
<td>86,52</td>
</tr>
<tr>
<td>8</td>
<td>kD</td>
<td>112359</td>
<td>18</td>
<td>0,34</td>
<td>34,16</td>
</tr>
<tr>
<td>8</td>
<td>SAH</td>
<td>1962645</td>
<td>307</td>
<td>18,17</td>
<td>85,95</td>
</tr>
</tbody>
</table>

**Engine with $\rho_{iso} = 0$**

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Tree</th>
<th>No. of Nodes</th>
<th>Max. Depth</th>
<th>Build Time</th>
<th>Render Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>kD</td>
<td>1056183</td>
<td>21</td>
<td>1,21</td>
<td>2,49</td>
</tr>
<tr>
<td>4</td>
<td>SAH</td>
<td>562161</td>
<td>130</td>
<td>14,79</td>
<td>4,16</td>
</tr>
<tr>
<td>8</td>
<td>kD</td>
<td>112359</td>
<td>18</td>
<td>0,34</td>
<td>5,93</td>
</tr>
<tr>
<td>8</td>
<td>SAH</td>
<td>427035</td>
<td>126</td>
<td>14,23</td>
<td>4,23</td>
</tr>
</tbody>
</table>

**Engine with $\rho_{iso} = 120$**
Evaluation

Engine Dataset
## Evaluation

### Results (2)

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Tree</th>
<th>No. of Nodes</th>
<th>Max. Depth</th>
<th>Build Time</th>
<th>Render Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Neghip with $\rho_{iso} = 10000$</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>kD</td>
<td>35151</td>
<td>16</td>
<td>0,004</td>
<td>2,97</td>
</tr>
<tr>
<td>4</td>
<td>SAH</td>
<td>31333</td>
<td>51</td>
<td>0,4</td>
<td>4,12</td>
</tr>
<tr>
<td>8</td>
<td>kD</td>
<td>4393</td>
<td>23</td>
<td>0,01</td>
<td>10,18</td>
</tr>
<tr>
<td>8</td>
<td>SAH</td>
<td>23225</td>
<td>51</td>
<td>0,38</td>
<td>4,47</td>
</tr>
<tr>
<td><strong>Neghip with $\rho_{iso} = 53000$</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>kD</td>
<td>31513</td>
<td>16</td>
<td>0,04</td>
<td>1,81</td>
</tr>
<tr>
<td>4</td>
<td>SAH</td>
<td>8543</td>
<td>28</td>
<td>0,34</td>
<td>2,87</td>
</tr>
<tr>
<td>8</td>
<td>kD</td>
<td>4393</td>
<td>13</td>
<td>0,02</td>
<td>4,83</td>
</tr>
<tr>
<td>8</td>
<td>SAH</td>
<td>7079</td>
<td>28</td>
<td>0,34</td>
<td>3,1</td>
</tr>
</tbody>
</table>
Evaluation

Neghip Dataset
## Results (3)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Theshold</th>
<th>Tree</th>
<th>Intersect Calls</th>
<th>Ray-Box Tests</th>
<th>Volume Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engine</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\rho_{iso} = 0$</td>
<td>4</td>
<td>kD</td>
<td>1.731.924</td>
<td>14.695.818</td>
<td>283.936.284</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>SAH</td>
<td>1.553.721</td>
<td>7.702.492</td>
<td>59.795.850</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>kD</td>
<td>1.975.115</td>
<td>54.407.797</td>
<td>1.592.065.190</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>SAH</td>
<td>1.563.913</td>
<td>11.351.200</td>
<td>158.410.322</td>
</tr>
<tr>
<td>$\rho_{iso} = 120$</td>
<td>4</td>
<td>kD</td>
<td>195.281</td>
<td>2.364.250</td>
<td>41.050.844</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>SAH</td>
<td>166.365</td>
<td>1.966.466</td>
<td>7.894.580</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>kD</td>
<td>244.561</td>
<td>6.554.875</td>
<td>334.116.520</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>SAH</td>
<td>176.210</td>
<td>2.274.389</td>
<td>18.520.868</td>
</tr>
<tr>
<td>Neghip</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\rho_{iso} = 10000$</td>
<td>4</td>
<td>kD</td>
<td>238.632</td>
<td>2.851.231</td>
<td>68.528.248</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>SAH</td>
<td>204.067</td>
<td>2.627.877</td>
<td>8.711.280</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>kD</td>
<td>276.515</td>
<td>10.205.981</td>
<td>753.244.902</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>SAH</td>
<td>208.623</td>
<td>3.208.909</td>
<td>37.134.946</td>
</tr>
<tr>
<td>$\rho_{iso} = 53000$</td>
<td>4</td>
<td>kD</td>
<td>99.157</td>
<td>1.815.862</td>
<td>38.505.276</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>SAH</td>
<td>91.435</td>
<td>2.574.532</td>
<td>4.513.244</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>kD</td>
<td>117.948</td>
<td>3.909.951</td>
<td>330.461.150</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>SAH</td>
<td>91.799</td>
<td>2.784.705</td>
<td>18.306.844</td>
</tr>
</tbody>
</table>
Conclusion

- normal kD-Tree performs better
- heuristic kD-Tree needs less intersection calls

→ hybrid kD-Tree

- empty space around center of volume data set
- cut away empty space cells with heuristic
- if ratio of SA to number of cells gets large enough use normal kD-Tree
The End.

Thank you for your attention.
G. Marmitt, A. Kleer, I. Wald, H. Friedrich, P. Slusallek: 
Fast and Accurate Ray Voxel Intersection Techniques 
for Iso-Surface Ray Tracing, Vision, Modeling and 
Visualization (VMV), 2004.

I. Wald, H. Friedrich, G. Marmitt, P. Slusallek, H-P. Seidel: 
Faster Isosurface Ray Tracing using Implicit KD-Trees, 
IEEE Transactions on Visualization and Computer 

V. Havran: Heuristic Ray Shooting Algorithms, PhD 