kD-Trees for Volume Ray-Casting

Anita Schilling

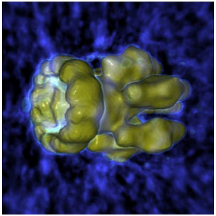
TECHNISCHE UNIVERSITÄT DRESDEN

Special Seminar for Computer Graphics
15. January 2009

Outline

Introduction

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

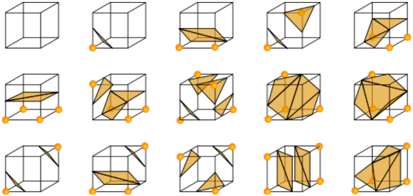


Volume Raycasting with CUDA
(Marsalek & Slusallek 2008)

Iso-Surface Extraction

Marching Cubes

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



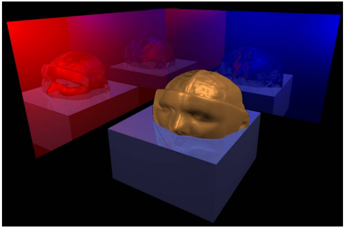
Iso-Surface Extraction

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)

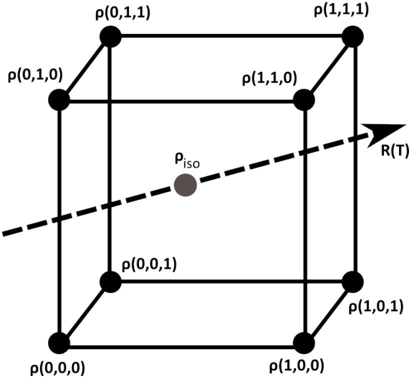
Direct Ray-Casting of Iso-Surface

Problems

- efficiently finding voxels hit by ray containing iso-surface
- correct intersection point calculation of ray with interpolated implicit surface of voxel

Ray-Voxel Intersection

- correct intersection point calculation is expensive

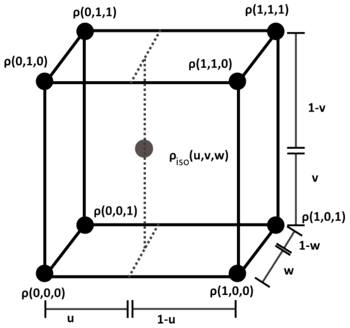


Accurate Intersection Method

- voxel data values ρ_{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
$$\begin{aligned} u_0 &= u, & u_1 &= 1 - u, \\ v_0 &= v, & v_1 &= 1 - v, \\ w_0 &= w, & w_1 &= 1 - w \end{aligned}$$



Accurate Intersection Method

- computing density $\rho(\mathbf{p})$ of any point $\mathbf{p} \in V$
- spatial location of voxel cell
  $V = [x_0 \dots x_1] \times [y_0 \dots y_1] \times [z_0 \dots z_1]$

1. transform $p = (x, y, z)$ into voxel unit coordinate system
   $$\mathbf{p}(u_0^p, v_0^p, w_0^p) = \left(\frac{x_1 - p_x}{x_1 - x_0}, \frac{y_1 - p_y}{y_1 - y_0}, \frac{z_1 - p_z}{z_1 - z_0} \right)$$

Accurate Intersection Method

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

2. transform ray $R(T)$ to voxel unit coordinate system $r(t) = a + tb$
   - $\mathbf{p}_{entry} = r(t_{in} = 0)$
   - $\mathbf{p}_{exit} = r(t_{out} = 1)$

Accurate Intersection Method

- 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

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

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

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

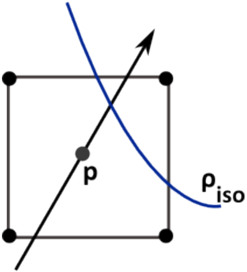
$$C = \sum_{ijk} (u_i^b v_j^a w_k^a + u_i^a v_j^b w_k^a + u_i^a v_j^a w_k^b) \cdot \rho_{ijk}$$

$$D = \sum_{ijk} u_i^a v_j^a w_k^a \cdot \rho_{ijk}$$

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

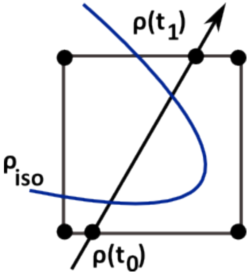


Linear Interpolation Method

- linearly interpolate intersection point on the ray

$$t_{hit} = t_{in} + (t_{out} - t_{in})\frac{\rho_{iso} - \rho_{in}}{\rho_{out} - \rho_{in}}$$

- 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 ρ_{iso}



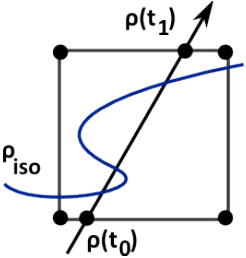
Neubauer's Method

- repeated linear interpolation

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

- if $sign(\rho(r(t)) - \rho_{iso}) = sign(\rho_0 - \rho_{iso})$
  - 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

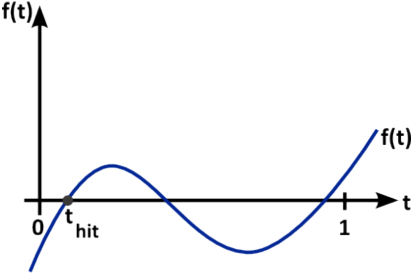
- only need first intersection with implicit surface
- repeated linear interpolation does find the correct root, if the start interval contains exactly one root

Accurate Intersection Method

- find ray parameter t where $\rho(t) = \rho_{iso}$

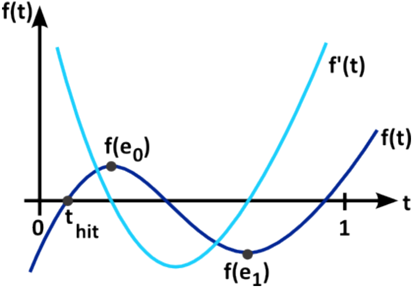
→ $f(t) = \rho(t) - \rho_{iso} = 0$ with $t \in [t_{in} = 0, t_{out} = 1]$,

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



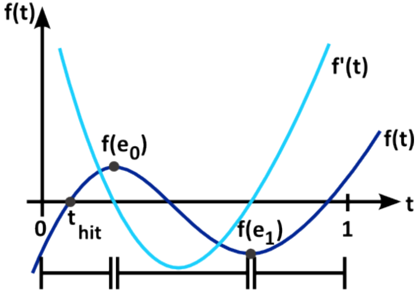
Accurate Root Finding Method

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



Accurate Root Finding Method

6. if $sign(f(t_{in})) \neq sign(f(e_0))$
   then interval contains exactly one root
   else advance ray to next segment

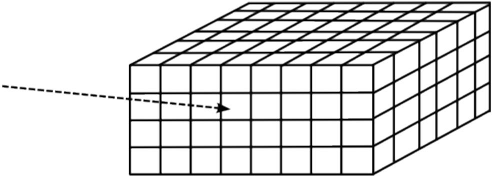


Accurate Root Finding Method

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 $p_{intersect} = R(T_{hit})$

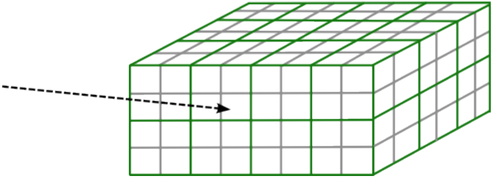
Find the right voxel for intersection

- test every voxel whether voxel contains iso-value and ray hits voxel



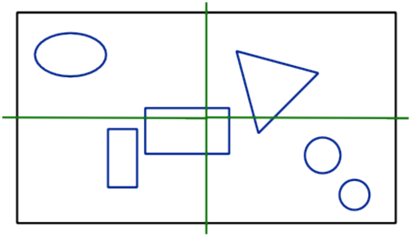
Uniform Grid Traversal

- 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

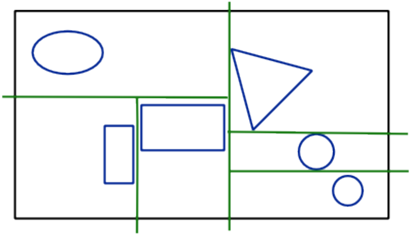


kD-Tree

- 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



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

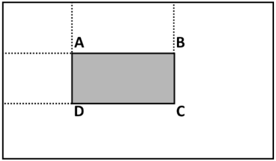
Surface Area of Iso-Surface

- 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

$$SA_{rectangle} = \sum_{x'=A_x, y'=A_y}^{B_x, D_y} i(x', y')$$

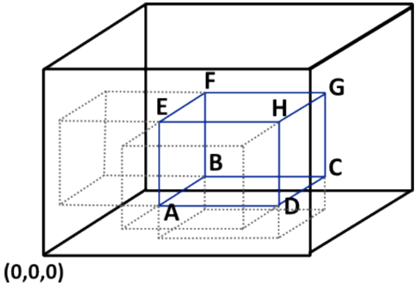


$$SA_{rectangle} = sat(A) + sat(C) - sat(B) - sat(D)$$

Surface Area of Iso-Surface

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

$$\begin{aligned} SA &= sat(G) + sat(E) + sat(B) + sat(D) \\ &\quad - sat(A) - sat(F) - sat(H) - sat(C) \end{aligned}$$



kDTree with Surface Area Heuristic

$$splitCost = \max \frac{SA_{left} \cdot \frac{SA_{left}}{noCells_{left}} + SA_{right} \cdot \frac{SA_{right}}{noCells_{right}}}{SA_{parent}}$$

$$splitCost = \max \frac{\frac{SA_{left}}{noCells_{left}} + \frac{SA_{right}}{noCells_{right}}}{noCells_{parent}}$$

$$splitCost = \min \frac{SA_{left} \cdot \frac{noCells_{left}}{noCells_{parent}} + SA_{right} \cdot \frac{noCells_{right}}{noCells_{parent}}}{SA_{parent}}$$

kD-Tree with Surface Area Heuristic

- 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 $\leq$ treshold then no split

Evaluation

Datasets

Dataset	Dimensions	Resolution
Inner Ear	$128 \times 128 \times 30$	8 bit
Mouse Skeleton	$201 \times 201 \times 326$	8 bit
Teddy Bear	$128 \times 128 \times 62$	16 bit
Tooth	$128 \times 128 \times 160$	12 bit
Engine	$256 \times 256 \times 110$	8 bit
Head	$256 \times 256 \times 225$	8 bit
Neghip	$64 \times 64 \times 64$	12 bit

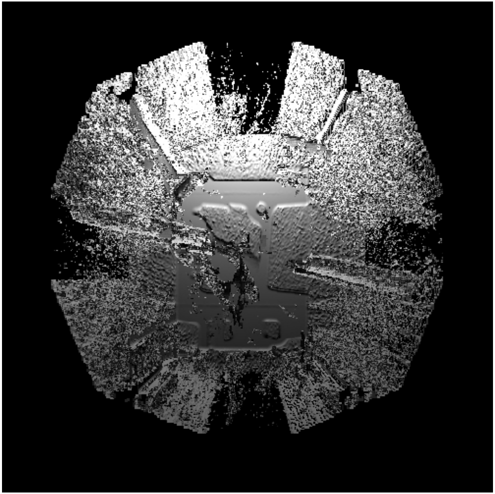
Evaluation

Results

Threshold	Tree	No. of Nodes	Max. Depth	Build Time	Render Time
Engine with $\rho_{iso} = 0$					
4	kD	1056183	21	1,31	15,27
4	SAH	3241961	311	19,34	86,52
8	kD	112359	18	0,34	34,16
8	SAH	1962645	307	18,17	85,95
Engine with $\rho_{iso} = 120$					
4	kD	1056183	21	1,21	2,49
4	SAH	562161	130	14,79	4,16
8	kD	112359	18	0,34	5,93
8	SAH	427035	126	14,23	4,23

Evaluation

Engine Dataset



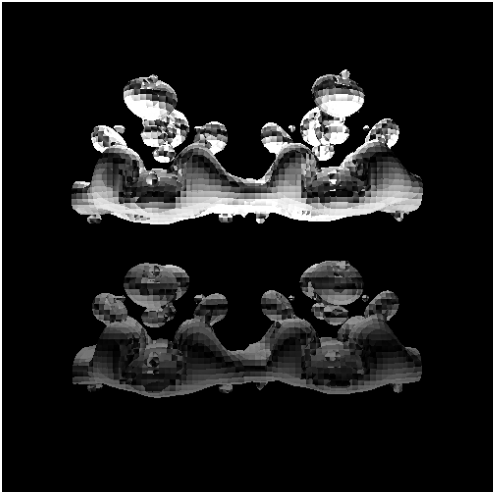
Evaluation

Results (2)

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

Evaluation

Neghip Dataset



Results (3)

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

In Progress

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.

Literature

- 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 Graphics (IEEE VIS), 2005.
- V. Havran: **Heuristic Ray Shooting Algorithms**, PhD dissertation, CVUT, Prague 2000.