Ray Tracing Components

- Shadows
- Reflection
- Refraction
- Recursive Ray Tracing
Recap: Ray Tracing

trace ray
Intersect all objects
color = ambient term
For every light
  cast shadow ray
  color += local shading term
If mirror
  color += color_{refl} *
  trace reflected ray
If transparent
  color += color_{trans} *
  trace transmitted ray

• Does it ever end?

Stopping criteria:
• Recursion depth
  – Stop after a number of bounces
• Ray contribution
  – Stop if reflected / transmitted contribution becomes too small
Recursion For Reflection: None
Recursion For Reflection: 1
Recursion For Reflection: 2
The Ray Tree

- $N_i$: surface normal
- $R_i$: reflected ray
- $L_i$: shadow ray
- $T_i$: transmitted (refracted) ray

Complexity?
Ray tree

- Visualizing the ray tree for single image pixel
Ray tree

- Visualizing the ray tree for single image pixel

This gets pretty complicated pretty fast!
Questions?
Ray Tracing Algorithm Analysis

• Lots of primitives
• Recursive
• Distributed Ray Tracing
  – Means using many rays for non-ideal/non-pointlike phenomena
    • Soft shadows
    • Anti-aliasing
    • Glossy reflection
    • Motion blur
    • Depth of field

\[
\text{cost} \approx \text{height} \times \text{width} \times \text{num primitives} \times \text{intersection cost} \times \text{size of recursive ray tree} \times \text{num shadow rays} \times \text{num supersamples} \times \text{num glossy rays} \times \text{num temporal samples} \times \text{num aperture samples} \times \ldots
\]

Can we reduce this?
Today

• Motivation
  – You need LOTS of rays to generate nice pictures
  – Intersecting every ray with every primitive becomes the bottleneck

• Bounding volumes

• Bounding Volume Hierarchies, Kd-trees

For every pixel
  Construct a ray from the eye
  For every object in the scene
    Find intersection with the ray
    Keep if closest
    Shade
Accelerating Ray Casting

• Goal: Reduce the number of ray/primitive intersections
Conservative Bounding Volume

- First check for an intersection with a conservative bounding volume
- **Early reject**: If ray doesn’t hit volume, it doesn’t hit the triangles!
Conservative Bounding Volume

- What does “conservative” mean?
  - Volume must be big enough to contain all geometry within
Conservative Bounding Regions

• Desiderata
  – Tight → avoid false positives
  – Fast to intersect
Ray-Box Intersection

- Axis-aligned box
- Box: \((X_1, Y_1, Z_1) \rightarrow (X_2, Y_2, Z_2)\)
- Ray: \(P(t) = R_o + tR_d\)
Naïve Ray-Box Intersection

- 6 plane equations: Compute all intersections
- Return closest intersection *inside the box*
  - Verify intersections are on the correct side of each plane: $Ax+By+Cz+D < 0$
Reducing Total Computation

- Pairs of planes have the same normal
- Normals have only one non-0 component
- Do computations one dimension at a time
Test if Parallel

- If $R_{dx} = 0$ (ray is parallel) AND
  $R_{ox} < X_1$ or $R_{ox} > X_2$ → no intersection
Find Intersections Per Dimension

• Basic idea
  – Determine an interval along the ray for each dimension
  – The intersect these 1D intervals (remember CSG!)
  – Done!
Find Intersections Per Dimension

• Basic idea
  – Determine an interval along the ray for each dimension
  – The intersect these 1D intervals (remember CSG!)
  – Done!

\[ \begin{align*}
  y &= Y_2 \\
  y &= Y_1 \\
  x &= X_1 \\
  x &= X_2
\end{align*} \]

Interval between \( X_1 \) and \( X_2 \)
Find Intersections Per Dimension

• Basic idea
  – Determine an interval along the ray for each dimension
  – The intersect these 1D intervals (remember CSG!)
  – Done!

Interval between $X_1$ and $X_2$

Interval between $Y_1$ and $Y_2$
Find Intersections Per Dimension

• Basic idea
  – Determine an interval along the ray for each dimension
  – The intersect these 1D intervals (remember CSG!)
  – Done!
Intersecting 1D Intervals
Intersecting 1D Intervals

Start = max of mins
Intersecting 1D Intervals

Start = max of mins

End = min of maxs
Intersecting 1D Intervals

If Start > End, the intersection is empty!

Start = max of mins

End = min of maxs
Find Intersections Per Dimension

- Calculate intersection distance $t_1$ and $t_2$
  - $t_1 = (X_1 - R_{ox}) / R_{dx}$
  - $t_2 = (X_2 - R_{ox}) / R_{dx}$
  - $[t_1, t_2]$ is the X interval
Then Intersect Intervals

- Init $t_{\text{start}}$ & $t_{\text{end}}$ with X interval
- Update $t_{\text{start}}$ & $t_{\text{end}}$ for each subsequent dimension
Then Intersect Intervals

- Compute $t_1$ and $t_2$ for $Y$...
Then Intersect Intervals

- **Update** $t_{\text{start}}$ & $t_{\text{end}}$ for each subsequent dimension
  - If $t_1 > t_{\text{start}}$, $t_{\text{start}} = t_1$
  - If $t_2 < t_{\text{end}}$, $t_{\text{end}} = t_2$
Then Intersect Intervals

- **Update** $t_{\text{start}}$ & $t_{\text{end}}$ for each subsequent dimension
  - If $t_1 > t_{\text{start}}$, $t_{\text{start}} = t_1$
  - If $t_2 < t_{\text{end}}$, $t_{\text{end}} = t_2$
Then Intersect Intervals

- Update $t_{\text{start}}$ & $t_{\text{end}}$ for each subsequent dimension
  
  - If $t_1 > t_{\text{start}}$, $t_{\text{start}} = t_1$
  
  - If $t_2 < t_{\text{end}}$, $t_{\text{end}} = t_2$

\[ x = X_1 \quad \text{and} \quad x = X_2 \]

\[ y = Y_1 \quad \text{and} \quad y = Y_2 \]
Is there an Intersection?

- If $t_{\text{start}} > t_{\text{end}} \rightarrow \text{box is missed}$
Is the Box Behind the Eyepoint?

- If $t_{\text{end}} < t_{\text{min}} \rightarrow \text{box is behind}$
Return the Correct Intersection

- If $t_{\text{start}} > t_{\text{min}}$ → closest intersection at $t_{\text{start}}$
- Else → closest intersection at $t_{\text{end}}$
  - Eye is inside box
Ray-Box Intersection Summary

- For each dimension,
  - If $R_{dx} = 0$ (ray is parallel) AND $R_{ox} < X_1$ or $R_{ox} > X_2$ → no intersection
- For each dimension, calculate intersection distances $t_1$ and $t_2$
  - $t_1 = (X_1 - R_{ox}) / R_{dx}$
  - $t_2 = (X_2 - R_{ox}) / R_{dx}$
  - If $t_1 > t_2$, swap
- Maintain an interval $[t_{start}, t_{end}]$, intersect with current dimension
  - If $t_1 > t_{start}$, $t_{start} = t_1$
  - If $t_2 < t_{end}$, $t_{end} = t_2$
- If $t_{start} > t_{end}$ → box is missed
- If $t_{end} < t_{min}$ → box is behind
- If $t_{start} > t_{min}$ → closest intersection at $t_{start}$
- Else → closest intersection at $t_{end}$
Efficiency Issues

- $1/R_{dx}$, $1/R_{dy}$ and $1/R_{dz}$ can be pre-computed and shared for many boxes
Bounding Box of a Triangle

\[(x_{\min}, y_{\min}, z_{\min}) = (\min(x_0, x_1, x_2), \min(y_0, y_1, y_2), \min(z_0, z_1, z_2))\]

\[(x_{\max}, y_{\max}, z_{\max}) = (\max(x_0, x_1, x_2), \max(y_0, y_1, y_2), \max(z_0, z_1, z_2))\]
Bounding Box of a Sphere

\[(x, y, z) = (x-r, y-r, z-r)\]

\[(x_{\text{max}}, y_{\text{max}}, z_{\text{max}}) = (x+r, y+r, z+r)\]
Bounding Box of a Plane

\[ (x_{\text{min}}, y_{\text{min}}, z_{\text{min}}) = (-\infty, -\infty, -\infty) \]

\[ (x_{\text{max}}, y_{\text{max}}, z_{\text{max}}) = (+\infty, +\infty, +\infty) \]

\[ ax + by + cz = d \]

*unless \( n \) is exactly perpendicular to an axis*
Bounding Box of a Group

$$(x_{\text{min}}, y_{\text{min}}, z_{\text{min}}) = (\min(x_{\text{min}_a}, x_{\text{min}_b}), \min(y_{\text{min}_a}, y_{\text{min}_b}), \min(z_{\text{min}_a}, z_{\text{min}_b}))$$

$$(x_{\text{max}}, y_{\text{max}}, z_{\text{max}}) = (\max(x_{\text{max}_a}, x_{\text{max}_b}), \max(y_{\text{max}_a}, y_{\text{max}_b}), \max(z_{\text{max}_a}, z_{\text{max}_b}))$$
Bounding box of transformed object IS NOT the transformation of the bounding box!

\[
(x_{\text{max}}, y_{\text{max}}, z_{\text{max}}) = \max\left(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7\right), \max(y_0, y_1, y_2, y_3, y_4, x_5, x_6, x_7), \max(z_0, z_1, z_2, z_3, z_4, x_5, x_6, x_7)
\]

\[
(x_{\text{min}}, y_{\text{min}}, z_{\text{min}}) = \min\left(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7\right), \min(y_0, y_1, y_2, y_3, y_4, x_5, x_6, x_7), \min(z_0, z_1, z_2, z_3, z_4, x_5, x_6, x_7)
\]

\[
(x'_{\text{max}}, y'_{\text{max}}, z'_{\text{max}}) = \max(x'_{\text{min}}, x'_{\text{max}}, y'_{\text{min}}, y'_{\text{max}}, z'_{\text{min}}, z'_{\text{max}})
\]

\[
(x'_{\text{min}}, y'_{\text{min}}, z'_{\text{min}}) = \min(x'_{\text{min}}, x'_{\text{max}}, y'_{\text{min}}, y'_{\text{max}}, z'_{\text{min}}, z'_{\text{max}})
\]
Questions?
Are Bounding Volumes Enough?

• If ray hits bounding volume, must we test all primitives inside it?
  – Lots of work, think of a 1M-triangle mesh
Bounding Volume Hierarchies

• If ray hits bounding volume, must we test all primitives inside it?
  – Lots of work, think of a 1M-triangle mesh
• You guessed it already, we’ll split the primitives in groups and build recursive bounding volumes
  – Like collision detection, remember?
Bounding Volume Hierarchy (BVH)

- Find bounding box of objects/primitives
- Split objects/primitives into two, compute child BVs
- Recurse, build a binary tree
Bounding Volume Hierarchy (BVH)

- Find bounding box of objects/primitives
- Split objects/primitives into two, compute child BVs
- Recurse, build a binary tree
Bounding Volume Hierarchy (BVH)

- Find bounding box of objects/primitives
- Split objects/primitives into two, compute child BVs
- Recurse, build a binary tree
Bounding Volume Hierarchy (BVH)

- Find bounding box of objects/primitives
- Split objects/primitives into two, compute child BVs
- Recurse, build a binary tree
Bounding Volume Hierarchy (BVH)

- Find bounding box of objects/primitives
- Split objects/primitives into two, compute child BVs
- Recurse, build a binary tree
Where to Split Objects?

- At midpoint of current volume  \textit{OR}
- Sort, and put half of the objects on each side  \textit{OR}
- Use modeling hierarchy
Questions?
Ray-BVH Intersection
Ray-BVH Intersection

- Find $t_{\text{start}}$ and $t_{\text{end}}$ for node
  - If no hit, return
Ray-BVH Intersection

• Compute $t_{\text{start}}, t_{\text{end}}$ for child nodes
  – Recursively check sub-volume with closer intersection first
Intersection with BVH

• Don't return intersection immediately if the other subvolume may have a closer intersection
  – Nodes can and will overlap!
Intersection with BVH

• Don't return intersection immediately if the other subvolume may have a closer intersection
  – Nodes can and will overlap!
Intersection with BVH

- Must also check the farther child node if closest hit so far is inside its bounding volume
BVH Discussion

• Advantages
  – easy to construct
  – easy to traverse
  – binary tree (=simple structure)

• Disadvantages
  – may be difficult to choose a good split for a node
  – poor split may result in minimal spatial pruning
BVH Discussion

• Advantages
  – easy to construct
  – easy to traverse
  – binary tree (simple structure)

• Disadvantages
  – may be difficult to choose a good split for a node
  – poor split may result in minimal spatial pruning

• Still one of the best methods
  – Recommended for your first hierarchy!
Questions?
Kd-trees

- Probably most popular acceleration structure
- Binary tree, axis-aligned splits
  - Each node splits space in half along an axis-aligned plane
- A space partition: The nodes do not overlap!
  - This is in contrast to BVHs
Data Structure

KdTreeNode:

KdTreeNode* backNode, frontNode //children
int dimSplit // either x, y or z
float splitDistance
    // from origin along split axis
boolean isLeaf
List of triangles //only for leaves

here dimSplit = 0 (x axis)
Kd-tree Construction

- Start with scene axis-aligned bounding box
- Decide which dimension to split (e.g. longest)
- Decide at which distance to split (not so easy)
Kd-tree Construction - Split

- Distribute primitives to each side
- If a primitive overlaps split plane, assign to both sides
Kd-tree Construction - Recurse

- Stop when minimum number of primitives reached
- Other stopping criteria possible
Questions?

• Further reading on efficient Kd-tree construction
  – Hunt, Mark & Stoll, IRT 2006
  – Zhou et al., SIGGRAPH Asia 2008
Kd-tree Traversal - High Level

- If leaf, intersect with list of primitives
- If intersects back child, recurse
- If intersects front child, recurse
Kd-tree Traversal, Naïve Version

- Could use bounding box test for each child
- But redundant calculation: bbox similar to that of parent node, plus axis aligned, one single split
Kd-tree Traversal, Smarter Version

- Get main bbox intersection from parent
  - $t_{\text{near}}$, $t_{\text{far}}$
- Intersect with splitting plane
  - easy because axis aligned
Kd-tree Traversal - Three Cases

- Intersects only back, only front, or both
- Can be tested by examining $t$, $t_{\text{start}}$ and $t_{\text{end}}$
Kd-tree traversal - three cases

- If $t > t_{\text{end}}$ => intersect only front
- If $t < t_{\text{start}}$ => intersect only back

Note: “Back” and “Front” depend on ray direction!
Kd-tree Traversal Pseudocode

travers(orig, dir, t_start, t_end):
    # adapted from Ingo Wald's thesis
    # assumes that dir[self.dimSplit] > 0
    if self.isLeaf:
        return intersect(self.listOfTriangles, orig, dir, t_start, t_end)
    t = (self.splitDist - orig[self.dimSplit]) / dir[self.dimSplit];
    if t <= t_start:
        # case one, t <= t_start <= t_end -> cull front side
        return self.backSideNode.traverse(orig, dir, t_start, t_end)
    elif t >= t_end:
        # case two, t_start <= t_end <= t -> cull back side
        return self.frontSideNode.traverse(orig, dir, t_start, t_end)
    else:
        # case three: traverse both sides in turn
        t_hit = self.frontSideNode.traverse(orig, dir, t_start, t)
        if t_hit <= t: return t_hit; # early ray termination
        return self.backSideNode.traverse(orig, dir, t, t_end)
Important Details

- For leaves, do NOT report intersection if $t$ is not in $[t_{\text{near}}, t_{\text{far}}]$.
  - Important for primitives that overlap multiple nodes!
  - Could check if already intersected (mailboxing), but this doesn’t pay off in practice

- Need to take direction of ray into account
  - Reverse back and front if the direction has negative coordinate along the split dimension

- Degeneracies when ray direction is parallel to one axis
Questions?
Where to Split?

- Example for baseline
- Note how this ray traverses easily: one leaf only
Split in the Middle

- Does not conform to empty vs. dense areas
- Inefficient traversal - Not so good!
Split in the Median

- Tries to balance tree, but does not conform to empty vs. dense areas
- Inefficient traversal - Not good
Optimizing Splitting Planes

• Most people use the Surface Area Heuristic (SAH)
  – MacDonald and Booth 1990, “Heuristic for ray tracing using space subdivision”, Visual Computer

• Idea: simple probabilistic prediction of traversal cost based on split distance

• Then try different possible splits and keep the one with lowest cost

• Further reading on efficient Kd-tree construction
  – Hunt, Mark & Stoll, IRT 2006
  – Zhou et al., SIGGRAPH Asia 2008
Is it Important to Optimize Splits?

• Given the same traversal code, the quality of Kd-tree construction can have a big impact on performance, e.g. a factor of 2 compared to naive middle split
  – But then, you should consider carefully if you need that extra performance
  – Could you optimize something else for bigger gain?
Efficient Implementation

• Not so easy, need ability to sort primitives along the three axes very efficiently and split them into two groups
• Plus primitives have an extent (bbox)

• Extra tricks include smarter tests to check if a triangle is inside a box
Hard-core efficiency considerations

- See e.g. Ingo Wald’s PhD thesis
  - http://www.mpi-inf.mpg.de/~wald/PhD/

- Calculation
  - Optimized barycentric ray-triangle intersection

- Memory
  - Make kd-tree node as small as possible
    (dirty bit packing, make it 8 bytes)

- Parallelism
  - SIMD extensions, trace 4 rays at a time, mask results where they disagree
Pros and Cons of Kd trees

- **Pros**
  - Simple code
  - Efficient traversal
  - Can conform to data

- **Cons**
  - Costly construction, not great if you work with moving objects
Questions?

- For extensions to moving scenes, see Real-Time KD-Tree Construction on Graphics Hardware, Zhou et al., SIGGRAPH 2008
Stack Studios, Rendered using Maxwell

Questions?
Regular Grid
Create Grid

- Find bounding box of scene
- Choose grid resolution $(n_x, n_y, n_z)$
- $grid_x$ need not $= grid_y$
Insert Primitives into Grid

- Primitives that overlap multiple cells?
- Insert into multiple cells (use pointers)
For Each Cell Along a Ray

- Does the cell contain an intersection?
  - Yes: return closest intersection
  - No: continue
Preventing Repeated Computation

• Option #1:
  – Perform computation only once, "mark" the object
• Option #2: live with redundant computation
  – Easier, recommended
Don't Return Distant Intersections

- If intersection \( t \) is not within the cell range, continue (there may be something closer)
Which Cells Should We Examine?

- Should we intersect the ray with each voxel?
- No! we can do better!

![Image of a grid with cells highlighted to illustrate the concept of intersecting rays with voxels.]
Where Do We Start?

- Intersect ray with scene bounding box
- Ray origin may be inside the scene bounding box

Cell \((i, j)\)

Cell \((i, j)\)

\(t_{\text{min}}\)

\(t_{\text{next}_x}\)

\(t_{\text{next}_y}\)

\(t_{\text{next}_x}\)

\(t_{\text{next}_y}\)
Is there a Pattern to Cell Crossings?

- Yes, the horizontal and vertical crossings have regular spacing.

\[ dt_x = \frac{\text{grid}_x}{\text{dir}_x} \]

\[ dt_y = \frac{\text{grid}_y}{\text{dir}_y} \]
What's the Next Cell?

if \( t_{next_x} < t_{next_y} \)

\[
i += \text{sign}_x
\]

\[
t_{min} = t_{next_x}
\]

\[
t_{next_x} += dt_x
\]

else

\[
j += \text{sign}_y
\]

\[
t_{min} = t_{next_y}
\]

\[
t_{next_y} += dt_y
\]

if \( \text{dir}_x > 0 \) \( \text{sign}_x = 1 \) else \( \text{sign}_x = -1 \)

if \( \text{dir}_y > 0 \) \( \text{sign}_y = 1 \) else \( \text{sign}_y = -1 \)
What's the Next Cell?

- 3DDDA – Three Dimensional Digital Difference Analyzer
- Similar to Line Rasterization
Pseudo-Code

create grid
insert primitives into grid
for each ray $r$
    find initial cell $c(i,j)$, $t_{\text{min}}$, $t_{\text{next}_x}$ & $t_{\text{next}_y}$
    compute $dt_x$, $dt_y$, $\text{sign}_x$ and $\text{sign}_y$
    while $c \neq \text{NULL}$
        for each primitive $p$ in $c$
            intersect $r$ with $p$
            if intersection in range found
                return
        $c = \text{find next cell}$
Ray Marching Visualization

sphere voxelization

primitive density

cells traversed

entered faces
Regular Grid Discussion

• Advantages?
  – very easy to construct
  – easy to traverse

• Disadvantages?
  – may be only sparsely filled
  – geometry may still be clumped
Questions?
Does Ray Tracing Simulate Physics?

- Photons go from the light to the eye, not the other way
  - What we do is *backward ray tracing*. 
Does Ray Tracing Simulate Physics?

- Ray Tracing is full of dirty tricks
- For example, shadows of transparent objects
  - Opaque? Surely not..
  - Multiply by transparency color?
    (ignores refraction & does not produce caustics)
Correct Transparent Shadow

- Using advanced refraction technique (photon mapping)
  - Refraction for illumination ("caustics") is usually not handled that well
“Forward” Ray Tracing

• Start from the light source: Shoot lots of “photons”
  – Very, very low probability to reach the eye/camera!
• What can we do about it?
  – Always send a ray to the eye…. still not efficient
  – More solutions later
Forward vs. Backward Ray Tracing

- A word of warning: The terms “forward” and “backward” are not quite standard
  - Some texts use “backward” when they mean “forward” in our sense
  - And vice versa

Walter et al., Single Scattering in Refractive Media with Triangle Mesh Boundaries, SIGGRAPH 2009
Does Ray Tracing Simulate Physics?

- We do backward ray tracing
- Fortunately, it turns out that there is a mathematical justification to going from eye towards the scene
  - We’ll look at it a little on the Monte Carlo lecture
- In any case, the real world doesn’t consist of just mirrors :-)
The Rendering Equation

• Clean mathematical framework for light transport simulation (Kajiya, 1986)

• We’ll see this later
  – You can take a peek here or here for a preview.

• At each point, outgoing light in one direction is the integral of incoming light in all directions multiplied by material reflectance.

The Rendering Equation is a physically-based model that agrees with reality to a good degree.
That’s All For Today

• Further reading:
  – Shirley: Realistic Ray Tracing
  – Dutre et al.: Advanced Global Illumination