Ray Tracing II & Acceleration Data Structures for Ray Casting
Transparency

- Cast ray in refracted direction
- Multiply by transparency coefficient (color)
Qualitative Refraction

From “Color and Light in Nature” by Lynch and Livingston
Refraction

Snell-Descartes Law:

\[ \eta_i \sin \Theta_i = \eta_T \sin \Theta_T \]

\[
\sin \Theta_T = \frac{\eta_i}{\eta_T} = \eta_r
\]

\[ I = N \cos \Theta_i - M \sin \Theta_i \]

\[ M = \frac{(N \cos \Theta_i - I)}{\sin \Theta_i} \]

\[ T = -N \cos \Theta_T + M \sin \Theta_T \]

\[ = -N \cos \Theta_T + (N \cos \Theta_i - I) \frac{\sin \Theta_T}{\sin \Theta_i} \]

\[ = -N \cos \Theta_T + (N \cos \Theta_i - I) \eta_r
\]

\[ = \left[ \eta_r \cos \Theta_i - \cos \Theta_T \right] N - \eta_r I \]

\[ = \left[ \eta_r \cos \Theta_i - \sqrt{1 - \sin^2 \Theta_T} \right] N - \eta_r I \]

\[ = \left[ \eta_r \cos \Theta_i - \sqrt{1 - \eta_r^2 \sin^2 \Theta_i} \right] N - \eta_r I \]

\[ = \left[ \eta_r \cos \Theta_i - \sqrt{1 - \eta_r^2 \sin^2 \Theta_i} \right] N - \eta_r I \]

\[ = \left[ \eta_r (N \cdot I) - \sqrt{1 - \eta_r^2 (1 - (N \cdot I)^2)} \right] N - \eta_r I \]

\[ \eta_i \sin \Theta_i = \eta_T \sin \Theta_T \]

- **Total internal reflection when the square root is imaginary**
- **Don’t forget to normalize!**

Plugging \( M \) into the equations.

Don’t forget to normalize!
Refraction & the Sidedness of Objects

- Make sure you know whether you’re entering or leaving the transmissive material:

  \[ \eta_T = \text{material index} \]

  \[ \eta_i = 1 \]

- Check dot product with normal

- Note: We won’t ask you to trace rays through intersecting transparent objects
Total Internal Reflection

Fig. 3.7A The optical manhole. From under water, the entire celestial hemisphere is compressed into a circle only 97.2° across. The dark boundary defining the edges of the manhole is not sharp due to surface waves. The rays are analogous to the crepuscular type seen in hazy air, Section 1.9. (Photo by D. Granger)

Fig. 3.7B The optical manhole. Light from the horizon (angle of incidence = 90°) is refracted downward at an angle of 48.6°. This compresses the sky into a circle with a diameter of 97.2° instead of its usual 180°.

From “Color and Light in Nature” by Lynch and Livingston
Cool Refraction Demo

- Enright, D., Marschner, S. and Fedkiw, R.,
Refraction and the Lifeguard Problem

- Running is faster than swimming

Lifeguard

Water

Beach

Run

Person in trouble

Swim
Wavelength

- Refraction is wavelength-dependent
  - Refraction increases as the wavelength of light decreases
  - violet and blue experience more bending than orange and red
- Newton’s experiment
- Usually ignored in graphics

Pink Floyd, *The Dark Side of the Moon*

Pittoni, 1725, Allegory to Newton
Dispersion

- Image by Henrik Wann Jensen using Photon Mapping
How does a Rainbow Work?

- From “Color and Light in Nature” by Lynch and Livingstone
Rainbow

- Refraction depends on wavelength
- Rainbow is caused by refraction + internal reflection + refraction
- Maximum for angle around 42 degrees

From “Color and Light in Nature” by Lynch and Livingstone
Questions?
Overview of Today

• 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

0 recursion

1 recursion

2 recursions
Questions?
CAD for lenses

- Has revolutionized lens design
- E.g. zooms are good now

From Hecht's Optics
Lens design, ray tracing

- Used to be done manually, by rooms full of engineers who would trace rays.
- Now software, e.g. Zemax
- More in 6.815/6.865

source: canon red book
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*
Forward Ray Tracing

• Start from the light source
  – But low probability to reach the eye

• What can we do about it?
  – Always send a ray to the eye… still not efficient
  – More solutions later
Does Ray Tracing Simulate Physics?

• Ray Tracing is full of dirty tricks
• For example, shadows of transparent objects:
  – opaque?
  – multiply by transparency color?
    (ignores refraction & does not produce caustics)
Correct Transparent Shadow

Image by Henrik Wann Jensen

Using advanced refraction technique
(refraction for illumination is usually not handled that well)
The Rendering Equation

- Clean mathematical framework for light-transport simulation
- We’ll see this later
- At each point, outgoing *light in one direction* is the integral of *incoming light in all directions* multiplied by reflectance property (BRDF)
Ray Tracing II & Acceleration Data Structures for Ray Casting
Shadows

• one shadow ray per intersection per point light source

MIT EECS 6.837, Durand
Shadows & Light Sources

http://www.davidfay.com/index.php
http://www.pa.uky.edu/~sciworks/light/preview/bulb2.htm

MIT EECS 6.837, Durand
Soft Shadows

• multiple shadow rays to sample area light source

MIT EECS 6.837, Durand
Antialiasing – Supersampling

- multiple rays per pixel
Reflection

- one reflection ray per intersection
Glossy Reflection

- multiple reflection rays

polished surface

Justin Legakis
Motion Blur

- Sample objects temporally
Depth of Field

- multiple rays per pixel

film

focal length

Justin Legakis
Ray Tracing Algorithm Analysis

- Ray casting
- Lots of primitives
- Recursive
- Distributed Ray Tracing Effects
  - 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?
Questions?
Today

• Motivation – Distribution Ray Tracing

• Bounding Boxes
  – of each primitive
  – of groups
  – of transformed primitives

• Spatial Acceleration Data Structures
Acceleration of Ray Casting

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

- First check for an intersection with a conservative bounding region
- Early reject
Conservative Bounding Regions

- tight → avoid false positives
- fast to intersect
Ray-Box Intersection

- Axis-aligned
- 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

- Calculate intersection distance $t_1$ and $t_2$
  
  - $t_1 = (X_1 - R_{ox}) / R_{dx}$
  
  - $t_2 = (X_2 - R_{ox}) / R_{dx}$
Maintain \( t_{\text{near}} \) & \( t_{\text{far}} \)

- Closest & farthest intersections \textit{on the object}
  - If \( t_1 > t_{\text{near}} \), \( t_{\text{near}} = t_1 \)
  - If \( t_2 < t_{\text{far}} \), \( t_{\text{far}} = t_2 \)
Is there an Intersection?

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

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

- If $t_{\text{near}} > t_{\text{min}}$ \rightarrow \text{closest intersection at } t_{\text{near}}$
- Else \rightarrow \text{closest intersection at } t_{\text{far}}
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 \( t_{near} \) and \( t_{far} \) (closest & farthest intersections so far)
    - If \( t_1 > t_{near} \), \( t_{near} = t_1 \)
    - If \( t_2 < t_{far} \), \( t_{far} = t_2 \)

- If \( t_{near} > t_{far} \) → box is missed
- If \( t_{far} < t_{min} \) → box is behind
- If \( t_{near} > t_{min} \) → closest intersection at \( t_{near} \)
- Else → closest intersection at \( t_{far} \)
Efficiency Issues

• $1/R_{dx}$, $1/R_{dy}$ and $1/R_{dz}$ can be pre-computed and shared for many boxes

• Unroll the loop
  – Loops are costly (because of termination test)
  – Avoid the $t_{\text{near}}$ & $t_{\text{far}}$ comparison for first dimension
Bounding Box of a Triangle

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

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

\[ (x_{\text{max}}, y_{\text{max}}, z_{\text{max}}) = (x+r, y+r, z+r) \]

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

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

\[n = (a, b, c)\]

\[ax + by + cz = d\]

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

* unless \(n\) is exactly perpendicular to an axis

MIT EECS 6.837, Durand
Bounding Box of a Group

\[(x_{\min a}, y_{\min a}, z_{\min a})\]

\[(x_{\max a}, y_{\max a}, z_{\max a})\]

\[(x_{\min b}, y_{\min b}, z_{\min b})\]

\[(x_{\max b}, y_{\max b}, z_{\max b})\]

\[(x_{\min}, y_{\min}, z_{\min}) = (\min(x_{\min a}, x_{\min b}), \min(y_{\min a}, y_{\min b}), \min(z_{\min a}, z_{\min b}))\]

\[(x_{\max}, y_{\max}, z_{\max}) = (\max(x_{\max a}, x_{\max b}), \max(y_{\max a}, y_{\max b}), \max(z_{\max a}, z_{\max b}))\]

MIT EECS 6.837, Durand
Bounding Box of a Transform

\[(x'_{\text{min}}, y'_{\text{min}}, z'_{\text{min}}) = (\min(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7), \min(y_0, y_1, y_2, y_3, y_4, x_5, x_6, x_7), \min(z_0, z_1, z_2, z_3, z_4, x_5, x_6, x_7))\]

\[(x'_{\text{max}}, y'_{\text{max}}, z'_{\text{max}}) = (\max(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7), \max(y_0, y_1, y_2, y_3, y_4, x_5, x_6, x_7), \max(z_0, z_1, z_2, z_3, z_4, x_5, x_6, x_7))\]
Special Case: Transformed Triangle

Can we do better?
Special Case: Transformed Triangle

$$(x_{\text{max}}, y_{\text{max}}, z_{\text{max}}) = (\max(x'_0, x'_1, x'_2), \max(y'_0, y'_1, y'_2), \max(z'_0, z'_1, z'_2))$$

$$(x_{\text{min}}, y_{\text{min}}, z_{\text{min}}) = (\min(x'_0, x'_1, x'_2), \min(y'_0, y'_1, y'_2), \min(z'_0, z'_1, z'_2))$$
Questions?
Bounding Volume Hierarchy

- Find bounding box of objects
- Split objects into two groups
- Recurse
Bounding Volume Hierarchy

- Find bounding box of objects
- Split objects into two groups
- Recurse
Bounding Volume Hierarchy

- Find bounding box of objects
- Split objects into two groups
- Recurse
Bounding Volume Hierarchy

• Find bounding box of objects
• Split objects into two groups
• Recurse
Bounding Volume Hierarchy

- Find bounding box of objects
- Split objects into two groups
- Recurse
Where to split objects?

- At midpoint  \textit{OR}
- Sort, and put half of the objects on each side  \textit{OR}
- Use modeling hierarchy

MIT EECS 6.837, Durand
Intersection with BVH

- Check sub-volume with closer intersection first
Intersection with BVH

• Don't return intersection immediately if the other subvolume may have a closer intersection
Bounding Volume Hierarchy Discussion

• Advantages
  – easy to construct
  – easy to traverse
  – binary

• 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
Questions?
Kd-trees

- Probably most popular acceleration structure
- Binary trees
- Axis-aligned splits
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

• Allocate primitives to each side
• If a primitive overlaps split, assign to both sides
Kd-tree construction - recurse

- Stop when minimum number of primitives reached
- Other stopping criteria possible
Questions?
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 - naive 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
  - tnear, tfar
- Intersect with splitting plane
  - easy because axis aligned
Kd-tree traversal - three cases

- Intersects only back, only front, or both
- Can be tested base on t, tnear and tfar
Kd-tree traversal - three cases

- If $t > t_{far}$ => intersect only back
- If $t < t_{near}$ => intersect only front
- Note: reversed if sign of ray direction negative
travers(orig, dir, t_near, t_far):
    #adapted from Ingo Wald’s thesis
    #assumes that dir[self.dimSplit] > 0
    if self.isLeaf:
        return intersect(self.listOfTriangles, orig, dir, t_near, t_far)
    d = (self.splitDist - orig[self.dimSplit]) / dir[self.dimSplit];
    if d <= t_near:
        # case one, d <= t_near <= t_far -> cull front side
        return self.frontSideNode.traverse(orig, dir, t_near, t_far)
    elif d >= t_far:
        # case two, t_near <= t_far <= d -> cull back side
        return self.backSideNode.traverse(orig, dir, t_near, t_far)
    else:
        # case three: traverse both sides in turn
        t_hit = self.frontSideNode.traverse(orig, dir, t_near, d)
        if t_hit <= d: return t_hit; # early ray termination
        return self.backSideNode.traverse(orig, dir, d, t_far)
Important details

• For leaves do NOT report intersection if not in t\_near \ t\_far.
  – Important for primitives that overlap multiple nodes
  – Sure, it means redundant calculations, but avoids extra book keeping and potential mistakes

• 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!
Importance

- Given the same traversal code, the quality of a kd tree construction can have a dramatic impact on performance, e.g. a factor of 2 compared to naive middle split
Split in the median

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

- Most people use the surface area heuristic
  - MacDonald and Booth 1990 “Heuristic for ray tracing using space subdivision”
- Idea: simple probabilistic prediction of traversal cost based on split distance
- Then try different possible splits and keep the one with lowest cost
Cost prediction

- Probability that we need to intersect a child
  - Area of the bbox of that child
    (exact for uniformly distributed rays)
- Cost of the traversal of that child
  - number of primitives (simplistic heuristic)
- This heuristic likes to put big densities of primitives in small-area nodes
Efficient implementation

- Not so easy, we need to be able 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.
Questions?
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?
Today

• Motivation – Distribution Ray Tracing
• Bounding Boxes
• Spatial Acceleration Data Structures
  – Regular Grid
  – Adaptive Grids
  – Hierarchical Bounding Volumes
Regular Grid
Create Grid

- Find bounding box of scene
- Choose grid resolution \((n_x, n_y, n_z)\)
- \(\text{grid}_x\) need not = \(\text{grid}_y\)
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!
Where Do We Start?

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

Cell \((i, j)\)
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 \( t_{\text{next}_x} < t_{\text{next}_y} \)
\[
\begin{align*}
    i & += \text{sign}_x \\
    t_{\text{min}} & = t_{\text{next}_x} \\
    t_{\text{next}_x} & += dt_x
\end{align*}
\]
else
\[
\begin{align*}
    j & += \text{sign}_y \\
    t_{\text{min}} & = t_{\text{next}_y} \\
    t_{\text{next}_y} & += dt_y
\end{align*}
\]

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

MIT EECS 6.837, Durand
Pseudo-Code

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

sphere voxelization

primitive density

entered faces

cells traversed

MIT EECS 6.837, Durand
Regular Grid Discussion

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

• Disadvantages?
  – may be only sparsely filled
  – geometry may still be clumped
Questions?
Today

• Motivation – Distribution Ray Tracing
• Bounding Boxes
• Spatial Acceleration Data Structures
  – Regular Grid
  – Adaptive Grids
  – Hierarchical Bounding Volumes
Adaptive Grids

• Subdivide until each cell contains no more than \( n \) elements, or maximum depth \( d \) is reached

Nested Grids

Octree/(Quadtree)
Primitives in an Adaptive Grid

• Can live at intermediate levels, or be pushed to lowest level of grid

Octree/(Quadtree)
Adaptive Grid Discussion

• Advantages?
  – grid complexity matches geometric density

• Disadvantages?
  – more expensive to traverse (especially octree)