Grid Acceleration & Texture Mapping
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 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.backSideNode.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)
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/](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?
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!
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 \((t_{next_x} < t_{next_y})\)

\[
\begin{align*}
    i &= +sign_x \\
    t_{min} &= t_{next_x} \\
    t_{next_x} &= +dt_x
\end{align*}
\]

Else

\[
\begin{align*}
    j &= +sign_y \\
    t_{min} &= t_{next_y} \\
    t_{next_y} &= +dt_y
\end{align*}
\]

\((dir_x, dir_y)\)

If \((dir_x > 0)\) \(sign_x = 1\) else \(sign_x = -1\)

If \((dir_y > 0)\) \(sign_y = 1\) else \(sign_y = -1\)
What's the Next Cell?

- **3DDDA** – Three Dimensional Digital Difference Analyzer

- Similar to Line rendering in 2D
  - but needs to go through all voxels
Where Do We Start?

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

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.
Primitives in an Adaptive Grid

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

• Advantages?
  – grid complexity matches geometric density

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

- Bounding box hierarchy
  - fast to traverse
  - tightly adapts to geometry
  - can be costly to construct (for tightness)

- kd tree
  - super fast to traverse
  - tightly adapts to geometry
  - more costly to construct (for tight construction)

- Grid
  - reasonably fast to traverse
  - not as tight
  - fast to construct (good for dynamic scenes)
Stretch!
Ray tracing recap

- Ray casting: compute visible point at pixel with ray-primitive intersection
- Ray tracing: add secondary rays for shadows, reflection, refraction
- Acceleration Data structures make ray-scene intersection more efficient
Surface appearance

• So far: BRDF
  – Given light and eye direction, how much light is reflected
  – Fresnel term: How much light is reflected vs. transmitted
  – Models angular variations of appearance
• What about spatial variations?
The Quest for Visual Realism

Model

Model + Shading

Model + Shading + Textures

At what point do things start looking real?

For more info on the computer artwork of Jeremy Birn see http://www.3drender.com/jbirn/productions.html
Texture Mapping

- Increase the apparent complexity of simple geometry
- Like wallpapering or gift-wrapping with stretchy paper
- Curved surfaces require extra stretching or even cutting
Texture mapping

3D model
with texture coordinates $u$, $v$
at each vertex

Texture map (2D image)

- 2D $u,v$ texture coordinates specify where the color of a point $P$ in a 3D triangle should be looked up in the 2D texture
Texture mapping

- Interpolate texture coordinates with barycentric coordinates:

\[ u_P = (1-\beta-\gamma)u_A + \beta u_B + \gamma u_C \]
Recap

- Ray cast, get visible point
- Get texture coordinates \((u, v)\) at that point
  - e.g. barycentric coordinates
- Look up texture color
  - Pixel \(u,v\) in texture map
Questions?
Texture Tiling

- Specify a texture coordinate \((u,v)\) at each vertex
- Canonical texture coordinates \((0,0) \rightarrow (1,1)\)
- Wrap around when coordinates >1

*tiles with visible seams*  
*seamless tiling (repeating)*
Problem

• For non-triangle geometry, not per vertex uv

• Solution: deduce uv from x,y,z
  – Various mappings are possible
Common Texture Coordinate Mappings

- Orthogonal
- Cylindrical
- Spherical
- Perspective Projection
- Texture Chart
Projective Textures

• Use the texture like a slide projector
• No need to specify texture coordinates explicitly
• A good model for shading variations due to illumination
• A fair model for reflectance (can use pictures)
Projective Texture Example

- Modeling from photographs
- Using input photos as textures

Figure from Debevec, Taylor & Malik
http://www.debevec.org/Research
MIT EECS 6.837, Durand
Texture Mapping & Illumination

- Texture mapping can be used to alter some or all of the constants in the illumination equation:
  - pixel color, diffuse color, alter the normal, …. 

\[ I_{total} = k_a I_{ambient} + \sum_{i=1}^{\text{lights}} I_i \left( k_d \left( \hat{N} \cdot \hat{L} \right) + k_s \left( \hat{V} \cdot \hat{R} \right)^{n_{\text{shiny}}} \right) \]

**Phong's Illumination Model**
Questions?
Shaders (Material class)

• Functions executed when light interacts with a surface
• Constructor:
  – set shader parameters
• Inputs:
  – Incident radiance
  – Incident & reflected light directions
  – Surface tangent basis (anisotropic shaders only)
• Output:
  – Reflected radiance
Shader

• Initially for production (slow) rendering
  – Renderman in particular
• Now used for real-time (Games)
  – Evaluated by graphics hardware
  – More later in the course
Questions?
Procedural Textures

\[ f(x,y,z) \rightarrow \text{color} \]

Image by Turner Whitted
Procedural Textures

• Advantages:
  – easy to implement in ray tracer
  – more compact than texture maps (especially for solid textures)
  – infinite resolution

• Disadvantages
  – non-intuitive
  – difficult to match existing texture
Perlin noise

- Pseudo-random function
  - But continuous
  - band pass

- Useful to add lots of visual detail

http://www.noisemachine.com/talk1/index.html
http://mrl.nyu.edu/~perlin/doc/oscar.html
http://mrl.nyu.edu/~perlin/noise/
http://en.wikipedia.org/wiki/Perlin_noise
http://freespace.virgin.net/hugo.elias/models/m_perlin.htm
  (not really Perlin noise but very good)
http://portal.acm.org/citation.cfm?id=325247

Ken Perlin
Requirements

• Pseudo random
• For arbitrary dimension
  – 4D is common for animation
• Smooth
• Band pass
• Little memory usage

• How would you do it?
Perlin noise

- Cubic lattice with pseudo gradient at vertices
- Set vertex values to zero
  - To avoid low frequencies
- Splines to interpolate the values
Algorithm

• Given an input point
• For each of its neighboring grid points:
  – Pick a "pseudo-random" gradient vector
  – Compute linear function (dot product)
• Take weighted sum, using ease curves
  – [demo]
Computing the pseudo-random gradient

- Precompute table of permutations \( P[n] \)
- Precompute table of gradients \( G[n] \)
- \( G = G[ ( i + P[ (j + P[k]) \mod n ] ) \mod n ] \)

- in practice only 256 gradients are stored!
  - But optimized so that they are well distributed
Noise
Sum $1/f$ noise

- That is, each octave $f$ has weight $1/f$
Sum $1/f$ |noise|
\[ \sin (x + \text{sum } \frac{1}{f} |\text{noise}|) \]
Comparison

- noise \quad \sin(x + \text{sum} \frac{1}{|\text{noise}|})
Question?
Noise for solid textures

• Wood
  – color = color map (r+noise)
  – http://www.connectedpixel.com/blog/texture/wood

• Marble
  – parallel plane + noise
  – Color = color map ( x + turbulence)
  – http://legakis.net/justin/MarbleApplet/
Corona

• The corona was made as follows:
  – Create a smooth gradient function the drops off radially from bright yellow to dark red.
  – Phase shift this function by adding a turbulence texture to its domain.
  – Place a black cutout disk over the image.

• Animation
  – Scale up over time
  – Use higher dim noise (for time)
    – http://www.noisemachine.com/talk1/imgs/flame500.html
Other cool usage: displacement, fur
Question?
Shaders

- Noise: one ingredient of shaders
- Shaders control diffuse color, but also specular components, maybe even roughness (exponent), transparency, etc.
- Shaders can be layered (e.g. a layer of dust, peeling paint, mortar between bricks).
- Notion of shade tree
  - Pretty much algebraic tree
- Assignment 5:
  checkerboard shader based on two shaders
Bottom line

- Programmable shader provide great flexibility
- Shaders can be extremely complex
  - 10,000 lines of code!
- Writing shaders is a black art
Questions?

Justin Legakis

MIT EECS 6.837, Durand

Justin Legakis