Introduction

- What this tutorial is about
- What it is not about

Overview

 Terrain Visualization

- Problem
  - Given: height values on regular 2D grid
  - Task: render with 60 Hz
- Brute-force solution
  - Render ~ 500 Mio tris
- Better solution
  - view-dependent dyn. LOD, stripes, cache locality
  - Idea: Quadtrees

Avoiding Cracks

- Cannot render quadrangles
- Probably not planar
- Cracks because of T-vertices
- Must render triangles

Subdivision Scheme

- Quadtree induces 4-8 mesh
  - 8
  - new
  - j
  - i

- Induces DAG
  - "vertex j is child of i" \( \iff \) j is created by splitting at i
  - Denote this by an edge \((i,j)\)
Dependency among Triangles

- Graph-theoretic condition
  Let $M^0$ be the complete DAG, let $M$ be a sub-graph of $M^0$.
  $M$ yields a crack-free terrain $\iff$ 
  $\forall j \in M : (i,j) \in M^0 \implies (i,j) \in M$

- Rendering condition:
  - Find criterion for vertices that has the "nesting property":
    criterion($j$) = "render it" $\implies$ 
    $\forall$ parents $i$: criterion($i$) = "render it"

Procedure for Rendering

submesh($i,j$)
if error($i$) < $\tau$ then 
  return 
if $B$, outside view then 
  return 
submesh($i,c$)
$V += p_i$
submesh($i,cr$)

Storing the Quadtree

- Don't use pointers
- Find numbering scheme with little "dead numbers"
- Observation: subdivision scheme induces 2 quadtrees

<table>
<thead>
<tr>
<th>Level</th>
<th>Green</th>
<th>Black</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Storing the "green" quadtree in the "black" one:

Movies

[Black screen]

[Image of the Australian Outback]
Isosurface Generation

- **Problem**
  - Given: scalar field \( f: \mathbb{R}^3 \rightarrow \mathbb{R} \)
  - Task: find polygonal repr. of \( f(x) = t \)
  - Discrete: curvilinear grid / regular grid
  - Space: physical / computational space
  - Task (discrete): find all cells with a node < \( t \) and a node > \( t \)

- **Simple algo ("marching cubes")**
  1. Compute sign for all nodes (\( \oplus > t \));
  2. Triangulate all cells according to LUT

Octrees over Volume Data

- **Leaf**: ptr to lower left node
- **Inner node**: ptr to first child
- All nodes \( v \) store \( v_{\min} \) and \( v_{\max} \)

Isosurface Generation with Octree

- Isosurface intersects volume assoc. with node \( v \) \( \iff \) \( v_{\min} < t < v_{\max} \)
- **Algo (obvious)**
  - Start with root
  - Recurse into nodes satisfying condition
- **Improvement**
  - Observation: edges are visited exactly 4 times
  - Keep hash table of edges

Ray Shooting

- **Applications**: ray tracing, radiosity, volume visualization, terrain following, etc.
- **Simplest solution**: grid
- **3D octree**
  - Bottom-up
  - Top-down

5D Octree for Rays

- **What is a ray?**
  - Point + direction = 5-dim. Object
- **Octree over rays**
  - "Direction cube"
  - One-to-one mapping for dir's:
    \( S^2 \leftrightarrow D = [-1,1]^2 \times \{x,y,z\} \)
  - All rays in universe \( U = [0,1]^2 \)
  - Node of 5D octree = beam in 3D
**Introduction**

- Construction
  - Start with root node \( U = [-1, 1]^2 \) and all objects associated
  - Partition node iff
    1. Too many objects, and
    2. Cell too large.
  - Partition set of objects

- Shooting rays
  1. Convert ray to 5D point
  2. Find leaf of octree
  3. Intersect ray with associated objects

- Optimizations ...

---

**Texture Synthesis**

- Properties of textures
  - Stationary under moving window
  - Locality of dependency

- Algorithm
  
  ```
  for all \( p \in \) new image do
    find \( p_i \in \) old image so that
    \[ \|p - N(p)\| = \min \] set \( p_i = p \)
  ```

---

**Nearest Neighbor Apps**

- Better independence from size of \( N(p) \)

---

**BSP Trees**

- Generalization of k-d trees

- Definition (recursive)
  - \( S = \) set of objects,
  - \( S(v) = \) objects assoc. with node \( v \),
  - \( T(S) = \) BSP for set \( S \)
  1. Case \( |S| = 1 \):
     - \( T = \) leaf \( v \) storing \( S(v) \)
  2. Case \( |S| = 3 \):
     - \( T = \) tree with root \( v \) storing \( h \), and \( S(v) \),
     - \( S^+\) = \( \{ x \in S \mid x \subseteq h \} \)
     - children for sets \( S^+(v) \) and \( S^-\)
     - \( S^-(v) = \{ x \in h | x \subseteq S \} \)

---

**Autopartitions**

- Properties
  - Each \( h \) = plane of one polygon
  - Each \( S(v) = \) that polygon

- Complexity
  - \( O(n \log n) \)
  - In 2D: proven
  - In 3D: experience for "well-behaved" geometry
BSPs for Object Representation

- Difference to orig. definition:
  - stop only when |S|=0
- Leaves
  - Homogenous convex cells
  - Either inside or outside
- Construction
  - Guided by heuristic

Boolean Operations

- Operations: ∩ ∪ ⊆

Algorithm
1. Split BSP by plane
2. Merge two BSPs
3. Compute operation on cells

Subalgorithm 1

- Split BSP T by plane H, polygon p at root of T
- Output two new BSPs
- Cases:
  1. T is leaf:
     - trivial ...
  2. p ⊂ H:
     - return children
  3. H completely on one side of p:
     - split one child, combine with other child
  4. H crosses p:
     - split both children, recombine across p

Subalgorithm 2

- Merge T₁ and T₂
- Output T with leaf cells C such that
  \[ C = \{ C \mid C = C_i \cap C_j, C_i, C_j \in C \} \]
- Algorithm
1. T₁ or T₂ is leaf: perform operation on cell
2. Else:
   - \[ T_1 \text{split} \rightarrow T'_1 \text{merge} \rightarrow T' \]
   - \[ T_2 \text{split} \rightarrow T'_2 \text{merge} \rightarrow T' \]

Subalgorithm 3

- The Cell Operation

<table>
<thead>
<tr>
<th>Op</th>
<th>T₁</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>∩</td>
<td>in</td>
<td>T₁₁</td>
</tr>
<tr>
<td></td>
<td>out</td>
<td>T₁₂</td>
</tr>
<tr>
<td>∪</td>
<td>in</td>
<td>T₂₁</td>
</tr>
<tr>
<td></td>
<td>out</td>
<td>T₂₂</td>
</tr>
<tr>
<td>\</td>
<td>in</td>
<td>T₁₂</td>
</tr>
<tr>
<td></td>
<td>out</td>
<td>T₁₂</td>
</tr>
<tr>
<td>⊆</td>
<td>in</td>
<td>T₁₂</td>
</tr>
<tr>
<td></td>
<td>out</td>
<td>T₂₂</td>
</tr>
</tbody>
</table>

Demos
Bounding Volume Hierarchies

- Definition (informal):
  - Tree, nodes carry BV
  - Leaves carry one (or more) “primitives”
  - BV of node contains BVs of all children
  - Leaf BV contains primitive
- Many variables
- Bounding Volumes
- Tightness

Applications
- Ray shooting
- Nearest-neighbor
- Frustum and occlusion culling
- Geographical data bases
- Collision detection

Construction
- Strategies
  - Bottom-up
  - Insertion
  - Top-down
  - Heuristic
- Interactive hierarchy construction

Collision Detection

Simultaneous traversal:
```
traverse(A,B)
if A,B do not overlap then
  return
if A and B are leaves then
  check primitives
else
  for all children A_i, B_j do
    traverse(A_i, B_j)
```

The recursion tree (what the algo really does):

Movies

Remaining primitives

A simple application

Thanks Folks
A Continuum of Data Structures

Quadtree  K-d tree  BSP tree  BV hierarchy