Spatial Data Structures for Computer Graphics

http://www.cse.iitb.ac.in/~sharat
November 2008
Spatial Data Structures for Computer Graphics

http://www.cse.iitb.ac.in/~sharat
November 2008

Acknowledgements:
Joint work with Biswarup Choudhury and Rhushabh Goradia
Spatial Data Structures for Computer Graphics

http://www.cse.iitb.ac.in/~sharat
November 2008

Acknowledgements:
Joint work with Biswarup Choudhury and Rhushabh Goradia
Most examples are from ViGIL IIT Bombay.
Talk Overview

Exhibit B: Which picture do you like?
Talk Overview

► Background about this talk
► Application in Vision
  • Exhibit A: kd-trees

Exhibit B: Which picture do you like?
Talk Overview

Exhibit B: Which picture do you like?

- Background about this talk
- Application in Vision
  - Exhibit A: kd-trees
- Application in Imaging
  - Exhibit B: Octrees
Talk Overview

- Background about this talk
- Application in Vision
  - Exhibit A: kd-trees
- Application in Imaging
  - Exhibit B: Octrees
- Application in Graphics
  - Exhibit C: Space Filling Curves, Compressed Octrees

Exhibit B: Which picture do you like?
Background

- Why this talk: Role of traditional CS and CSE in graphics

Exhibit C: Visibility Map
Background

- Why this talk: Role of traditional CS and CSE in graphics
- Not a theory talk (algorithms are correct, but may not be optimal in a big-Oh sense)

Exhibit C: Visibility Map
Background

- Why this talk: Role of traditional CS and CSE in graphics

- Not a theory talk (algorithms are correct, but may not be optimal in a big-Oh sense)

- Use of images, and points as primitives in computer graphics

Exhibit C: Visibility Map
Background

- Why this talk: Role of traditional CS and CSE in graphics
- Not a theory talk (algorithms are correct, but may not be optimal in a big-Oh sense)
- Use of images, and points as primitives in computer graphics
- Please feel free to interrupt and ask questions at any stage
Background

- Why this talk: Role of traditional CS and CSE in graphics
- Not a theory talk (algorithms are correct, but may not be optimal in a big-Oh sense)
- Use of images, and points as primitives in computer graphics
- Please feel free to interrupt and ask questions at any stage

Vision, Graphics, and Imaging: Forward and Inverse Problems
Talk Overview

✔ Background about this talk

→ Application in Vision
  • Exhibit A: kd-trees

→ Application in Imaging
  • Exhibit B: Octrees

→ Application in Graphics
  • Exhibit C: Space Filling Curves, Compressed Octrees
Problem Definition: Object Pose

Object Pose: Given images of a static object, how to create the illusion of realistic motion of the object along any arbitrary path — composed realistically in arbitrary environments?
Problem Definition: Object Pose

Object Pose: Given images of a static object, how to create the illusion of realistic motion of the object along any arbitrary path — composed realistically in arbitrary environments?

[Play Video]
Problem Definition: Object Pose

**Object Pose:** Given images of a static object, how to create the illusion of realistic motion of the object along any arbitrary path — composed realistically in arbitrary environments?
1. Object Pose Solution

Image Acquisition (Studio)

Computing Shape

Path specification (Run-Time)

 Computing Color
Clearly, we need to compute the shape and color from previously acquired close by images. ... nearest neighbour computation in two dimensions.
1. Object Pose Solution

Image Acquisition (Studio)

Path specification (Run-Time)

Computing Shape

Computing Color

Clearly, we need to compute the shape and color from previously acquired close by images .... nearest neighbour computation in two dimensions [Play Video]
View Interpolation via Visual Hull
The Visual Hull

Carve

Carve again
2. Computer Vision In Action

Hull

Compared to the real thing
More details on the method

Question: Can we use the studio pictures ALSO for color computation?
Color Computation

Requires solution to: Given images of an object captured under a set of lighting conditions, how to efficiently render the scene under new illumination configurations?
Color Computation

Requires solution to: Nearest Neighbors, and Bi-chromatic Nearest Neighbors
Output Frame
3. **kd-trees**

- Dimension of data is $k$ (but common to say k-d tree of dimension 3 instead of 3d-tree).
- kd-trees are binary trees
- Designed to handle spatial data in a simple way
- For $n$ points, $O(n)$ space, $O(\log n)$ height (if balanced), supports range and nearest-neighbor queries.

- Node consists of
  - Two child pointers,
  - Satellite information (such as name).
  - A key: Either a single float representing a coordinate value, or a pair of floats (representing a dimension of a rectangle)
Basic Idea Behind kd-trees

Construct a binary tree

- At each step, choose one of the coordinate as a basis of dividing the rest of the points
- For example, at the root, choose $x$ as the basis
  - Like binary search trees, all items to the left of root will have the $x$-coordinate less than that of the root
  - All items to the right of the root will have the $x$-coordinate greater than (or equal to) that of the root
- Choose $y$ as the basis for discrimination for the root’s children
- And choose $x$ again for the root’s grandchildren

Note: Equality (corresponding to right child) is significant
Example: Construct kd-tree Given Points

- Coordinates of points are (35,90), (70,80), (10,75), (80,40), (50,90), (70,30), (90,60), (50,25), (25,10), (20,50), and (60,10)

- Points may be given one a time, or all at once.

- Data best visualized as shown below
Example: kdtree Insertion
Nearest Neighbors: The scenario

- “Find the nearest Pizza Hut.” (Compare with the McDonald problem).
- Assume kd-tree T given, and C is the region associated with a node.
- Input p is a point
- Searching for point p in T helps
  - In one dimension, T is very useful: the closest neighbor is from the set of nodes visited (MANY nodes are pruned)
  - In higher dimensions, T is not as useful (the closest neighbor may be far away).
- Nevertheless, pruning is possible.
- General strategy: Collect partial results, judicial traversal, and prune.
What If We Locate Point?

We visit (35,90), (70,80), \ldots, and fall off (70,30)

Closest point is nowhere near this path. We must visit both subtrees.
Nearest Neighbor: Pruned version

- Maintain the rectangle $r$ associated with a node
- Compute a lower bound on the distance from the query $q$ to the rectangle
  - Distance between $q$ and any point in $r$ is at least $\text{lowerbound}(r, q)$
  - Do not compute all distances between $q$ and every point in $r$
- $\text{lowerbound}()$ helps because if the lower bound is larger than the distance computed so far, we do not consider many points
- Must compute $\text{lowerbound}()$ quickly
Nearest Neighbor: Pruned Version

```c
float lowerbound(Rectangle r, Point p) {
    if (r.inside(p)) return 0;
    if (r.left(p)) return r.minX - p.x;
    ...
}
```

```c
Result process(KDNode k, int cd, Rectangle r, Result res) {
    if (k == null) return res;
    if (lowerbound(r, query) >= res.distance) return res;
    ...
}
```

- If the lower bound is larger than the distance computed so far, exit!
- Otherwise compute the distance with the current node
- Process the two children in order!
Pruned Version: Root (35,90)
Pruned Version: (70,80)
Pruned Version: (80,40)
Pruned Version: (70,30)
Pruned Version: (50,25)
Pruned Version: (60,10)

Note: (90,60) and (50,90) will be skipped next!
Pruned Version: (10,75)
Pruned Version: (25,10)
Pruned Version: Answer
Which Nodes Are Processed?
Any Improvement Possible?

• Nine nodes examined. Can we do better?
• Yes, if decision to pick a node is based on dynamic changing costs instead of initial left-right decision.
• Use a priority queue. (Seven nodes examined instead of nine)
Talk Overview

✔ Background about this talk
✔ Application in Vision
  ● Exhibit A: kd-trees
  → Application in Imaging
    ● Exhibit B: Octrees
→ Application in Graphics
  ● Exhibit C: Space Filling Curves, Compressed Octrees

Exhibit B: Which picture do you like?
4. Color Quantization

- By “quantization,” here we mean the process of changing the given \( N \) number of colors to \( K \) colors. In the picture below, an image with \( N=33694 \) colors is changed to one with \( K=64 \) colors.
4. Color Quantization

- By “quantization,” here we mean the process of changing the given $N$ number of colors to $K$ colors. In the picture below, an image with $N=33694$ colors is changed to one with $K=64$ colors.

- Why? Lightweight; thus easy to transmit, store, share
4. Color Quantization

- By “quantization,” here we mean the process of changing the given $N$ number of colors to $K$ colors. In the picture below, an image with $N=33694$ colors is changed to one with $K=64$ colors.

- Why? Lightweight; thus easy to transmit, store, share
- Mechanics? Create as output a lookup table
- Can be done in any color space
4. Color Quantization

- By “quantization,” here we mean the process of changing the given \( N \) number of colors to \( K \) colors. In the picture below, an image with \( N=33694 \) colors is changed to one with \( K=64 \) colors.

- Why? Lightweight; thus easy to transmit, store, share
- Mechanics? Create as output a lookup table
- Can be done in any color space
- Basic challenge: Efficient implementation of reverse lookup table, good representation of colors
Digression: Web Safe Colors

- 216 colors have been accepted as “safe” to display. Some of these colors are gray
- Each channel can take only values 00, 33, 66, 99, CC, and FF
- Figures shows the colors organized in descending RGB values. Value at top left is FFFFFF; first item in the second row has the value FFCCFF
Uniform Quantization

- First find the maximum and minimum range and allocate numbers R, G, and B such that R*G*B<=K

- \textit{Uniformly} divide the range. For example, the range \([r_{\min}, r_{\max}]\) is divided into R parts.

- Map these R values to actual colors. For example, if R=4, then we might chose 1, 2, 4 and 8
Uniform Quantization

• First find the maximum and minimum range and allocate numbers R, G, and B such that $R \times G \times B \leq K$

• *Uniformly* divide the range. For example, the range $[r_{\min}, r_{\max}]$ is divided into R parts.

• Map these R values to actual colors. For example, if $R=4$, then we might chose 1, 2, 4 and 8

• Advantages:
Uniform Quantization

• First find the maximum and minimum range and allocate numbers R, G, and B such that R*G*B ≤ K

• **Uniformly** divide the range. For example, the range $[r_{\text{min}}, r_{\text{max}}]$ is divided into R parts.

• Map these R values to actual colors. For example, if R=4, then we might chose 1, 2, 4 and 8

• Advantages: Easy to implement

• Drawbacks:
Uniform Quantization

- First find the maximum and minimum range and allocate numbers R, G, and B such that R*G*B ≤ K

- Uniformly divide the range. For example, the range $[r_{\text{min}}, r_{\text{max}}]$ is divided into R parts.

- Map these R values to actual colors. For example, if R=4, then we might chose 1, 2, 4 and 8

- Advantages: Easy to implement

- Drawbacks: Distribution of colors is ignored resulting in banding or false contours
Popularity Algorithm

- Multi-pass algorithm
Popularity Algorithm

- Multi-pass algorithm
- Count number of cells having a particular set of color, and take the top K colors
Popularity Algorithm

- Multi-pass algorithm
- Count number of cells having a particular set of color, and take the top K colors
- But first quantize the colors ($2^{32} \rightarrow 2^{15}$)
Popularity Algorithm

- Multi-pass algorithm
- Count number of cells having a particular set of color, and take the top K colors
- But first quantize the colors \(2^{32} \rightarrow 2^{15}\)
- Second pass: For each pixel, find the “closest” color
Popularity Algorithm

- Multi-pass algorithm
- Count number of cells having a particular set of color, and take the top K colors
- But first quantize the colors ($2^{32} \rightarrow 2^{15}$)
- Second pass: For each pixel, find the “closest” color
- Advantages:
Popularity Algorithm

- Multi-pass algorithm
- Count number of cells having a particular set of color, and take the top K colors
- But first quantize the colors \(2^{32} \rightarrow 2^{15}\)
- Second pass: For each pixel, find the “closest” color
- Advantages: Easy to implement, popular colors show up in the resulting image
Popularity Algorithm

- Multi-pass algorithm
- Count number of cells having a particular set of color, and take the top K colors
- But first quantize the colors ($2^{32} \rightarrow 2^{15}$)
- Second pass: For each pixel, find the “closest” color
- Advantages: Easy to implement, popular colors show up in the resulting image
- Disadvantages:
Popularity Algorithm

- Multi-pass algorithm
- Count number of cells having a particular set of color, and take the top K colors
- But first quantize the colors \((2^{32} \rightarrow 2^{15})\)
- Second pass: For each pixel, find the “closest” color
- Advantages: Easy to implement, popular colors show up in the resulting image
- Disadvantages: Expensive; clusters of ‘unpopular’ but significant colors are ignored
Median-Cut Algorithm

- Goal: Adaptively subdivide color block into K parts so that each sub-block has the same number of colors
Median-Cut Algorithm

- Goal: Adaptively subdivide color block into K parts so that each sub-block has the same number of colors
- Step 1: Divide the longest dimension by a plane into two parts so that there are equal number of colors (in this dimension)
- Recursively apply the same method until the number of sub-blocks is K
- Color reduction efficiently implemented using kd-Trees
- Inverse color lookup is still slow
Octree based quantization

Inverse color lookup is done using the bits of the incoming color.

<table>
<thead>
<tr>
<th>R</th>
<th>G</th>
<th>B</th>
<th>child:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>
Octree based quantization

Inverse color lookup is done using the bits of the incoming color.

<table>
<thead>
<tr>
<th>R</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

child: 5 3 6 0 7 1 1 6

Surprisingly, color reduction is also done using the same oct-tree
Talk Overview

- Background about this talk
- Application in Vision
  - Exhibit A: kd-trees
- Application in Imaging
  - Exhibit B: Octrees
- Application in Graphics
  - Exhibit C: Space Filling Curves, Compressed Octrees
5. Point Models

- Model surfaces as points
- Each point has attributes: [coordinates, normal, reflectance, emissivity]
5. Point Models

- Model surfaces as points
- Each point has attributes: [coordinates, normal, reflectance, emissivity]
- Immediate question: Why not triangles, why points? And how do we get these points?
Polygons v/s points: LOD

- Level of Detail (LOD) based hierarchy is simpler in point based models
Visibility Between Point Pairs

VISIBILITY IN POLYGONAL MODELS

VISIBILITY IN POINT MODELS
Visibility Between Point Pairs

- View dependent visibility versus view independent visibility
Visibility Between Point Pairs

- View dependent visibility versus view independent visibility

- Although view dependent visibility based point based rendering solutions exist,
• View dependent visibility versus view independent visibility

• Although view dependent visibility based point based rendering solutions exist, we present the first global illumination solution for point models based on the view independent paradigm.
Visibility Between Point Pairs

View Independent Visibility calculation between point pairs is essential to give correct Global Illumination results as a point receives energy from other point only if it is visible.
Hierarchical Visibility

Hierarchical Visibility enables quick answers to visibility queries, thus enabling a faster GI solution.
Hierarchical Visibility

Key Notion: We define a **Visibility Map (V-map)** for the resulting tree to enable *quick* answers to visibility queries.
6. Visibility Map

- The visibility map for a tree is a collection of visibility links for every node in the tree.
- The visibility link for any node \( N \) is a set \( L \) of nodes.
- Every point in any node in \( L \) is guaranteed to be visible from every point in \( N \).
What is a Visibility Map (V-map)?

With respect to at any level,

- **COMPLETELY INVISIBLE**
- **COMPLETELY VISIBLE**
- **PARTIALLY VISIBLE**
What is a Visibility Map (V-Map)?

With respect to any level,

- **COMPLETELY INVISIBLE**
- **COMPLETELY VISIBLE**
- **PARTIALLY VISIBLE**
What is a Visibility Map (V-Map)?

With respect to at any level,
- **COMPLETELY INVISIBLE**
- **COMPLETELY VISIBLE**
- **PARTIALLY VISIBLE**
Visibility Map Queries?

Visibility map entertain efficient answers:

1. Is point $x$ visible from point $y$?

2. What is the visibility status of $u$ points around $x$ with respect to $v$ points around $y$?
   - Repeat a “primitive” point-point visibility query $uv$ times
   - V-map gives the answer with $O(1)$ point-point visibility queries.

3. Given a point $x$ and a ray $R$, determine the first object of intersection.

4. Is point $x$ in the shadow (umbra) of a light source?

All queries answered with a simple octree traversal.
Quadtrees

Given a set of points, we need to know how to build octrees, or
Quadtrees

Given a set of points, we need to know how to build octrees, or for that matter, quadtrees

A quadtree built on a set of 10 points in 2D
Quadtrees

Given a set of points, we need to know how to build octrees, or for that matter, quadtrees

A quadtree built on a set of 10 points in 2D

Important: Interested in a parallel algorithm for building octrees
Compressed Quadtrees

Each node in compressed octree is either a leaf or has at least two children

This is going to be very useful
7. Space Filling Curves

- We recursively bisect space into $2^k \times 2^k \times 2^k$ non-overlapping cells of equal size.
- Key Idea: Mapping of these cells to a 1D linear ordering
7. Space Filling Curves

- We recursively bisect space into $2^k \times 2^k \times 2^k$ non-overlapping cells of equal size.

- Key Idea: Mapping of these cells to a 1D linear ordering

Each cell is thus assigned an index — which can be done cleverly.

The Sfc index for $k=2$ for a cell in 3 Dimensions ($d=3$) with coordinates $(3, 1, 2) = (11, 01, 10)$ is $101110 = 13$
Octrees and Sfcs

- Leaves when sorted by Sfc indices represent bottom traversal
- In fact, this represents a post order traversal of a quadtree
Octrees and Sfcs

- Leaves when sorted by Sfc indices represent bottom traversal
- In fact, this represents a post order traversal of a quadtree

Given two nodes \( c_1 \) and \( c_2 \),
Octrees and Sfcs

- Leaves when sorted by Sfc indices represent bottom traversal
- In fact, this represents a post order traversal of a quadtree

- Better still, can be viewed as multiple Sfcs at various resolutions
- Even better, parent can be generated from child’s Sfc

Given two nodes $c_1$ and $c_2$, The longest common prefix of the Sfc indices (that’s a multiple of the dimension d) represents the Sfc index of the LCA.
Talk Summary

✔ Spatial data structures is core in Vi,G,I
✔ Parallel algorithms on GPU are doable
✔ Application in Vision (Exhibit A: kd-trees)
✔ Application in Imaging (Exhibit B: Oc-trees)
✔ Application in Graphics (Exhibit C: Compressed Octrees)

Hope you liked it.. Email questions to sharat @ iitb.ac.in