Course 16
Geometric Data Structures for Computer Graphics
Generic Dynamization

Dr. Elmar Langetepe
Institut für Informatik I
Universität Bonn
Motivation

- Some geometric data structures were considered
- So far we assumed that the structure is static
- Fixed set of objects
- Objects may changes over time, Delete, Insert
- Need special update techniques for Insert and Delete for every data structure
- **Now:** One Delete/Insert techniques suitable for many data structures
- **Generic dynamization**
Motivation

- Consider *Static* geometric data structure for \( n \) fixed objects
- Assume: Static version is easy to implement
- Assume: Efficient *Range Queries* on fixed set
- Assume: Objects changes over time, Delete, Insert

Implement Delete/Insert only once
For many data structures (no special dynamization)
Model: $V \in TStat$

$V$ is a static geometric data structure of Type $TStat$

With object set $D$

Operations:

$\text{build}(V, D)$: Build the structure $V$ of type $TStat$ with all data objects in the set $D$.

$\text{query}(V, q)$: Gives the answer (objects of $D$) to a query to $V$ with query object $q$.

$\text{extract}(V, D)$: Collects all data objects $D$ of $V$ in a single set and returns a pointer to it.
Example: Balanced $k$-d-tree

Balanced $k$-d-tree of set $D$:

**build**: $O(n \log n)$

**query**, orthogonal range: $O(\sqrt{n} + a)$
Dynamic structure \( W \in T_{Dyn} \)

Operations:

**Build**\((W, D)\): Build the structure \( W \) of type \( T_{Dyn} \) with data objects in the set \( D \).

**Query**\((W, q)\): Gives the answer (objects of \( D \)) to a query to \( W \) with query object \( q \).

**Extract**\((W, D)\): Collects all data objects \( D \) of \( W \) in a single set and returns a pointer to it.

**Insert**\((W, d)\): Insert object \( d \) into \( W \).

**Delete**\((W, d)\): Delete \( d \) out of \( W \).
Model of the dynamization

<table>
<thead>
<tr>
<th>TStat</th>
<th>TDyn</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time:</td>
</tr>
<tr>
<td></td>
<td>Build(W,D)</td>
</tr>
<tr>
<td></td>
<td>Query(W,D)</td>
</tr>
<tr>
<td></td>
<td>Extract(W,D)</td>
</tr>
<tr>
<td></td>
<td>Insert(W,d)</td>
</tr>
<tr>
<td></td>
<td>Delete(W,d)</td>
</tr>
<tr>
<td></td>
<td>Time:</td>
</tr>
<tr>
<td></td>
<td>$B_V(n)$</td>
</tr>
<tr>
<td></td>
<td>$Q_V(n)$</td>
</tr>
<tr>
<td></td>
<td>$E_V(n)$</td>
</tr>
<tr>
<td></td>
<td>$I_W(n)$</td>
</tr>
<tr>
<td></td>
<td>$D_W(n)$</td>
</tr>
</tbody>
</table>
Simple *Throw-Away* solution

\[
\text{Insert}(W, d) : \quad \text{extract}(W, D); \ \text{\textbf{build}}(W, D \cup \{d\})
\]

\[
\text{Delete}(W, d) : \quad \text{extract}(W, D); \ \text{\textbf{build}}(W, D \setminus \{d\}).
\]

Ineffective, approach must cope with \textit{small} changes.
Requirements

- Query operation: **Decomposable**
  - \( V = V_1 \cup V_2 \cup \cdots \cup V_j \Rightarrow (query(V_i, d) \Rightarrow query(V, d)) \)
  - Almost all geometric DS, **Example: k-d tree**

- Time-Functions **increase monotonically in** \( n \)
  - Functions like: \( n, n \log n, n^2, 2^n, \sqrt{n} \)
  - **Example: k-d tree,**
    \[ B_V(n) = O(n \log n), \ E_V(n) = O(n), \ Q_V(n) = O(\sqrt{n} + a) \]

- A few others, normally fulfilled.
Amortized Insert: Binary structure

- Set $D$ of objects, $|D| = n$
- Decompose $D$ into sets $D_i$
- Binary representation of $n$:
  - $n = a_l 2^l + a_{l-1} 2^{l-1} + \ldots + a_1 2 + a_0$ mit $a_i \in \{0, 1\}$
  - Binary representation: $a_l a_{l-1} \ldots a_1 a_0$
- $a_i = 1$ $\Rightarrow$ build static structure $V_i$ with $2^i$ elements of $D$. 
Amortized Insert: Binary structure

Set $D$ of objects, $|D| = n$

$|D| = n = 11 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$

Binary structure $W_n$, consists of some static structures $V_i$
k-d tree Binary structure

\[ 11 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \]
Binary structure: Operations cost

\[ 2^3 \begin{array}{cccc} \text{V}_3 \\ \text{V}_2 \\ \text{V}_1 \\ \text{V}_0 \end{array} \]

**Build**\((W,D)\): 
\( B_W(n) \in O(B_V(n)) \) (\textit{k-d-tree}: \( O(n \log n) \))

**Extract**\((W,D)\): 
\( E_W(n) \leq \log n \ E_V(n) \) (\textit{k-d-tree}: \( O(n \log n) \))

**Query**\((W,D)\): 
\( Q_W(n) \leq \log n \ Q_v(n) \) (\textit{k-d-tree}: \( O(\log n(\sqrt{n} + a)) \))
Amortized Insert

Insert new element \( d \): Structural changes of the \emph{binary representation}.

Example:

\[
\begin{array}{cccc}
2^3 & & & V_3 \\
2^2 & & & \\
2^1 & & V_1 \\
2^0 & & & \\
\end{array}
\quad
\begin{array}{cccc}
2^3 & & & V_3 \\
2^2 & & & V_2 \\
2^1 & & & \\
2^0 & & & \\
\end{array}
\]

\[W_{11} \xrightarrow{} W_{12}\]

To Do:

\[
\text{extract}(V_0, D_0); \quad \text{extract}(V_1, D_1); \quad \text{build}(V_2, D);
\]

Next insert without changes of old parts!

Reconstruction partially!
Example: k-d tree Binary structure

\[ |D| = 11 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \]

\[ |D \cup \{ l \}| = 12 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 \]
Amortized Insert: Costs

Amortized time: Start with empty structure

Sequence $S$ of $|S| = s$ operations, $k$ Insert operations

$S = (\text{Insert, } op2, \text{op1, Insert, Insert, Insert, } op1, \ldots)$

\[
\frac{\text{tot. cost of } k \text{ Insert oper.}}{k} \leq \bar{I}(s)
\]

Means: Insert in $\bar{I}(s)$ amortized time.

One can show:

\[
\bar{I}_W(s) \in O\left(\frac{\log s}{s} B_V(s)\right).
\]

($k$-$d$-tree: $O(\log^2 s)$)
Amortized Delete

- First: Stand alone, without generic Insert(Bin. Str.)
- Weak-Delete operation on static structure
- Example: $k$-$d$-tree
  - Mark point as deleted
  - Proceed as before
  - Occasional reconstruct
Weak Delete: \textit{k-d-tree}

|D| = 7

|V| = O(11)

Reconstruct completely IFF \( D \) has only the half-size of \( V \)
Amortized Delete: Results

- Requires Static structure with $weak.Delete(V, d)$ operation
- Cost function: $WD_V(n)$ ($k$-$d$-tree: $O(\log n)$)
- $r$ size of the actual data set, $s$ length of operation sequence
- Dynamization by Occasional reconstruction

\[
B_W(r) = B_V(r) (\text{ $k$-$d$-tree: } O(r \log r))
\]
\[
E_W(r) \in O(E_V(r)) (\text{ $k$-$d$-tree: } O(r))
\]
\[
Q_W(r) \in O(Q_V(r)) (\text{ $k$-$d$-tree: } O(\sqrt{r} + a))
\]

$S = (Insert, op1, Insert, op2, Delete, op2, Delete, op1, \ldots)$
Start with empty structure
Amortized Delete: $|S| = s$
\[
\overline{D}_W(s) \in O\left(\frac{WD_V(s) + B_V(s)}{s}\right) (\text{ $k$-$d$-tree: } O(\log s))
\]
**Weak.Delete**($W, d$): Find the structure $V_i$ of binary structure $W$ implemented by a searchtree $T$ for all elements.
Results: Amortized Insert/Delete

- $r$ size of the actual data set, $s$ length of operation sequence
- $S = (\text{Insert, } op2, \text{Insert, } \text{Delete, } op2, \text{Delete, } \text{Insert, } op1, \ldots)$

Amortized time for insertion:

$$\overline{I}_W(s) \in O \left( \log s \frac{B_V(s)}{s} \right) \text{ (k-d-tree: } O(\log^2 s)), $$

Amortized time for deletion:

$$\overline{D}_W(s) \in O \left( \log s + WD_V(s) + \frac{B_V(s)}{s} \right) \text{ (k-d-tree: } O(\log s)). $$

Other operations:

- $B_W(r) = B_V(r) \text{ (k-d-tree: } O(r \log r))$
- $E_W(r) \in O(\log r E_V(r)) \text{ (k-d-tree: } O(r \log r))$
- $Q_W(r) \in O(\log r Q_V(r)) \text{ (k-d-tree: } O(\log r (\sqrt{r} + a)))$
Conclusion

• Simple generic dynamization techniques
• Easy to implement: Binary structure/Occasional reconstruction
• Amortized Delete and Insert
• Applicable for many geometric data structures
• Efficient: $\log$ factor
• Does not waste storage
• Worst-Case sensitive: Amortize the reconstruction itself