12. Data Structures for Graphics

How would we be able to answer the following queries in constant time?

- Given a polygon, what are the adjacent polygons?
- Given an edge, which 2 polygons share it?
- Given a vertex, which faces share it?
- Given a vertex, which edges share it?
Can the following data structure fulfill the requirement?
Can the following data structure fulfill the requirement?

An alternative:

12. Data Structures for Graphics
12. Data Structures for Graphics

Can the following data structure fulfill the requirement?

An alternative:

- **Face Table**
- **Edge Index Table**
- **Vertex Table**
- **Edge Table**
12. Data Structures for Graphics

Can the following data structure fulfill the requirement?

An alternative:

Edge Table

Vertex Table

Edge Index Table
12. Data Structures for Graphics

Can the following data structure fulfill the requirement?

An alternative:

**Vertex Table**

<table>
<thead>
<tr>
<th>V₀</th>
<th>V₁</th>
<th># of degree</th>
<th># of degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>X₀</td>
<td>Y₀</td>
<td>Z₀</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Edge Index Table**

<table>
<thead>
<tr>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

**Edge Table**

* *
12.1 Winged-Edge Data Structure

- To avoid using variable-length data structures
- Hide the implementation behind a class interface
Example: representing a tetrahedron

### Edge Table

<table>
<thead>
<tr>
<th>Edge Name</th>
<th>Vertices from</th>
<th>Vertices to</th>
<th>Faces left</th>
<th>Faces right</th>
<th>Clockwise Pred</th>
<th>Clockwise Succ</th>
<th>Counter-Clockwise Pred</th>
<th>Counter-Clockwise Succ</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>2</td>
<td>A</td>
<td>D</td>
<td>e</td>
<td>d</td>
<td>f</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>3</td>
<td>B</td>
<td>D</td>
<td>c</td>
<td>e</td>
<td>a</td>
<td>f</td>
</tr>
<tr>
<td>f</td>
<td>3</td>
<td>1</td>
<td>C</td>
<td>D</td>
<td>d</td>
<td>c</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>c</td>
<td>3</td>
<td>4</td>
<td>B</td>
<td>C</td>
<td>e</td>
<td>b</td>
<td>f</td>
<td>d</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
<td>4</td>
<td>C</td>
<td>A</td>
<td>c</td>
<td>f</td>
<td>a</td>
<td>e</td>
</tr>
<tr>
<td>e</td>
<td>2</td>
<td>4</td>
<td>A</td>
<td>B</td>
<td>d</td>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
</tbody>
</table>
Example: representing a tetrahedron

- **Vertex-Edge Table**
  - Vertex | Edge
  - 1     | d
  - 2     | b
  - 3     | b
  - 4     | c

- **Face-Edge Table**
  - Face | Edge
  - A    | d
  - B    | e
  - C    | d
  - D    | b
12.1 Winged-Edge Data Structure

- How to find edges adjacent to a given vertex $v$?

- Vertex-Edge Table (to find $c$)

- Edge Table (to find $g$)

- Edge Table (to find $f$)
12.1 Winged-Edge Data Structure

- How to find edges adjacent to a given vertex $v$?

1. **Edge Table** (to find $e$)
2. **Edge Table** (to find $d$)
3. **Edge Table** (to find $c$, stop)
12.1 Winged-Edge Data Structure

- How to find **faces** adjacent to a given vertex $v$?

- **Vertex-Edge Table** (to find $c$)

- **Edge Table** (to find $B$ and $g$)

- **Edge Table** (to find $E$ and $f$)
12.1 Winged-Edge Data Structure

- How to find **faces** adjacent to a given vertex \( v \)?

- **Edge Table** (to find \( D \) and \( e \))
- **Edge Table** (to find \( C \) and \( d \))
- **Edge Table** (to find \( A \) and \( c \), stop)
12.2 Scene Graphs

- Managing objects in desired positions

Transformations + tree
12.2 Scene Graphs

- A tree structure with each node representing an arm and each edge representing a joint

- Matrix to apply to an object is the product of the matrices in the chain from the object to the root of the tree

- Use matrix stack to achieve efficient implementation
12.2 Scene Graphs

\[ M_0 = T_0 \]
\[ M_1 = R(\theta)T_1 \]
\[ M_2 = \cdots \]
Function $\text{traverse}(\text{node})$

- push ($M_{\text{local}}$)
- draw object using composite matrix from stack
- $\text{traverse}$(left child)
- $\text{traverse}$(right child)
- pop()
End of Data Structures