

## Retained Mode Graphics

---



## Two approaches to graphics

---



- Say "how"
  - Immediate mode: explicit commands
  
- Say "what"
  - Retained mode: define and change model

## What's in a Geometric Model

---



- Geometry
  - Spatial layout
  - Attributes
- Topology
  - Connectivity
  - Structure
- Application-specific data

## Model part of app data structure

---

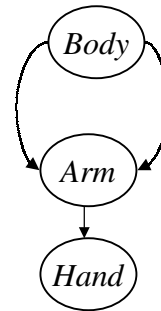


- App + graphical data structures
  - App ds changes => graphical ds changes  
=> update screen
- Modern libs let you integrate them
  - Inventor, Java-3D

## Hierarchy in Geometric Models

---

- Build model from pieces
  - Define each in "natural" coordinates
  - Relate them to logical parents
  - Modularity
- Define a DAG
  - Nodes: part of model
  - Edges: relationships



## Hierarchy Inheritance

---

- Transforms
  - Nested Coordinate systems
- Attributes
  - "red car"

## Recall Assignments 2 and 3

---



- You have a (simple) RM data structure!
  
- Multiple uses
  - Rendering
  - Picking

## Retained Mode Packages

---



- Maintain graphical data structure
- Traverse graphical data structure
  - Auto redisplay, picking, intelligent handling of special nodes, optimizations
  
- Goal: Make your life easier!
  - You should be thinking: this is what I would just be doing anyway

## Model Traversal: Special Nodes

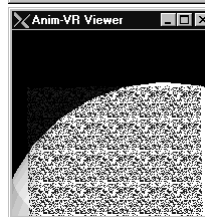
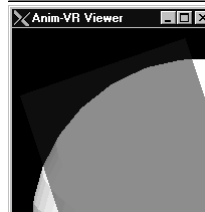
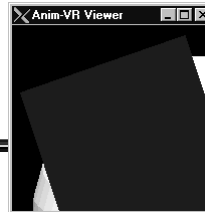
---

- Lights
- Transparency
- Cameras

## Recall: Simple Transparency

---

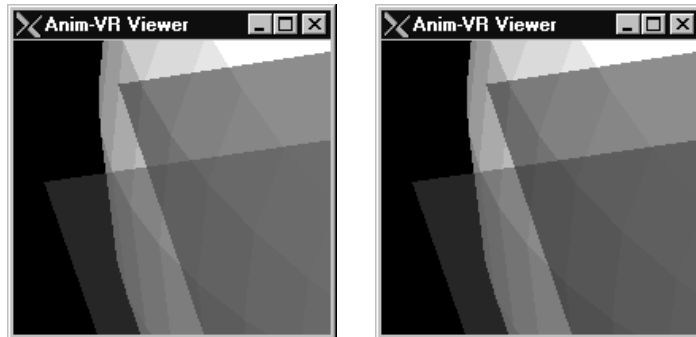
- Interpolated transparency
  - ie. OpenGL alpha values
  - $I = (1 - k_t) I_A + k_t I_B$
- Screen door transparency



## Naive Rendering Fails: RM can possibly handle this

---

- Even if rendered after opaque
- If more than one, order matters!



## Consider OpenGL Display Lists

---

- `glNewList(int, mode)`
- `glEndList()`
- `glCallList(int)`
  
- `int glGenLists(num)`

## Consider OpenGL Display Lists

---



- Provide some of this
  - Contain geometry, attributes
  - Hierarchy (DAG)
  - Embed name stack commands for picking

## Why aren't OpenGL DLs enough?

---



- "Simple replay, output only"
  - No intelligent handling, no optimizations
    - Lights? Cameras? Action?
  - No editing or automatic repair
  - Picking?
- Expensive to create/change
  - ⇒ Not real hierarchy
    - C macros vs. procedures

## Problems with DL model

---



- Attributes, geometry mixed together
  - Programmer must know where everything is
  - Less opportunity for optimization
  - Cannot integrate application data

## Object oriented model

---



- Inventor, Java-3D, Repo-3D
- Objects for each "object"
  - Internal DAG nodes for structure
  - Leaf nodes are "things"
  - Properties attached to node

## Major Objects

---



- Geometry
  - Spheres, boxes, polygons, cameras, lights, ...
- Properties
  - Transformations
  - Attributes (rendering style, color, etc.)
  - Geometric Properties
- Groups
  - Structure

## Special Group: "Root"

---



- Attach a hierarchy to a window
- Need
  - Camera
  - Window
  - Viewport

## Properties

---

- Every property has a default value
- Attached to nodes
  - Define property value for graph rooted at current node

## Advantage of Objects: Indirection of method calls

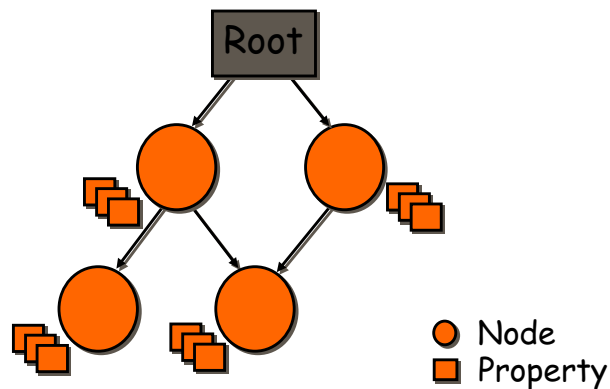
---

- Property value?
  - *type* get()
- Incorporate time!
  - *type* get(int time)

## Variations of properties (Obliq-3D)

- For *every* type of property (color, transformation, float, int, ...)
  - Constant
  - Synchronous time-based animation
  - Arbitrary function

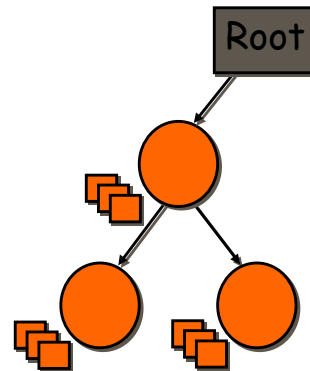
## Example Graph



## Graph Traversal: Rendering

---

- Save/restore state
- Traverse multiple times
- Decide when to render



## Graph Traversal: Picking

---

- How to incorporate OpenGL Picking?

## Optimizations: Elision

---



- Pruning
- Culling
- LOD

## Optimization: Rendering

---



- State changes are expensive
- Display lists are expensive to create by faster to draw