FPV: fast protein visualization using Java 3D™

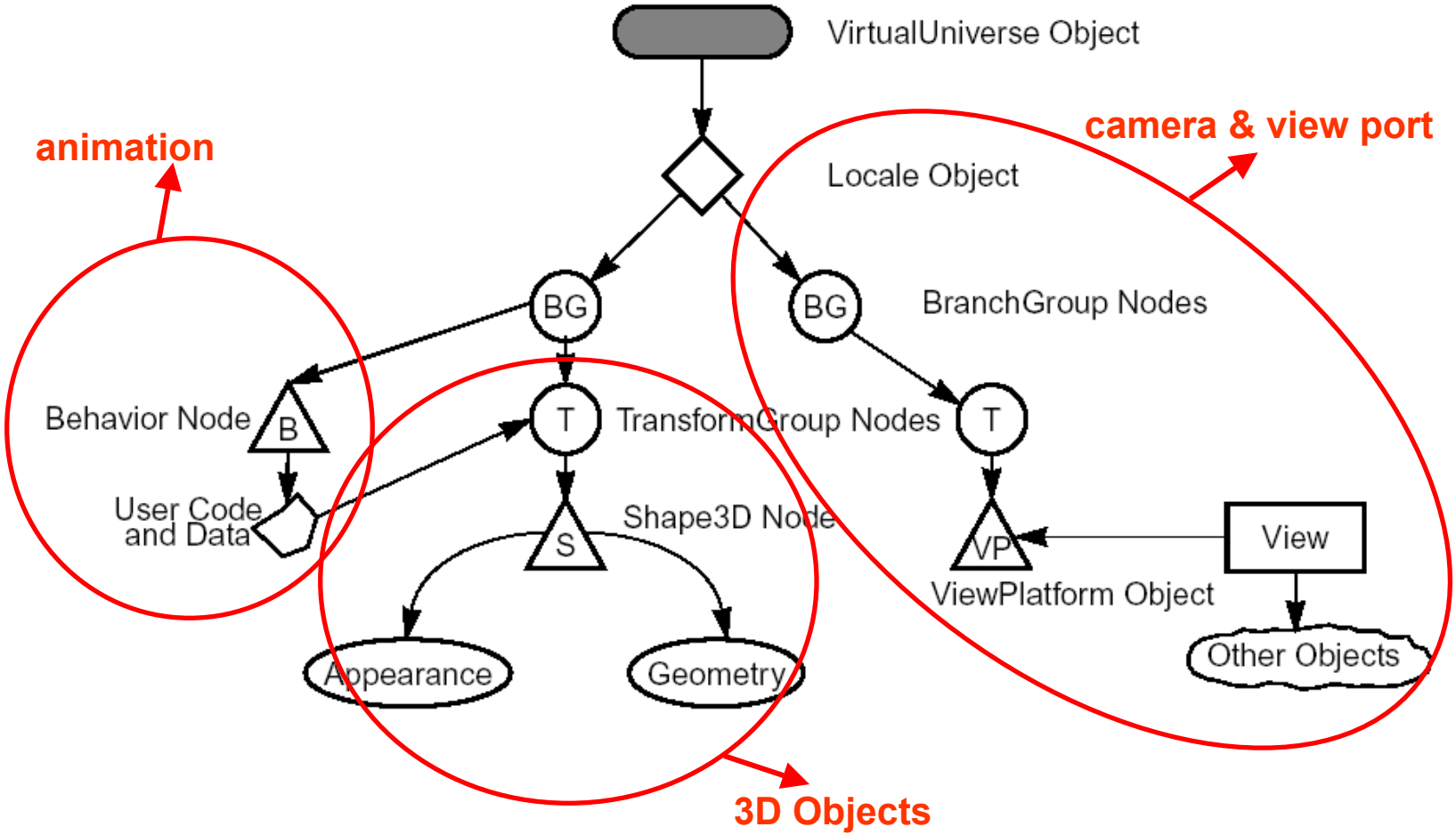
Tolga Can, Yujun Wang, Yuan-Fang Wang and Jianwen Su

Introduction

- Visualization of proteins is crucial in understanding their functions and structural relationships to other proteins
- Our goal is to build a user community without boundaries → Java provides portability
- Java 3D is a scene-graph based graphics library built upon OpenGL/Direct3D
- We tackle some problems related to visualizing complex molecular scenes using Java 3D

Java3D API

- **Scene-Graph based graphics rendering**



Problems with Java 3D

- **Interaction is slower compared to other systems built using C/C++ and OpenGL**
  - ⇨ **Lower frame rates**
- **Out of memory errors for complex scenes**
  - ⇨ **Possible reason: scene-graph based graphics API → Complex scenes may need more memory for the additional scene-graph constructs, e.g.**
    - ❑ Shape3D objects
      - ◆ Container for geometry and appearance
    - ❑ TransformGroup objects
      - ◆ For specifying location, orientation in the virtual world

An Example: memory problem

- **From java3d-interest mailing list (Feb 26 2003):**
  - **Trying to create *n* Shape3Ds with the individual TGs. This is for a molecular viewer where the geometry(would be shared) will be spheres or cylinders. The scene graph looks like this:**
    - BG->TG(*n* of them)->Shape3D(*n* of them)(No Geometry or Appearance added)
  - **Here is the memory needed for such a scene:**

n	Memory Used (MB)
1000	3.7
2000	8.1
3000	12.5
28000	115

  - **The above memory requirement is only for the empty shape3D objects and the geometry hasn't been attached yet. A TG, Shape3D tuple requires ~4K memory**

Does Java3D provide a solution?

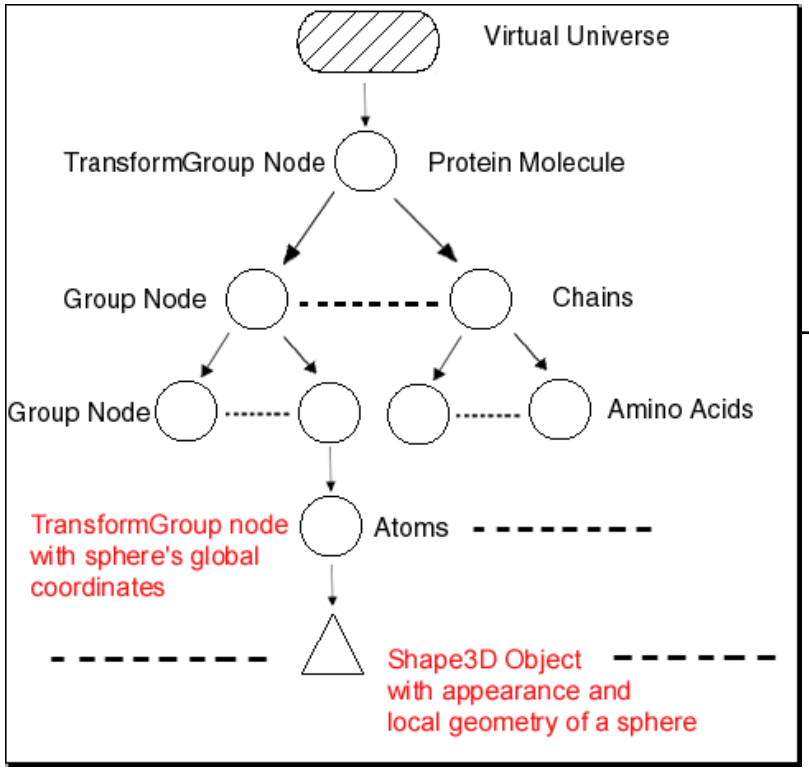
- Java 3D provides BranchGroup.compile() method to make optimizations on the scene graph:
  - ⇨ Scene-graph flattening (combining TransformGroup nodes)
  - ⇨ Combining Shape3D nodes
- But Java3D's *compile* method does not solve memory overflow problem because they are called "after" creating the scene graph (i.e. root branch group)
- We need a solution that works on the fly during the creation of the scene!!

Observations

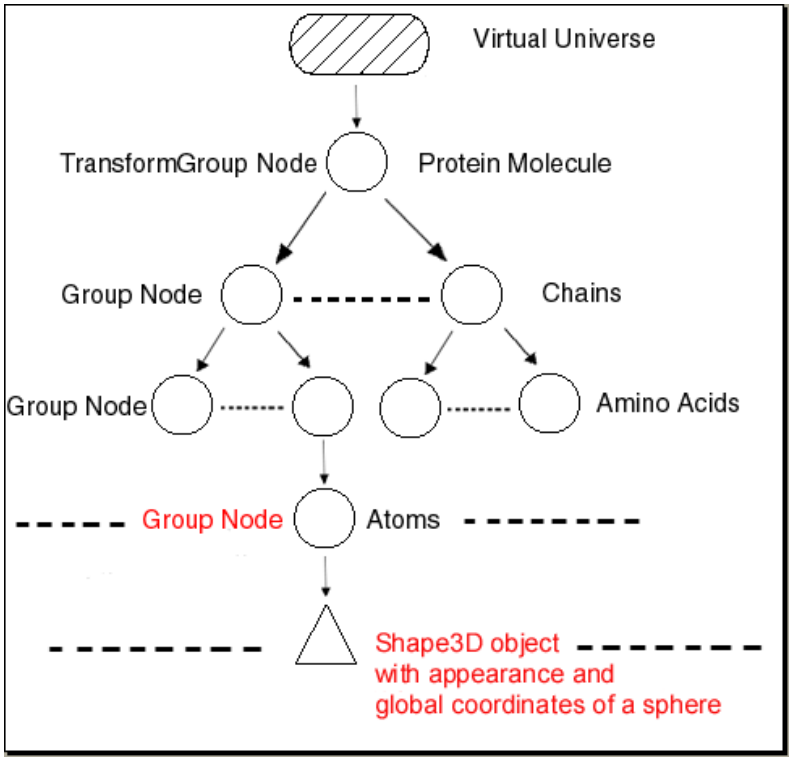
- Molecular scenes are usually static if you are not doing molecular dynamics → we can reduce TransformGroup objects
- Organic molecules contain limited number of different atoms → we can reduce Shape3D objects
- Different models types have different memory consumption
  - ⇨ Space-fill model may have tens of thousands atoms, hence we need to be careful in constructing the scene graph
  - ⇨ Ribbon model is already less memory consuming because the number of SSEs (secondary structure elements) is far less than the number of atoms in a protein molecule
  - ⇨ Bonds model may contain lines (i.e. bonds) in the order of atoms in the molecule so we need to be careful here, too.

Proposed Solution: first step

- **Converting TransformGroup nodes to Group Nodes**



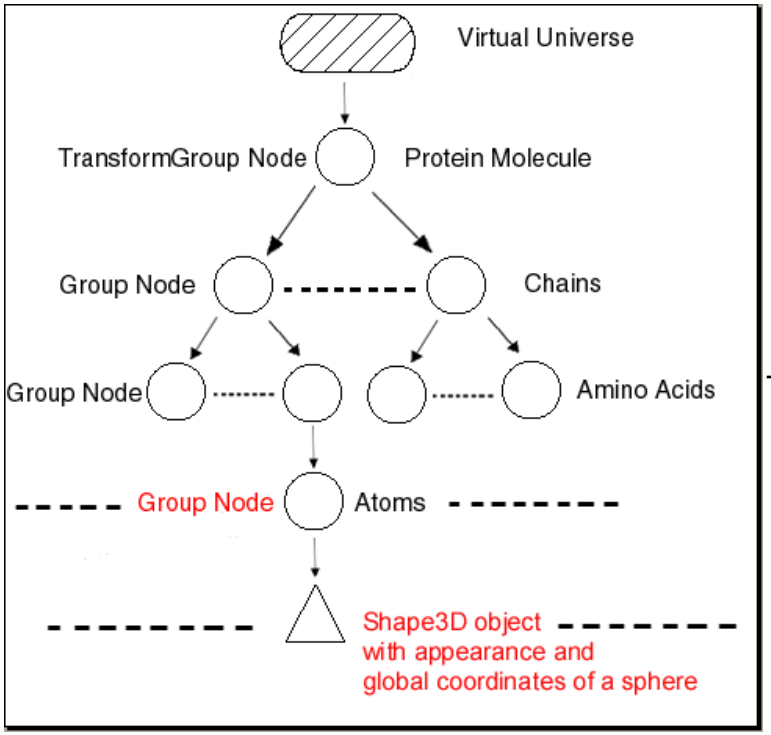
Intuitive Way of creating a molecular scene



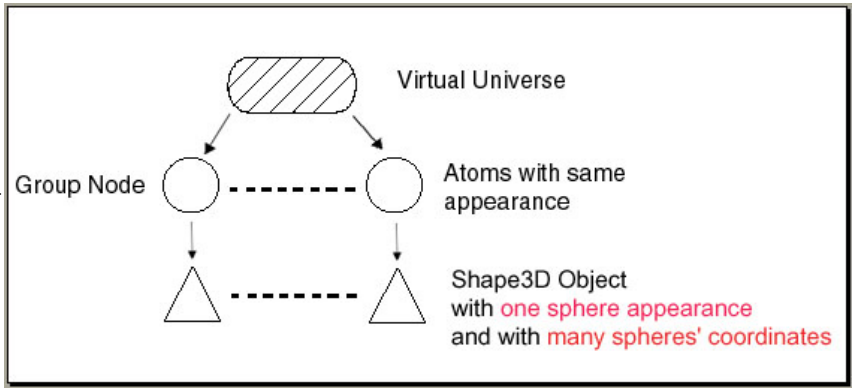
Scene-graph after applying first step

Proposed Solution: second step

- Getting rid of Group & Shape3D objects

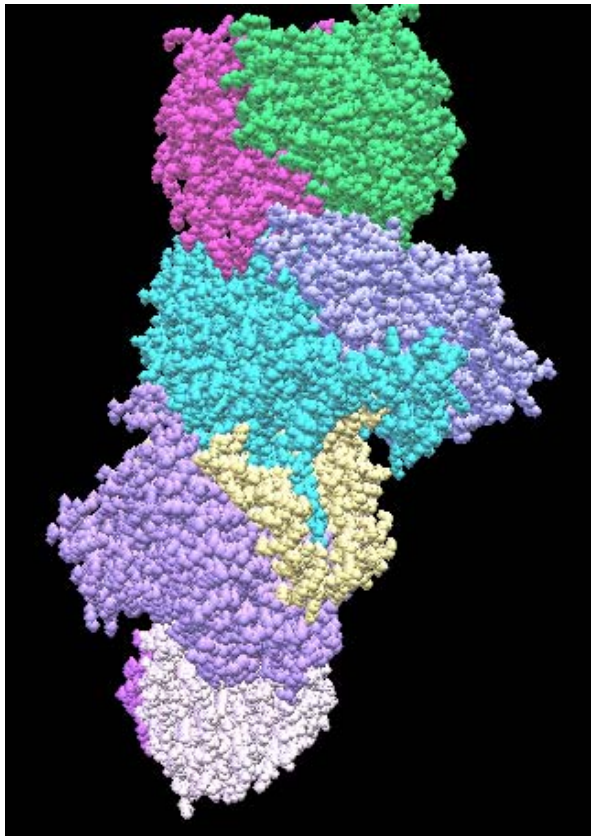


Scene-graph after applying first step

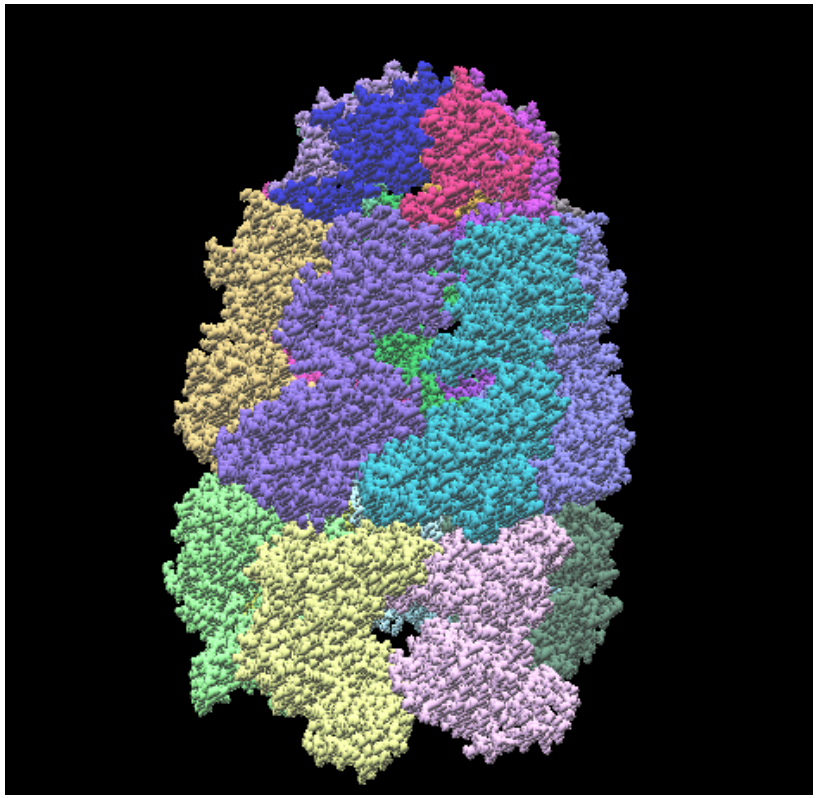


Flattened scene-graph

Viewing Large Molecules



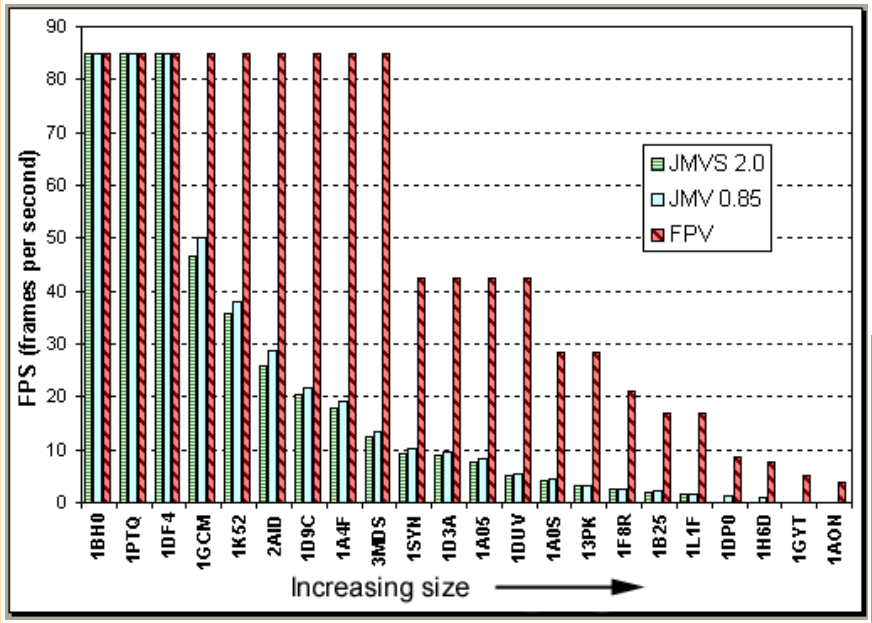
Nitrogenase Molybdenum-Iron Protein
PDB ID: 1n2c
24190 atoms (3182 residues)
@14 fps



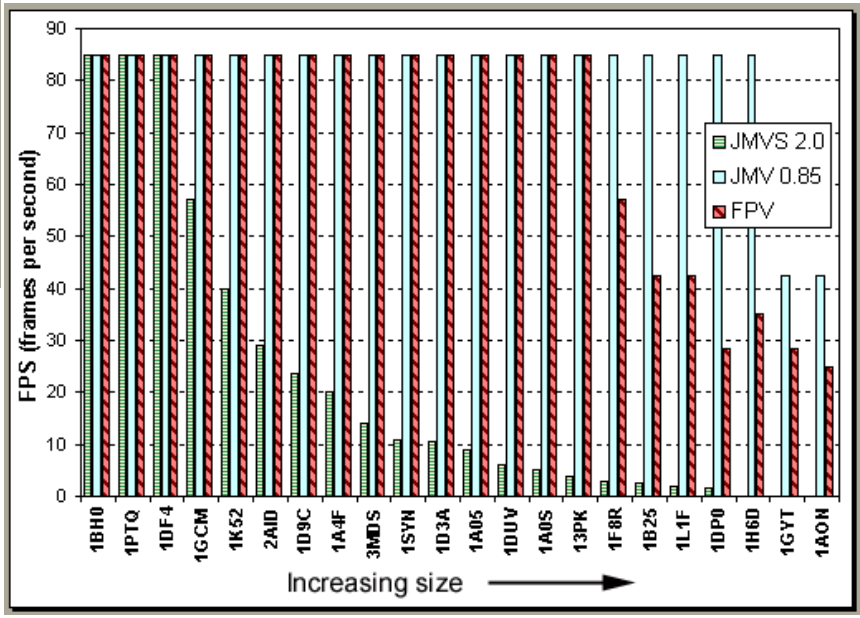
GroEL-GroES complex of the bacterium E. coli
PDB ID: 1aon
58688 atoms (8337 residues)
@3.7 fps

Performance Evaluation: RT Interaction

- Comparison to 2 existing tools based on Java 3D



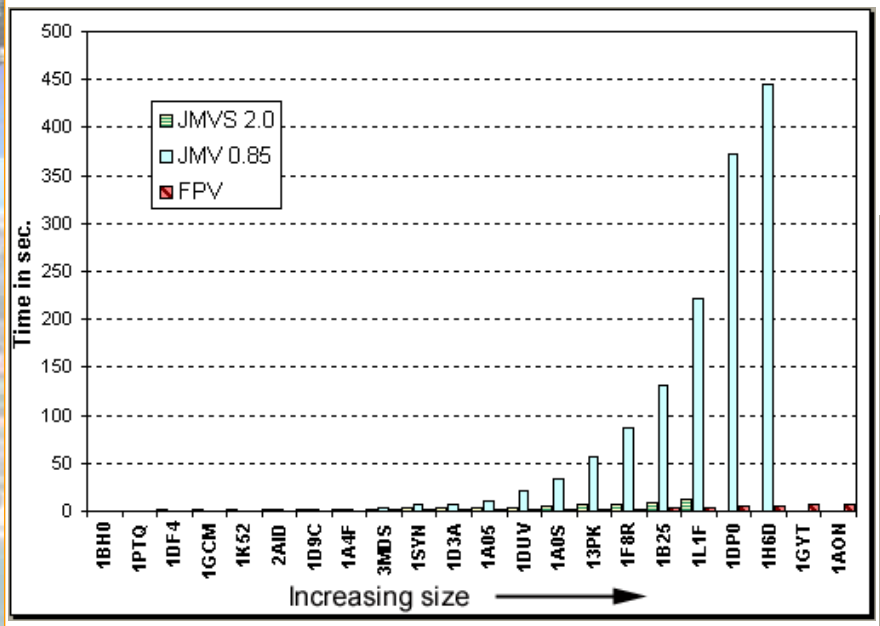
Space-fill model Rendering Performance



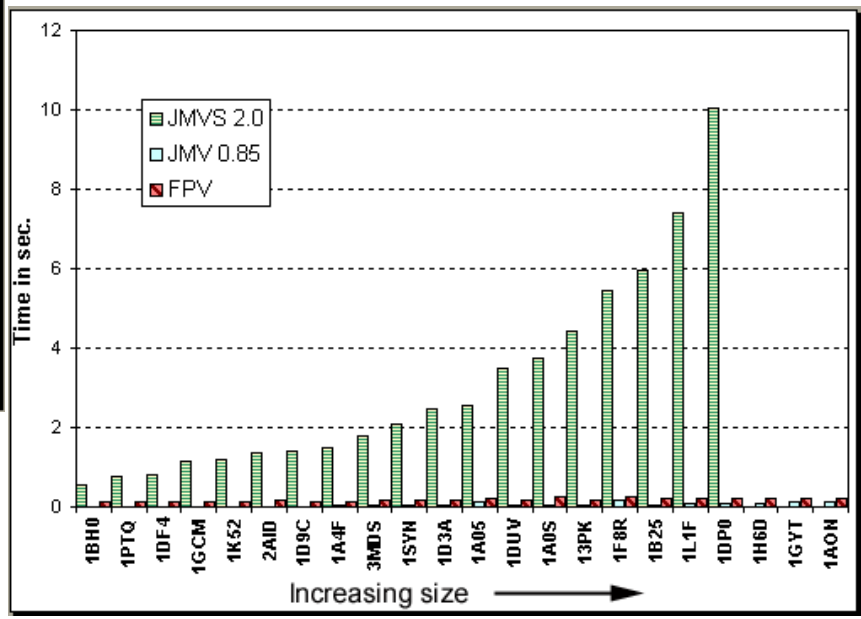
Bonds Model Rendering Performance

Performance Evaluation: Scene Building

- **Applying the techniques does not introduce an overhead.**



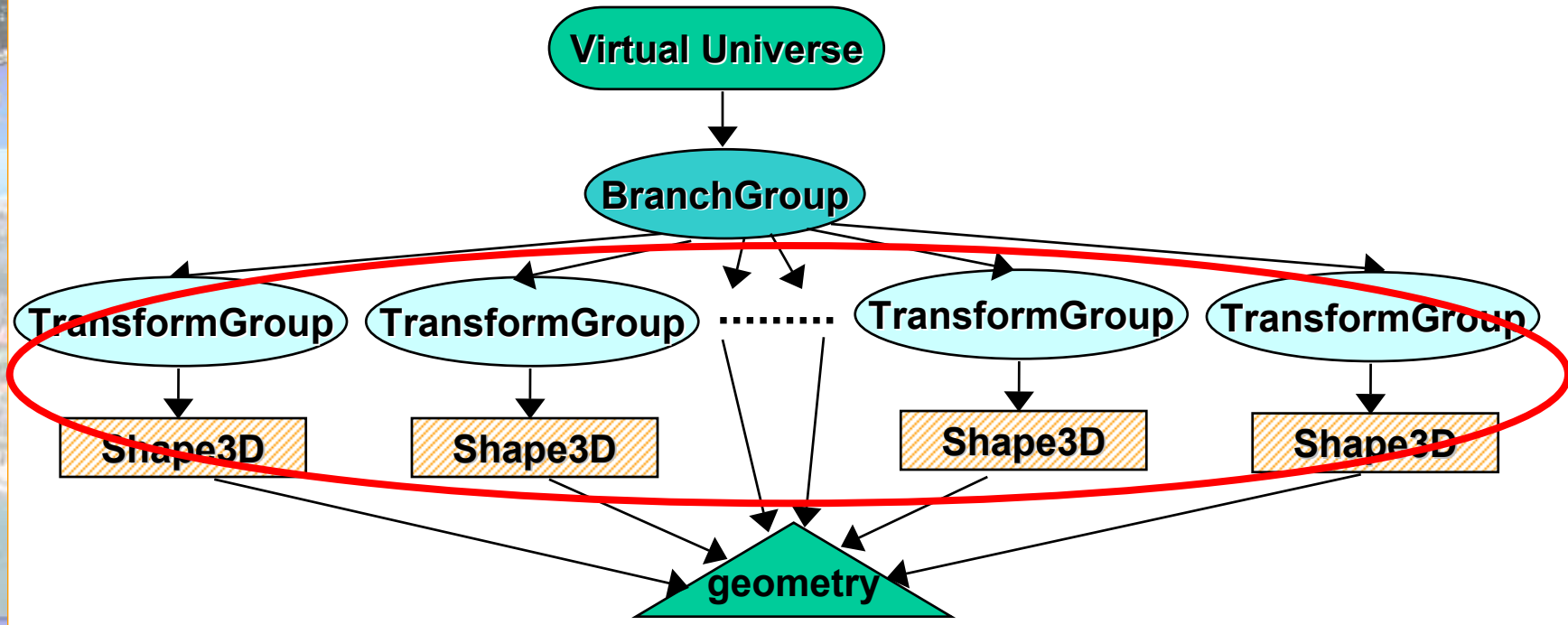
Space-fill model scene-building times



Bonds Model scene-building times

Another Solution?

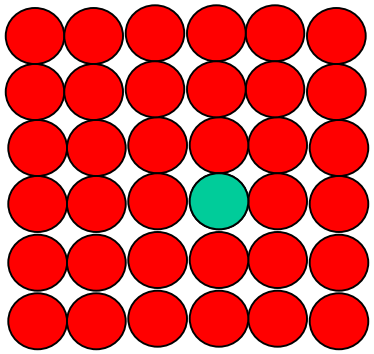
- What about sharing geometry?

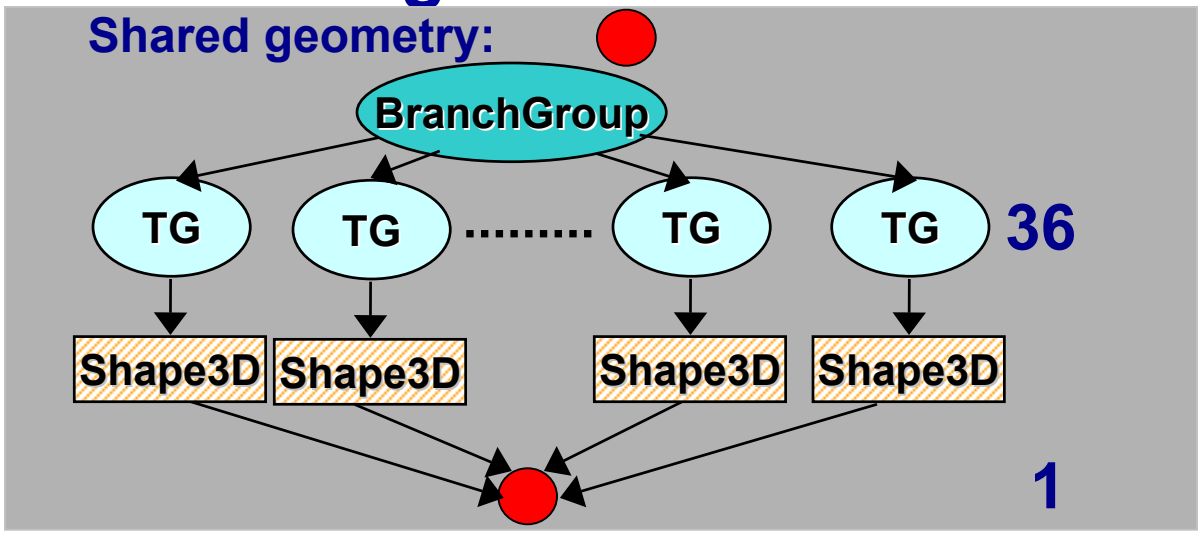


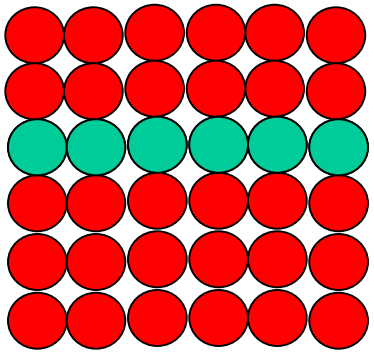
- But even if the geometry is shared, just Shape3Ds and TransformGroups may require huge amount of memory.

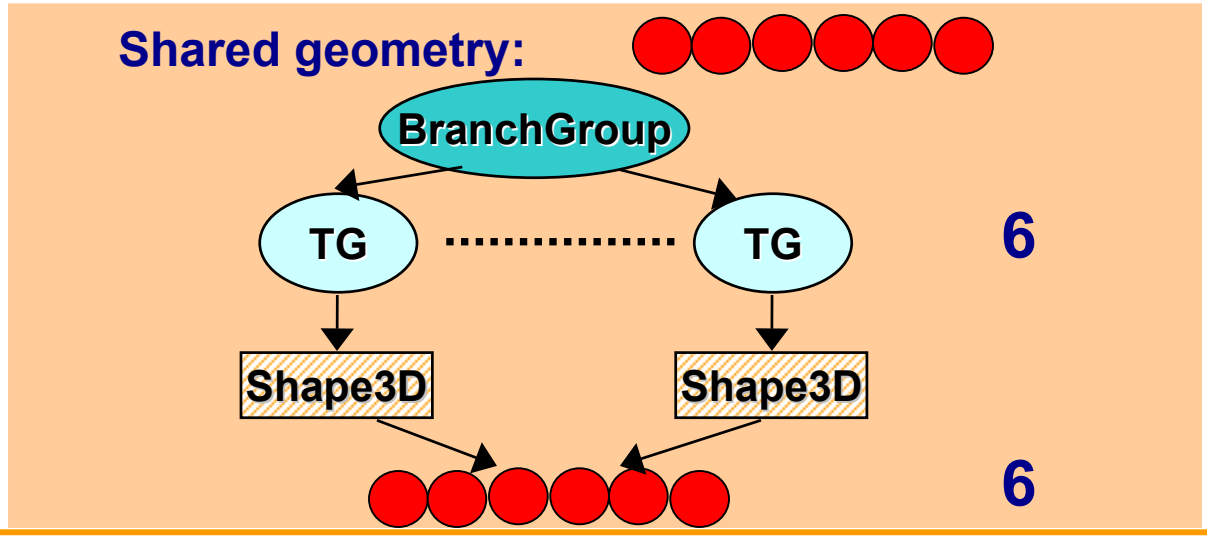
Combining Geometries vs. Sharing Geometries

- Consider the following scene:



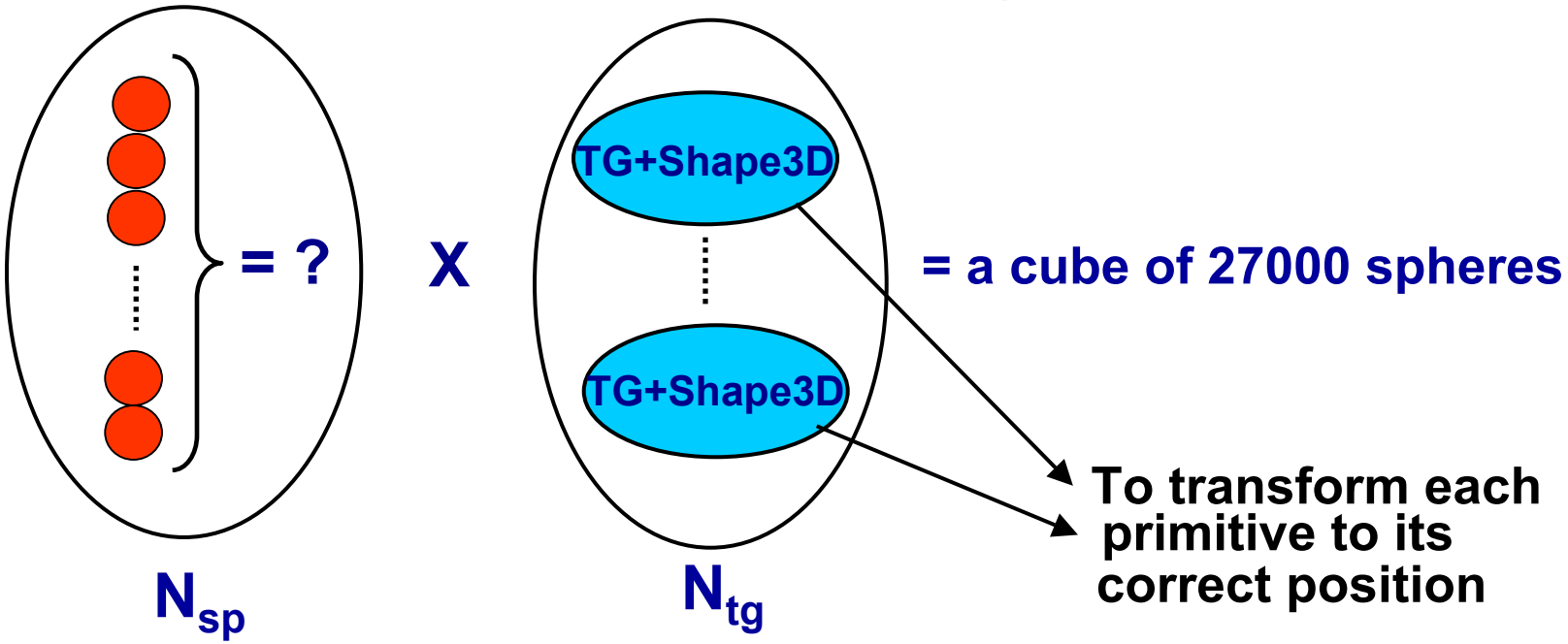






Optimal Solution (I)

- **Example problem:**
  - ⇨ Rendering a cube consisting of 27000 spheres
  - ⇨ Size of a single sphere geometry = ~8K
  - ⇨ Size of TG & Shape3D tuple = ~4K
  - ⇨ What is the optimum number of spheres to combine as a geometry primitive for sharing? (N_{sp} = ?)

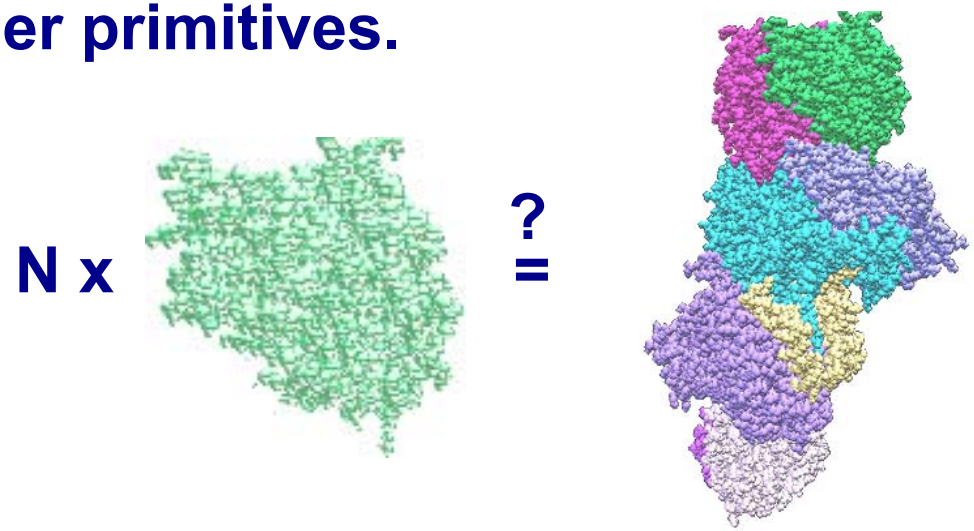


Optimal Solution (II)

- N_{sp} x N_{tg} = 27000
- Minimize total memory: 8K•N_{sp} + 4K•N_{tg}
  - ⇨ $N_{tg} = \sqrt{(8/4)*27000} \approx 232$
  - ⇨ Memory used ≈ 2MB
  - ⇨ In general: $N_{tg} = \sqrt{(sizeGeom/sizeTGS3D)*sizeofScene}$
- ASSUMPTION: you have to be able to share the combined geometries to construct the whole scene, i.e. construct the cube by using 232 sphere pieces

Applicable to Molecular Scenes?

- **We cannot apply that solution to a molecular scene.**
  - ⇨ **We need to be able to represent the whole molecule by large combined geometric primitives of spheres.** **Not Quite Possible**
  - ⇨ **Only reusable primitives we can have are amino acids. Beyond that it's not easy to find repeating bigger primitives.**



Conclusions

- **Proposed solution for molecular scenes:**
  - ⇨ **Reduce number of TG & Shape3D objects**
  - ⇨ **Since geometry cannot be shared, reduce resolution of geometry instead**
- **The proposed technique helps load and view complex molecular scenes containing thousands of atoms**
- **It doesn't introduce any overhead in scene-graph building**
- **Can be incorporated into existing tools easily**

Current and Future Work

- **Building a collaborative environment for users (potentially on different platforms)**
  - ⇨ **Distributed Visualization**
  - ⇨ **Distant Learning**
  - ⇨ **Annotation of protein structures to facilitate collaboration**
    - ❑ Attaching textual, multimedia, hyperlinks to structures

Thank you for your attention!

For More Information:

Tolga Can
Department of Computer Science
University of California at Santa Barbara
Santa Barbara, CA 93106, U.S.

Email: tcan@cs.ucsb.edu
URL: http://www.cs.ucsb.edu/~tcan/fpv/