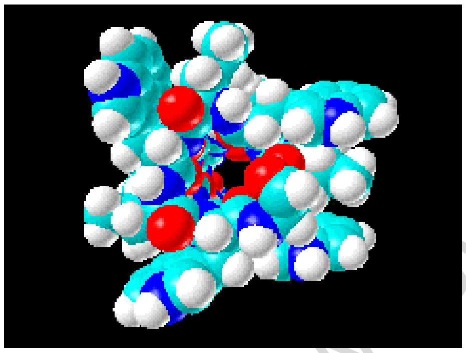
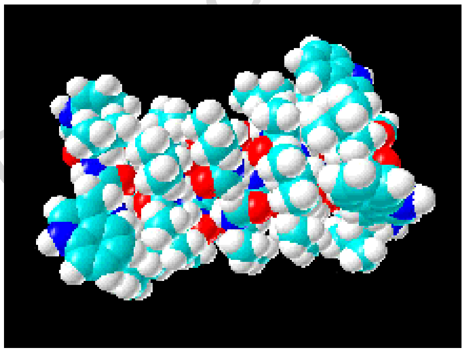
JAVA 3D APPLICATIONS

1. The application areas of Java 3D

- Scientific visualization
- Information visualization
- Medical visualization
- Geographical information systems (GIS)
- Computer-aided design (CAD)
- Animation
- Education

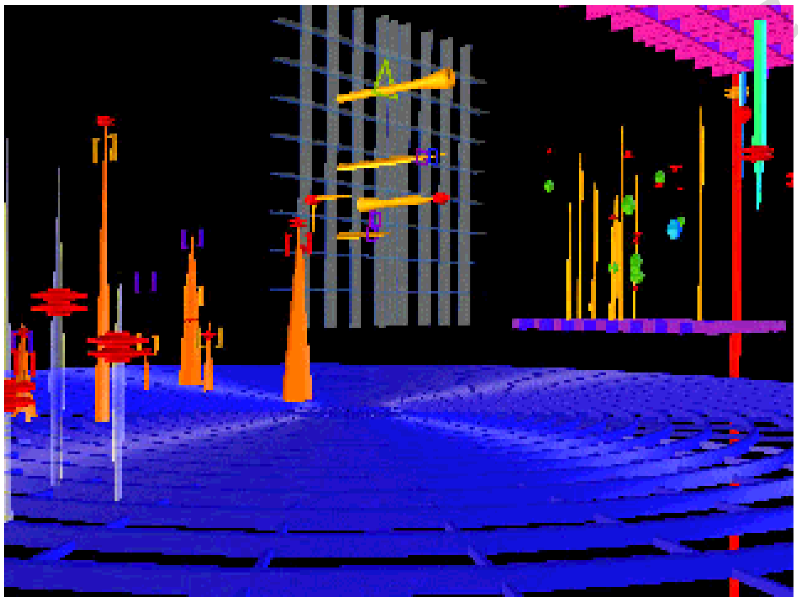
XYUAN@UWINDSOR.CA

2. Examples: Scientific Visualization

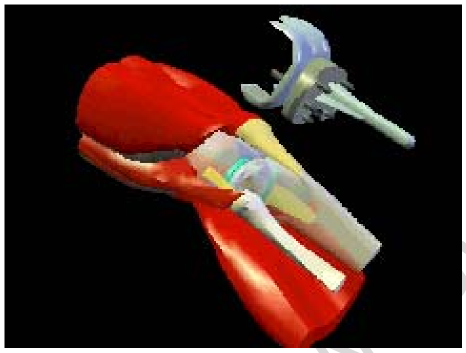



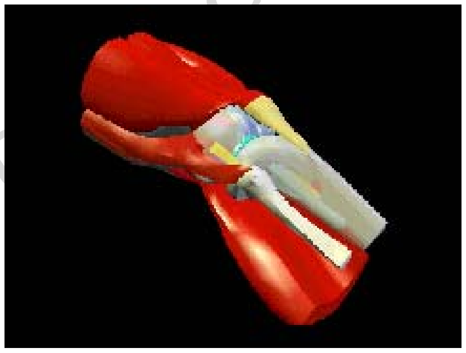
XYUAN@UWINDSOR.CA

3. Examples: Abstract Data (Financial)



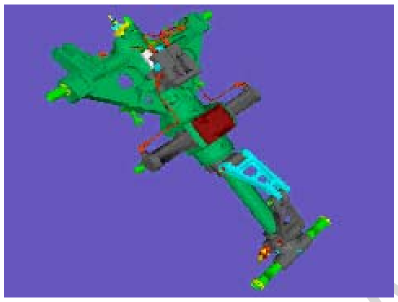
4. Examples: Medical Education





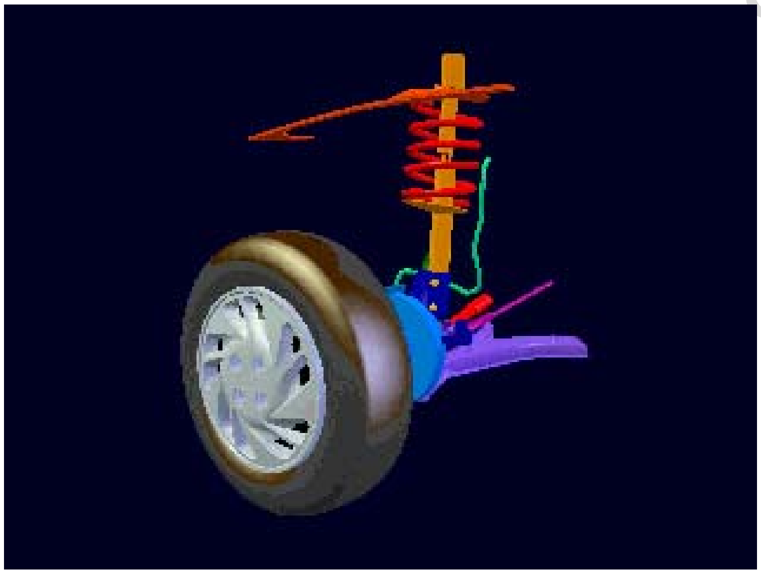
XYUANSWIND.CA

5. Examples: CAD



XYUANDUWINDSOR.CA

6. Examples: Mechanical Analysis



7. Examples: Computer Animations

XYUAN@UWINDSOR.CA

8. Performance Considerations:

- Use low polygon geometry with high quality textures.
- Use high quality textures with transparency and 2D geometry.
- Use tiling and small texture sizes when possible.
- When creating multiple instances, use `SharedGroup` and `Link` nodes.
- Use `Behavior` and `Alpha` nodes only where necessary.
- Turn `Timer` nodes on and off where appropriate.

9. Object Creations:

- Examine every "`new`" statement in your program.
- Ask if it is really necessary?

XYUAN@UWINDSOR.CA

- If a "`new`" statement is executed at every timer or render time then replace it with a static node.
- Try to create all nodes before a user starts interacting with your simulation.
- Avoid using the "`new`" statement after a user begins interacting with your simulation.
- For garbage collection to take place use class `WeakHashMap`
  - The use of a `WeakHashMap` object associates pairs of objects as keys and values. The value is made eligible for garbage collection when the key becomes weakly reachable.
- For garbage collection to take place, unregister event listener classes when they are no longer needed.

XYUAN@UWINDSOR.CA

10. There are a number of things in the Java 3D API that were included specifically to increase performance.
    - *Capability bits* are the applications way of describing its intentions to the Java 3D implementation. The implementation examines the capability bits to determine which objects may change at run time. Many optimizations are possible with this feature.
    - The default `isFrequent` bit indicates that the application may frequently access or modify those attributes permitted by the associated capability bit. This can be used by Java 3D as a hint to avoid certain optimizations that could cause those accesses or modifications to be expensive.

XYUAN@UWINDSOR.CA

- The are two compile methods in Java 3D. They are in the `BranchGroup` and `SharedGroup` classes. Once an application calls `compile()`, only those attributes of objects that have their capability bits set may be modified. The implementation may then use this information to "compile" the data into a more efficient rendering format.
- Many Java 3D object require a *bounds* associated with them. These objects include `Lights`, `Behaviors`, `Fogs`, `Clips`, `Backgrounds`, `BoundingLeafs`, `Sounds`, `Soundscapes`, `ModelClips`, and `AlternateAppearance`. These bounds limit the spatial scope of the specific object, allowing the implementation to quickly disregard the processing of any objects that are out of the spatial scope.

XYUAN@UWINDSOR.CA

- All state required to render a specific object in Java 3D is completely defined by the direct path from the root node to the given leaf. That means that leaf nodes have no effect on other leaf nodes, and therefore may be rendered in any order. There are a few ordering requirements for direct descendents of `OrderedGroup` nodes or `Transparent` objects. But, most leaf nodes may be reordered to facilitate more efficient rendering.
- `OrderedGroup` supports an indirection table to allow the user to specify the order that the children should be rendered. This will speed up order update processing, eliminating the expensive attach and detach cycle.

XYUAN@UWINDSOR.CA

- A `Shape3D` node has a reference to a `Geometry` and an `Appearance`. An `Appearance` `NodeComponent` is simply a collection of other `NodeComponent` references that describe the rendering characteristics of the geometry. Because the `Appearance` is nothing but a collection of references, it is much simpler and more efficient for the implementation to check for rendering characteristic changes when rendering. This allows the implementation to minimize state changes in the low level rendering API.

XYUAN@UWINDSOR.CA

11. Tips and Tricks

- Move Object vs. Move `ViewPlatform`
  - If the application simply needs to transform the entire scene, transform the `ViewPlatform` instead. This changes the problem from transforming every object in the scene into only transforming the `ViewPlatform`.
- Capability bits
  - Only set them when needed. Many optimizations can be done when they are not set. So, plan out application requirements and only set the capability bits that are needed.

XYUAN@UWINDSOR.CA

- Bounds and Activation Radius
  - Consider the spatial extent of various leaf nodes in the scene and assign bounds accordingly. This allows the implementation to prune processing on objects that are not in close proximity.
  - Automatic bounds calculations for geometric objects is fine.
  - Bounds computation does consume CPU cycles. If an application does a lot of geometry coordinate updates, to improve performance, it is better to turn off auto bounds compute. The application will have to do the bounds update itself.

XYUAN@UWINDSOR.CA

- Change Number of Shape3D Nodes
  - In the current implementation there is a certain amount of fixed overhead associated with the use of the `Shape3D` node. In general, the fewer `Shape3D` nodes that an application uses, the better. However, combining `Shape3D` nodes without factoring in the spatial locality of the nodes to be combined can adversely effect performance by effectively disabling view frustum culling. An application programmer will need to experiment to find the right balance of combining `Shape3D` nodes while leveraging view frustum culling. The `.compile` optimization that combines shape node will do this automatically, when possible.

XYUAN@UWINDSOR.CA

- Geometry Type and Format
  - Most rendering hardware reaches peak performance when rendering long triangle strips. Unfortunately, most geometry data stored in files is organized as independent triangles or small triangle fans (polygons). The Java 3D utility package includes a stripifier utility that will try to convert a given geometry type into long triangle strips. Application programmers should experiment with the stripifier to see if it helps with their specific data. If not, any stripification that the application can do will help.
  - Another option is that most rendering hardware can process a long list of independent triangles faster than

XYUAN@UWINDSOR.CA

a long list of single triangle triangle fans. The stripifier in the Java 3D utility package will be continually updated to provided better stripification.

- Sharing `Appearance/Texture/Material NodeComponents`
  - To assist the implementation in efficient state sorting, and allow more shape nodes to be combined during compilation, applications can help by sharing `Appearance/Texture/Material NodeComponent` objects when possible.
- Geometry by reference
  - Using geometry by reference reduces the memory needed to store a scene graph, since Java 3D avoids creating a copy in some cases.

XYUAN@UWINDSOR.CA

- However, using this features prevents Java 3D from creating display lists (unless the scene graph is compiled), so rendering performance can suffer in some cases. It is appropriate if memory is a concern or if the geometry is writable and may change frequently. The interleaved format will perform better than the non-interleaved formats, and should be used where possible. In by-reference mode, an application should use arrays of native data types; referring to *TupleXX[]* arrays should be avoided.

X.YUAN@UWINDSOR.CA

- Texture by reference and Y-up
  - Using texture by reference and Y-up format may reduce the memory needed to store a texture object,

since Java 3D avoids creating a copy in some cases. When a copy of the by-reference data is made in Java3D, users should be aware that this case will use twice as much memory as the by copy case. This is due to the fact that Java3D internally makes a copy in addition to the user's copy to the reference data.

X.YUAN@UWINDSOR.CA

- Drawing 2D graphics using J3DGraphics2D
  - The `J3DGraphics2D` class allows you to mix 2D and 3D drawing into the same window. However, this can be very slow in many cases because Java 3D needs to buffer up all of the data and then composite it into the back buffer of the Canvas3D.
  - A new method, `drawAndFlushImage`, is provided to ac-

celerate the drawing of 2D images into a Canvas3D. To use this, it is recommended that an application create their own `BufferedImage` of the desired size, use Java2D to render into their BufferedImage, and then use the new `drawAndFlushImage` method to draw the image into the Canvas3D.

- Application Threads
  - The built in threads support in the Java language is very powerful, but can be deadly to performance if it is not controlled. Applications need to be very careful in their threads usage.
    - First, try to use them in a demand driven fashion. Only let the thread run when it has a task to do.

XYUAN@UWINDSOR.CA

Free running threads can take a lot of cpu cycles from the rest of the threads in the system — including Java 3D threads.

- Next, be sure the priority of the threads are appropriate. Most Java Virtual Machines will enforce priorities aggressively. Too low a priority will starve the thread and too high a priority will starve the rest of the system. If in doubt, use the default thread priority.
- Finally, see if the application thread really needs to be a thread. Would the task that the thread performs be all right if it only ran once per frame? If so, consider changing the task to a Behavior that

XYUAN@UWINDSOR.CA

wakes up each frame.

- Java 3D Threads
  - Java 3D uses many threads in its implementation, so it also needs to implement the precautions listed above. In almost all cases, Java 3D manages its threads efficiently. They are demand driven with default priorities. There are a few cases that don't follow these guidelines completely.
- Switch Nodes for Occlusion Culling
  - If the application is a first person point of view application, and the environment is well known, `Switch` nodes may be used to implement simple occlusion culling. The children of the switch node that are not

XYUAN@UWINDSOR.CA

currently visible may be turned off. If the application has this kind of knowledge, this can be a very useful technique.

- Switch Nodes for Animation
  - Most animation is accomplished by changing the transformations that effect an object. If the animation is fairly simple and repeatable, the flip-book trick can be used to display the animation. Simply put all the animation frames under one switch node and use a `SwitchValueInterpolator` on the switch node. This increases memory consumption in favor of smooth animations.
- OrderedGroup Nodes

XYUAN@UWINDSOR.CA

  - `OrderedGroup` and its subclasses are not as high performing as the unordered group nodes. They disable any state sorting optimizations that are possible. If the application can find alternative solutions, performance will improve.

- LOD Behaviors
  - For complex scenes, using LOD Behaviors can improve performance by reducing geometry needed to render objects that don't need high level of detail. This is another option that increases memory consumption for faster render rates.

- Picking
  - If the application doesn't need the accuracy of ge-

XYUAN@UWINDSOR.CA

ometry based picking, use bounds based picking. For more accurate picking and better picking performance, use `PickRay` instead of `PickCone/PickCylnder` unless you need to pick line/point. `PickCanvas` with a tolerance of 0 will use `PickRay` for picking.

- D3D users only
  - Using Quad with Polygon line mode is very slow. This is because DirectX doesn't support Quad. Breaking down the Quad into two triangles causes the diagonal line to be displayed. Instead Java 3D draws the polygon line and does the hidden surface removal manually.

XYUAN@UWINDSOR.CA