Visualizing Biomolecular Complexes with VMD

John Stone

Theoretical and Computational Biophysics Group

University of Illinois at Urbana-Champaign

http://www.ks.uiuc.edu/Research/vmd/

Centre for High Performance Computing,

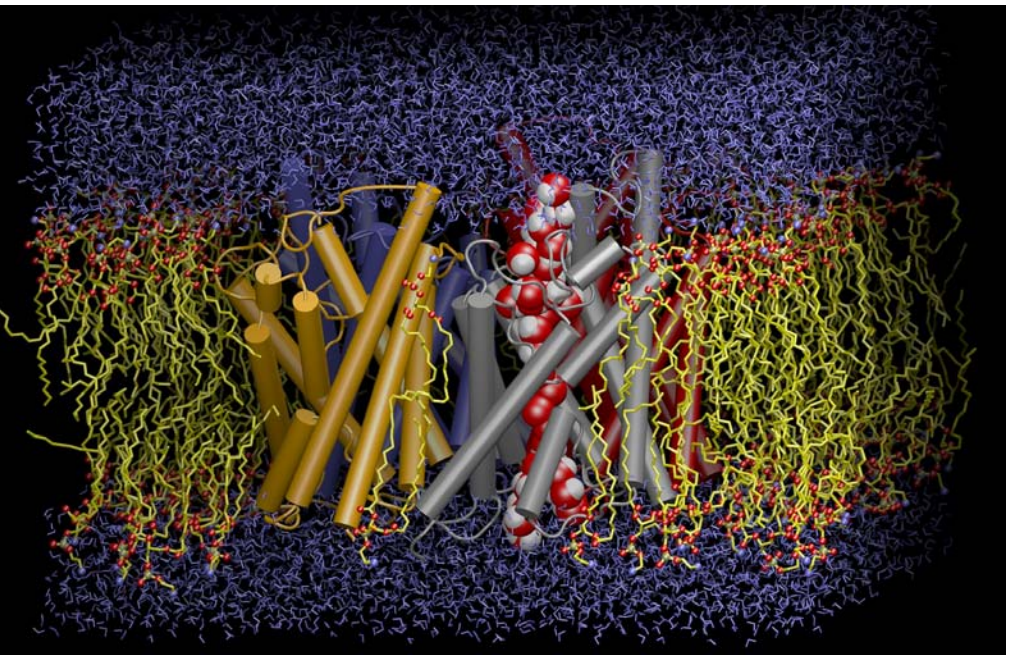
CSIR Rosebank Campus, Cape Town, South Africa, October 27, 2008

Overview

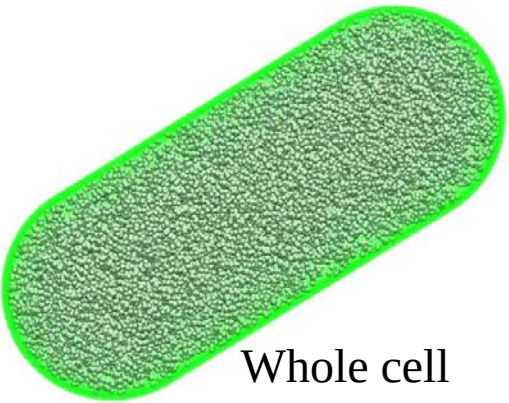
- Brief intro to VMD
- Challenges presented by the driving science projects
- Graphics technology driving molecular visualization capabilities and performance
- Future directions for molecular visualization

VMD

- VMD – “Visual Molecular Dynamics”
- Visualization of molecular dynamics simulations, sequence data, volumetric data, quantum chemistry data, particle systems
- User extensible with scripting and plugins
- http://www.ks.uiuc.edu/Research/vmd/

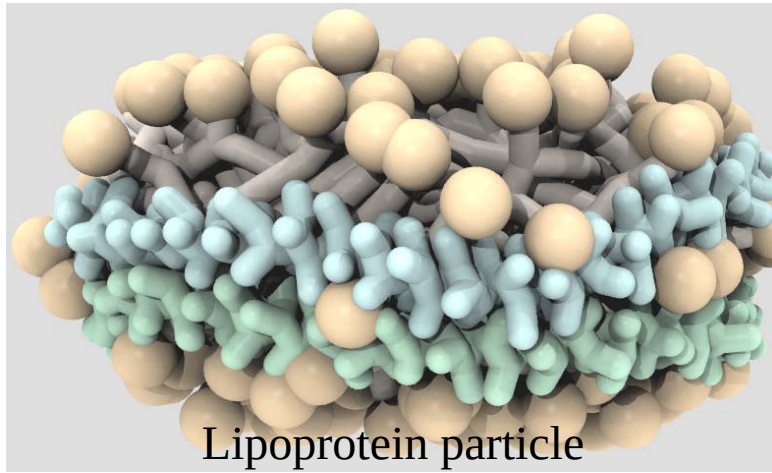


VMD Advanced Data Handling



Whole cell

Atomic, CG, Particle:
Coordinates, Trajectories,
Energies, Forces,
Secondary Structure, …



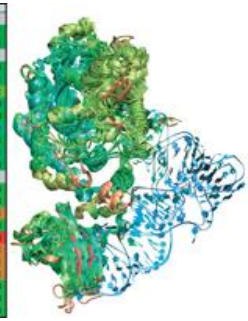
Lipoprotein particle

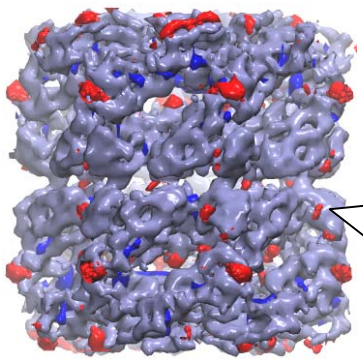
Graphics, Geometry

VMD

Annotations

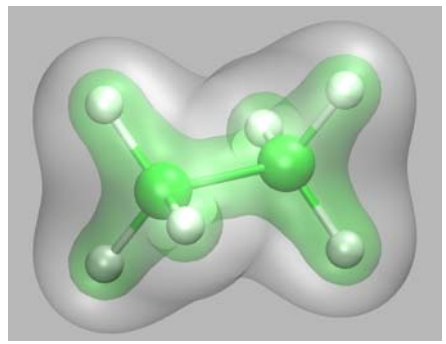
Sequence Data:
Multiple Alignments,
Phylogenetic Trees





GroEL

Volumetric Data:
Density maps,
Electron orbitals,
Electrostatic potential, …



Ethane

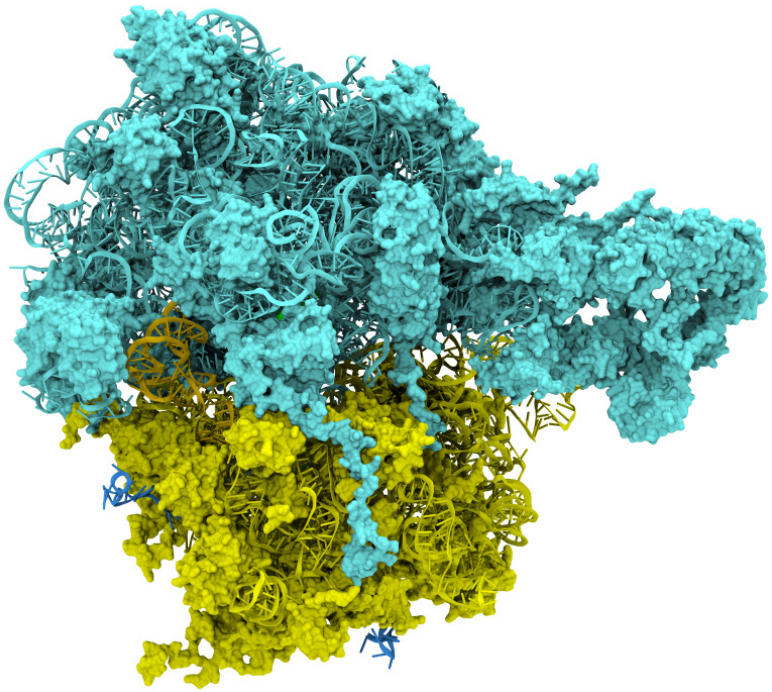
VMD Takes Advantage of Emerging Technological Opportunities

- 8- and 12-core CPUs common by 2010…
- Graphics processors (GPUs) have over 240 processing units, and frequently achieve speedups of 8-30x vs. CPUs
- Parallel processing is now **required** to increase performance
- Several VMD algorithms are now parallelized for multi-core CPUs and GPUs
- Continued developments will more broadly benefit rendering and analysis features of VMD

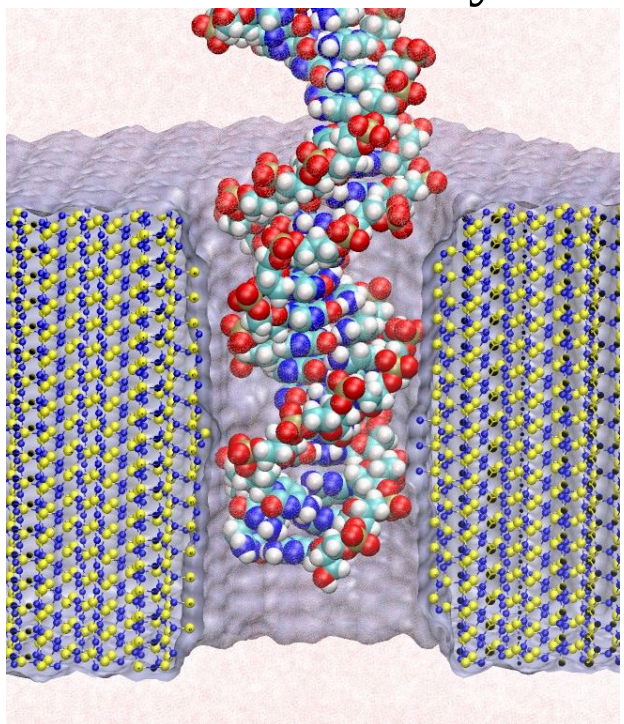
Goal: A Computational Microscope

- Study the molecular machines in living cells

Ribosome: synthesizes proteins from genetic information, target for antibiotics

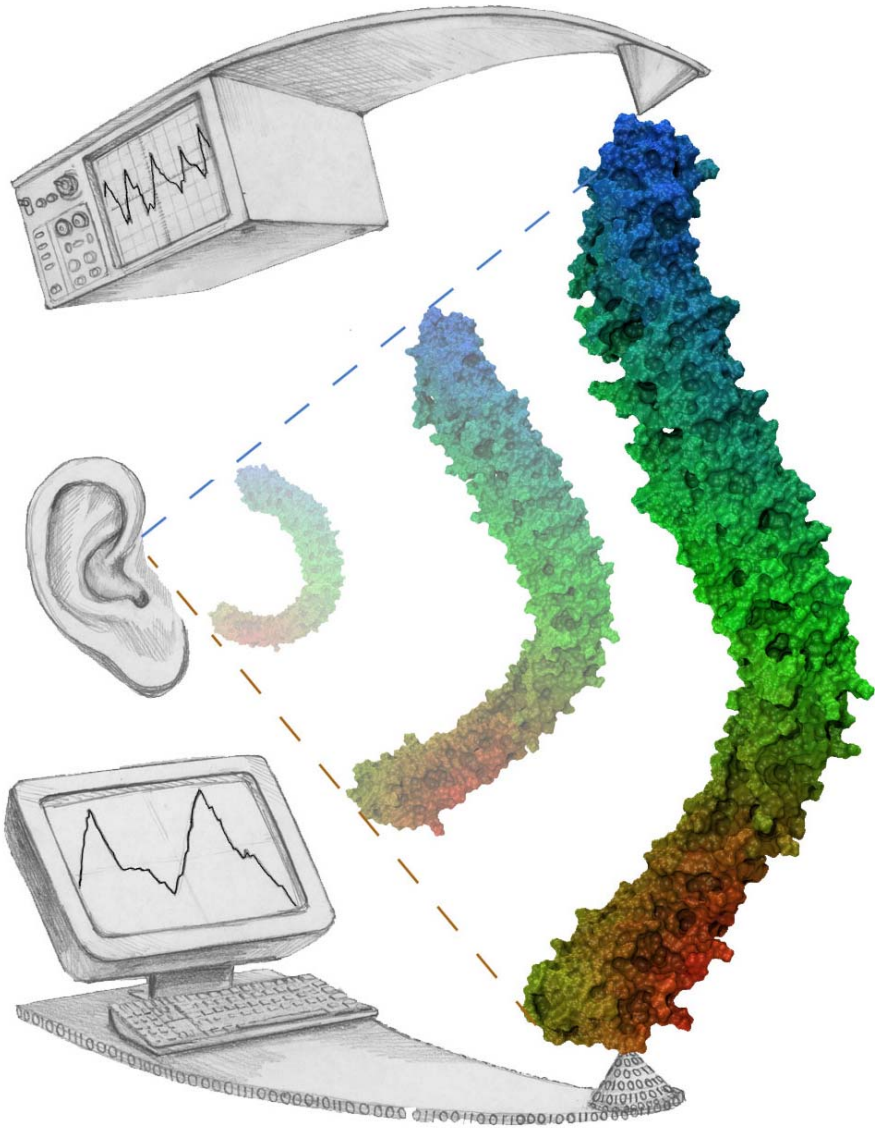


Silicon nanopore: bionanodevice for sequencing DNA efficiently



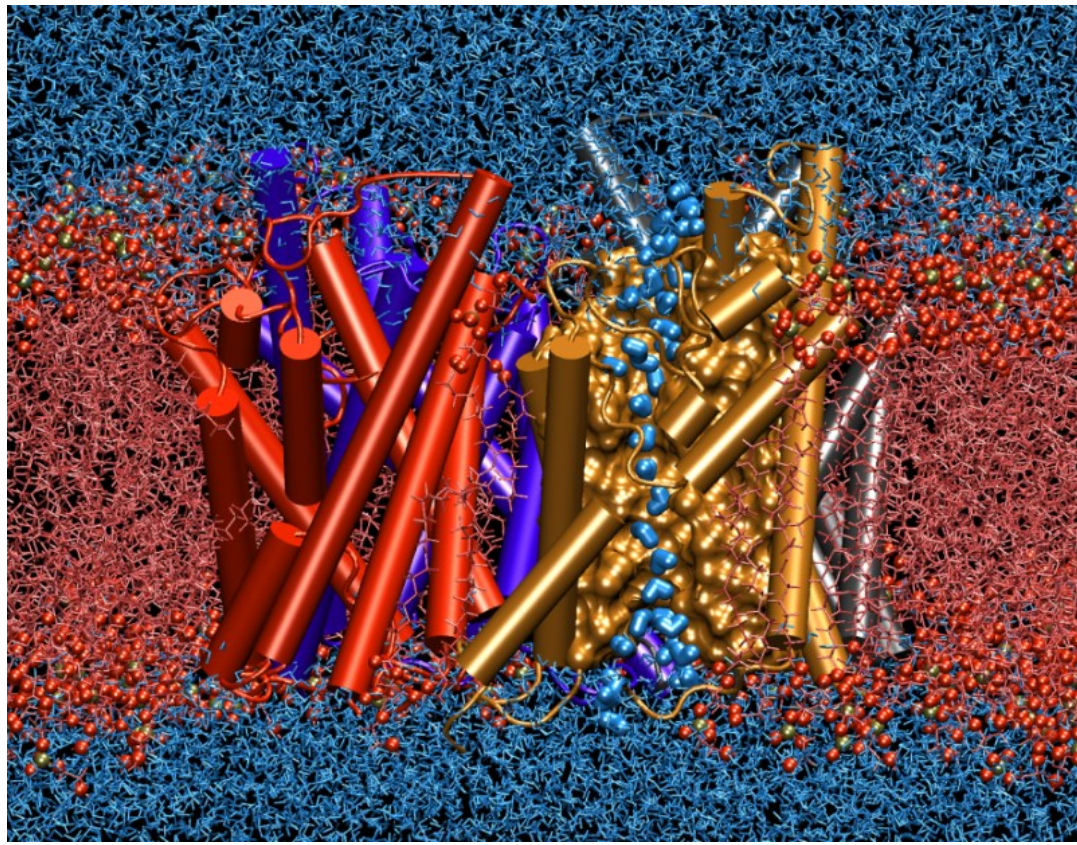
What the Computational Microscope Sees

- Sees the calculated motions of the molecular machines
- Ankyrin repeat protein being stretched out…
- Results compared between computer simulation and atomic force microscope experiments



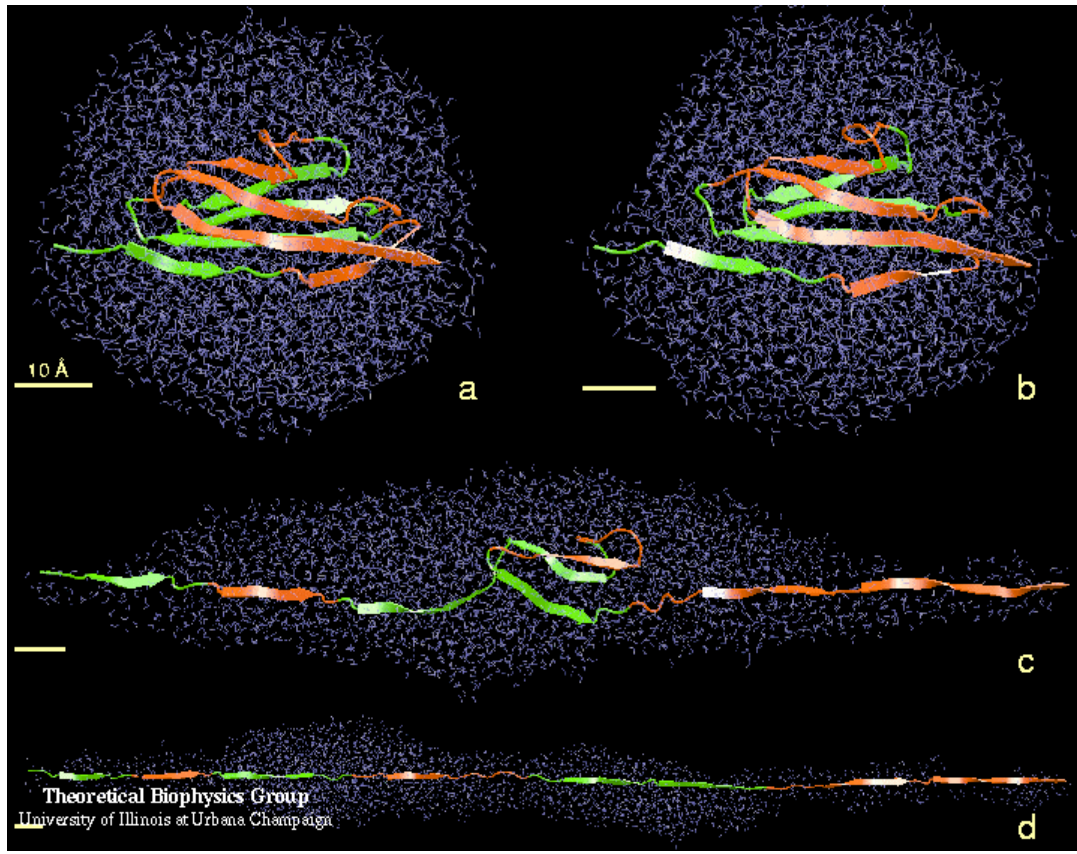
Structure Building, Simulation Preparation

- Obtain atomic structure, e.g. from the Protein Data Bank
- Integrate structure into its native biological environment:
  - Membrane
  - Water
  - Ions
- Display and analyze the prepared system



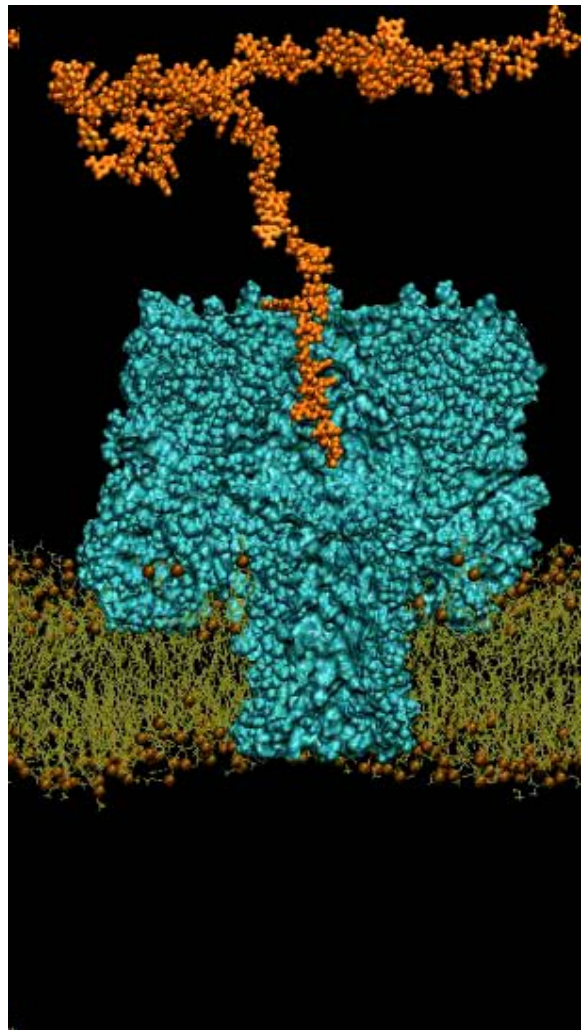
Molecular Dynamics

- Classical mechanical simulation of atomic motions (F=ma)
- Molecular dynamics calculations save trajectories of atomic coordinates as the simulation progresses
- Researchers study trajectories by analyzing force profiles, energies, structural changes etc



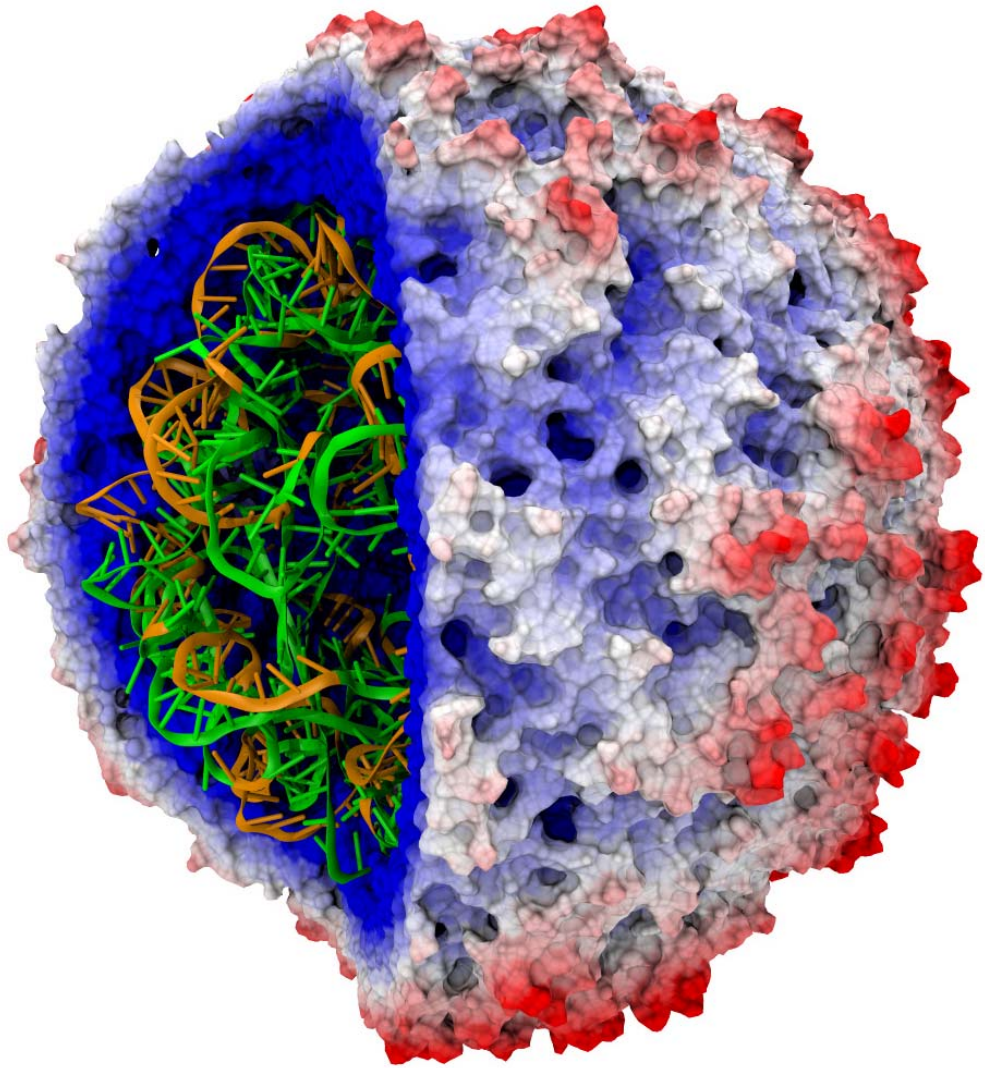
Simulation of Biological Molecules

- All-atom models of proteins, membranes, DNA, in water solution
- Classical mechanics N-body algorithms are O(N log N) at best
- N = 100K to 10M atoms today, soon up to 100M atoms
- 512 CPU jobs often run on remote supercomputers for weeks at a time for a 10ns simulation
- Visualization and analysis require workstations with 4-32 GB of RAM, 1-4 CPUs, high-end graphics accelerators

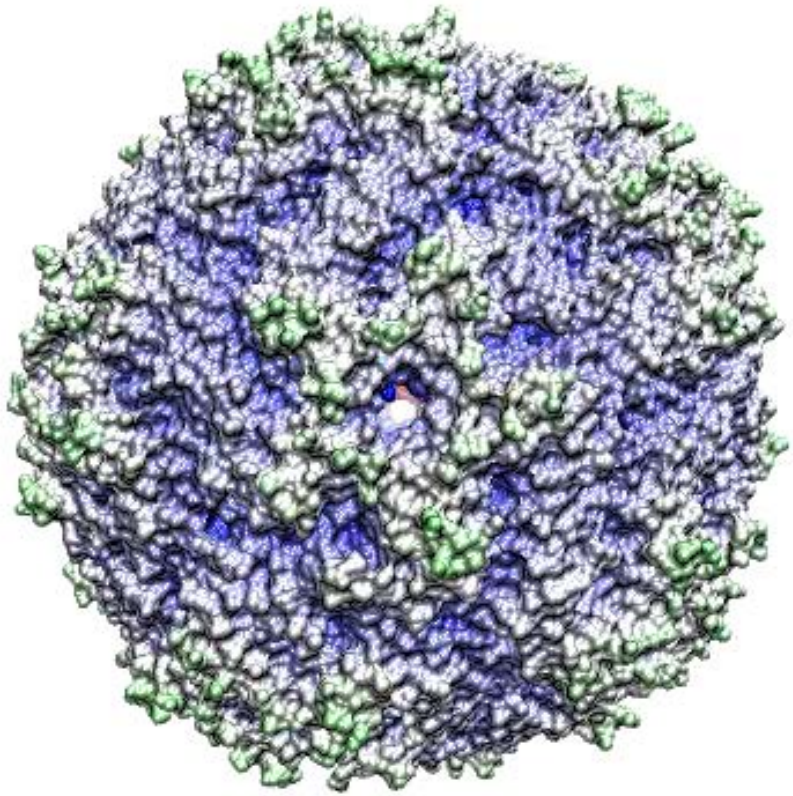


The Computational Microscope in Virology

- Simulations lead to better understanding of the mechanics of viral infections
- Better understanding of infection mechanics at the molecular level may result in more effective treatments for diseases
- Since viruses are large, their computational viewing requires tremendous resources, in particular large parallel computers

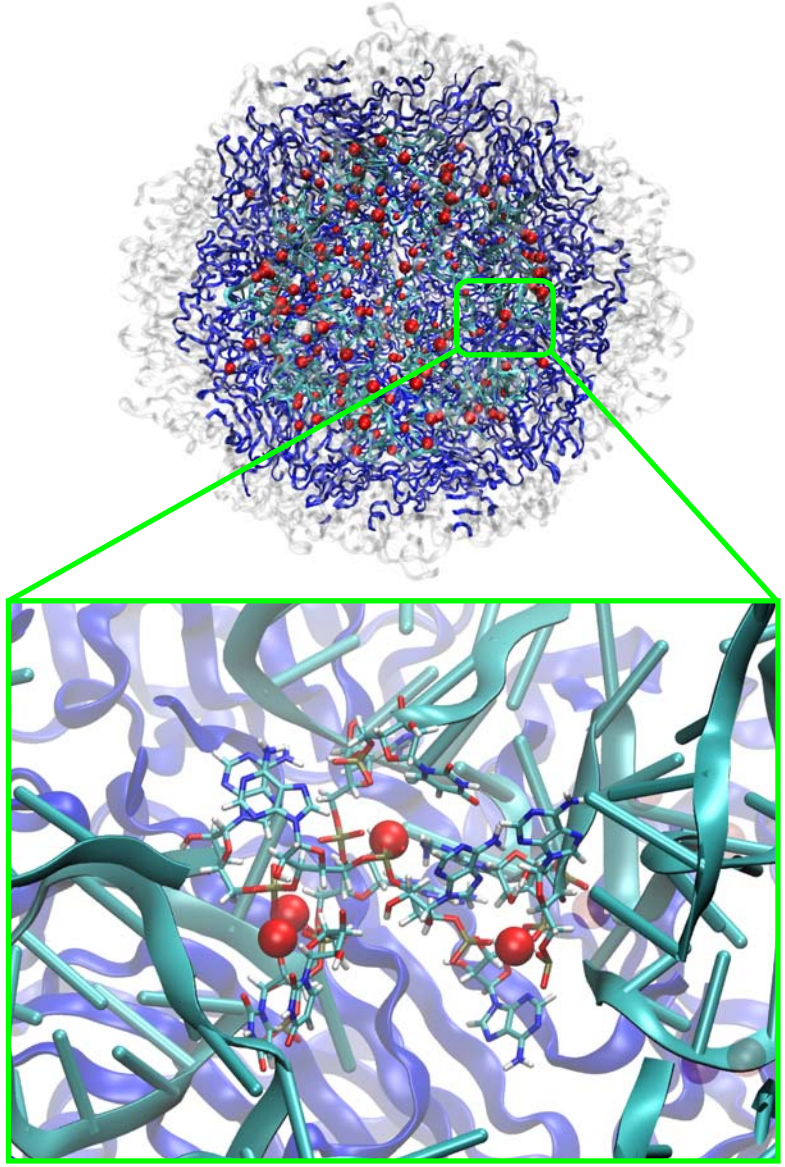


Building, Viewing, Analyzing the Virus: VMD Software



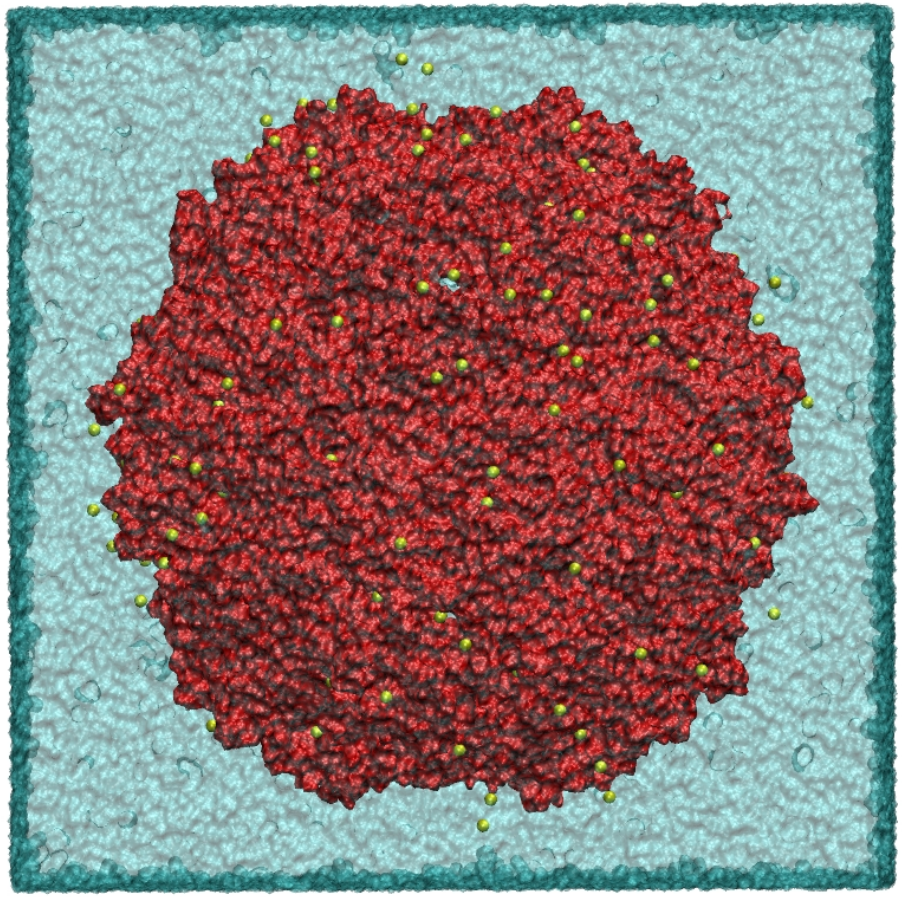
Preparing the Virus for Simulation

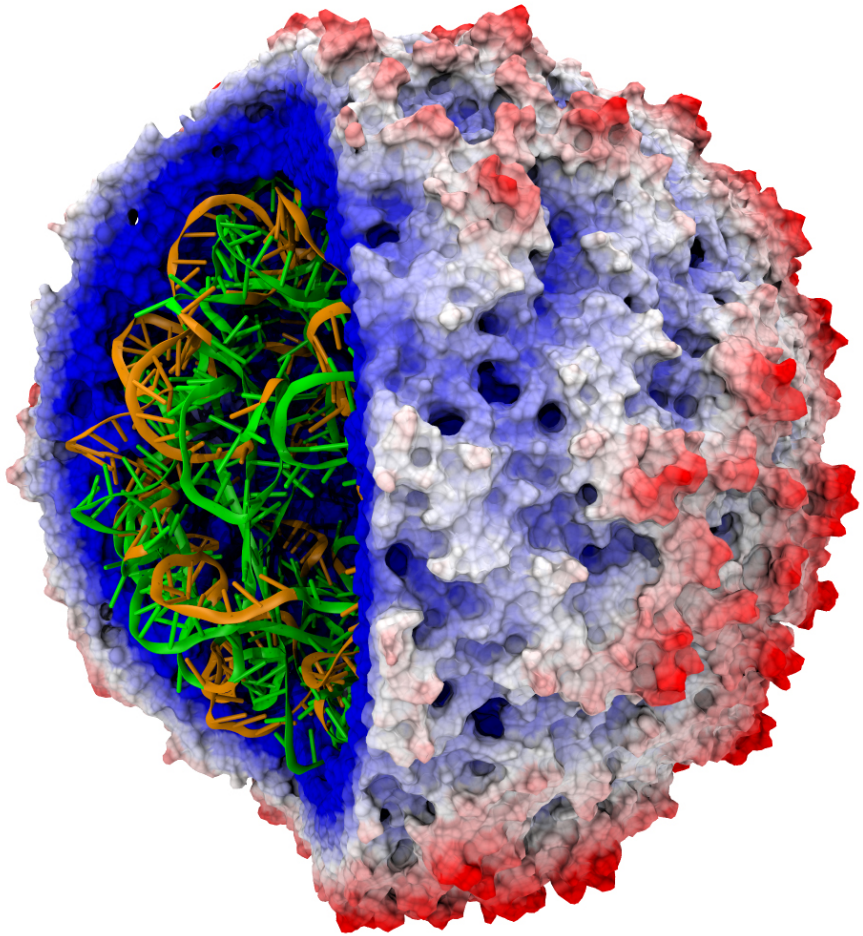
- Key task: placement of ions inside and around the virus
- Virus ion placement ran for 110 CPU-hours on SGI Altix Itanium2
- Same calculation took 27 GPU-minutes on GeForce 8800GTX with CUDA implementation
- Over 240 times faster: ion placement can now be done on a desktop machine!
- New linear-time GPU algorithm (multilevel summation) will speed this up even further



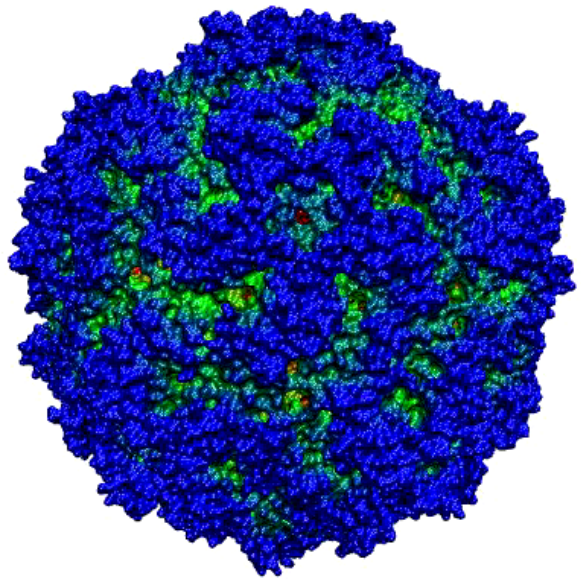
Complete Virus Model

Satellite Tobacco Mosaic Virus 932,508 atoms





Simulating the Virus: NAMD Software



STMV simulation with 1M atoms: 5.93 ns per day on 2,000 processors of Cray XT3

Timeline: Graphics Hardware Used for Molecular Visualization

60's and 70's:

- Mainframe-based vector graphics on Tektronix terminals,
- Evans & Sutherland image generators

80's:

- Transition to raster graphics on Unix workstations, Mac, PC
- Space-filling molecular representations
- Stereoscopic rendering

90's - 2002:

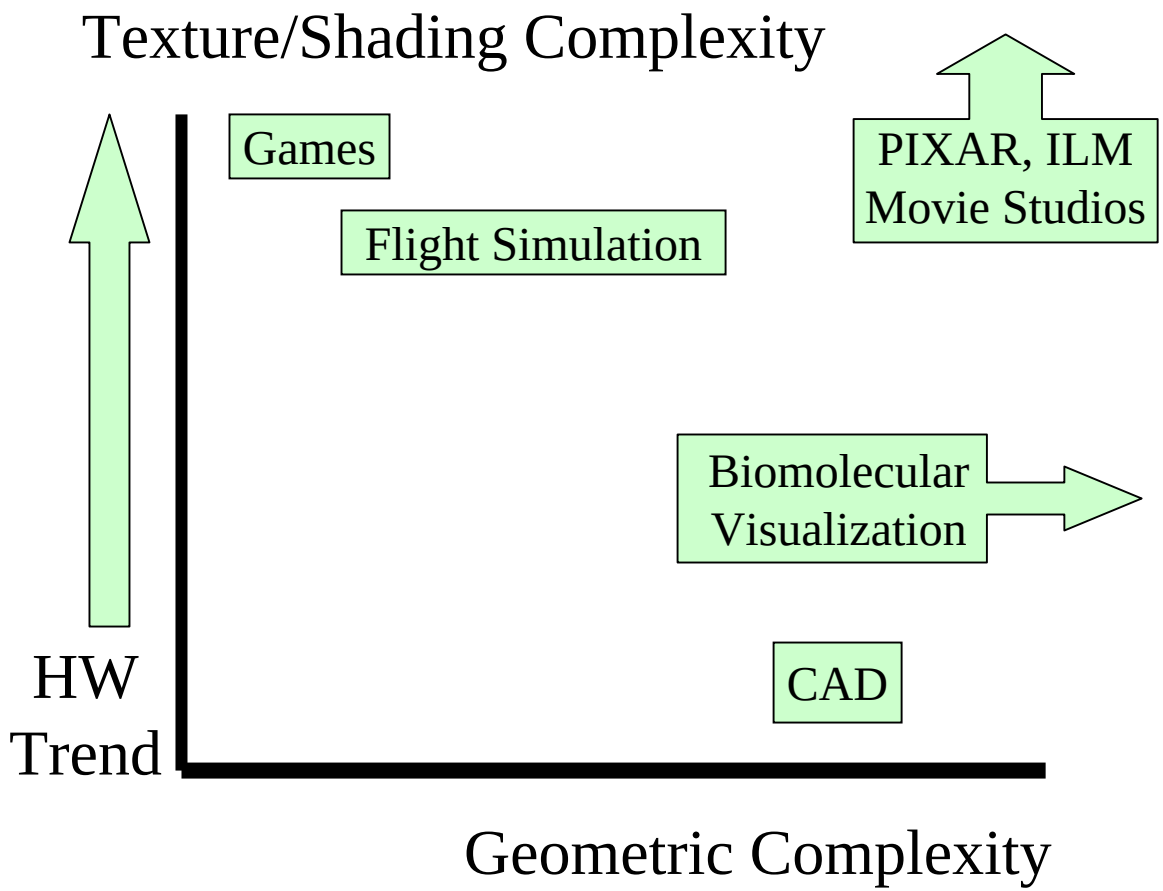
- 3rd-generation raster graphics systems
- Depth-cueing
- Texture mapping: coloring by potential, density, etc
- Full-scene antialiasing

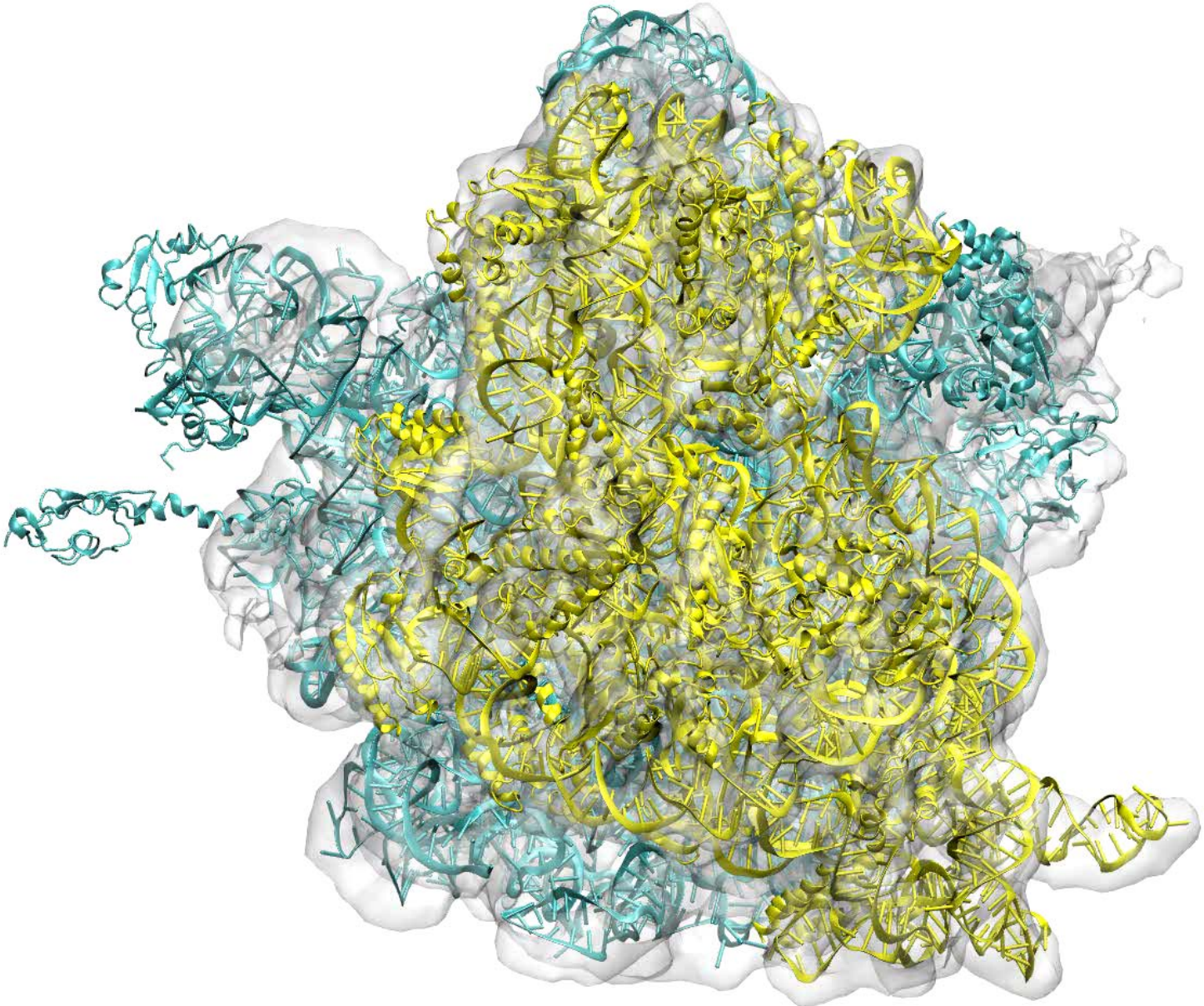
2002-present:

- Programmable shading, GPU-acceleration of geometry calculations, …

Comparison of Molecular Visualization with Other Graphics Intensive Applications

- Geometric complexity often limits molecular visualization performance
- All atoms move every simulation timestep, thwarts many LOD simplification techniques
- Commodity graphics hardware is tuned for requirements of games
- Solution: Use sophisticated shading instead of explicit geometry where possible



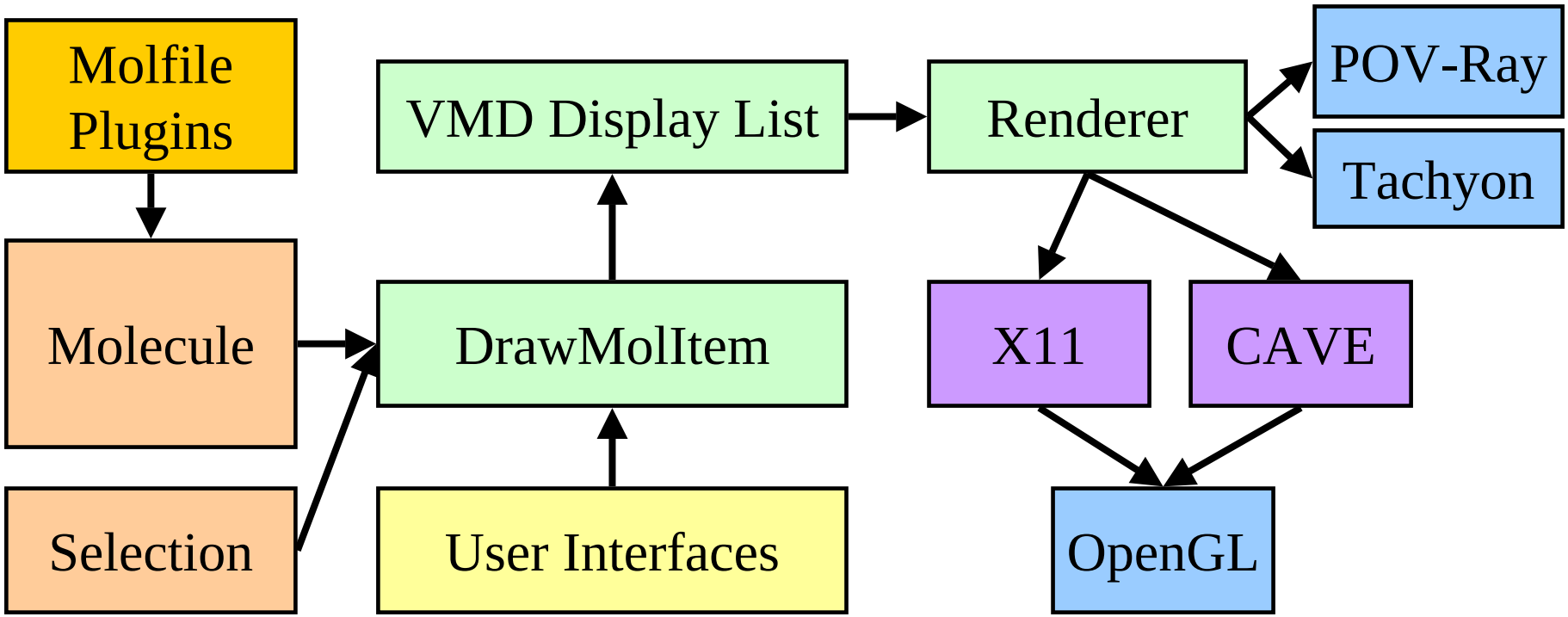


National Center for Research Resources

itute, UIUC

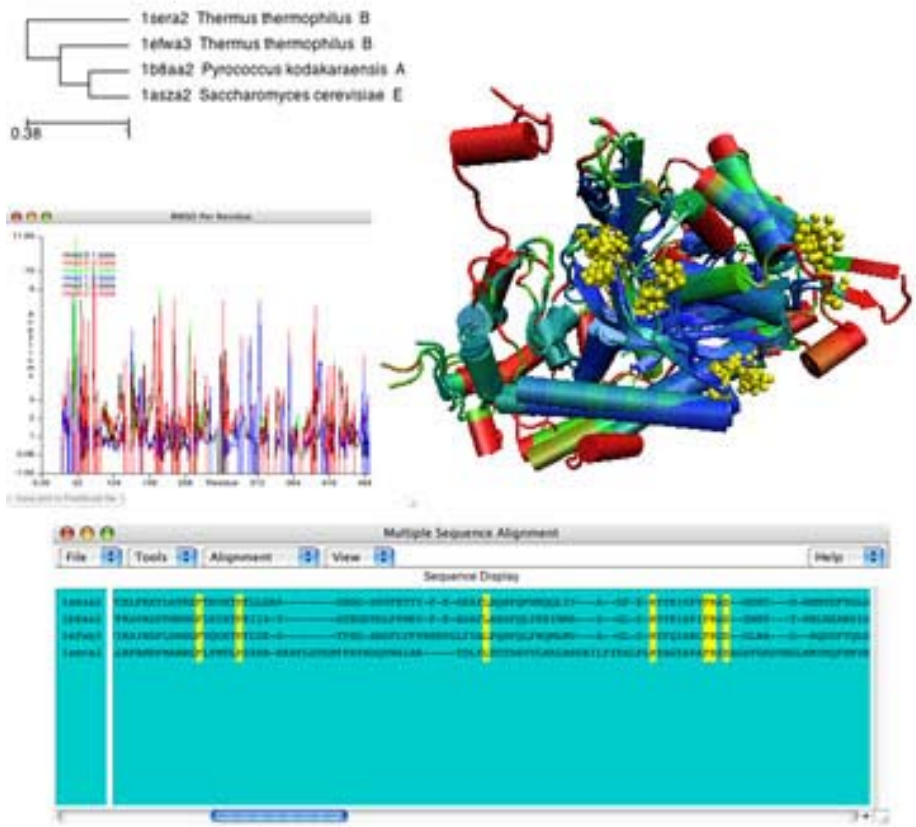
VMD Visualization Engine

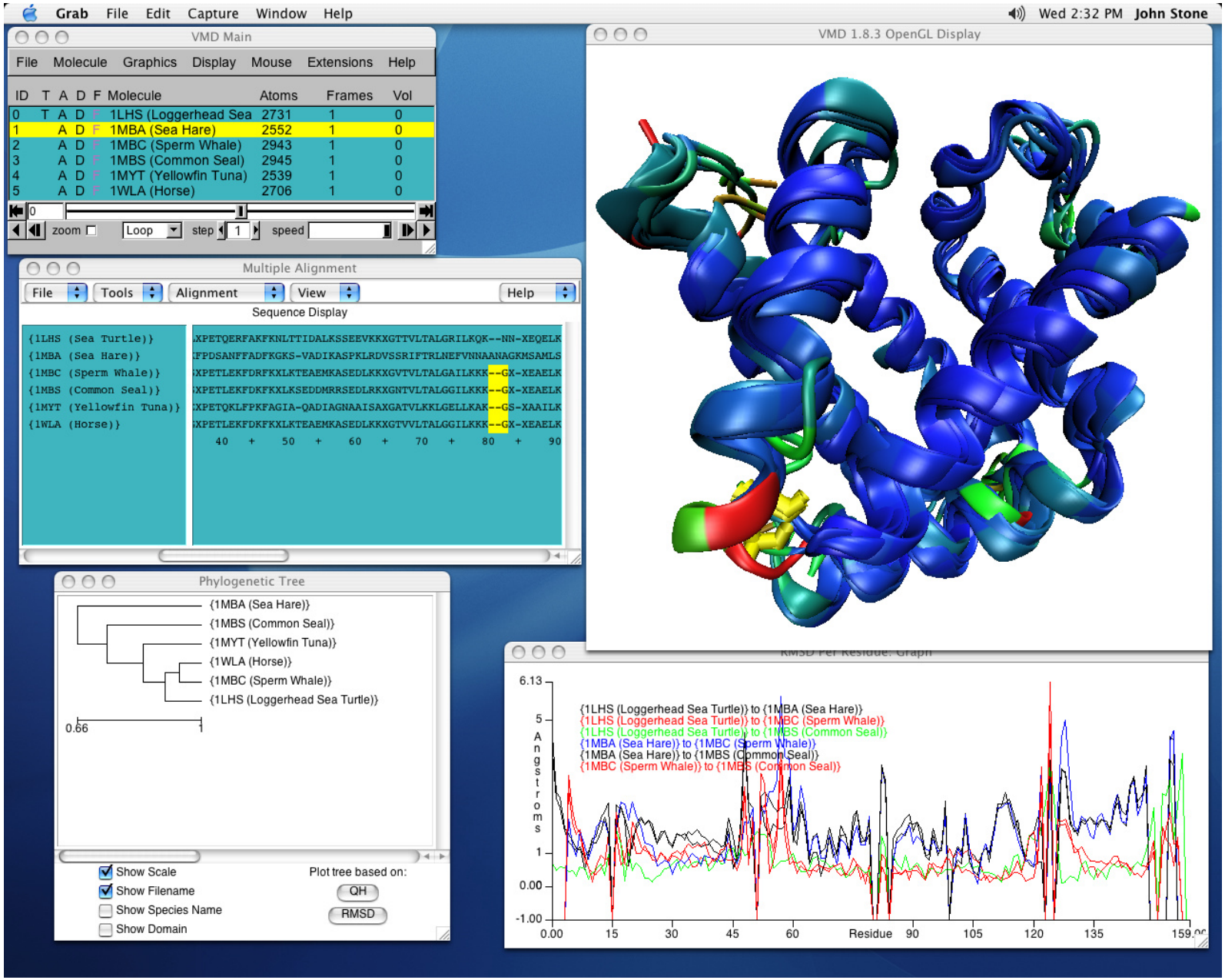
- Simplified diagram only illustrates key stages
- VMD caches reusable data at each stage



Multi-modal Visualization

- Aligned sequences and structures, phylogeny
- Simultaneous use of shape, color, topology, and interactivity
- Multiple simultaneous representations
- Multiple data display modalities
- Selections in one modality can be used to highlight or select in others




Grab File Edit Capture Window Help
Wed 2:32 PM John Stone
VMD Main
File Molecule Graphics Display Mouse Extensions Help
ID T A D F Molecule Atoms Frames Vol
0 A D F 1LHS (Loggerhead Sea 2731 1 0
1 A D F 1MBA (Sea Hare) 2552 1 0
2 A D F 1MBC (Sperm Whale) 2943 1 0
3 A D F 1MBS (Common Seal) 2945 1 0
4 A D F 1MYT (Yellowfin Tuna) 2539 1 0
5 A D F 1WLA (Horse) 2706 1 0
zoom Loop step 1 speed
VMD 1.8.3 OpenGL Display
Multiple Alignment
File Tools Alignment View Help
Sequence Display
{1LHS (Sea Turtle)}
{1MBA (Sea Hare)}
{1MBC (Sperm Whale)}
{1MBS (Common Seal)}
{1MYT (Yellowfin Tuna)}
{1WLA (Horse)}
40 + 50 + 60 + 70 + 80 + 90
Phylogenetic Tree
{1MBA (Sea Hare)}
{1MBS (Common Seal)}
{1MYT (Yellowfin Tuna)}
{1WLA (Horse)}
{1MBC (Sperm Whale)}
{1LHS (Loggerhead Sea Turtle)}
0.66
1
Show Scale
Show Filename
Show Species Name
Show Domain
Plot tree based on:
QH
RMSD
RMSD Per Residue: Graph
6.13
5
Angstroms
1
0.00
-1.00
{1LHS (Loggerhead Sea Turtle)} to {1MBA (Sea Hare)}
{1LHS (Loggerhead Sea Turtle)} to {1MBC (Sperm Whale)}
{1LHS (Loggerhead Sea Turtle)} to {1MBS (Common Seal)}
{1MBA (Sea Hare)} to {1MBC (Sperm Whale)}
{1MBA (Sea Hare)} to {1MBS (Common Seal)}
{1MBC (Sperm Whale)} to {1MBS (Common Seal)}
0.00 15 30 45 60 Residue 90 105 120 135 159.00

Displaying 10M Atom Complexes

- Uncompressed atom coordinates 120MB (3 floats/atom)
- Avoid traversal of per-atom data, hierarchical data structures are a must
- Caching, lazy evaluation, multithreading, overlapped rendering with computation
- Geometry caching, symmetry/instancing accelerate static structure display
- Representation geometry often 10-50x size of atom coordinate data, but is largely ephemeral

Programmable Graphics Hardware

Groundbreaking research systems:

AT&T Pixel Machine (1989):

82 x DSP32 processors

UNC PixelFlow (1992-98):

64 x (PA-8000 +

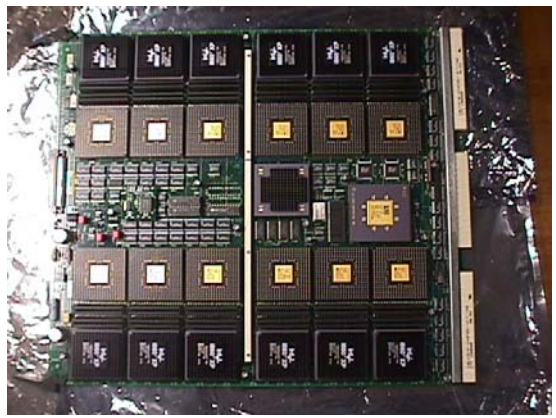
8,192 bit-serial SIMD)

SGI RealityEngine (1990s):

Up to 12 i860-XP processors perform vertex operations (*u*code), fixed-func. fragment hardware

All mainstream GPUs now incorporate programmable processors

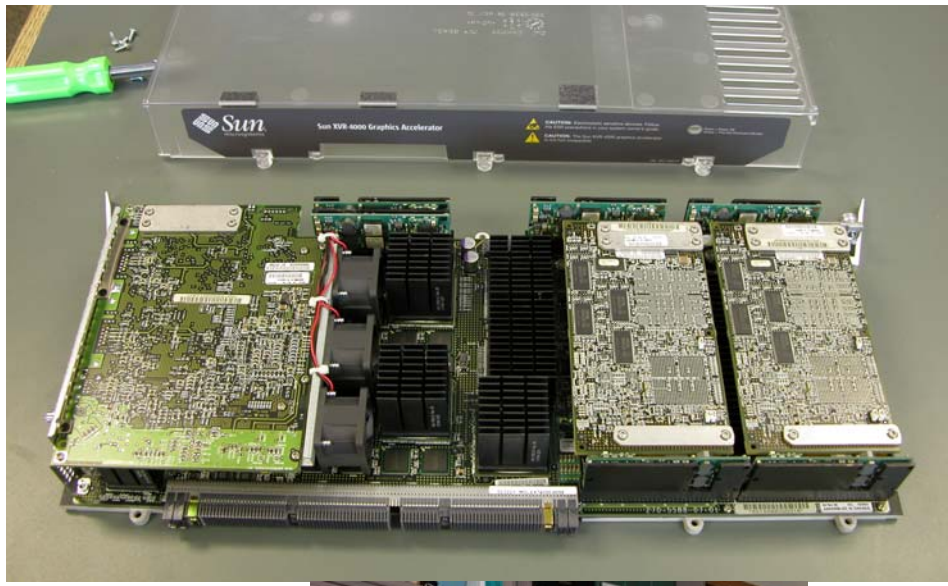
UNC PixelFlow Rack



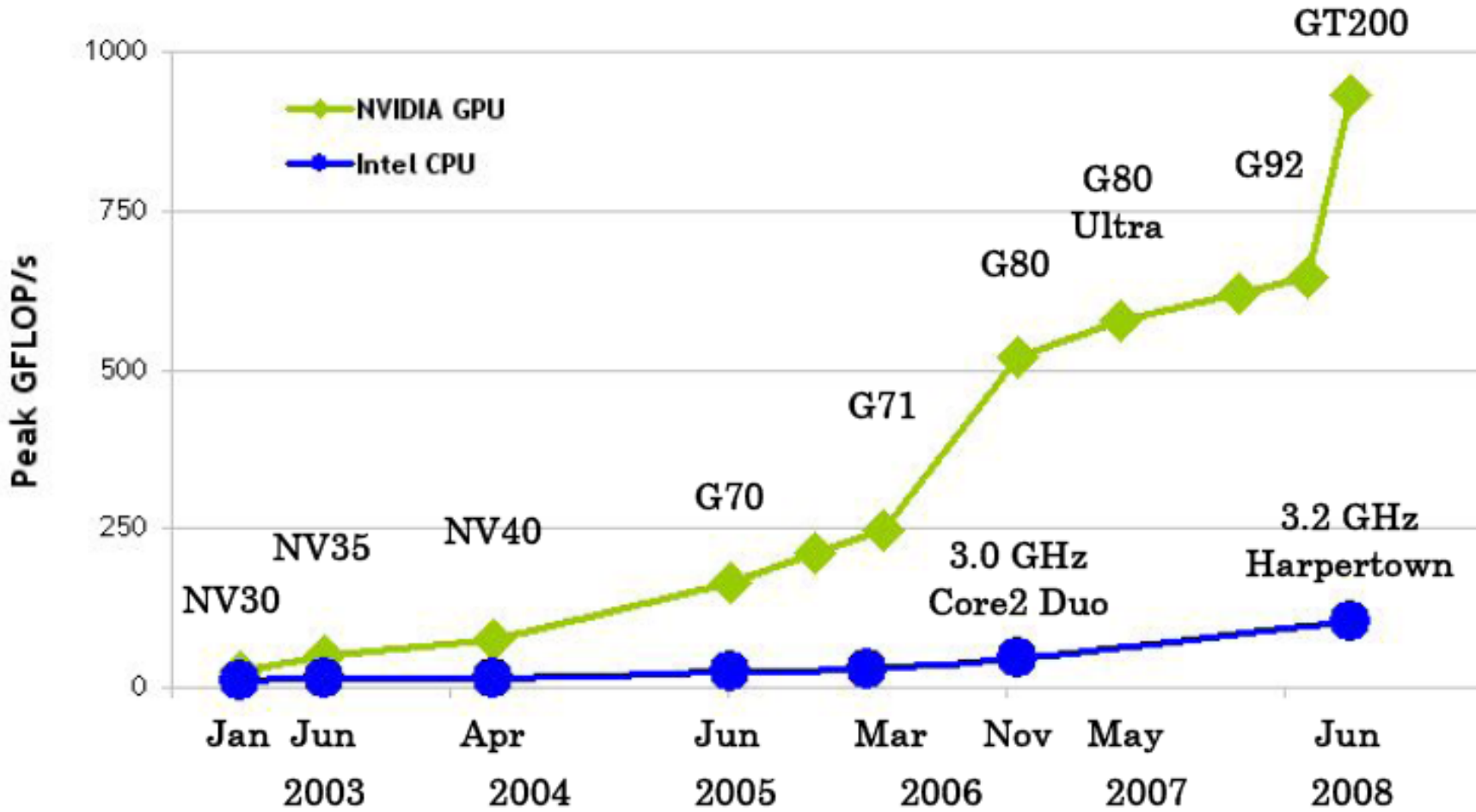
Reality Engine Vertex Processors

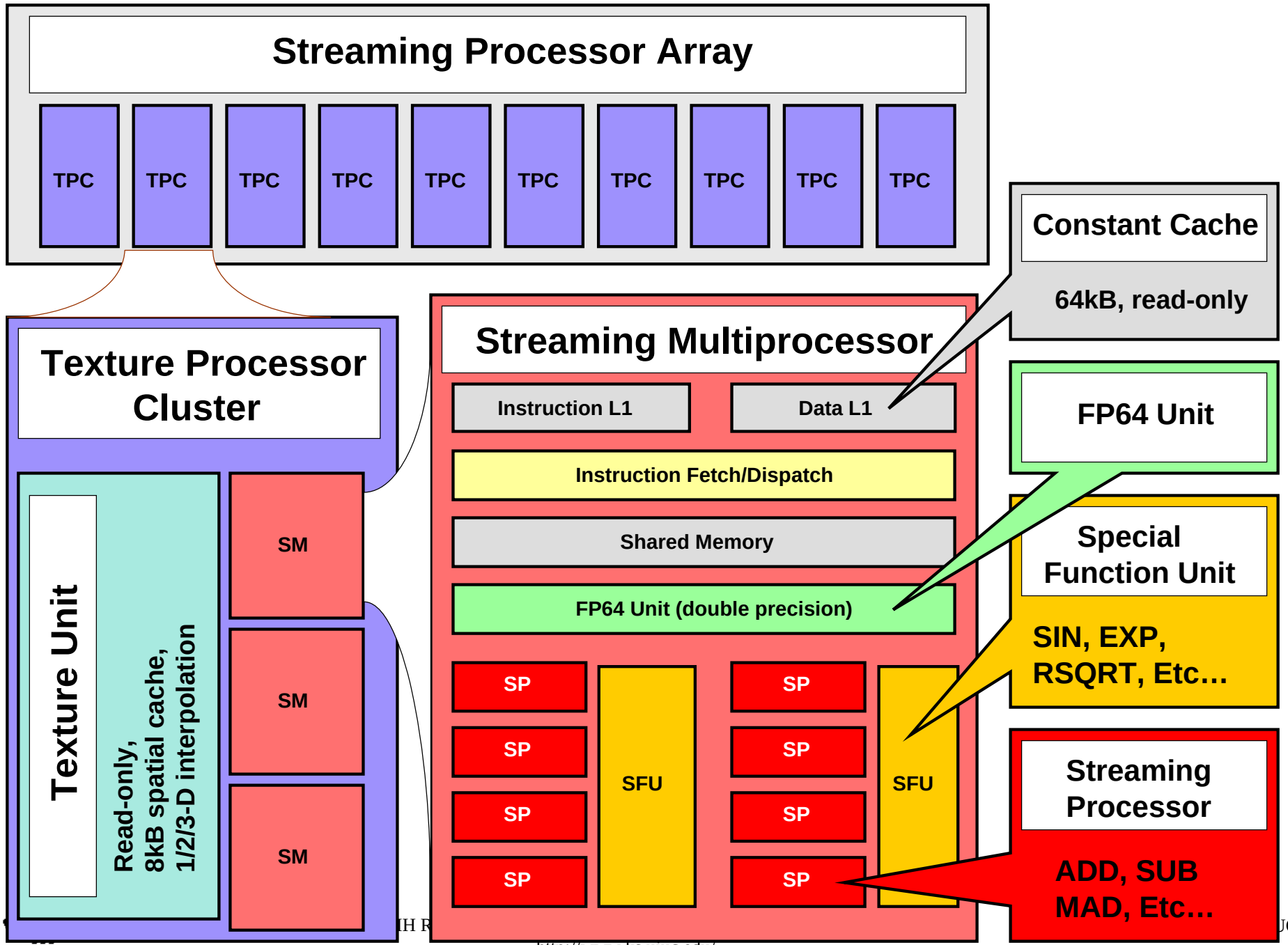
Early Experiments with Programmable Graphics Hardware in VMD

- Sun XVR-1000/4000 (2002)
  - 4xMAJC-5200 CPUs
  - 1GB Texture RAM
  - 32MB *u*code RAM
  - 1 Teraflop Antialiasing Filter Pipeline
- Custom *u*code and OpenGL extension for rendering spheres
  - Draw only half-spheres, with solid side facing the viewer
  - 1-sided lighting
  - Host CPU only sends arrays of radii, positions, colors
  - Fast DMA engines copy arrays from system memory to GPU
  - Overall performance was over twice as fast, host CPU load significantly decreased



Peak Single-precision Arithmetic Performance Trend, GPU vs. CPU




Streaming Processor Array
TPC
TPC
TPC
TPC
TPC
TPC
TPC
TPC
TPC
TPC
Texture Processor Cluster
Texture Unit
Read-only, 8kB spatial cache, 1/2/3-D interpolation
SM
SM
SM
Streaming Multiprocessor
Instruction L1
Data L1
Instruction Fetch/Dispatch
Shared Memory
FP64 Unit (double precision)
SP
SP
SP
SP
SFU
SP
SP
SP
SP
SFU
Constant Cache
64kB, read-only
FP64 Unit
Special Function Unit
SIN, EXP, RSQRT, Etc…
Streaming Processor
ADD, SUB MAD, Etc…

Evolution of Molecular Graphics Software Design

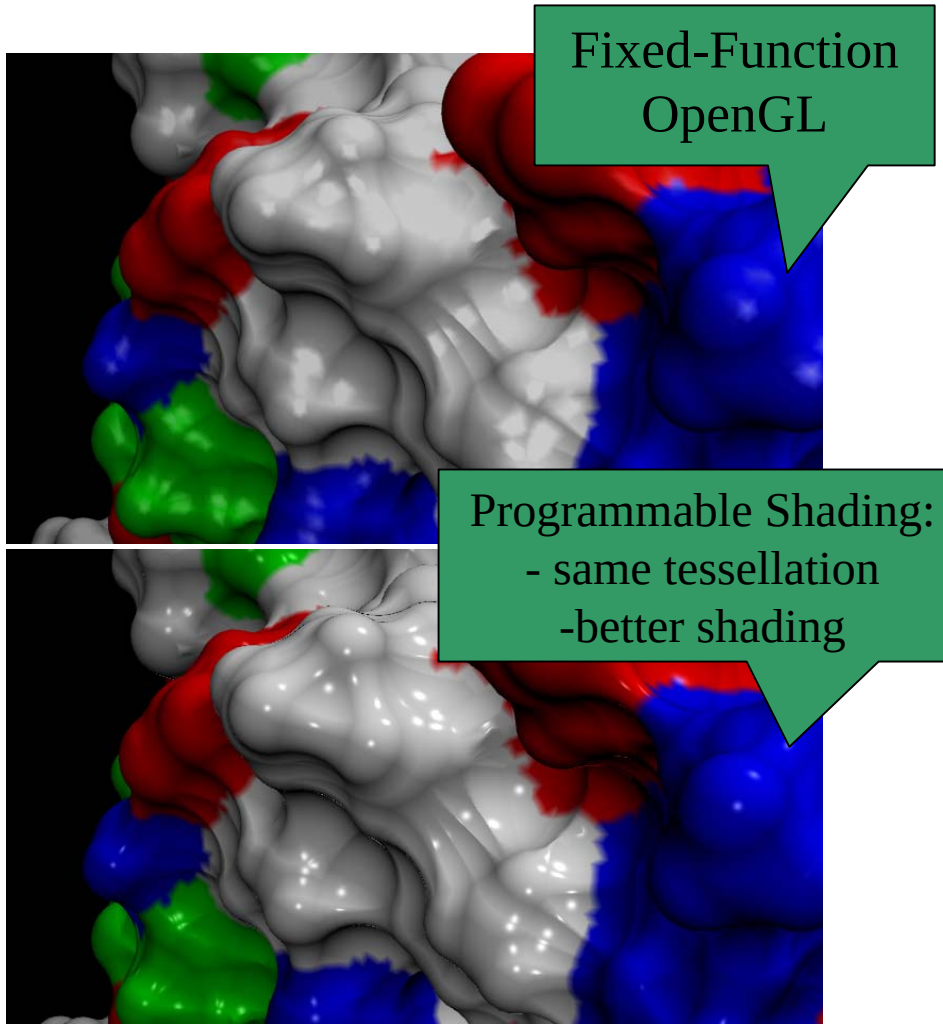
- Given ongoing trends:
  - Increasing size and complexity of simulations
  - Increased demand for interactivity in all stages of modeling, visualization, and analysis
  - Relative floating point performance and memory bandwidth of GPUs vs CPUs
- Highly parallelizable work must shift to GPUs
- CPU cores retain high level functions and coarse parallelism not suited to GPUs

Offloading More of Molecular Graphics and Modeling Work to the GPU

- Graphics related work first to go, adjust algorithms to favor higher GPU utilization:
  - GPU must generate geometry itself, not enough CPU-GPU bandwidth otherwise, particularly for trajectory animation
  - Use programmable shading rather than finely tessellated surface geometry
  - Proxy objects, ray casting
- Begin moving highly parallel modeling algorithms to the GPU (e.g. CUDA)

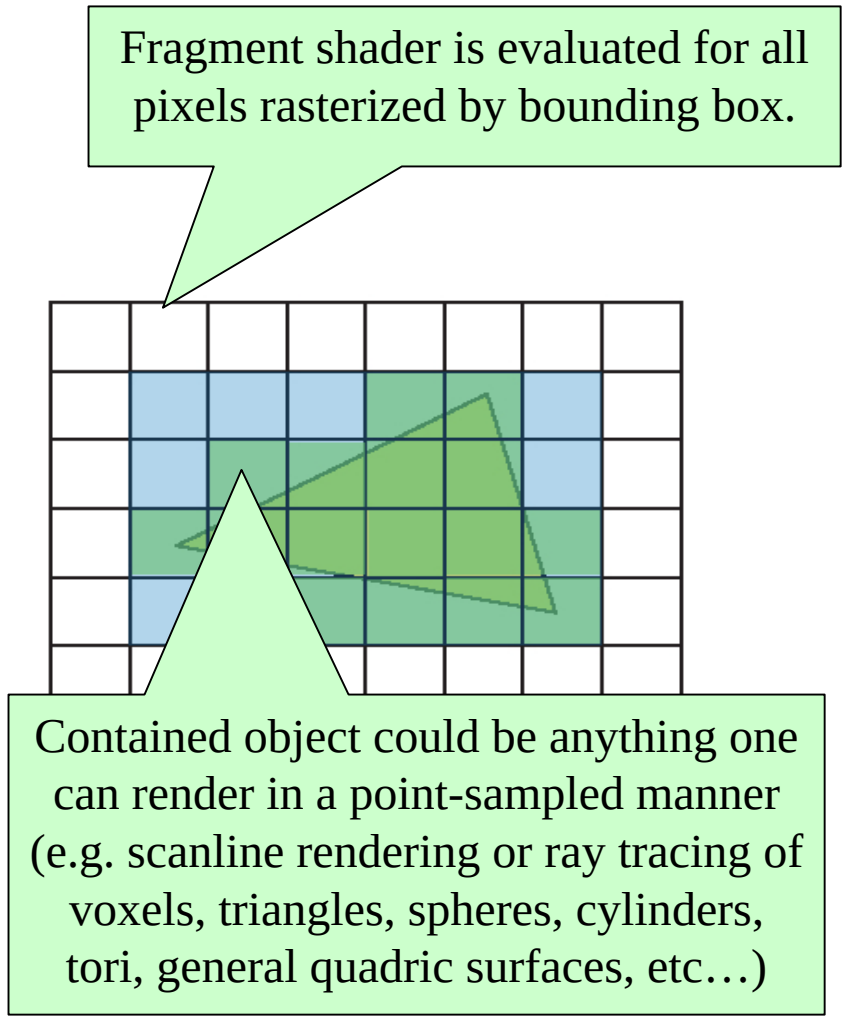
Benefits of Programmable Shading

- Potential for superior image quality with better shading algorithms
- Direct rendering of:
  - Quadric surfaces
  - Density map data, solvent surfaces
- Offload work from host CPU to GPU



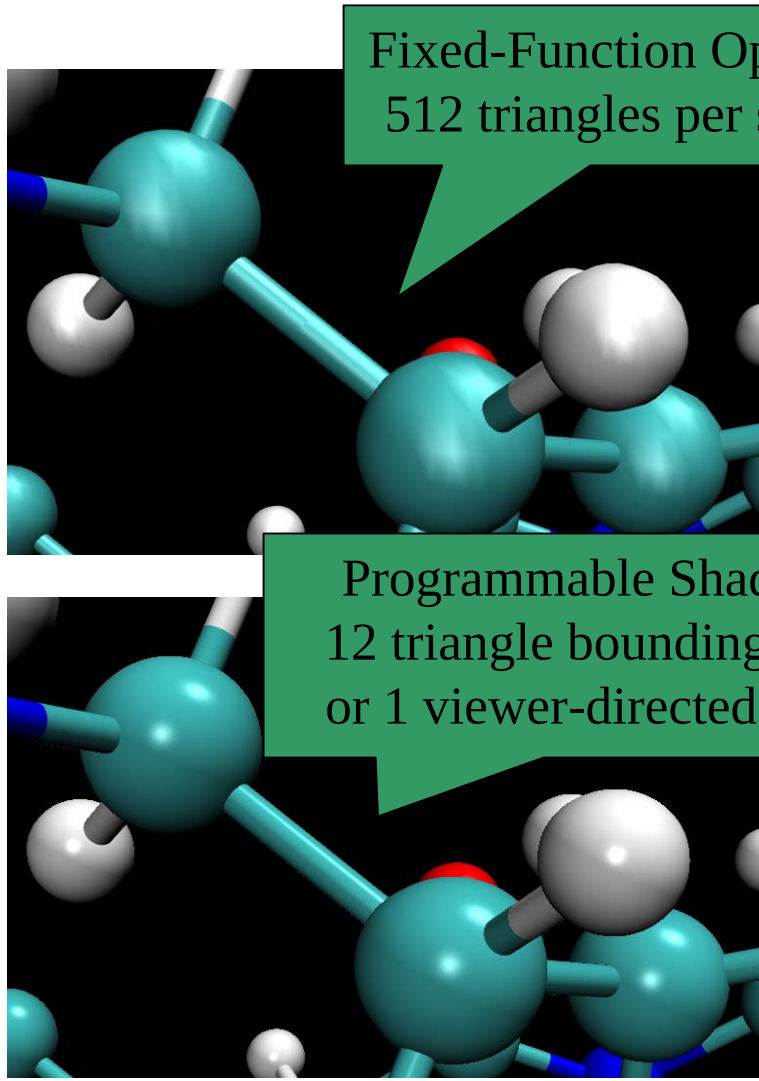
Rendering Non-polygonal Data with Present-day Programmable Shading

- Algorithms mapped to vertex/fragment shading model available in current hardware
- Render by drawing bounding box or a viewer-directed quad containing shape/data
- Vertex shader sets up
- Fragment shader performs all the work

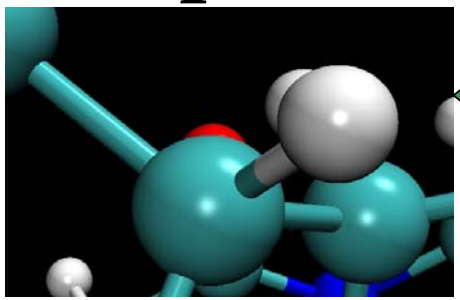


Ray Traced Sphere Rendering with Programmable Shading

- Fixed-function OpenGL requires curved surfaces to be tessellated with triangles, lines, or points
- Fine tessellation required for good results with Gouraud shading; performance suffers
- Static tessellations look bad when viewer zooms in
- Dynamic tessellation too costly when animating huge trajectories
- Programmable shading solution:
  - Ray trace spheres in fragment shader
  - GPU does all the work
  - Spheres look good at all zoom levels
  - Rendering time is proportional to pixel area covered by sphere
  - Overdraw is a bigger penalty than for triangulated spheres



Sphere Fragment Shader



- Written in OpenGL Shading Language
- High-level C-like language with vector types and operations
- Compiled dynamically by the graphics driver at *runtime*
- Compiled machine code executes on GPU

```
//
// VMD Sphere Fragment Shader (not for normal geometry)
//
void main(void) {
  vec3 raydir = normalize(V);
  vec3 spheredir = spherepos - rayorigin;

  // Perform ray-sphere intersection tests based on the code in Tachyon
  float b = dot(raydir, spheredir);
  float temp = dot(spheredir, spheredir);
  float disc = b*b + sphereradsq - temp;

  // only calculate the nearest intersection, for speed
  if (disc <= 0.0)
    discard; // ray missed sphere entirely, discard fragment

  // calculate closest intersection
  float tnear = b - sqrt(disc);

  if (tnear < 0.0)
    discard;

  // calculate hit point and resulting surface normal
  vec3 pnt = rayorigin + tnear * raydir;
  vec3 N = normalize(pnt - spherepos);

  // Output the ray-sphere intersection point as the fragment depth
  // rather than the depth of the bounding box polygons.
  // The eye coordinate Z value must be transformed to normalized device
  // coordinates before being assigned as the final fragment depth.
  if (vmdprojectionmode == 1) {
    // perspective projection = 0.5 + (hfpn + (f * n / pnt.z)) / diff
    gl_FragDepth = 0.5 + (vmdprojparms[2] + (vmdprojparms[1] * vmdprojparms[(
3];
  } else {
    // orthographic projection = 0.5 + (-hfpn - pnt.z) / diff
    gl_FragDepth = 0.5 + (-vmdprojparms[2] - pnt.z) / vmdprojparms[3];
  }

#ifdef TEXTURE
  // perform texturing operations for volumetric data
  // The only texturing mode that applies to the sphere shader
```

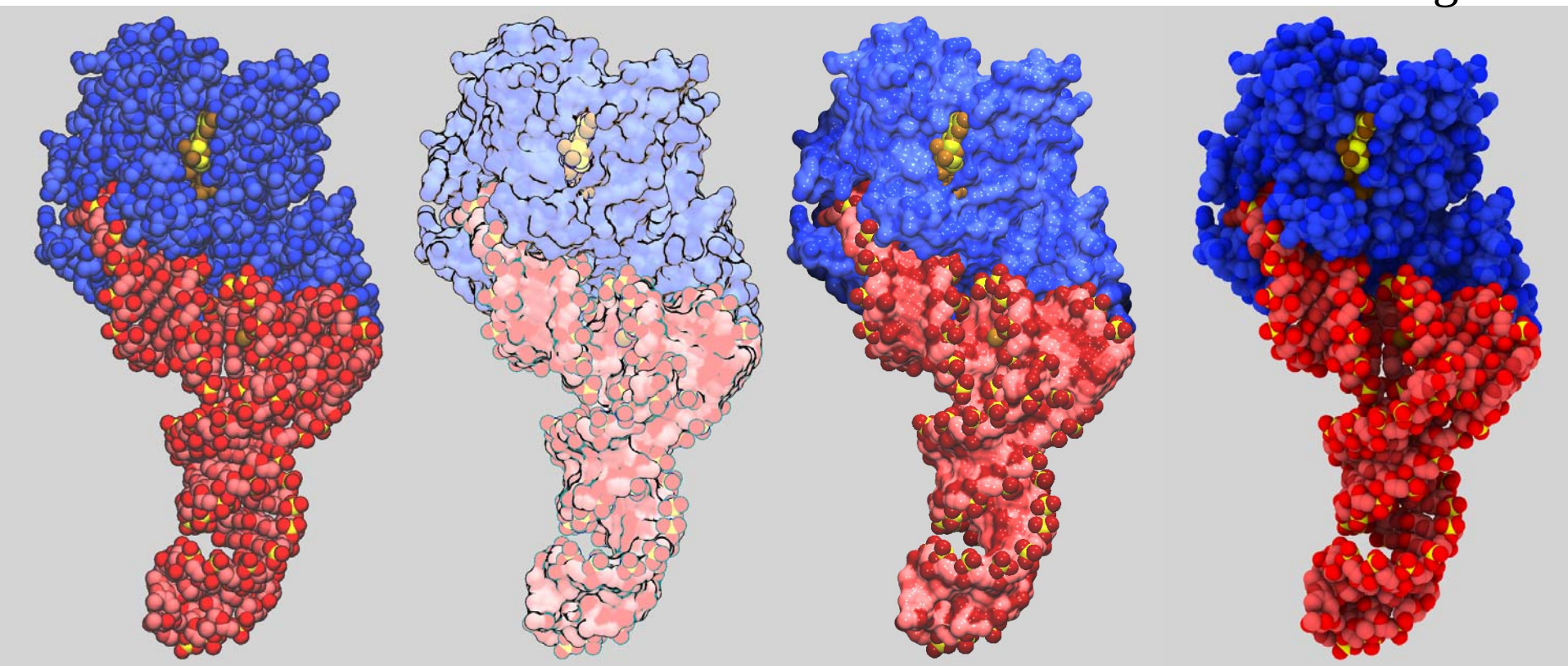
Shading Comparison: EF-Tu

Outline Shader

"Goodsell" Shader

Glossy Shader

Ambient Occlusion, Shadowing



VMD Interactive OpenGL Rendering

VMD/Tachyon

Efficient 3-D Texturing of Large Datasets

- MIP mapping, compressed map data
- Perform volumetric color transfer functions on GPU rather than on the host CPU
  - perform all range clamping and density-to-color mapping on GPU
  - update color transfer function without re-downloading large texture maps

Strategies for Working Within GPU Hardware Constraints

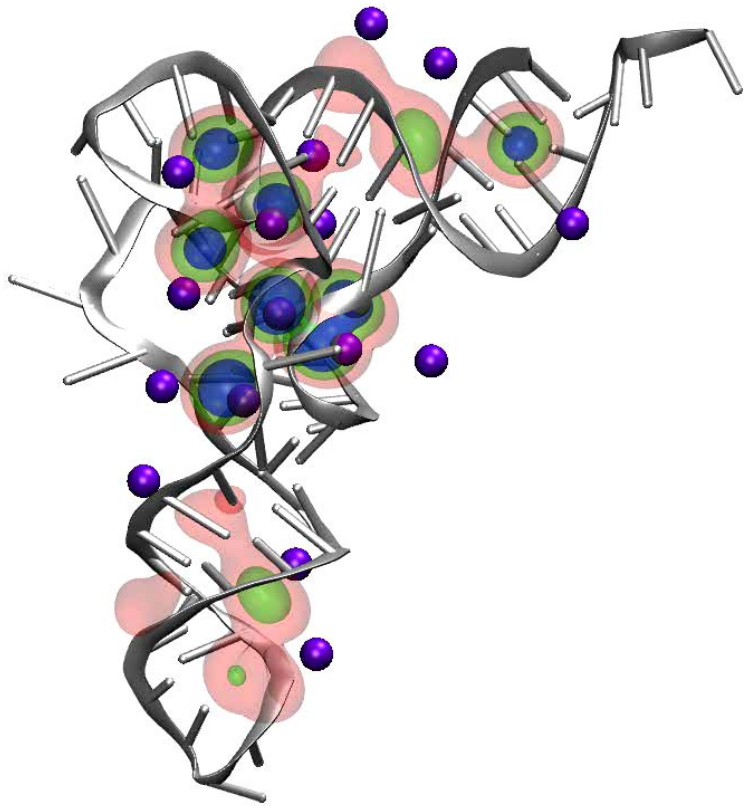
- Most GPUs <= 1GB RAM currently
- Use bricked data, multi-level grids, view-dependent map resolution
- Use occlusion culling to prevent rendering of bricks that aren't visible, thus avoiding texture download/access
- Use reduced precision FP types for surface normal / gradient maps

Further GPU Acceleration

- Atomic representation tessellation, spline calculations, atom selections, spatial queries computed entirely on GPU
- Direct rendering of isosurfaces from volumetric data via ray casting (e.g. electron density surfaces, codes exist already)
- Computation and direct rendering of metaball (“Blob”) approximation of molecular surfaces via ray casting (codes exist already)

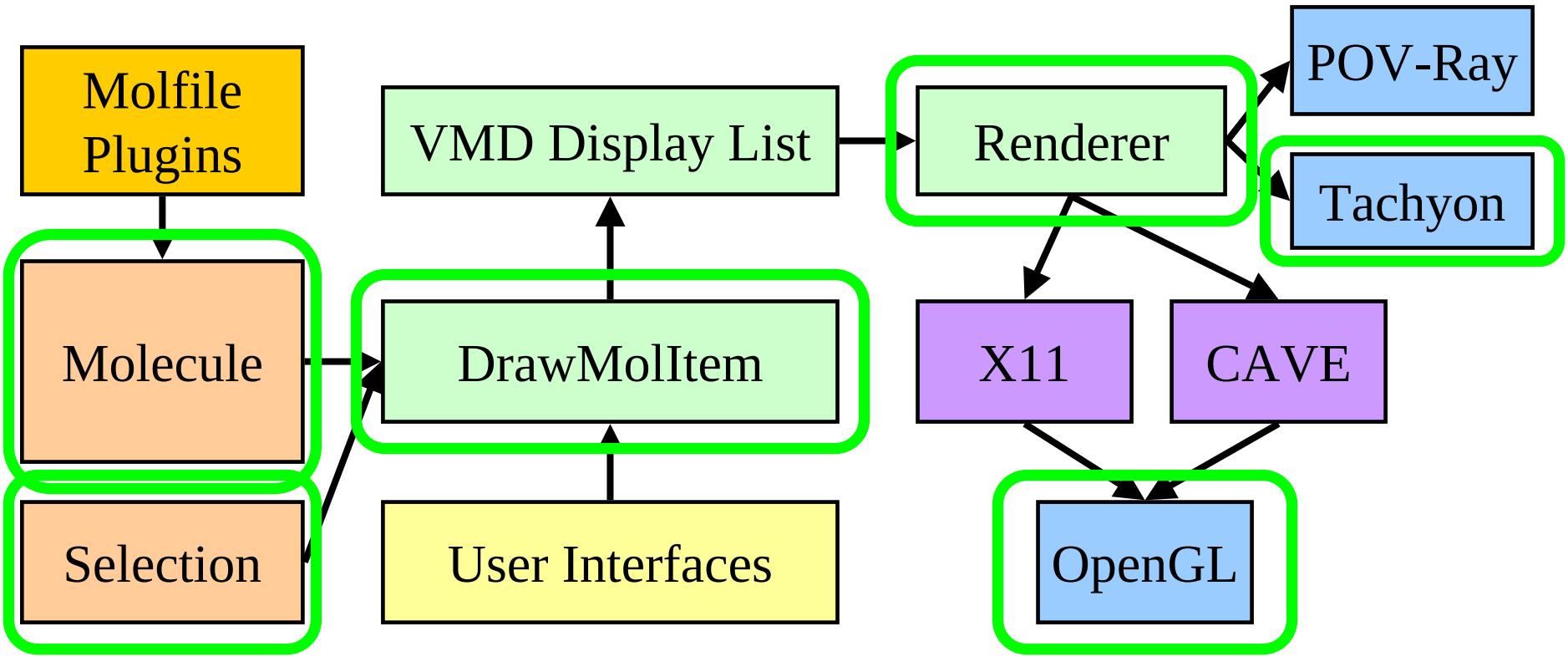
Computing Volumetric Properties

- Compute density, distance, occupancy, potential maps for a frame or averaged over a trajectory
- Well suited to GPU acceleration
- Example: display binding sites for diffusively bound ions as probability density isosurfaces



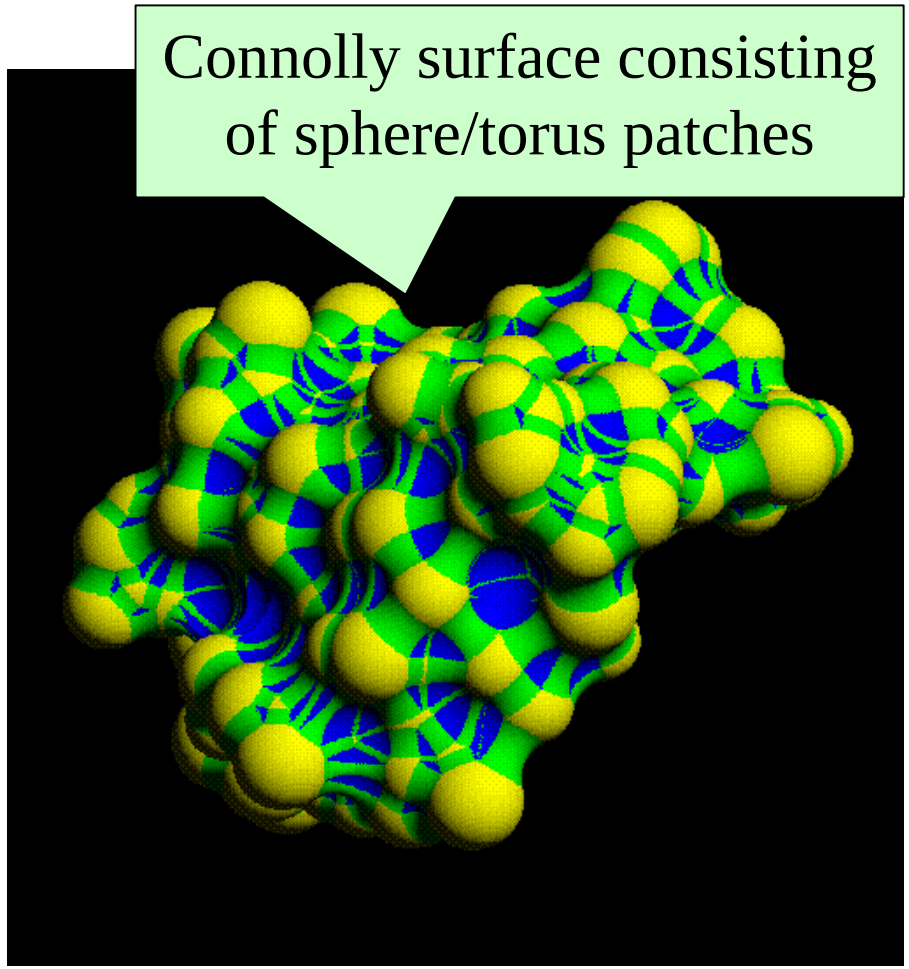
tRNA magnesium ion occupancy

VMD Visualization Engine, GPU Acceleration Opportunities



The Wheel of Reincarnation: Revival of Molecular Graphics Techniques?

- Graphics hardware is making another trip around the wheel of reincarnation (Myer and Sutherland CACM ’68)
- Visualization techniques that weren’t triangle-friendly lost favor in the 90’s may return
- Algorithms that mapped poorly to fixed-function OpenGL are often easier to implement with programmable shading
- Non-polygonal geometry can now be rendered entirely on the GPU itself

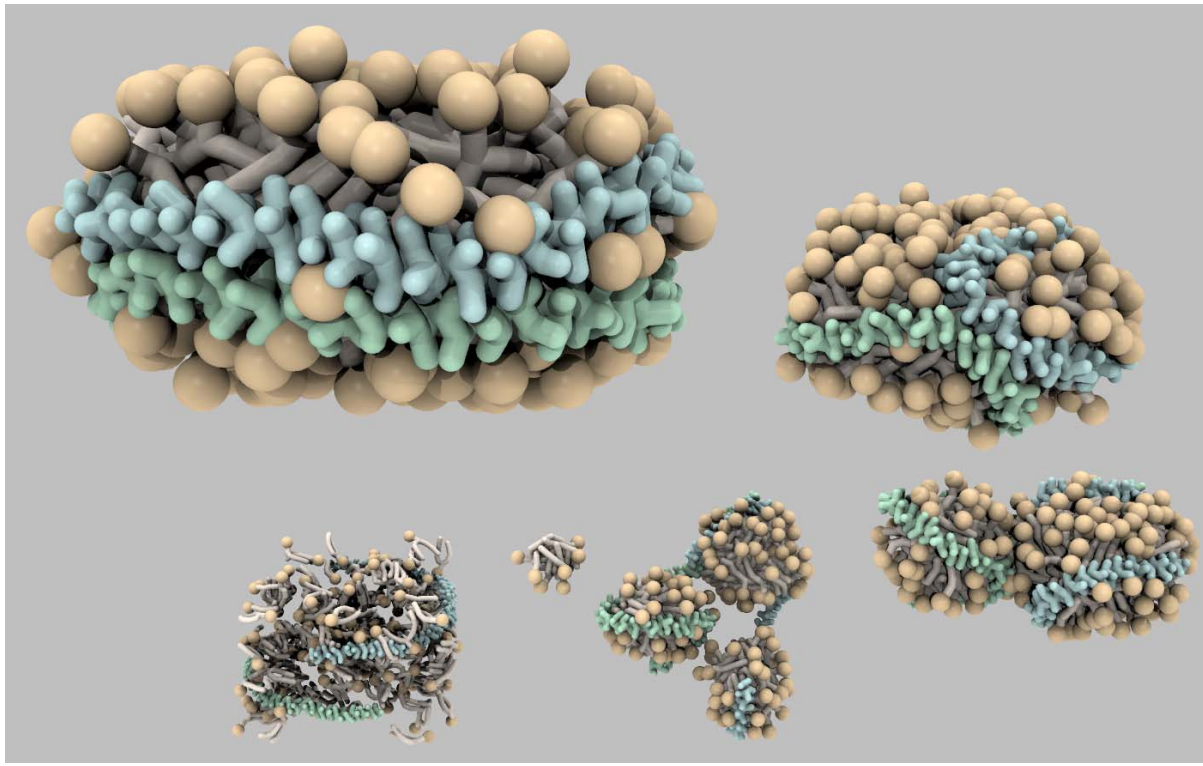


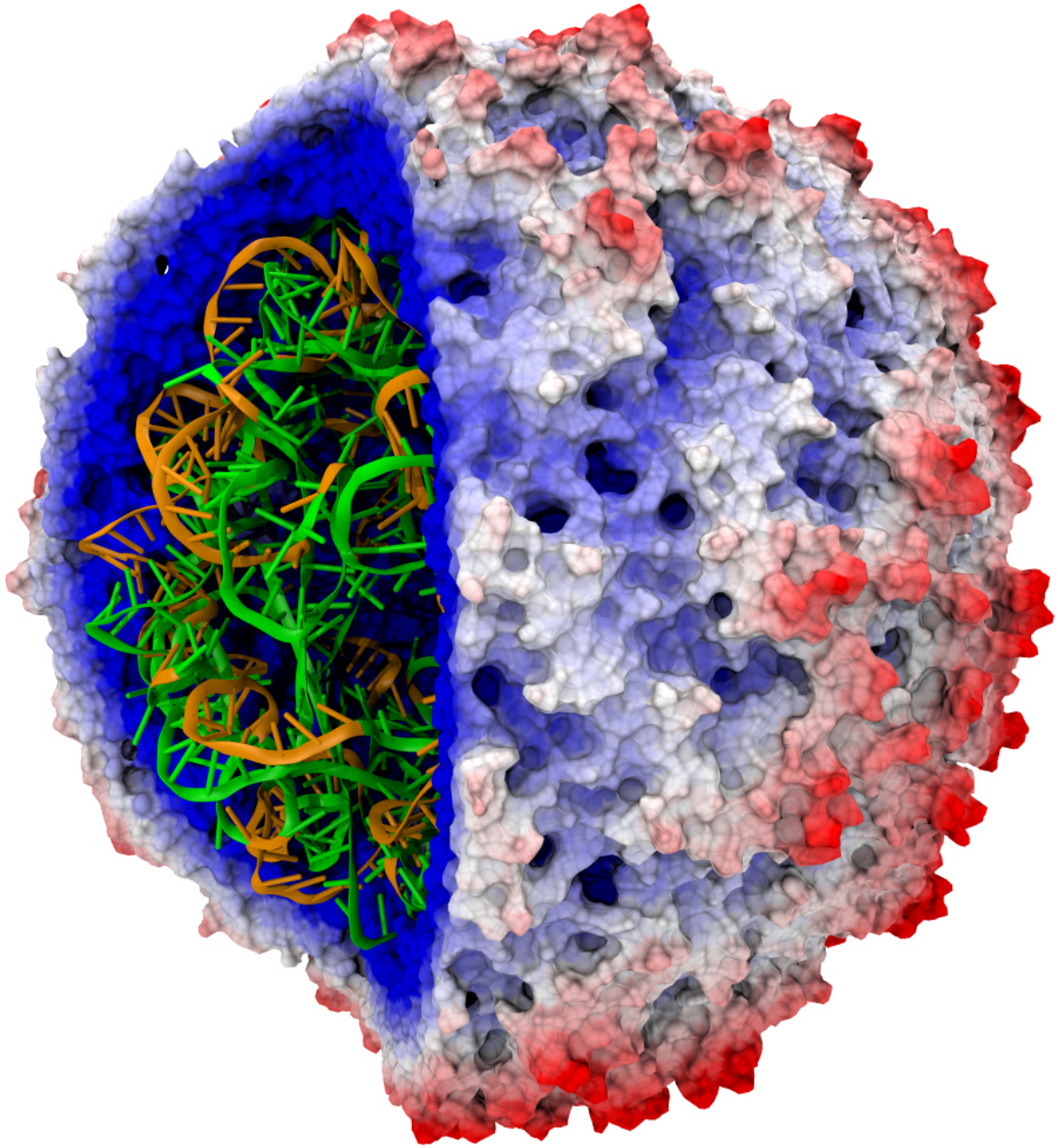
Next-Gen Graphics Architectures

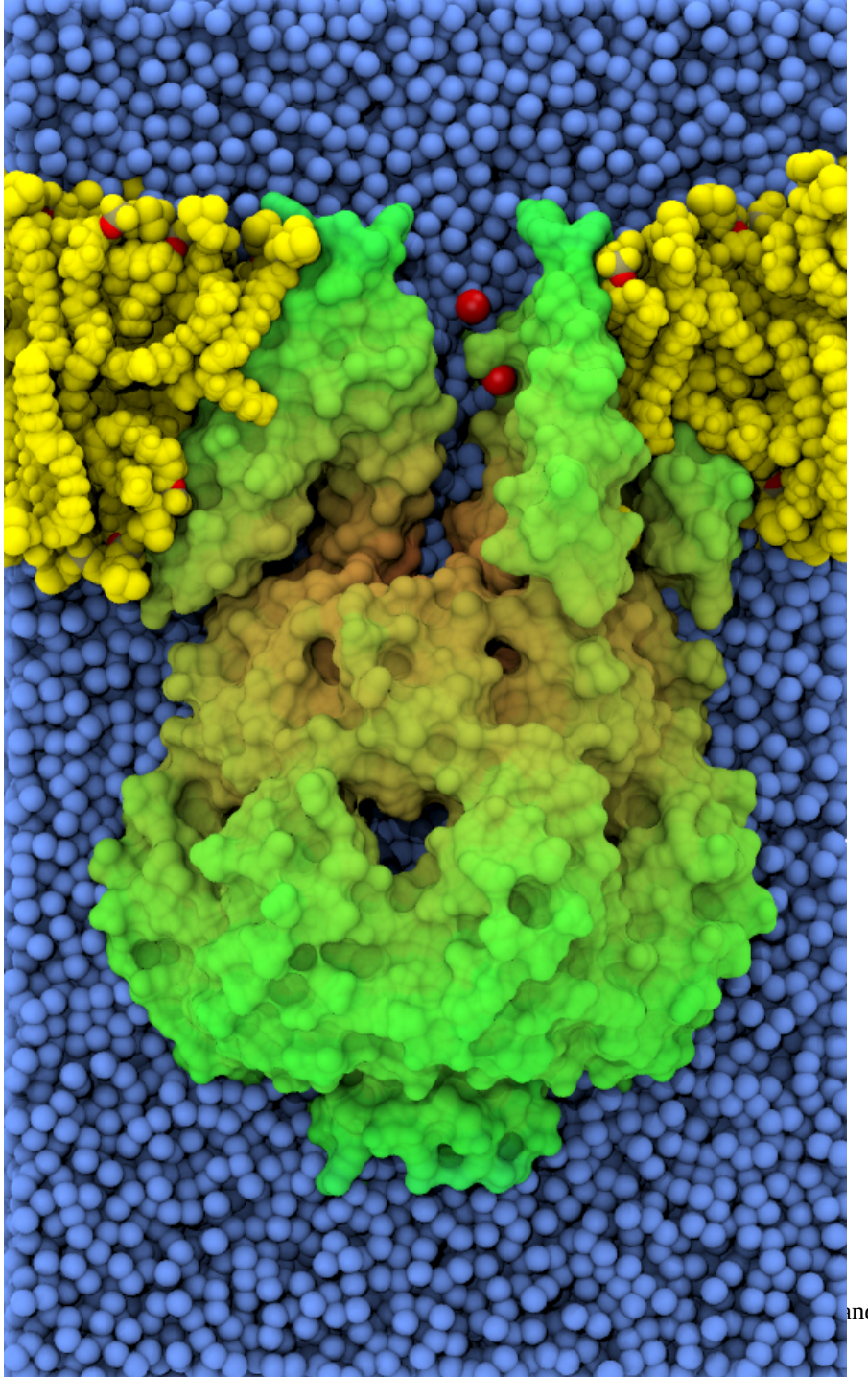
- New programmable pipeline stages: geometry shader, pre-tessellation vertex shader
- Predicated rendering commands, conditions evaluated in hardware (culling operations, etc)
- Mixed OpenGL, CUDA, etc.

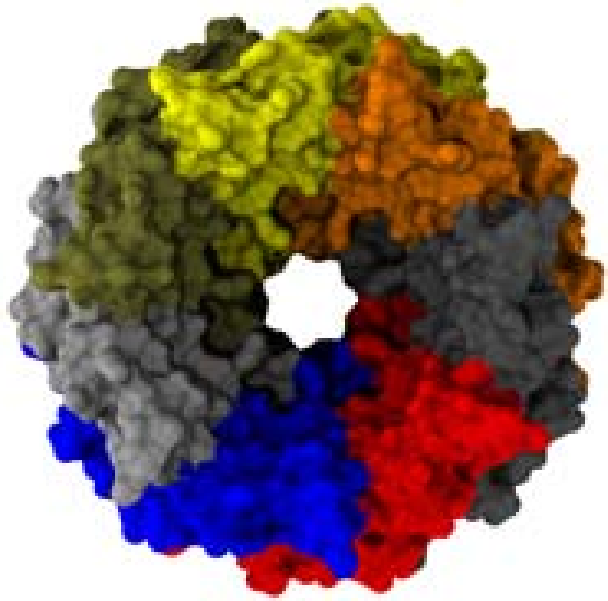
Higher Quality Rendering: VMD/Tachyon Ambient Occlusion Lighting

- Omnidirectional diffuse lighting
- Improved shape perception
- Tachyon tuned for use by VMD
- Tachyon AO lighting works with all VMD representations

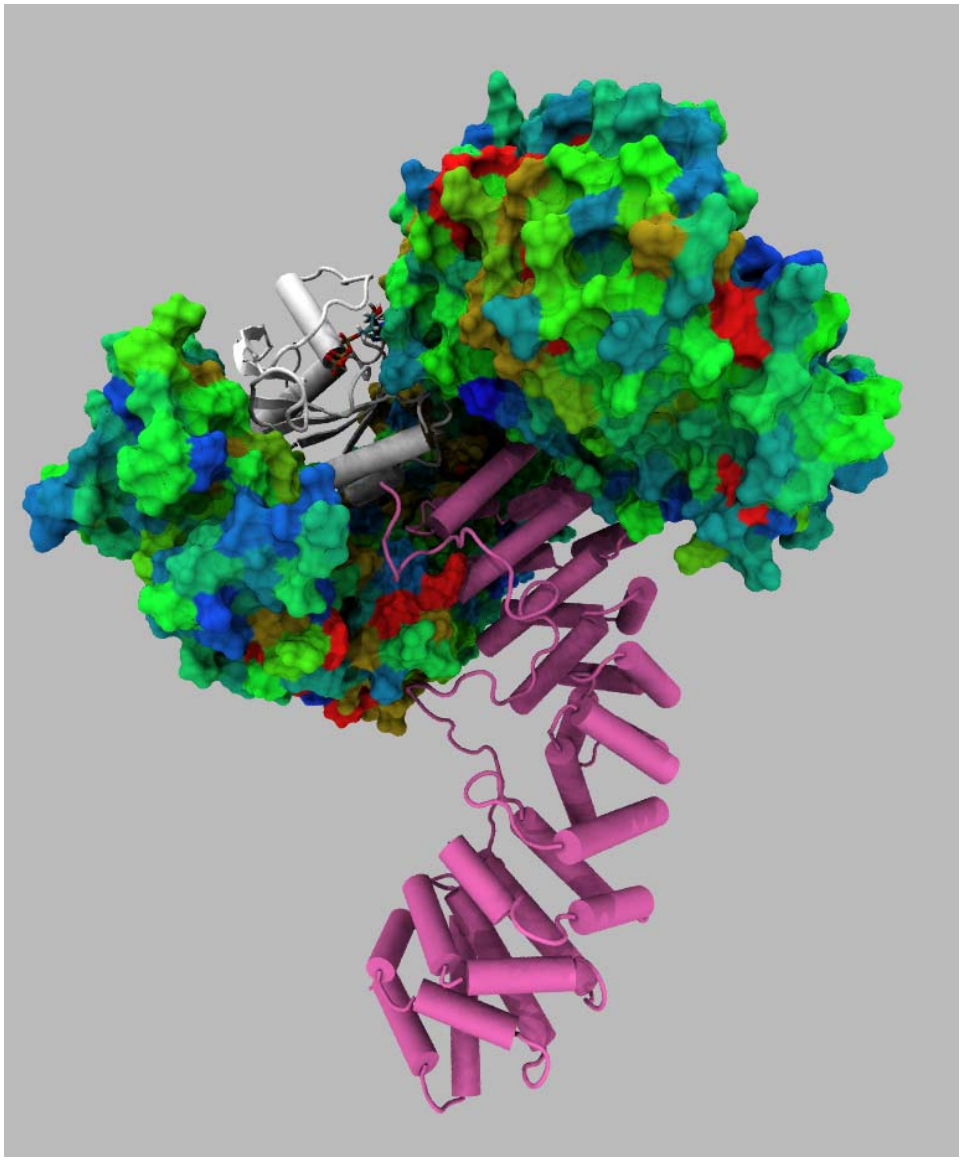




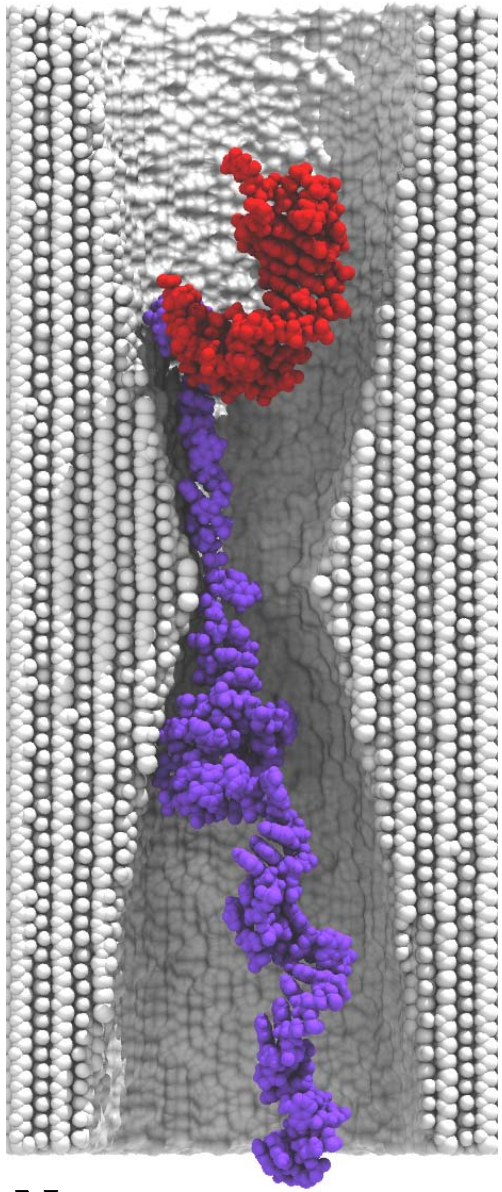




National Center for Research Resources



Exportin Cse1p



Nanopore

Interactive Display with AO and Other Demanding Techniques

- Ultimately the user wants to be able to display their structures at interactive rates
- Ambient occlusion lighting computationally costly:
  - Usable on very small static molecular structures, with some limitations
  - Published GPU algorithms don’t scale well for large all-atom trajectories yet
  - Future GPUs and improved algorithms will likely make interactive AO usable for large structures in a few years

Upcoming Challenges

- Petascale simulations will generate trajectories too large to download from the supercomputers
- Much more analysis will have to be done prior to visualization of the results, to help focus on the interesting data