INTEL® HPC DEVELOPER CONFERENCE

Visualization and Analysis of Biomolecular Complexes on Upcoming KNL-based HPC Systems

John E. Stone

Theoretical and Computational Biophysics Group

Beckman Institute for Advanced Science and Technology

University of Illinois at Urbana-Champaign

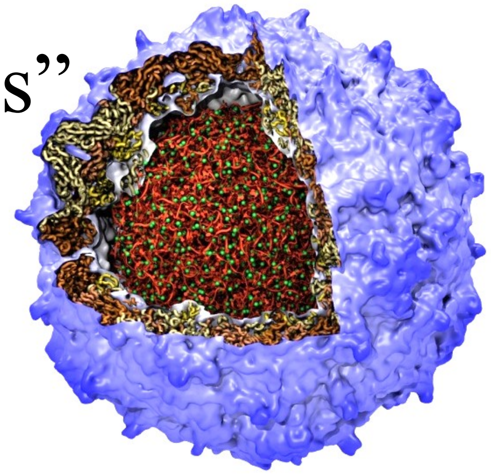
http://www.ks.uiuc.edu/Research/vmd/

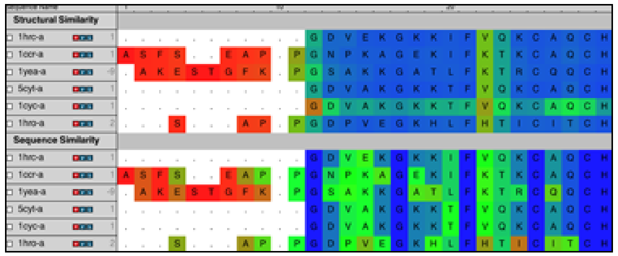
Intel HPC Developer Conference, Sheraton Hotel

Sunday, Nov 13th, 2016, Salt Lake City, UT

VMD – “Visual Molecular Dynamics”

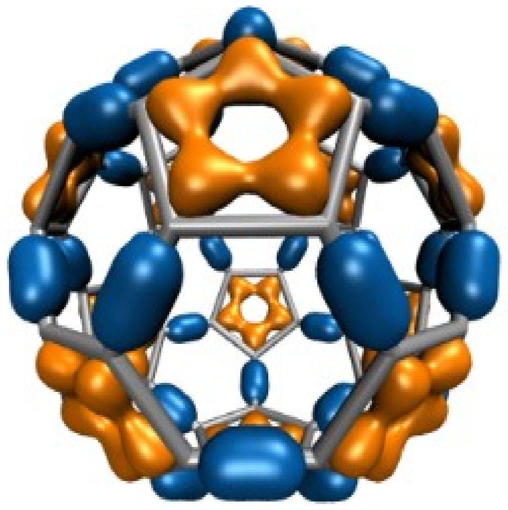
- Visualization and analysis of:
  - molecular dynamics simulations
  - quantum chemistry calculations
  - particle systems and whole cells
  - sequence data
- User extensible w/ scripting and plugins
- http://www.ks.uiuc.edu/Research/vmd/

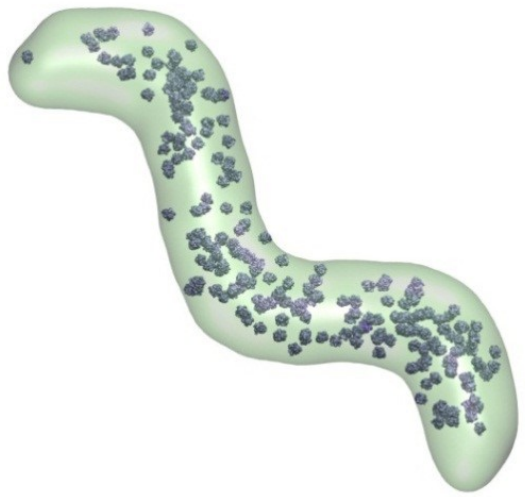

Poliovirus

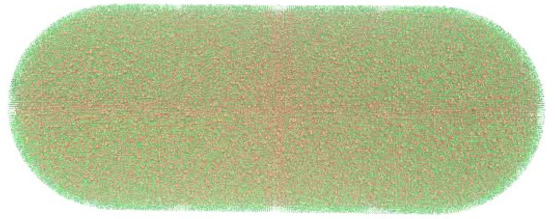


```
Sequence Name     1                   10                  20
Structural Similarity
1hrc-a    1   . . . . . . . . . . . G D V E K G K K I F V Q K C A Q C H
1ccr-a    1   A S F S . . E A P . P G N P K A G E K I F K T K C A Q C H
1yea-a   -9   . A K E S T G F K . P G S A K K G A T L F K T R C Q Q C H
5cyt-a    1   . . . . . . . . . . . G D V A K G K K T F V Q K C A Q C H
1cyc-a    1   . . . . . . . . . . . G D V A K G K K T F V Q K C A Q C H
1hro-a    2   . . . S . . . A P . P G D P V E G K H L F H T I C I T C H
Sequence Similarity
1hrc-a    1   . . . . . . . . . . . G D V E K G K K I F V Q K C A Q C H
1ccr-a    1   A S F S . . E A P . P G N P K A G E K I F K T K C A Q C H
1yea-a   -9   . A K E S T G F K . P G S A K K G A T L F K T R C Q Q C H
5cyt-a    1   . . . . . . . . . . . G D V A K G K K T F V Q K C A Q C H
1cyc-a    1   . . . . . . . . . . . G D V A K G K K T F V Q K C A Q C H
1hro-a    2   . . . S . . . A P . P G D P V E G K H L F H T I C I T C H
```

Ribosome Sequences


Electrons in
Vibrating Buckyball

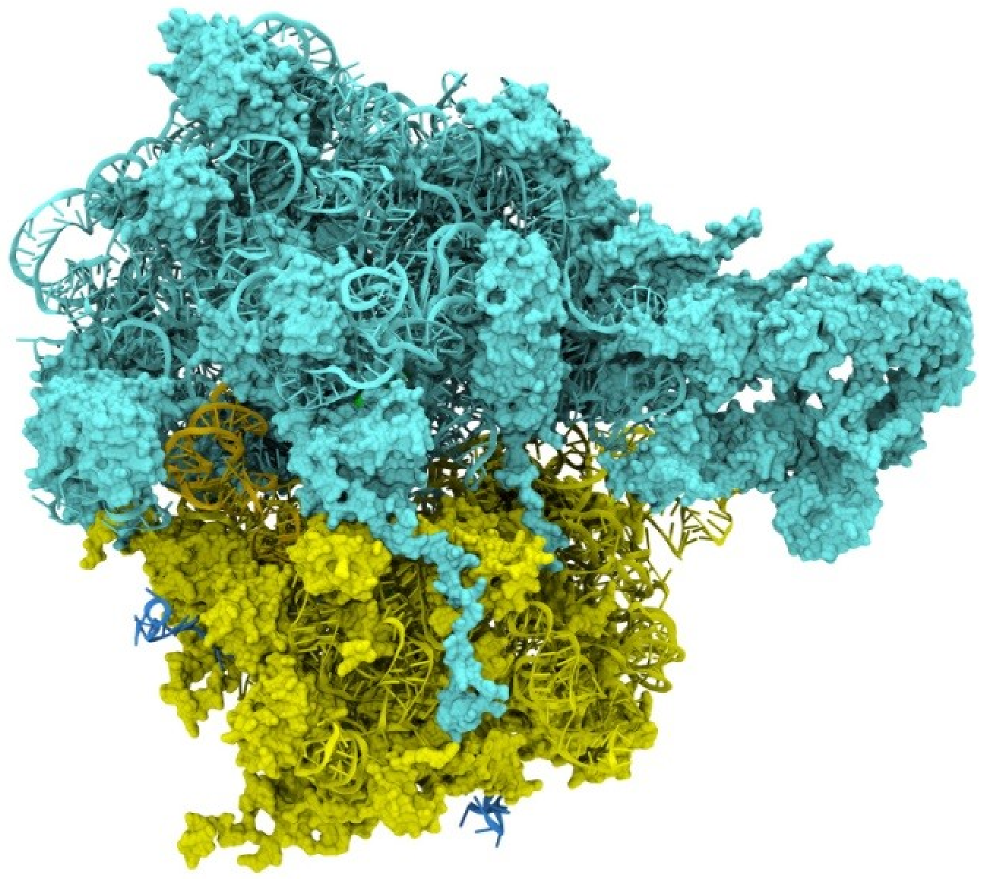

Cellular Tomography
Cryo-electron Microscopy


Whole Cell Simulations

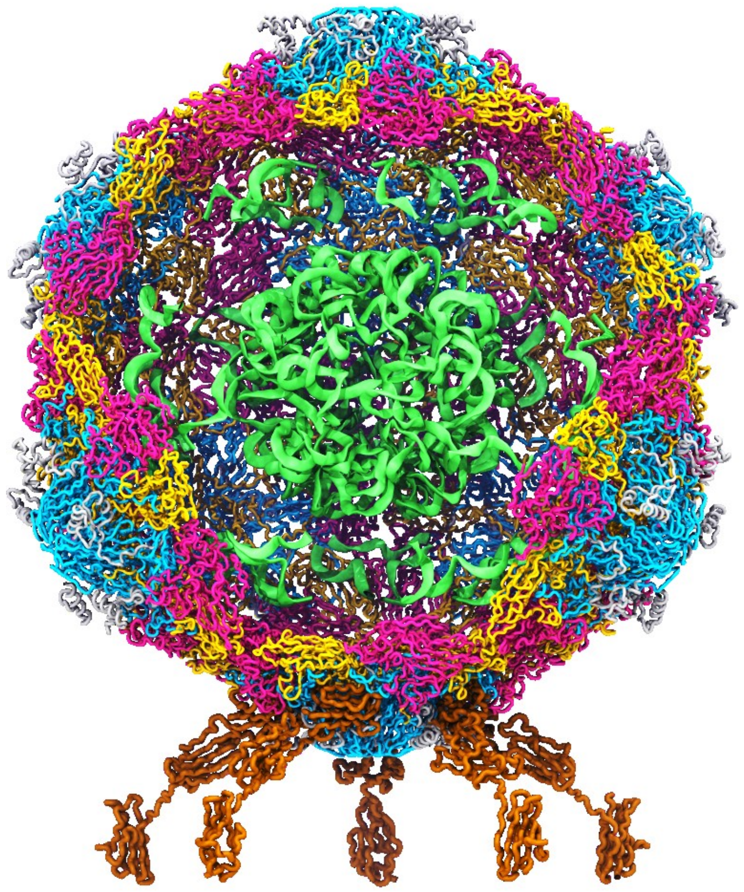
Goal: A Computational Microscope

Study the molecular machines in living cells

Ribosome: target for antibiotics



Poliovirus



VMD 1.9.3 on New HPC Systems: TACC Stampede-2 and ANL Theta

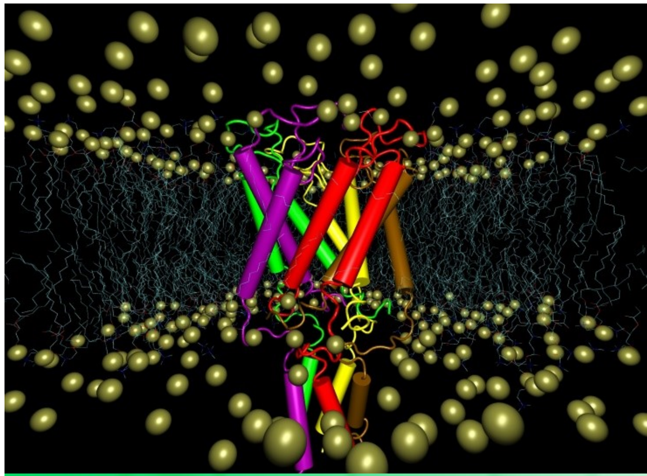
- Enable large scale analysis and viz. tasks in-place
- Challenge: adaptations for MIC architecture
- Approach:
  - Incorporate OSPRay for ray tracing on MIC
  - Change CPU threading for large core counts
  - MIC optimizations for key viz/analysis kernels

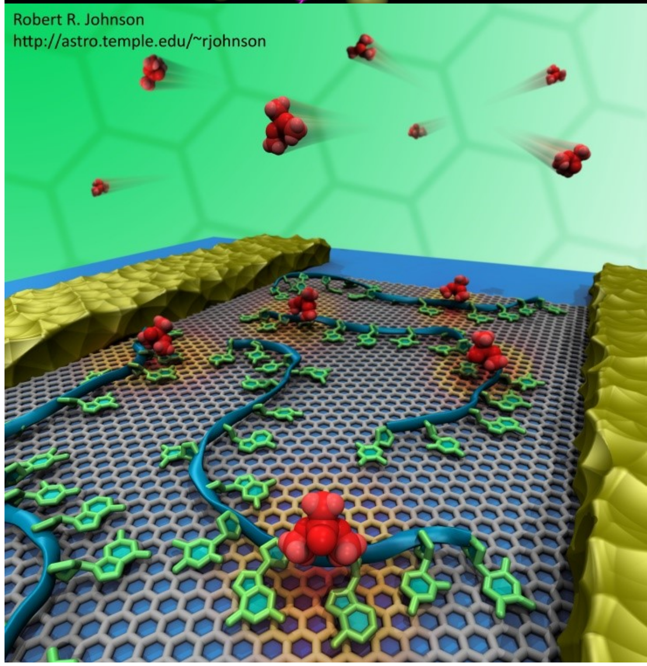
NSF: TACC Stampede-2

DOE: Argonne Theta

Ray Tracing in VMD

- Support for ray tracing of VMD molecular scenes began in1995
- Tachyon parallel RT engine interfaced with VMD (1999)
- Tachyon embedded as an internal VMD rendering engine (2002)
- Built-in support for large scale parallel rendering (2012)
- Refactoring of VMD to allow fully interactive ray tracing as an alternative to OpenGL (2014)





Tachyon Ray Tracing Engine

- Originally developed on Intel iPSC/860 hypercube (1994)
- First support for MPI (1995)
- Multithreading for Intel Paragon XP/S, large SGI and Sun shared memory machines (1995)
- In-situ CFD visualization (1996)
- Support for OpenMP w/ Kuck and Associates KCC (1998)
- Co-developed w/ VMD, 1998-present

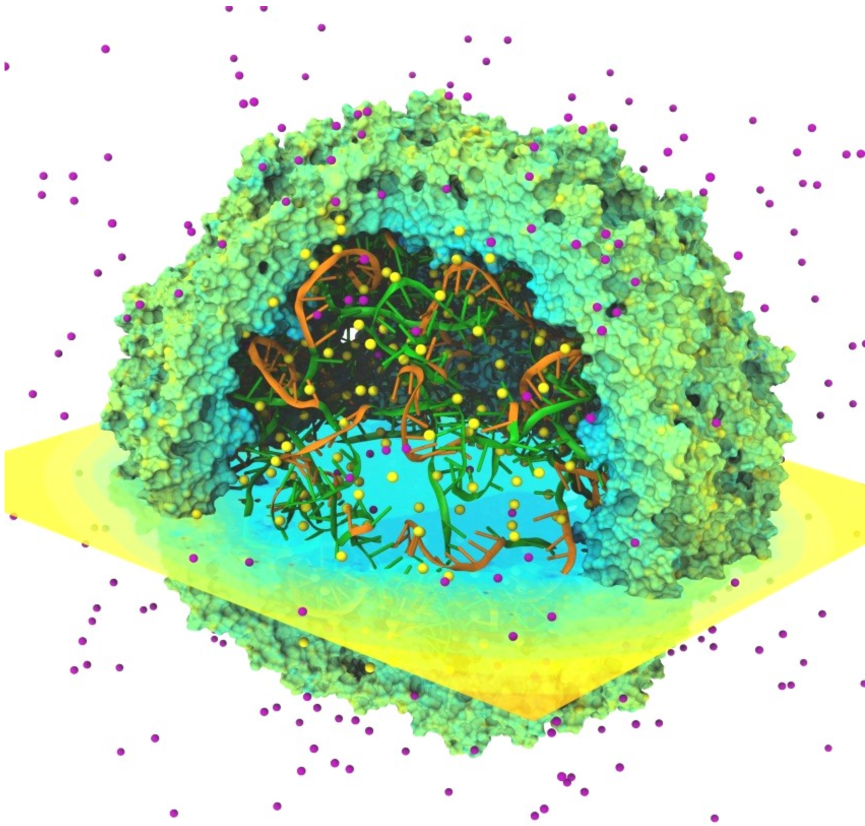
Rendering of Numerical Flow Simulations Using MPI. John Stone and Mark Underwood. Second MPI Developers Conference, pages 138-141, 1996.

An Efficient Library for Parallel Ray Tracing and Animation. John E. Stone Master's Thesis, University of Missouri-Rolla, Department of Computer Science, April 1998.

Early Experiences Scaling VMD Molecular Visualization and Analysis Jobs on Blue Waters. John E. Stone, Barry Isralewitz, and Klaus Schulten.. Extreme Scaling Workshop (XSW), pp. 43-50, 2013.

Biomolecular Visualization Challenges

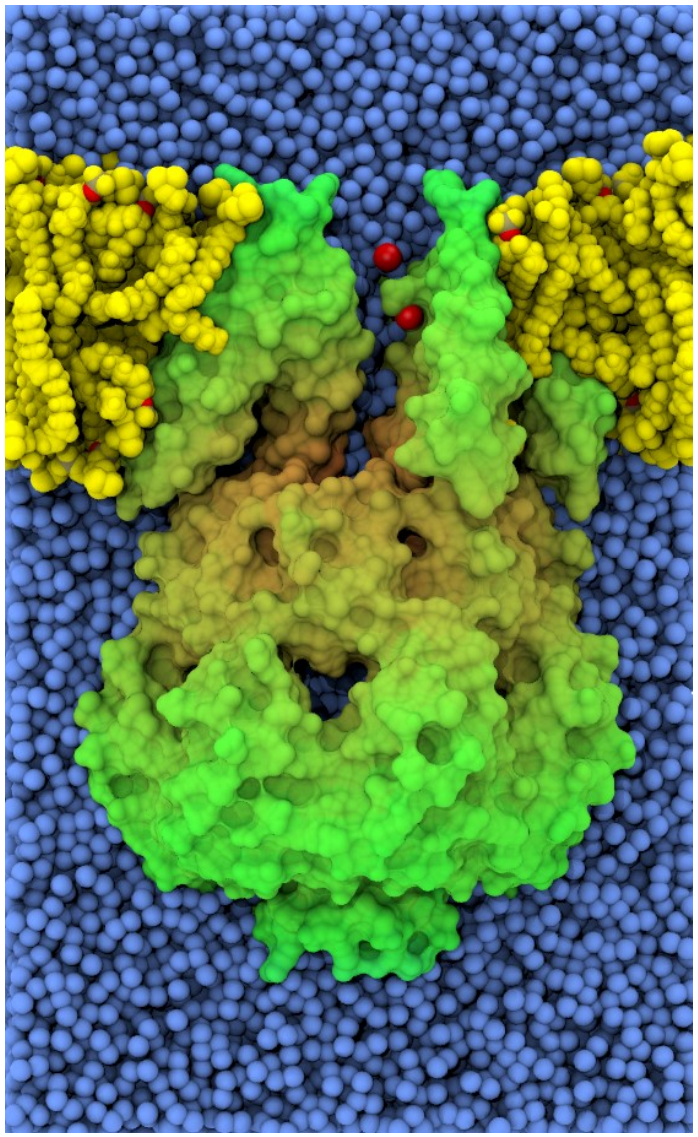
- Geometrically complex scenes
- Spatial relationships important to see clearly: fog, shadows, AO helpful
- Often show a mix of structural and spatial properties
- Time varying!



Geometrically Complex Scenes

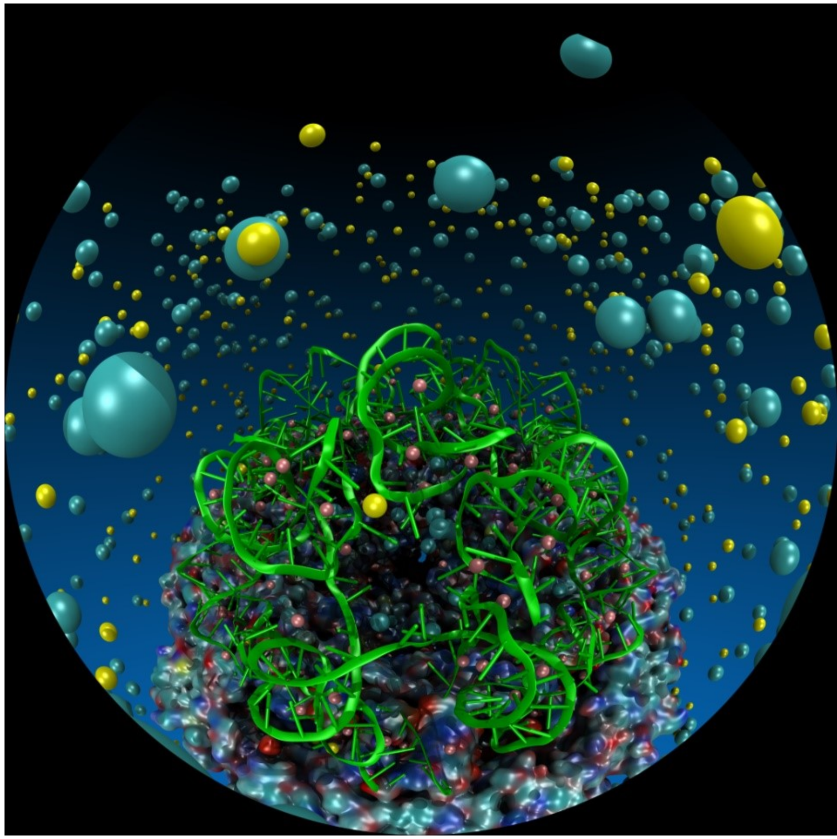
Ray tracing techniques well matched to molecular viz. needs:

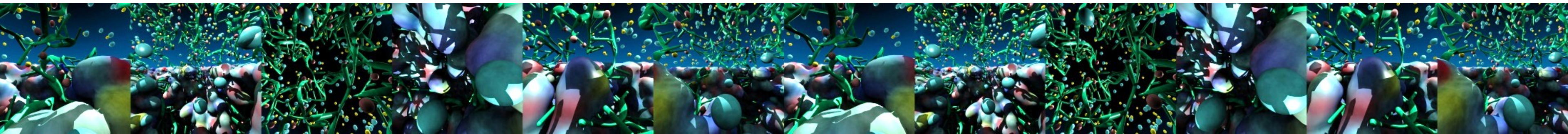
- Curved geometry, e.g. spheres, cylinders, toroidal patches, easily supported
- Greatly reduced memory footprint vs. polygonalization
- Runtime scales only moderately with increasing geometric complexity
- Occlusion culling is "free", RT acceleration algorithms do this and much more



Ray Tracing for Stereoscopic Planetarium Dome Masters, Panoramic Displays

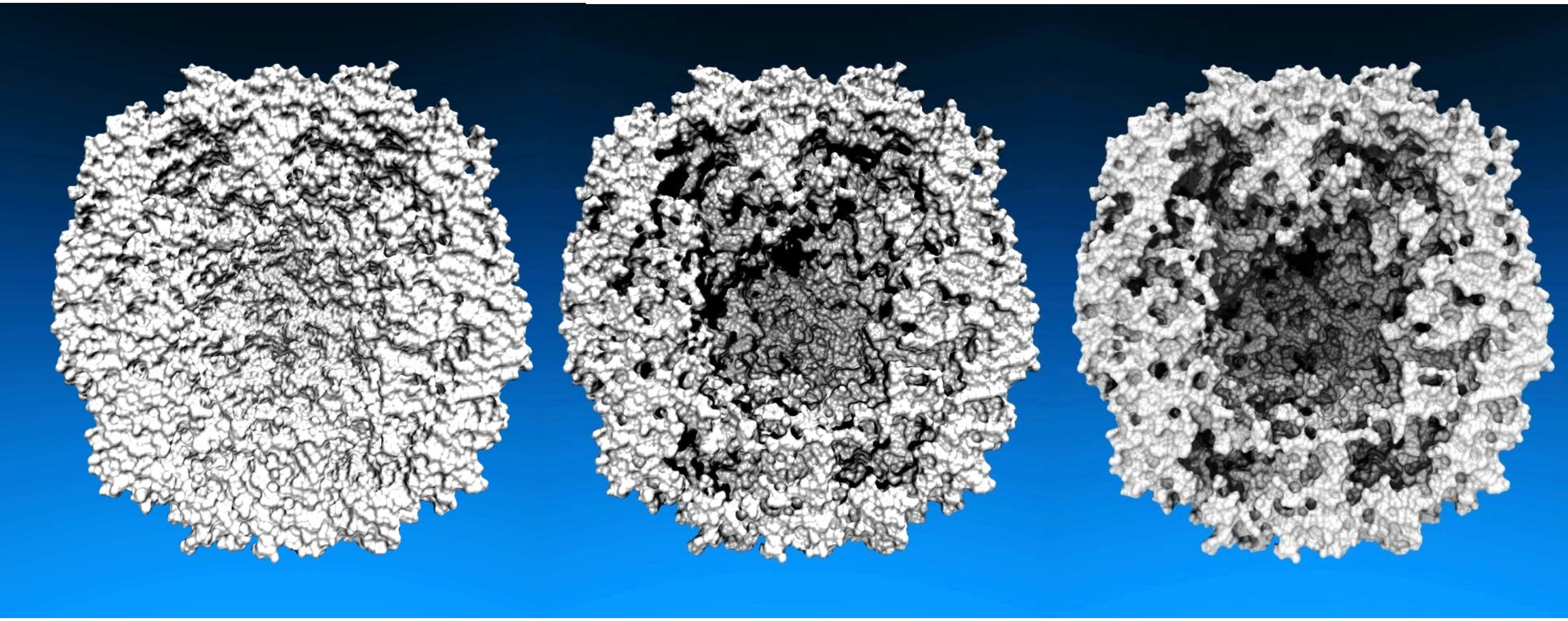
- **RT aptly suited to 360° panoramic rendering**
- **Single-pass rendering** of stereo pairs, spheremaps, cubemaps, planetarium dome masters
- Stereo panoramas require spherical camera projection scheme that is (very) poorly suited to rasterization
- Easy to correct for VR headset lens distortions, e.g. Oculus Rift, Google Cardboard





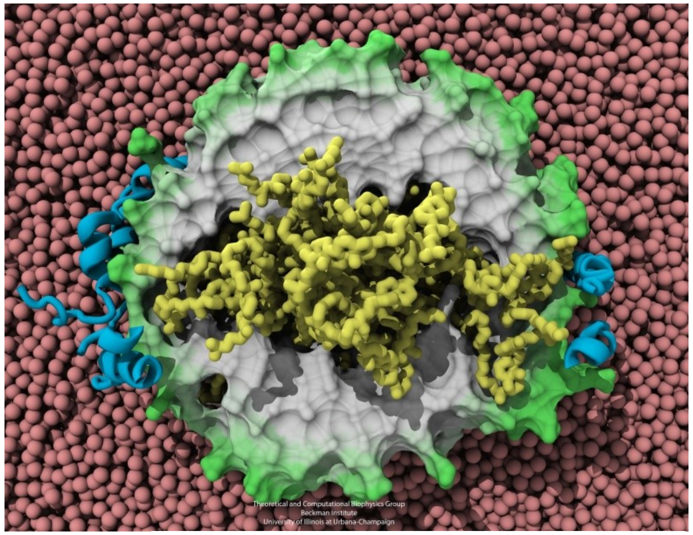
Ray Tracing Naturally Supports Advanced Lighting and Shading Techniques

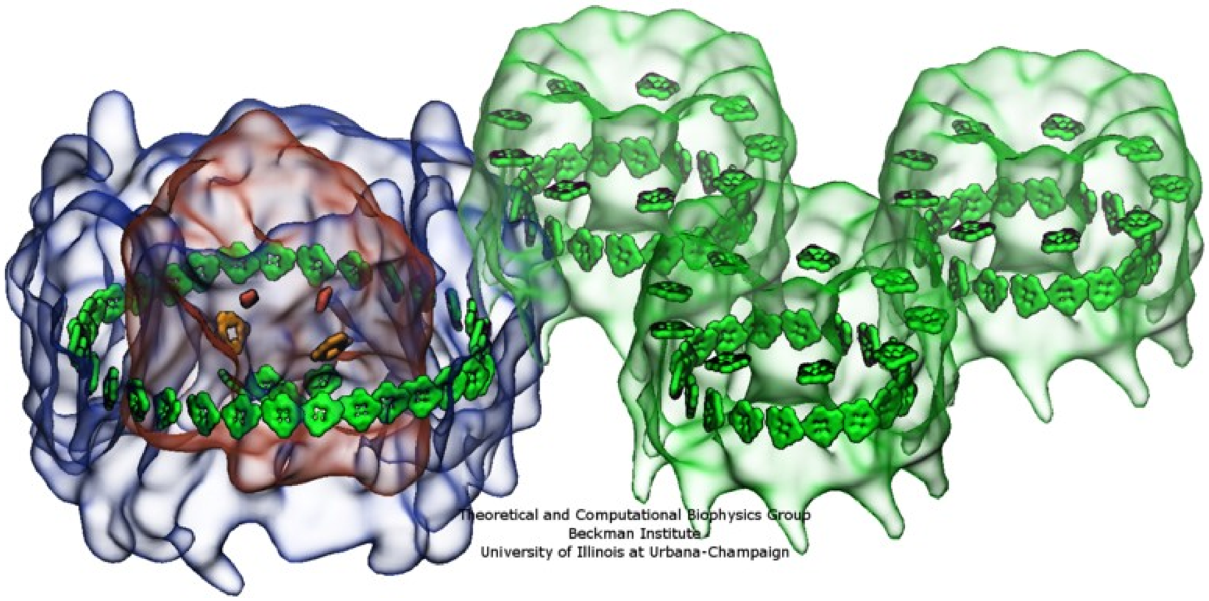
Two lights, no shadows: typical of OpenGL	**Two lights, hard shadows, 1 shadow ray per light**	**Two lights, shadows, ambient occlusion w/ 144 AO rays/hit**

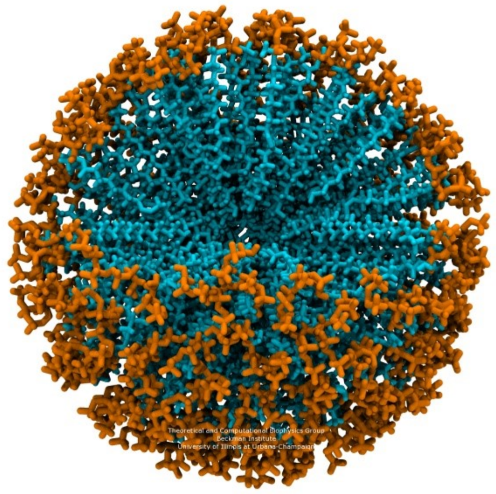


Benefits of Advanced Lighting and Shading Techniques

- Exploit visual intuition
- Spend computer time in exchange for scientists' time, make images that are more easily interpreted

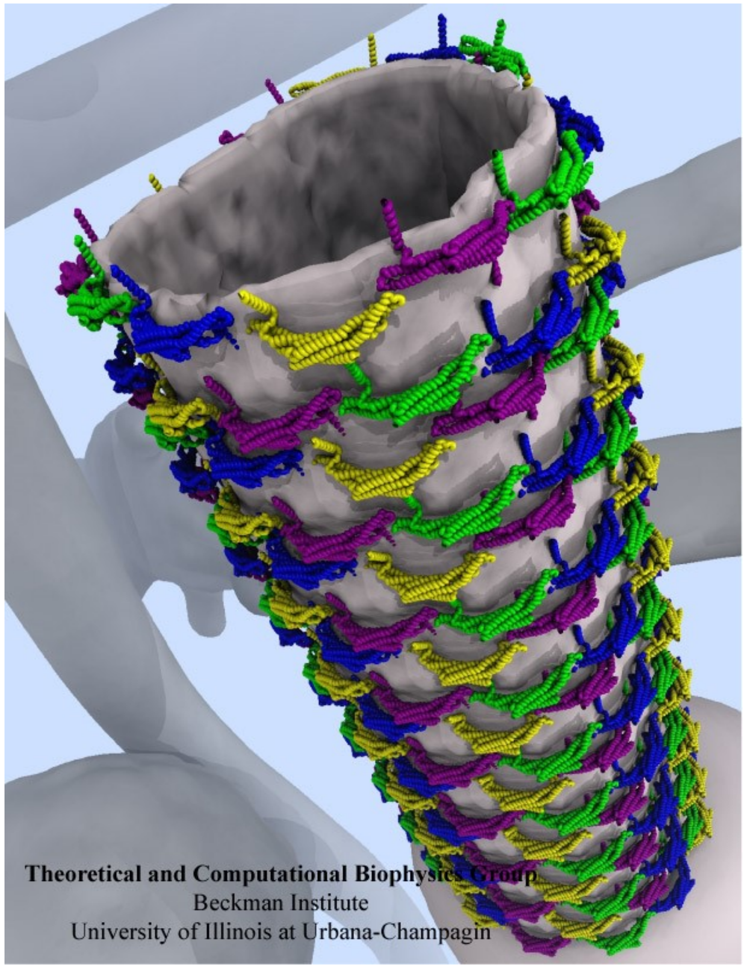

Theoretical and Computational Biophysics Group
Beckman Institute
University of Illinois at Urbana-Champaign

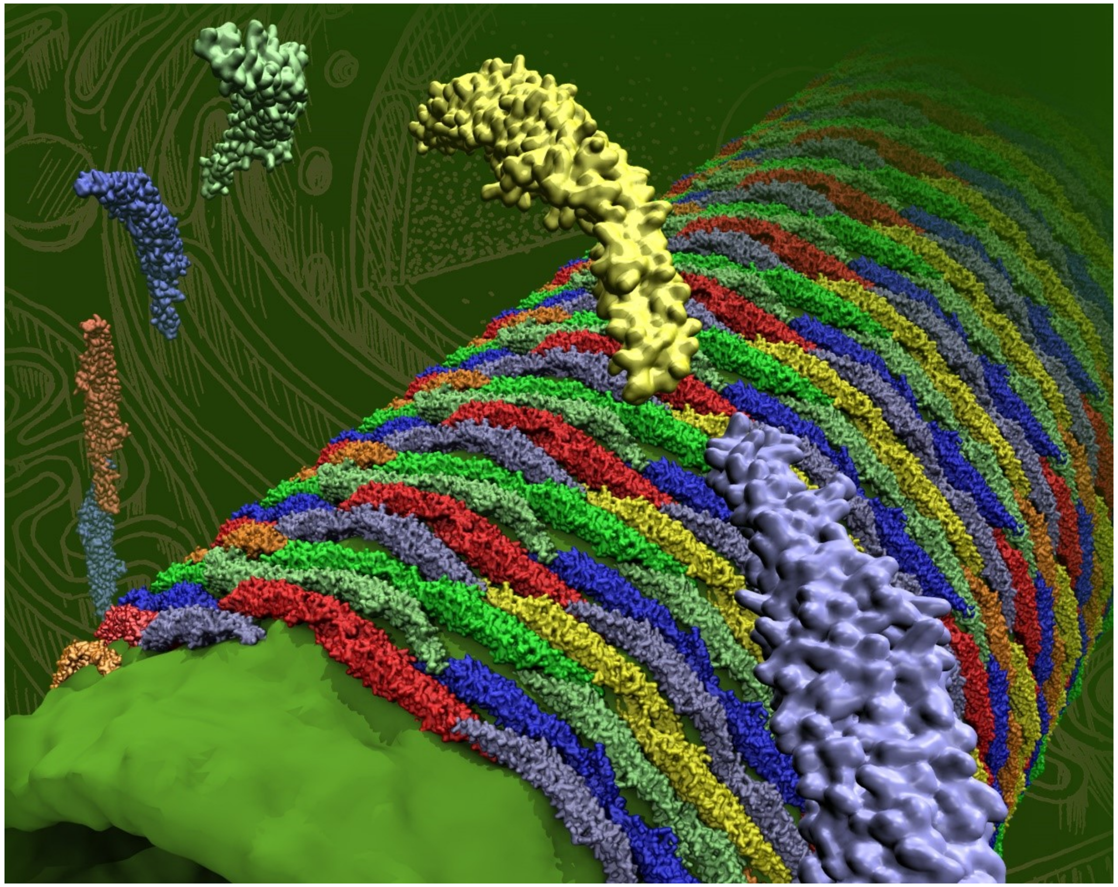

Theoretical and Computational Biophysics Group
Beckman Institute
University of Illinois at Urbana-Champaign


Theoretical and Computational Biophysics Group
Beckman Institute
University of Illinois at Urbana-Champaign

Beckman Institute,
U. Illinois at Urbana-Champaign

Ray Tracing Large Biomolecular Complexes: Large Physical Memory Required (128GB)


Theoretical and Computational Biophysics Group
Beckman Institute
University of Illinois at Urbana-Champagin



Ray Tracing Performance

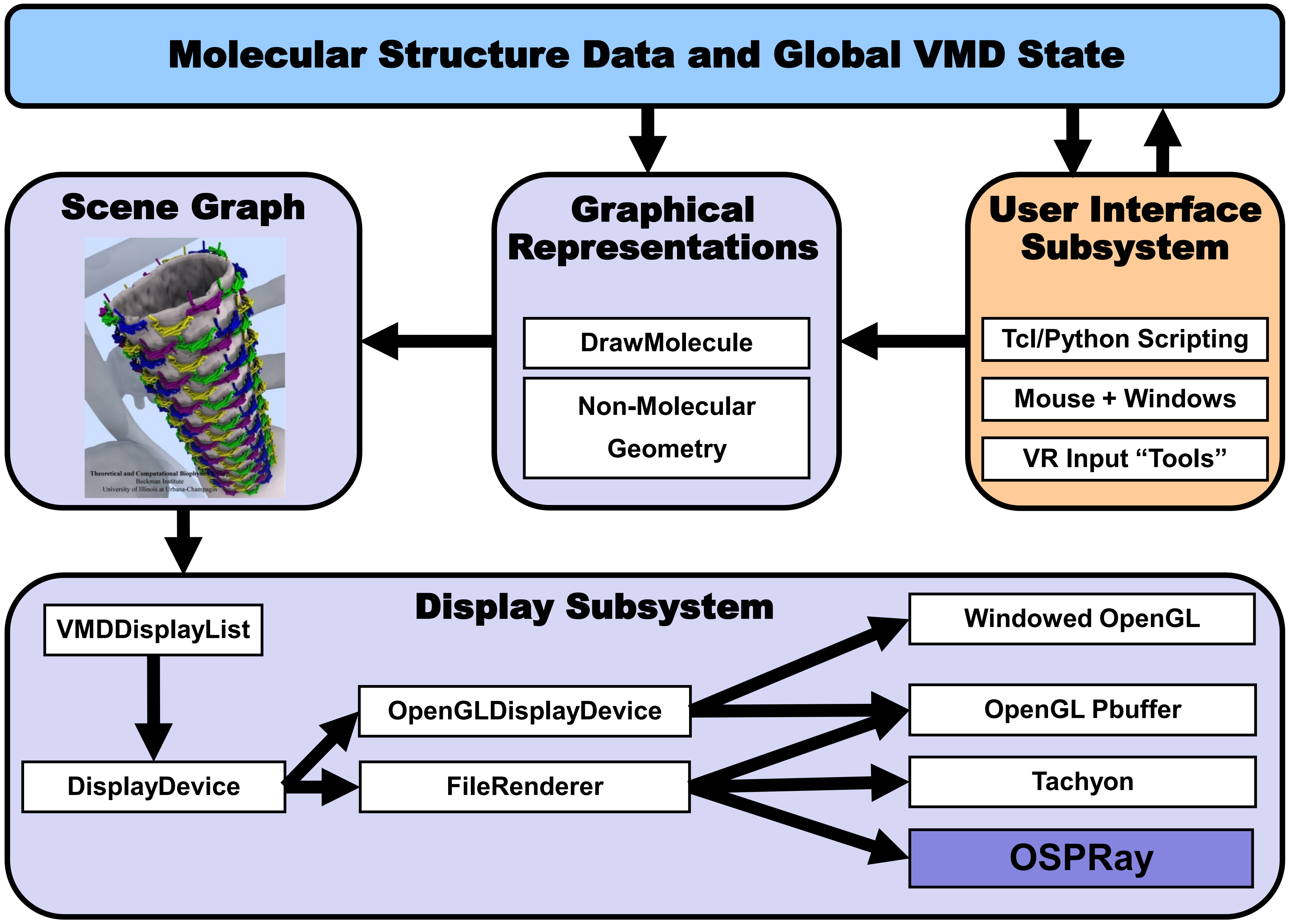
- Well suited to massively parallel hardware
- Peak performance requires full exploitation of SIMD/vectorization, multithreading, efficient use of memory bandwidth
- Traditional languages and compilers not currently up to the task:
  - Efficacy of compiler autovectorization for Tachyon and other classical RT codes **is very low…**
  - Core ray tracing kernels have to be explicitly designed for the target hardware, SIMD, etc.

Fast Ray Tracing Frameworks

- Applications focus on higher level RT ops
- SPMD-oriented languages and compilers address the shortcomings of traditional tools
- Intel RT frameworks provide performance-critical algorithms on IA hardware:
  - Embree: triangles only, basic kernels
  - OSPRay: general RT framework, includes complete renderer implementations

Initial OSPRay support in VMD

- Support researchers with allocations at supercomputer centers with machines based on Knights Landing or Intel® Xeon® processors
- OSPRay functionality general enough for rendering requirements of the majority of VMD scenes
  - Initial VMD-OSPRay development uses general purpose OSPRay renderers not specific to VMD
  - OSPRay built-in renderers could be used by any visualization tool
  - VMD compensates for currently-unimplemented geometry types and mesh formats through automatic internal conversion where possible

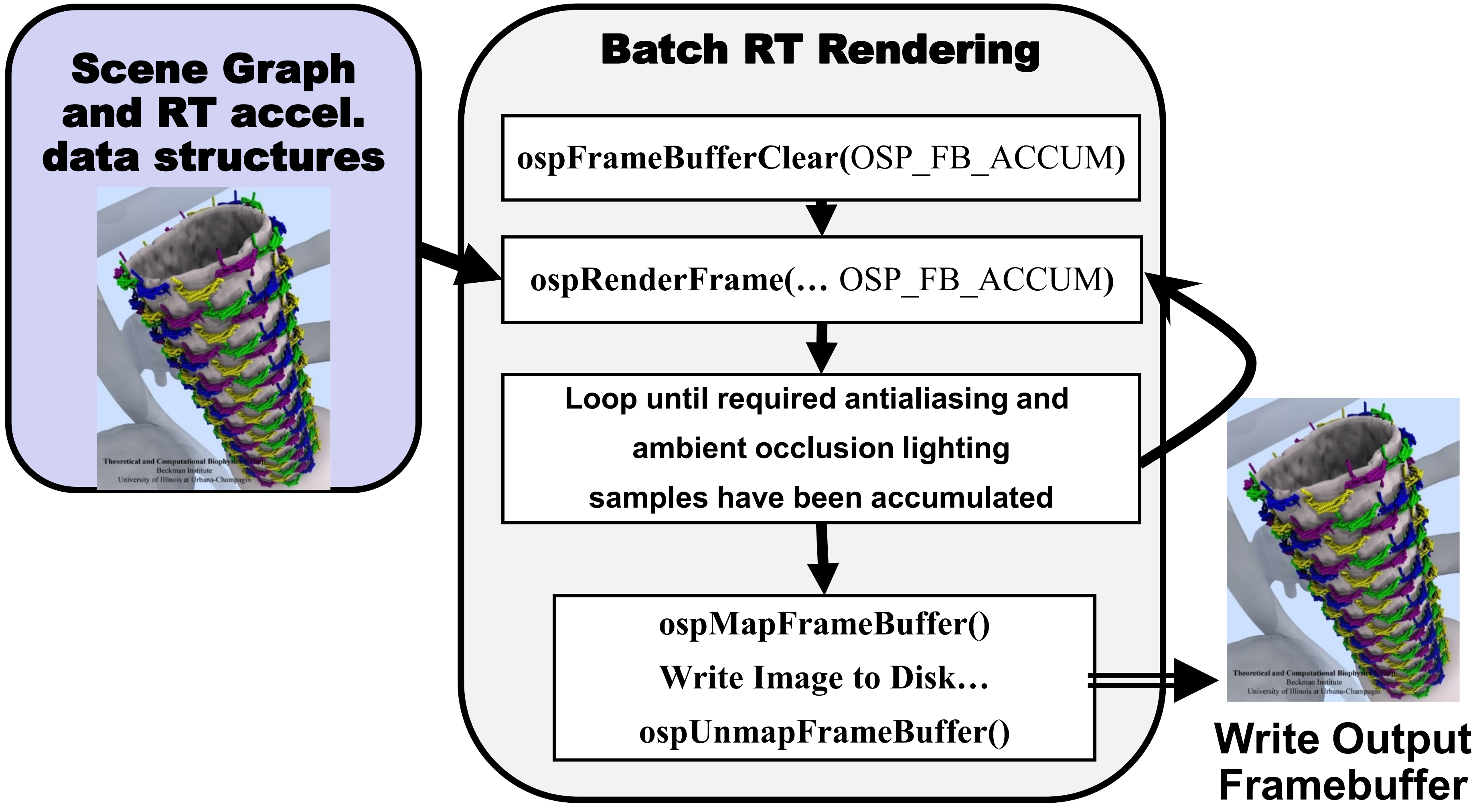

Molecular Structure Data and Global VMD State
Scene Graph
Theoretical and Computational Biophysics Group
Beckman Institute
University of Illinois at Urbana-Champagin
Graphical Representations
DrawMolecule
Non-Molecular Geometry
User Interface Subsystem
Tcl/Python Scripting
Mouse + Windows
VR Input "Tools"
Display Subsystem
VMDDisplayList
DisplayDevice
OpenGLDisplayDevice
FileRenderer
Windowed OpenGL
OpenGL Pbuffer
Tachyon
OSPRay

VMD Scene Graph in OSPRay

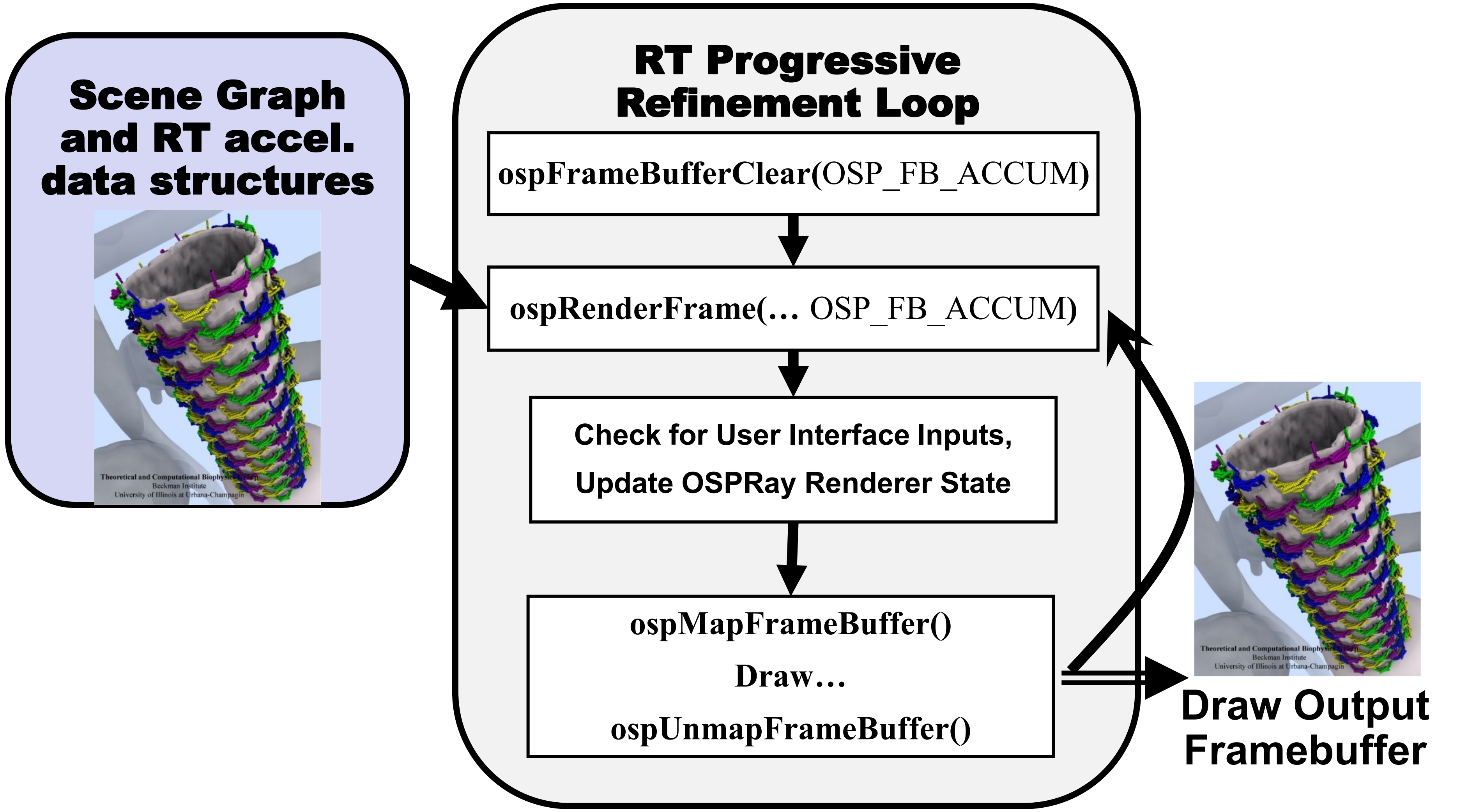
- VMD 1.9.3: On-the-fly scene graph conversion:
  - VMD flattens internal scene graph
  - Transforms geom. to eye space
  - Maps to native OSPRay geom. and materials
- Ongoing work:
  - Many opportunities for reduction of memory footprint, avoiding data layout reformatting
  - Achieving closer or identical shading where possible
  - Streamlining implementation

VMD-OSPRay Offline/Batch Mode

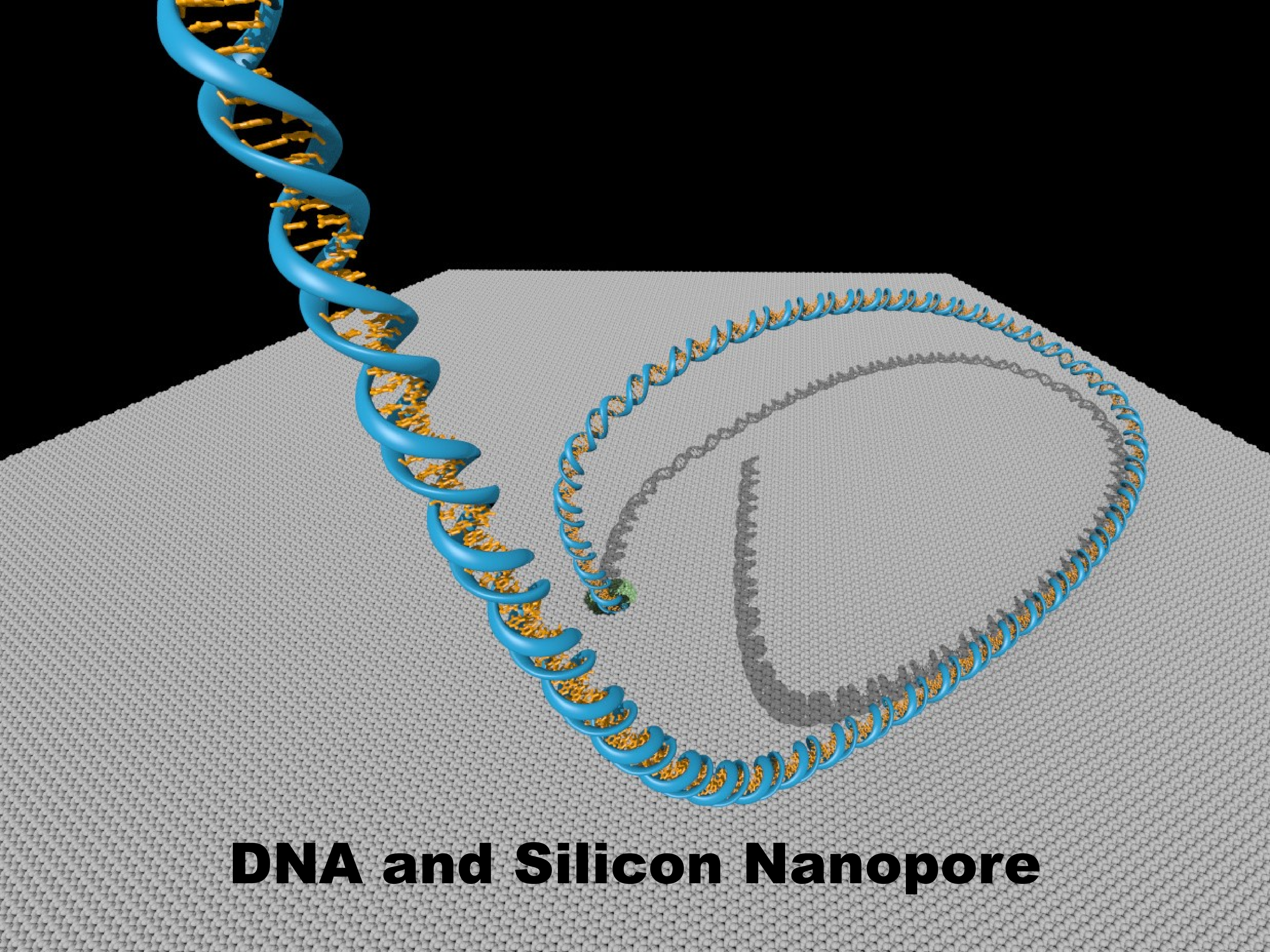
Ray Tracing Loop


Scene Graph and RT accel. data structures
Batch RT Rendering
ospFrameBufferClear(OSP_FB_ACCUM)
ospRenderFrame(… OSP_FB_ACCUM)
Loop until required antialiasing and ambient occlusion lighting samples have been accumulated
ospMapFrameBuffer()
Write Image to Disk…
ospUnmapFrameBuffer()
Write Output Framebuffer
Theoretical and Computational Biophysics Group
Beckman Institute
University of Illinois at Urbana-Champagin

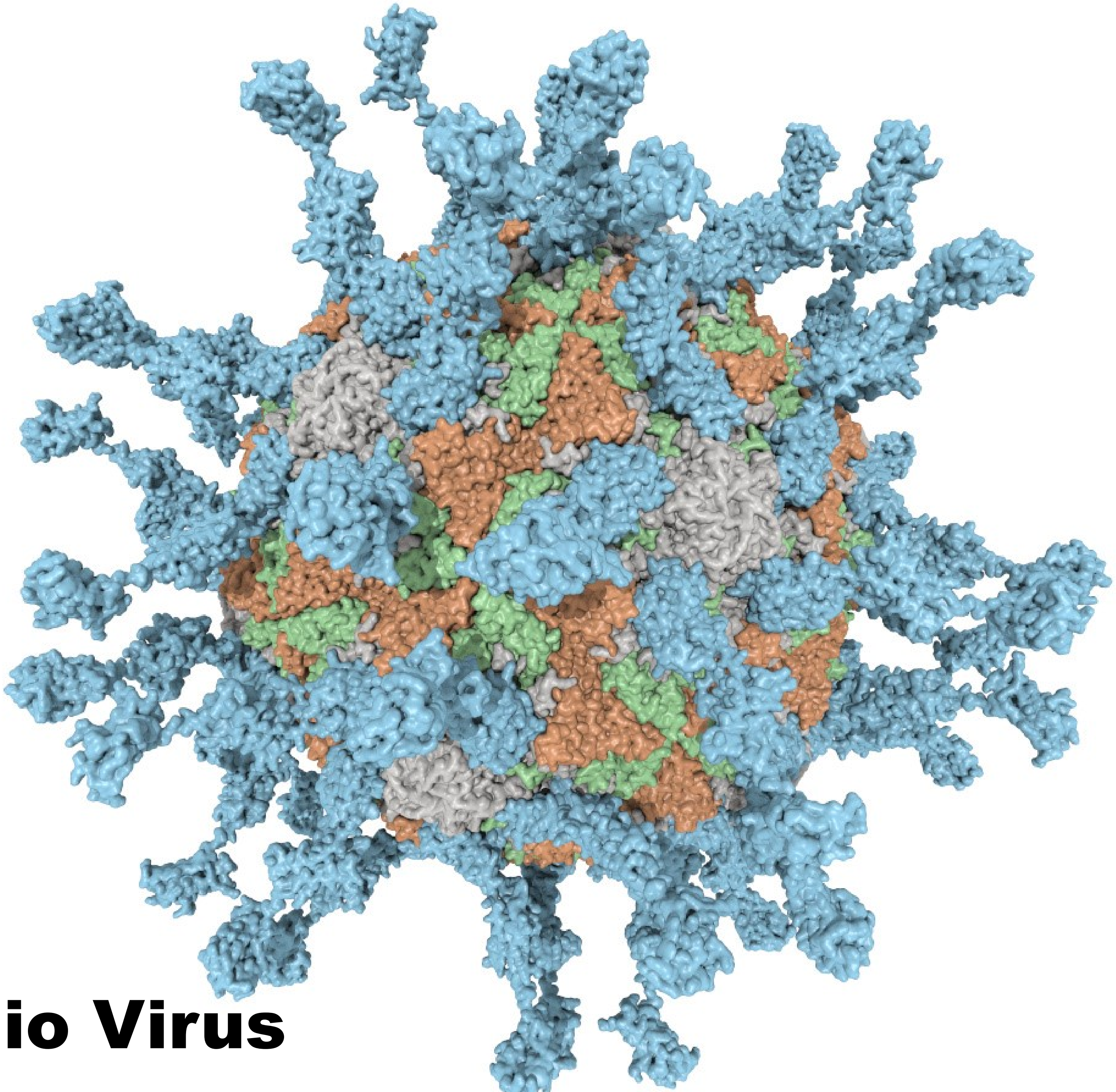
VMD-OSPRay Interactive Ray Tracing with Progressive Refinement


Scene Graph and RT accel. data structures
Theoretical and Computational Biophysics Group
Beckman Institute
University of Illinois at Urbana-Champaign
RT Progressive Refinement Loop
ospFrameBufferClear(OSP_FB_ACCUM)
ospRenderFrame(… OSP_FB_ACCUM)
Check for User Interface Inputs, Update OSPRay Renderer State
ospMapFrameBuffer()
Draw…
ospUnmapFrameBuffer()
Draw Output Framebuffer

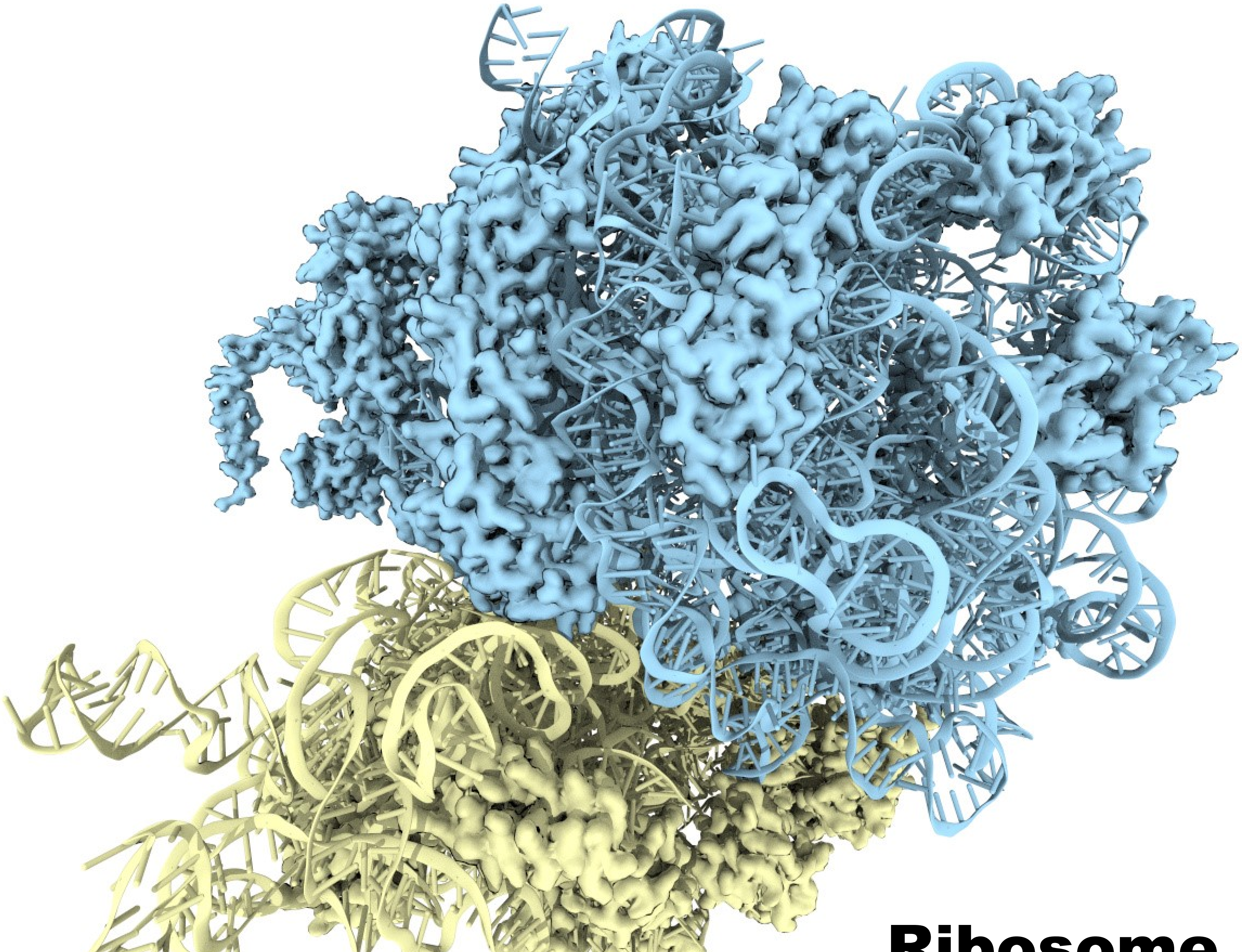
Early OSPRay Renderings with VMD 1.9.3



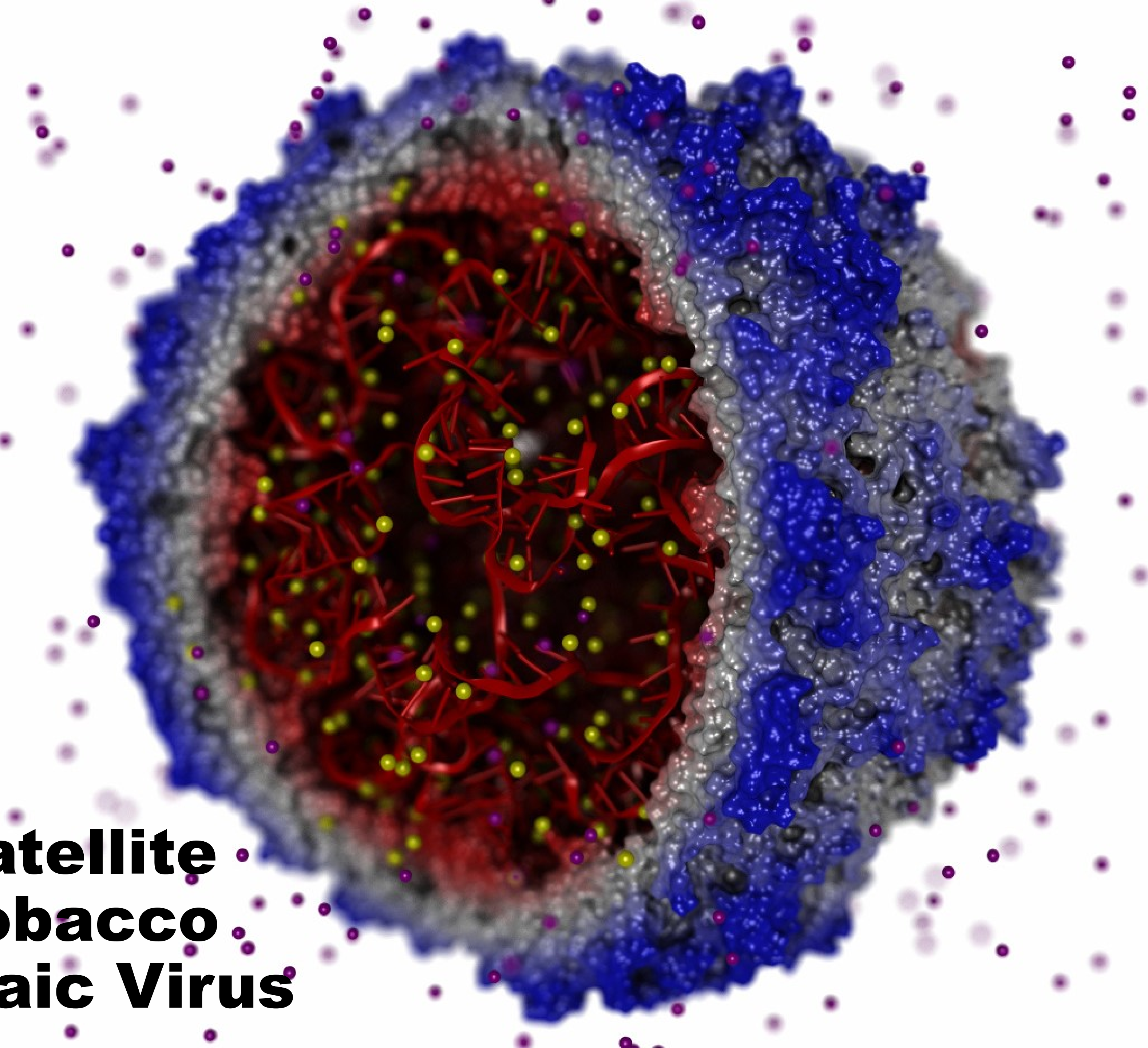
DNA and Silicon Nanopore



Polio Virus

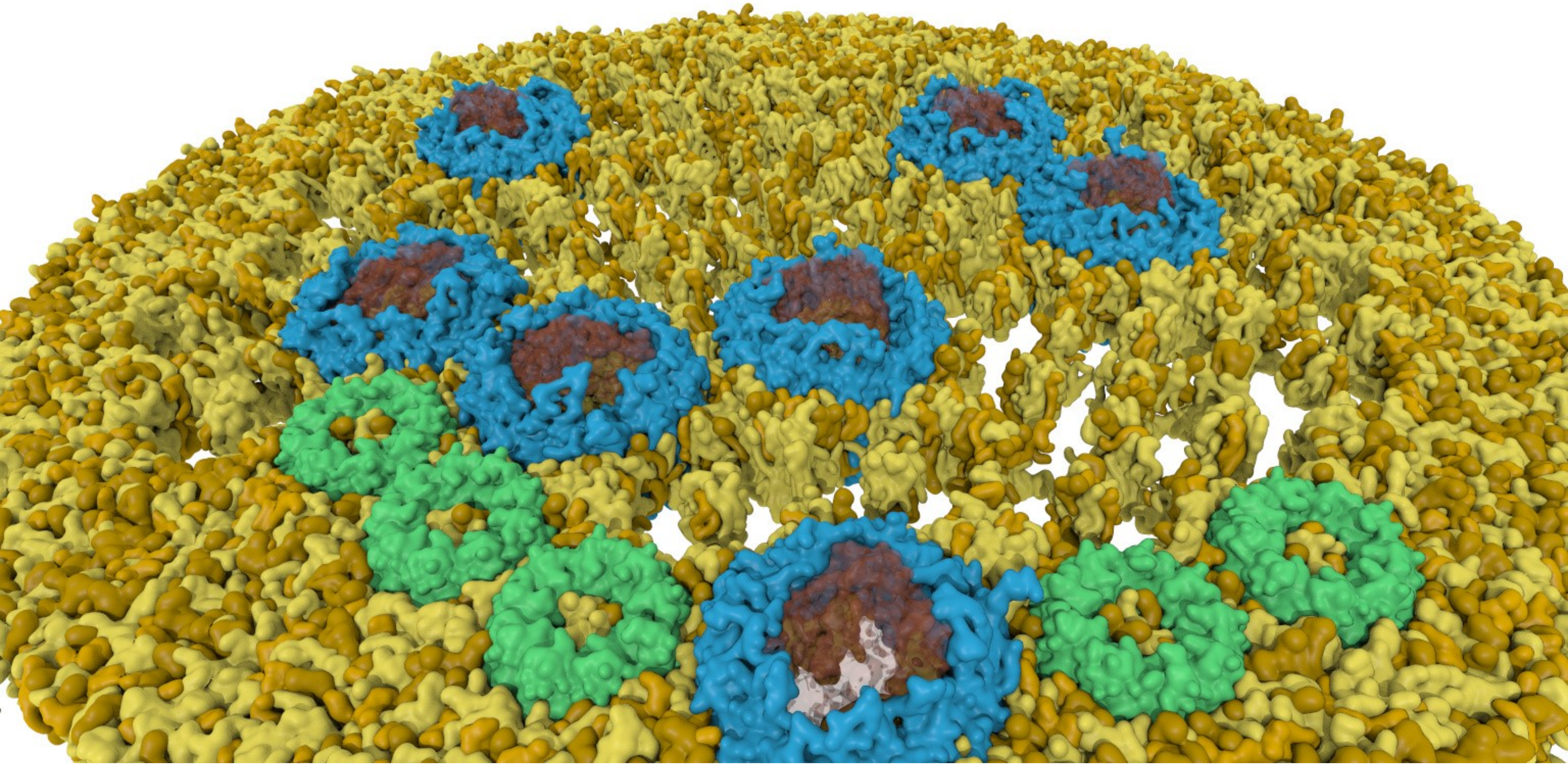


Ribosome



Satellite Tobacco Mosaic Virus

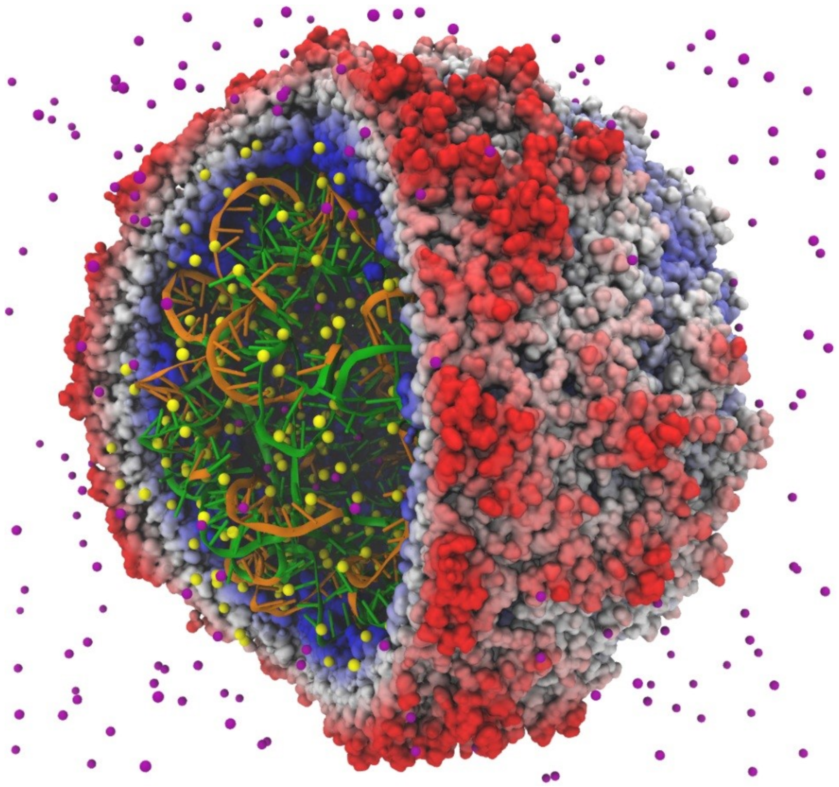
Planar Photosynthetic Membrane Patch



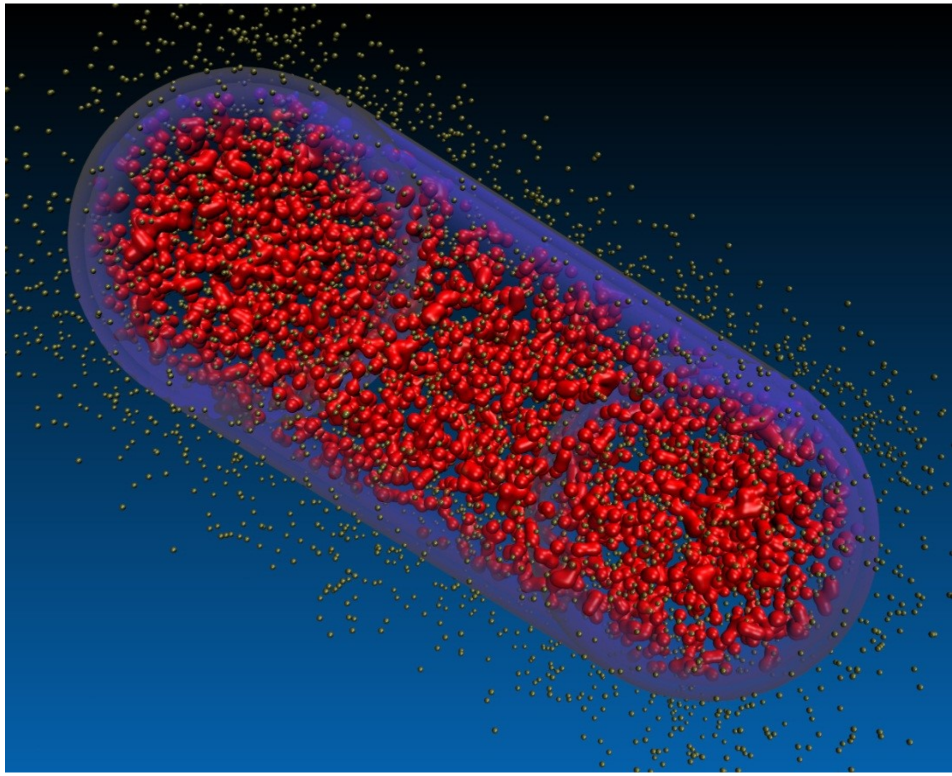
Early AVX-512 Kernels on KNL

VMD “QuickSurf” Representation

- Uses multi-core CPUs and GPU acceleration to enable **smooth real-time animation** of MD trajectories
- Linear-time algorithm, scales to millions of particles, as limited by memory capacity



Satellite Tobacco Mosaic Virus



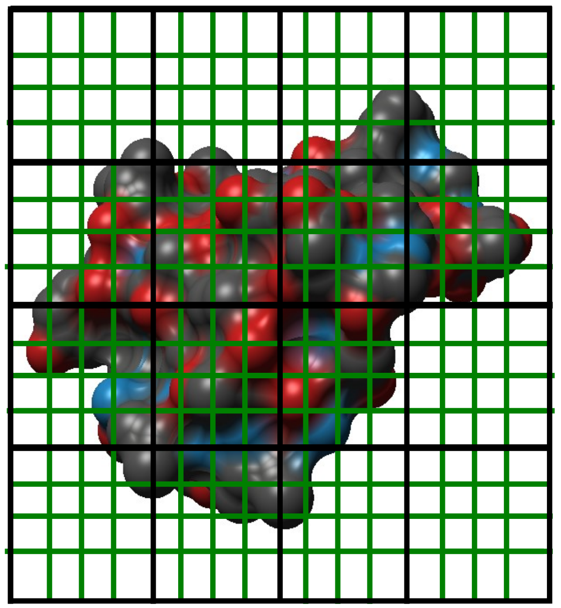
Lattice Cell Simulations

QuickSurf Algorithm Overview

- Build spatial acceleration data structures, optimize data for SIMD, GPUs
- Compute 3-D density map, 3-D volumetric texture map:

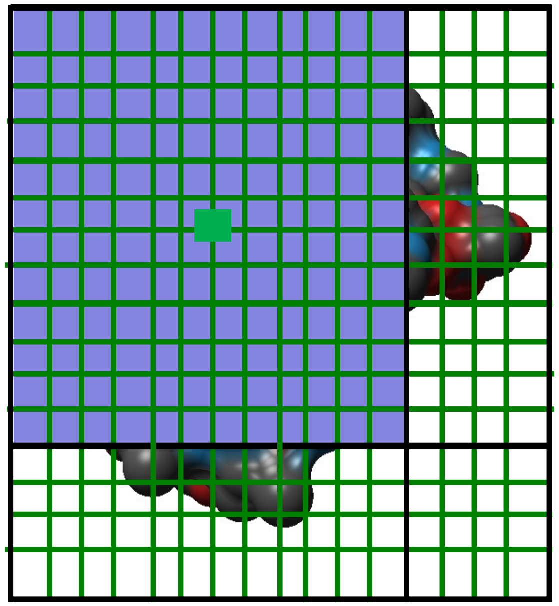
$$\rho(\vec{r}; \vec{r}_1, \vec{r}_2, \ldots, \vec{r}_N) = \sum_{i=1}^{N} e^{\frac{-|\vec{r}-\vec{r}_i|^2}{2\alpha^2}}$$

- Extract isosurface for a user-defined density value


3-D density map lattice, spatial acceleration grid, and extracted surface

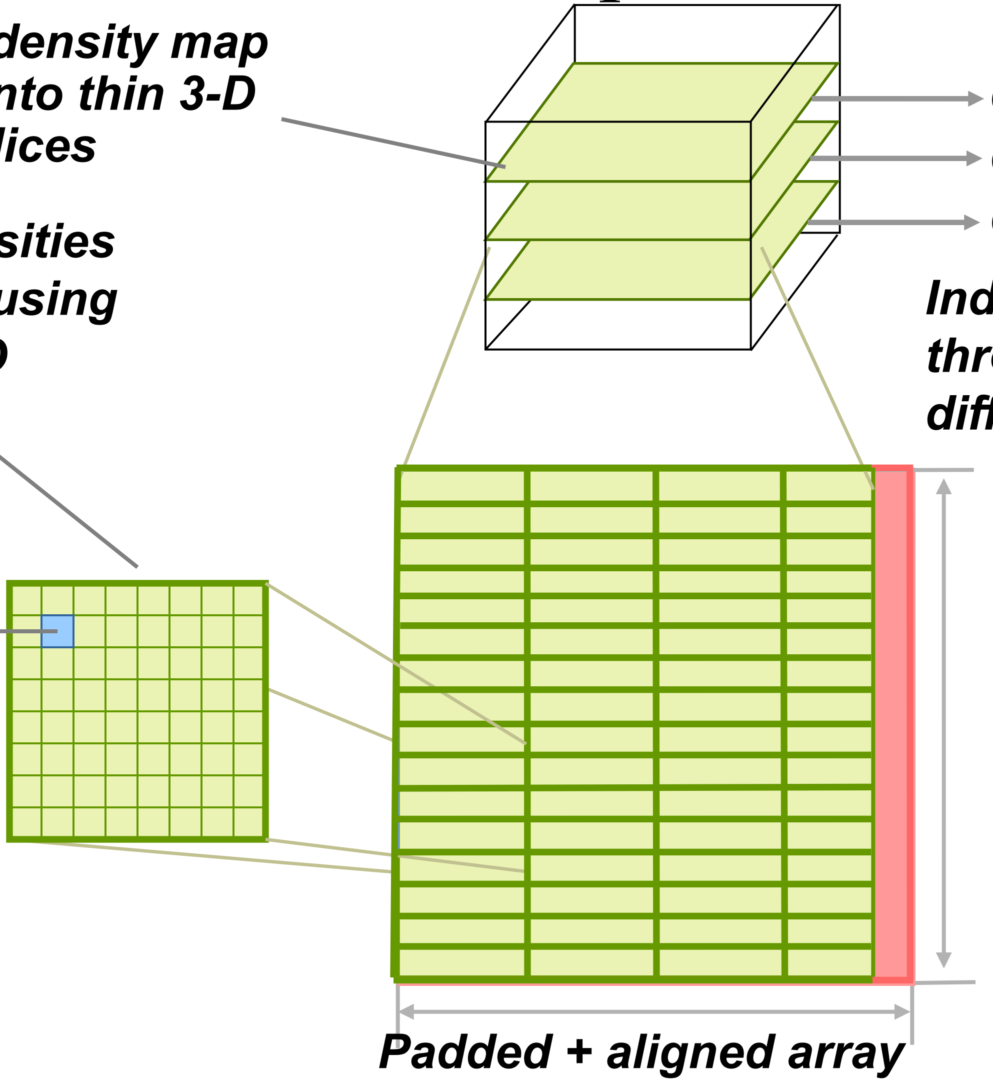
QuickSurf ***Scatter*** Density Map Algorithm

- Existing CPU algorithm targets small 4- to 8-core CPUs, with a ***scatter*** algorithm: each atom loops over a square region surrounding the atom, accumulating densities into grid…
- Output conflicts for independent CPU threads resolved by ***privatization***
- For small CPU thread counts, say 16 or fewer, this approach was great..
- For large (256) CPU thread counts, e.g., KNL, this leaves much to be desired, **but it's a starting point…**



Atom and the neighboring density map lattice points it accumulates into

QuickSurf CPU Density Map Parallel Decomposition



QuickSurf 3-D density map decomposes into thin 3-D slabs/slices

Vectors of densities are computed using hardware SIMD instructions

Independent CPU threads operate on different planes

Each CPU thread computes 1, 4, 8, or 16 density map lattice points per loop iteration: C, SSE, AVX2 or AVX-512ER

SIMD lanes producing results that are used

Padding:

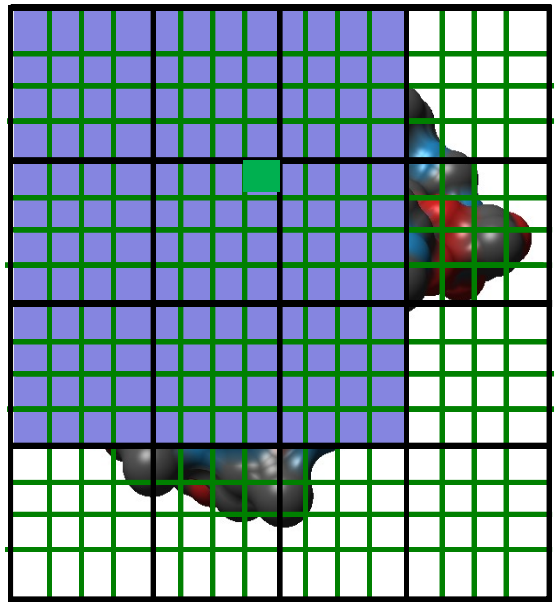
Inactive SIMD lanes or region of discarded output used to guarantee aligned vector loads+stores

QuickSurf ***Scatter*** Loop with AVX512ER on Xeon Phi

```
// Use AVX512ER when we have a multiple-of-16 to compute
__m512 dy2dz2_16 = _mm512_set1_ps(dy2dz2);
__m512 dx_16 = _mm512_add_ps(_mm512_set1_ps(dx), _mm512_load_ps(&sxdelta16[0]));
for (; (x+15)<=xmax; x+=16,dx_16=_mm512_add_ps(dx_16, gridspacing16_16)) {
    __m512 r2 = _mm512_fmadd_ps(dx_16, dx_16, dy2dz2_16);
    // use fast exp2() approximation instruction,  inputs already negated and in base 2
    y = _mm512_exp2a23_ps(_mm512_mul_ps(r2, arinv_16));
    float *ufptr = &densitymap[addr + x];
    d = _mm512_loadu_ps(ufptr);
    _mm512_storeu_ps(ufptr, _mm512_add_ps(d, y));
}
```

QuickSurf *Gather* Density Map Algorithm

- Ongoing work: adapt GPU ***gather*** algorithm for Xeon Phi
- Eliminate need for privatization of large density map grids, scale to much larger thread counts
- Spatial acceleration grid cells are sized to match the cutoff radius for the exponential, beyond which density contributions are negligible
- Density map lattice points computed by summing density contributions from particles in 3x3x3 grid of neighboring spatial acceleration cells

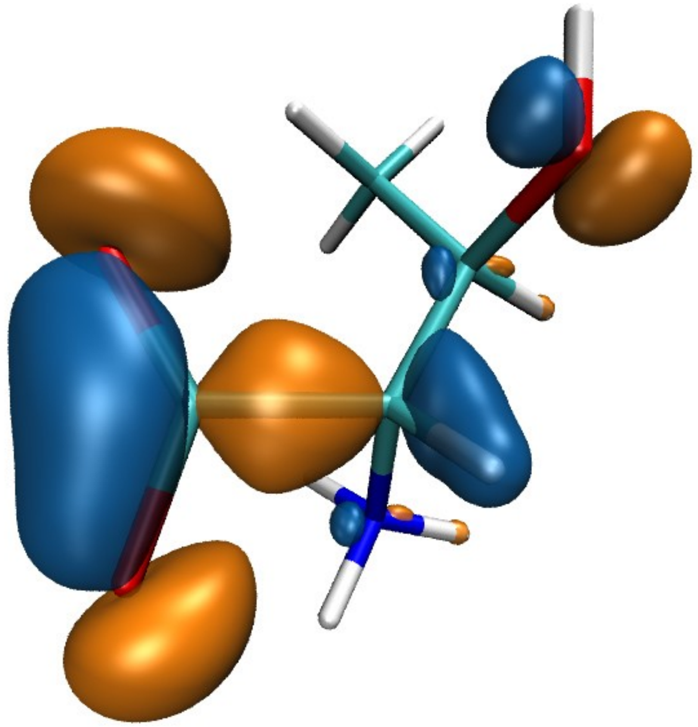

3-D density map lattice point and the neighboring spatial acceleration cells it references

QuickSurf Density Map Kernel Snippet…

```
for (zab=zabmin; zab<=zabmax; zab++) {
  for (yab=yabmin; yab<=yabmax; yab++) {
   for (xab=xabmin; xab<=xabmax; xab++) {
    int abcellidx = zab * acplanesz + yab * acncells.x + xab;
    uint2 atomstartend = cellStartEnd[abcellidx];
    if (atomstartend.x != GRID_CELL_EMPTY) {
     for (unsigned int atomid=atomstartend.x; atomid<atomstartend.y; atomid++) {
      float4 atom = sorted_xyzr[atomid];
      float dx = coorx - atom.x;       float dy = coory - atom.y;      float dz = coorz - atom.z;
      float dxy2 = dx*dx + dy*dy;
      float r21 = (dxy2 + dz*dz) * atom.w;
      densityval1 += exp2f(r21);
       /// Loop unrolling and register tiling benefits begin here……
      float dz2 = dz + gridspacing;
      float r22 = (dxy2 + dz2*dz2) * atom.w;
      densityval2 += exp2f(r22);
      /// More loop unrolling ….
```

Animating Molecular Orbitals

- Animation of (classical mechanics) molecular dynamics trajectories provides insight into simulation results
- To do the same for QM or QM/MM simulations one must compute MOs at ~**10 FPS** or more
- **Wide SIMD hardware with fast exponential instructions** makes this possible (GPUs and Xeon Phi)



High Performance Computation and Interactive Display of Molecular Orbitals on GPUs and Multi-core CPUs. J. Stone, J. Saam, D. Hardy, K. Vandivort, W. Hwu, K. Schulten, *2nd Workshop on General-Purpose Computation on Graphics Processing Units (GPGPU-2), ACM International Conference Proceeding Series*, volume 383, pp. 9-18, 2009.

Molecular Orbital Computation and Display Process

One-time initialization

Initialize Pool of Worker Threads

Read QM simulation log file, trajectory

Preprocess MO coefficient data
eliminate duplicates, sort by type, etc…

For each trj frame, for each MO shown

For current frame and MO index,
retrieve MO wavefunction coefficients

Compute 3-D grid of MO wavefunction amplitudes
Most performance-demanding step

Extract isosurface mesh from 3-D MO grid

Apply user coloring/texturing
and render the resulting surface

MO Kernel for One Grid Point (Naive C)

```
…
for (at=0; at<numatoms; at++) {
  int prim_counter = atom_basis[at];
  calc_distances_to_atom(&atompos[at], &xdist, &ydist, &zdist, &dist2, &xdiv);
  for (contracted_gto=0.0f, shell=0; shell < num_shells_per_atom[at]; shell++) {
    int shell_type = shell_symmetry[shell_counter];
    for (prim=0; prim < num_prim_per_shell[shell_counter];  prim++) {
      float exponent       = basis_array[prim_counter      ];
      float contract_coeff = basis_array[prim_counter + 1];
      contracted_gto += contract_coeff * expf(-exponent*dist2);
      prim_counter += 2;
    }
    for (tmpshell=0.0f, j=0, zdp=1.0f; j<=shell_type; j++, zdp*=zdist) {
      int imax = shell_type - j;
      for (i=0, ydp=1.0f, xdp=pow(xdist, imax); i<=imax; i++, ydp*=ydist, xdp*=xdiv)
        tmpshell += wave_f[ifunc++] * xdp * ydp * zdp;
    }
    value += tmpshell * contracted_gto;
    shell_counter++;
  }
} …..
```

Loop over atoms

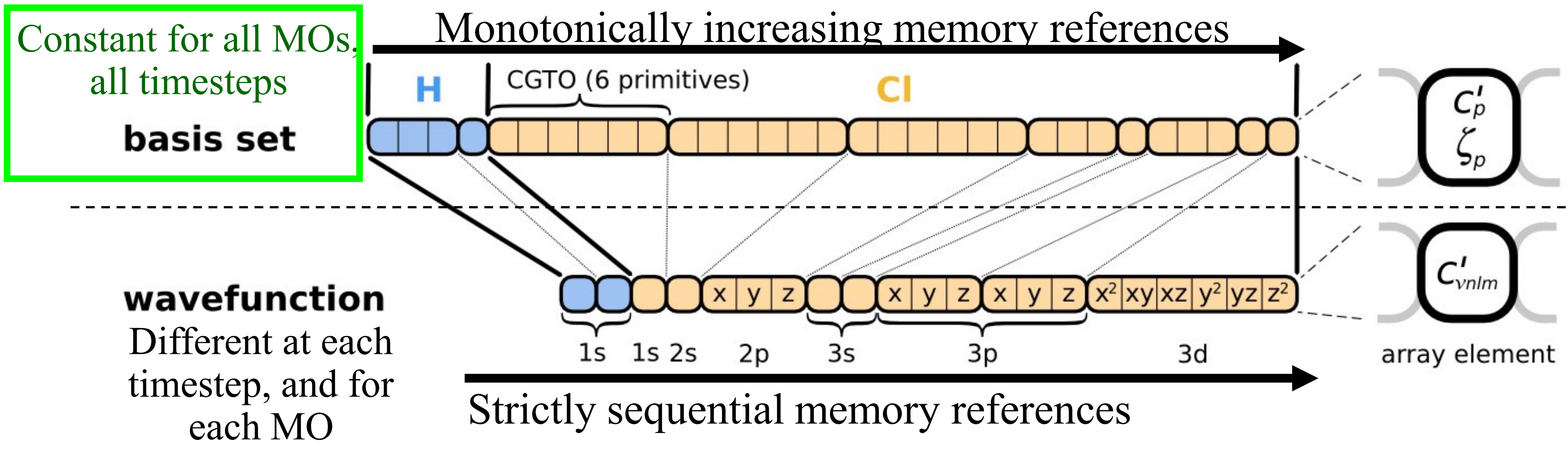
Loop over shells

Loop over primitives: largest component of runtime, due to expf()

Loop over angular momenta

(unrolled in real code)

Traversal of Atom Type, Basis Set, Shell Type, and Wavefunction Coefficients



Constant for all MOs, all timesteps
basis set
Monotonically increasing memory references
H
CGTO (6 primitives)
Cl
C'_p
ζ_p
wavefunction
Different at each timestep, and for each MO
x y z x y z x y z
x^2 xy xz y^2 yz z^2
1s 1s 2s 2p 3s 3p 3d
$C'_{\nu nlm}$
array element
Strictly sequential memory references

- Loop iterations always access same or consecutive array elements: yields good L1 cache performance

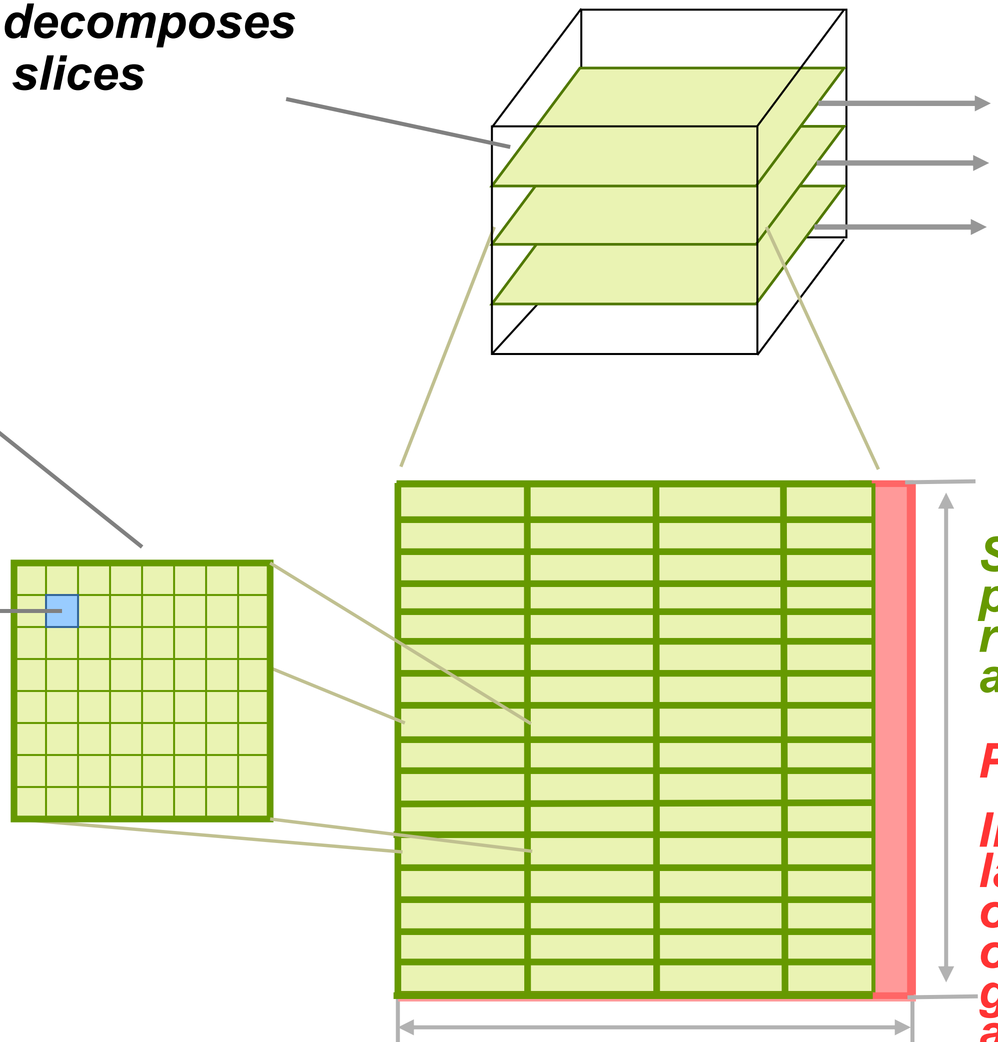
MO Kernel Snippet:
Unrolled Angular Momenta Loop

```
   /* multiply with the appropriate wavefunction coefficient */
   float tmpshell=0;
   switch (shelltype) {
    case S_SHELL:
     value += const_wave_f[ifunc++] * contracted_gto;
     break;
[… P_SHELL case …]
    case D_SHELL:
     tmpshell += const_wave_f[ifunc++] * xdist2;
     tmpshell += const_wave_f[ifunc++] * xdist * ydist;
     tmpshell += const_wave_f[ifunc++] * ydist2;
     tmpshell += const_wave_f[ifunc++] * xdist * zdist;
     tmpshell += const_wave_f[ifunc++] * ydist * zdist;
     tmpshell += const_wave_f[ifunc++] * zdist2;
     value += tmpshell * contracted_gto;
     break;
[... Other cases: F_SHELL, G_SHELL, etc …]
} // end switch
```

Loop unrolling:

- Saves registers
- Reduces loop control overhead
- Increases arithmetic intensity

MO CPU Parallel Decomposition



Lattice decomposed across many CPU threads

Vectors of wavefunction amplitudes are computed using hardware SIMD instructions

Each CPU thread computes 1, 4, 8, or 16 MO lattice points per loop iteration: C, SSE, AVX2 or AVX-512ER

SIMD lanes producing results that are used

Padding:

Inactive SIMD lanes or region of discarded output used to guarantee aligned vector loads+stores

Padded + aligned array

AVX-512ER MO CGTO Loop

```
int maxprim = num_prim_per_shell[shell_counter];
int shelltype = shell_types[shell_counter];
for (prim=0; prim<maxprim; prim++) {
    float exponent       = basis_array[prim_counter    ];
    float contract_coeff = basis_array[prim_counter + 1];

    // contracted_gto += contract_coeff * exp(exponent*dist2);
    __m512 expval = _mm512_mul_ps(_mm512_set1_ps(exponent * MLOG2EF), dist2);

    // expf() approximation required, use (base-2) AVX-512ER instructions…
    __m512 retval = _mm512_exp2a23_ps(expval);

    __m512 ctmp = _mm512_mul_ps(_mm512_set1_ps(contract_coeff), retval);
    contracted_gto = _mm512_add_ps(contracted_gto, ctmp);
    prim_counter += 2;
}
```

AVX-512ER MO Wavefunction Loop

```
/* multiply with the appropriate wavefunction coefficient */
__m512 ts = _mm512_set1_ps(0.0f);
switch (shelltype) {
  case S_SHELL:
    value = _mm512_add_ps(value,  _mm512_mul_ps(_mm512_set1_ps(wave_f[ifunc++]), cgto));
         break;

  case P_SHELL:
    ts = _mm512_add_ps(ts, _mm512_mul_ps(_mm512_set1_ps(wave_f[ifunc++]),  xdist));
    ts = _mm512_add_ps(ts, _mm512_mul_ps(_mm512_set1_ps(wave_f[ifunc++]), ydist));
    ts = _mm512_add_ps(ts, _mm512_mul_ps(_mm512_set1_ps(wave_f[ifunc++]), zdist));
    value = _mm512_add_ps(value, _mm512_mul_ps(ts, cgto));
    break;

  case D_SHELL:
     ….
```

AVX-512ER+FMA MO Wavefunction Loop

```
/* multiply with the appropriate wavefunction coefficient */
__m512 ts = _mm512_set1_ps(0.0f);
switch (shelltype) {
  // use FMADD instructions
  case S_SHELL:
    value = _mm512_fmadd_ps(_mm512_set1_ps(wave_f[ifunc++]), cgto, value);
    break;

  case P_SHELL:
    ts = _mm512_fmadd_ps(_mm512_set1_ps(wave_f[ifunc++]), xdist, ts);
    ts = _mm512_fmadd_ps(_mm512_set1_ps(wave_f[ifunc++]), ydist, ts);
    ts = _mm512_fmadd_ps(_mm512_set1_ps(wave_f[ifunc++]), zdist, ts);
    value = _mm512_fmadd_ps(ts, cgto, value);
    break;
```

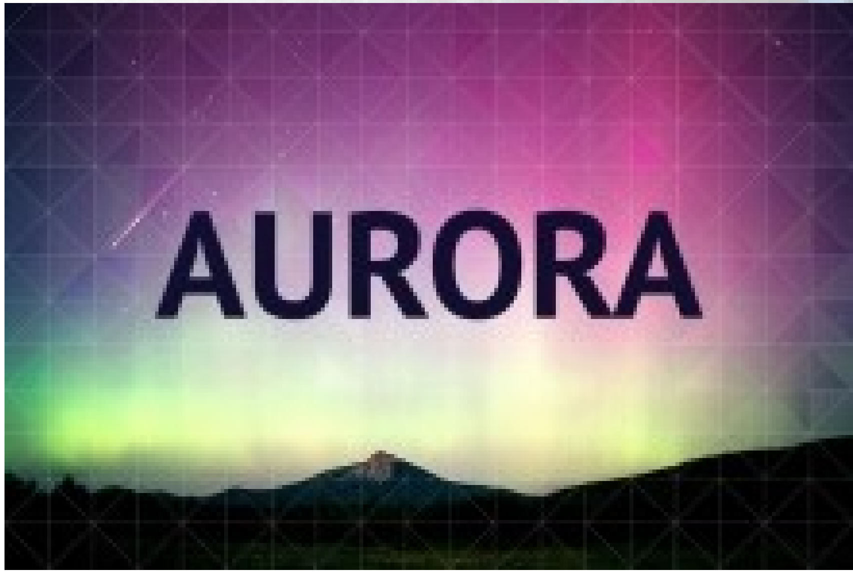
Performance of AVX-512ER Instrinsics vs. Autovectorization on KNL

• Intel C++ '15 autovectorization **(fail)**:	**220+ sec**
• Hand-coded SSE2 w/ existing thread scheme:	**48.5 sec**
• Hand-coded AVX-512ER w/ existing thread scheme:	**6.3 sec**
• Hand-coded AVX-512ER, refactoring thread pool:	**0.2 sec**
• **Hand-coded AVX-512ER tuned thread pool:**	**0.131 sec**
• **Hand-coded AVX-512ER+FMA tweaks:**	**0.107 sec**

Further improvement will require more attention to details of cache behaviour and further tuning of low-level threading constructs for Xeon Phi/KNL

Future Work

- Many more AVX-512 kernels…
- Continue optimization of OSPRay renderer class
- Runtime loading of VMD-specific OSPRay shader extension modules
- Interactive ray tracing of time-varying molecular geometry
- Support upcoming ANL Aurora machine



Acknowledgements

- Theoretical and Computational Biophysics Group, University of Illinois at Urbana-Champaign
- Funding:
  - NSF OCI 07-25070
  - NSF PRAC "The Computational Microscope"
  - NIH support: 9P41GM104601, 5R01GM098243-02

NIH BTRC for Macromolecular Modeling and Bioinformatics

1990-2017

Beckman Institute
University of Illinois at
Urbana-Champaign

INTEL® HPC DEVELOPER CONFERENCE