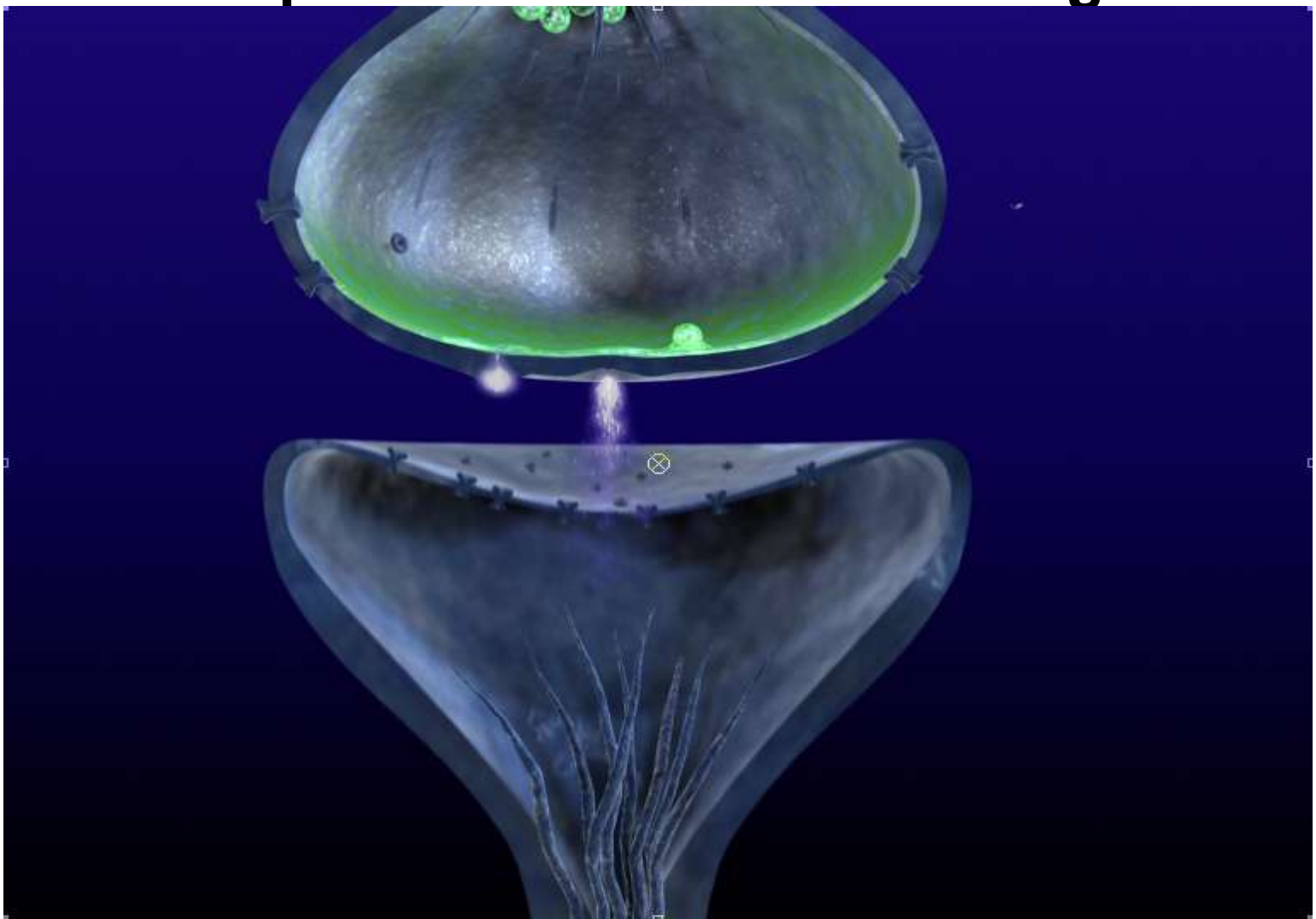
Electrostatic Interactions are key to biological function

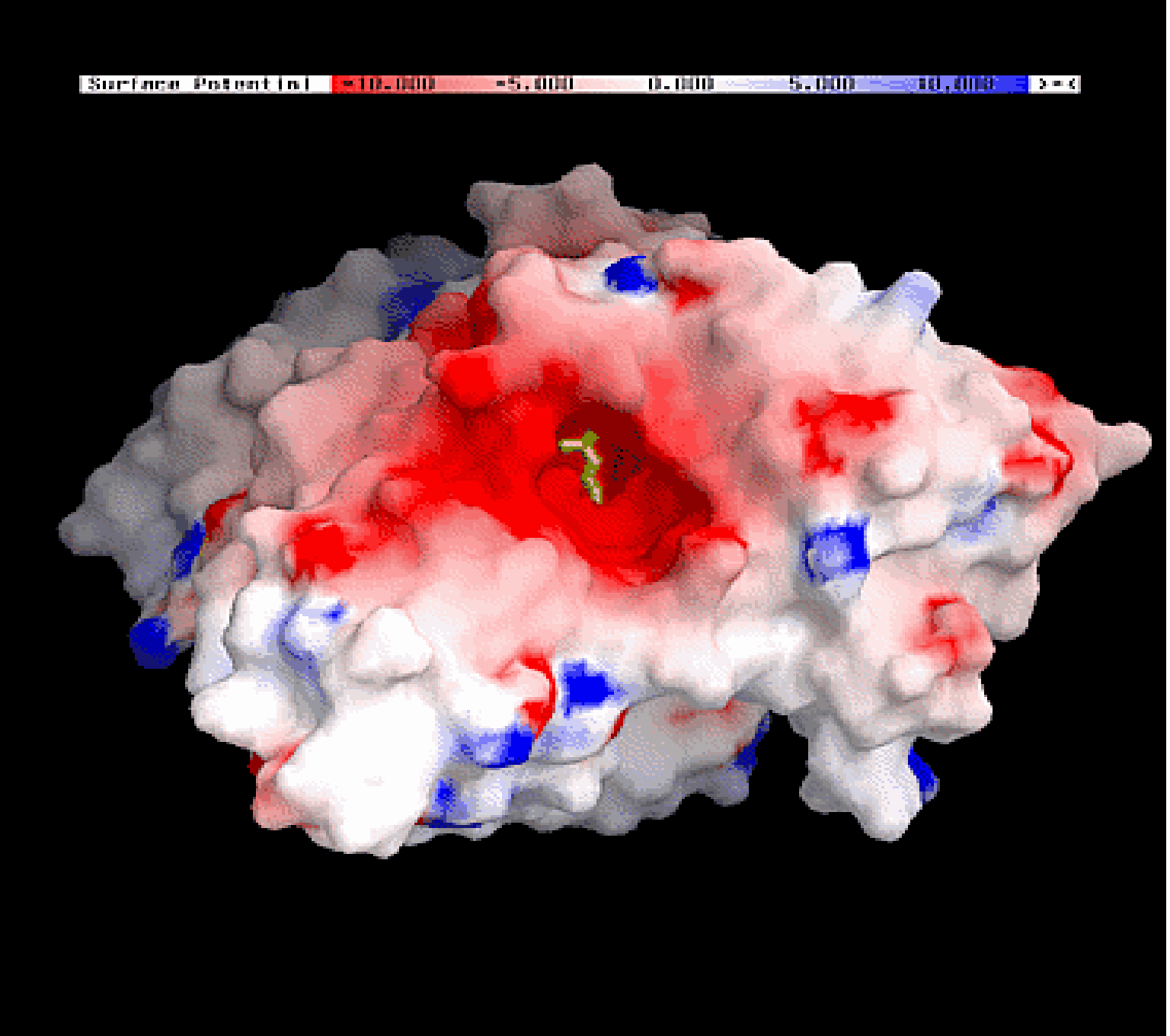
The strongest force in Nature

(at least on atomic scale and above)

- **Example: DNA winding around histones is held by opposing charges. (histone core charge +200, DNA = -400 units.)**
- **Example: Acetylcholine docking into acetylcholinestarase is steered by charges.**
- **Example: Many active sites in proteins are charged.**

Example: Function of a synapse (nerve junction) requires electrostatic steering.





Molecular surface of acetyl choline esterase molecule (structure by Sussman et al.) color coded by electrostatic potential. The view is directly into the active site and acetyl choline is present in a bond representation. Note the depth of the pocket, its negative nature corresponding to the postive charge on the acetyl choline *(small worm-like thing inside the red spot)*

***Curare* - the poison in blow-darts**

Electrostatic potential is traditionally obtained by solving Poisson-Boltzmann (PB) equation numerically.

Some popular Real Space PB Solvers designed specifically for Biomolecular structures:

DelPhi – proprietary finite difference PB solver

http://trantor.bioc.columbia.edu/delphi/

MEAD – open-source finite difference BP solver

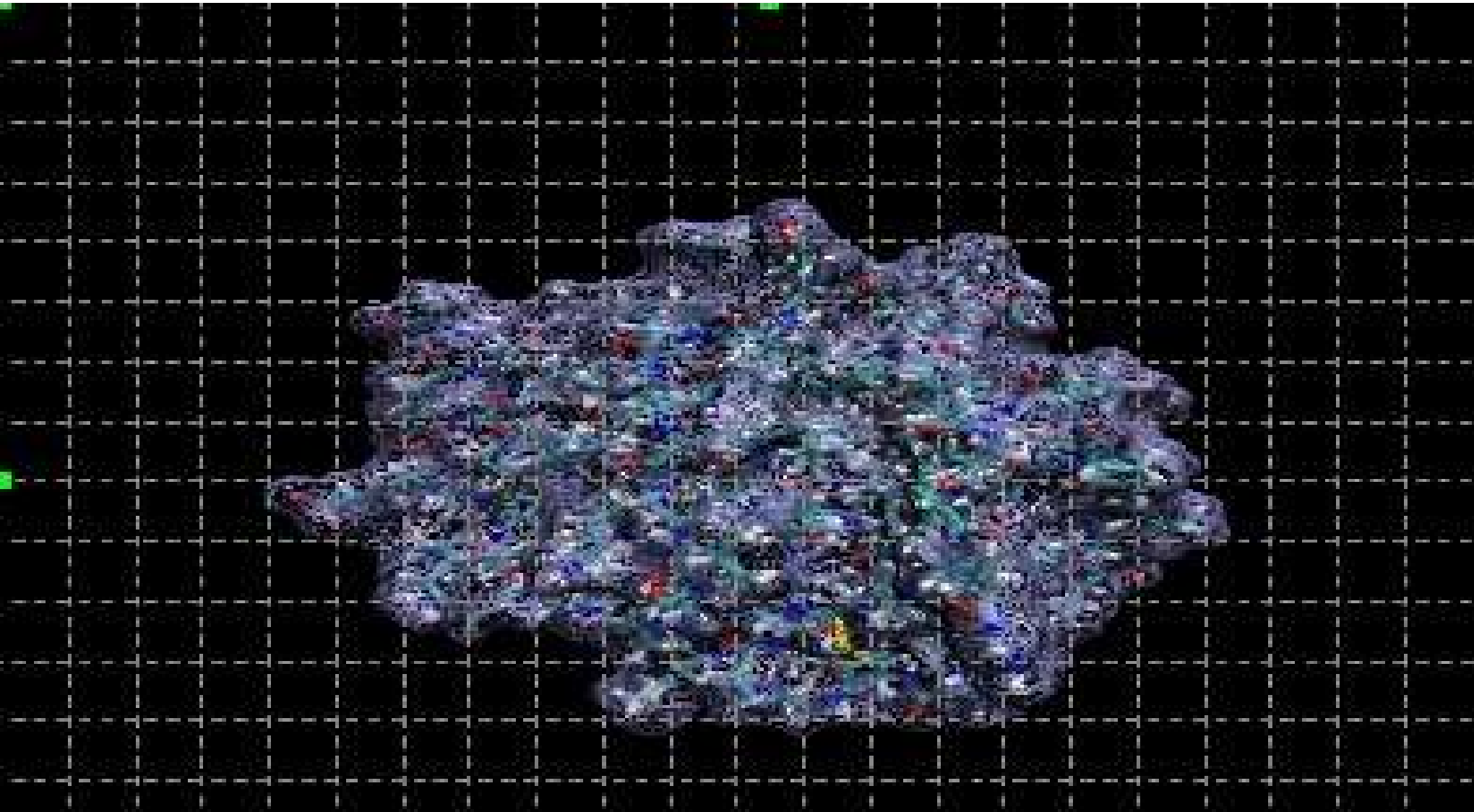
http://www.scripps.edu/mb/bashford/

APBS – open-source adaptive mesh PB solver

http://agave.wustl.edu/apbs/

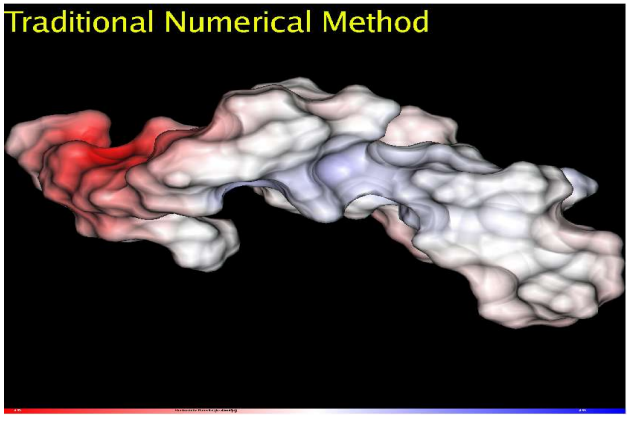
Of these, we chose DelPhi as a reference for comparison because it is considered to be standard in the field.

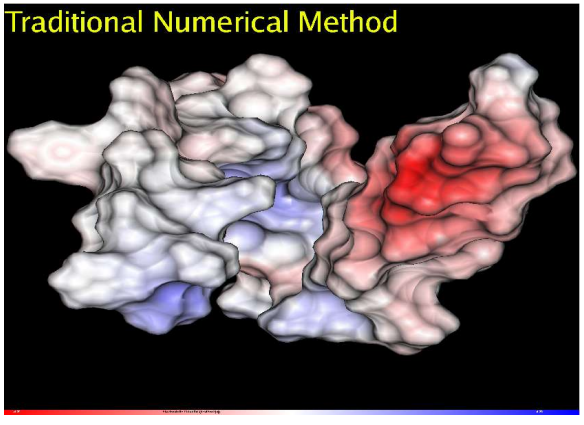
Traditional Approach : To obtain electrostatic potential, solve the Poisson-Boltzn $\nabla \varepsilon(r) \nabla \Phi(\vec{r}) = -4\pi \rho_m(r)$ grid.

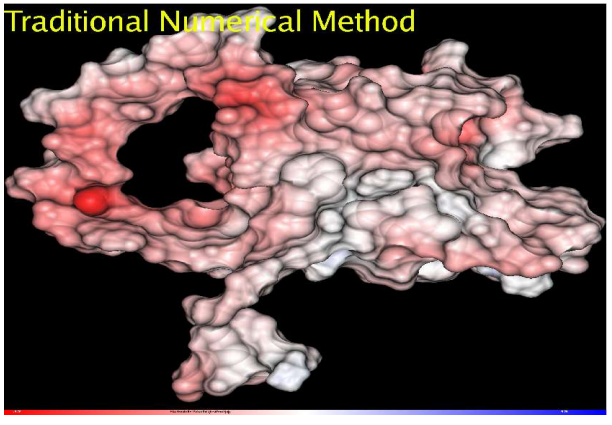


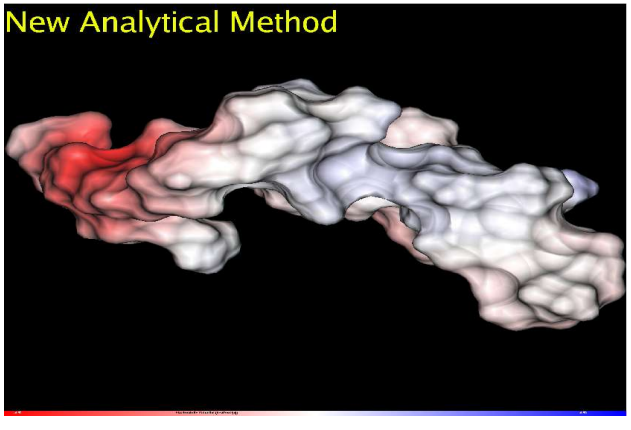
Agreement Between NPB and New ALPB

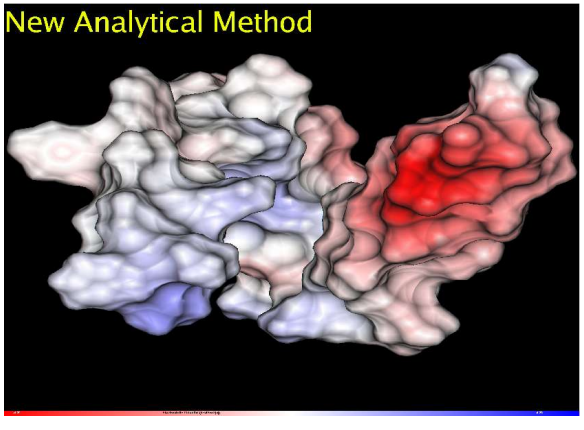
Numerical

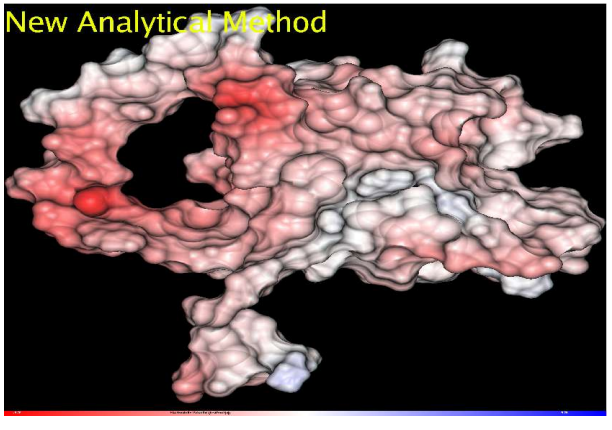






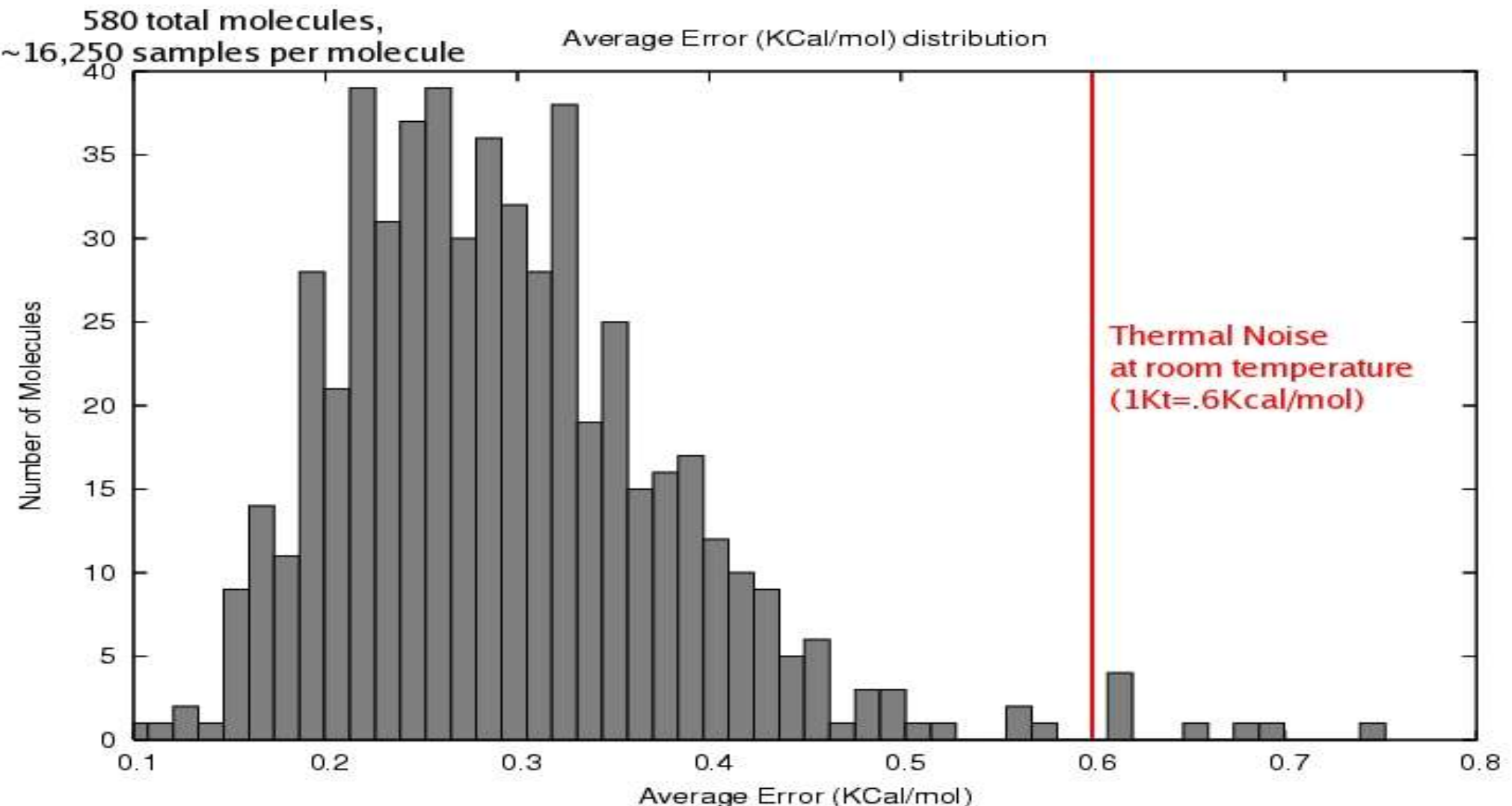






Analytical

ALPB: Distribution of Average Absolute Error on a per-molecule basis.



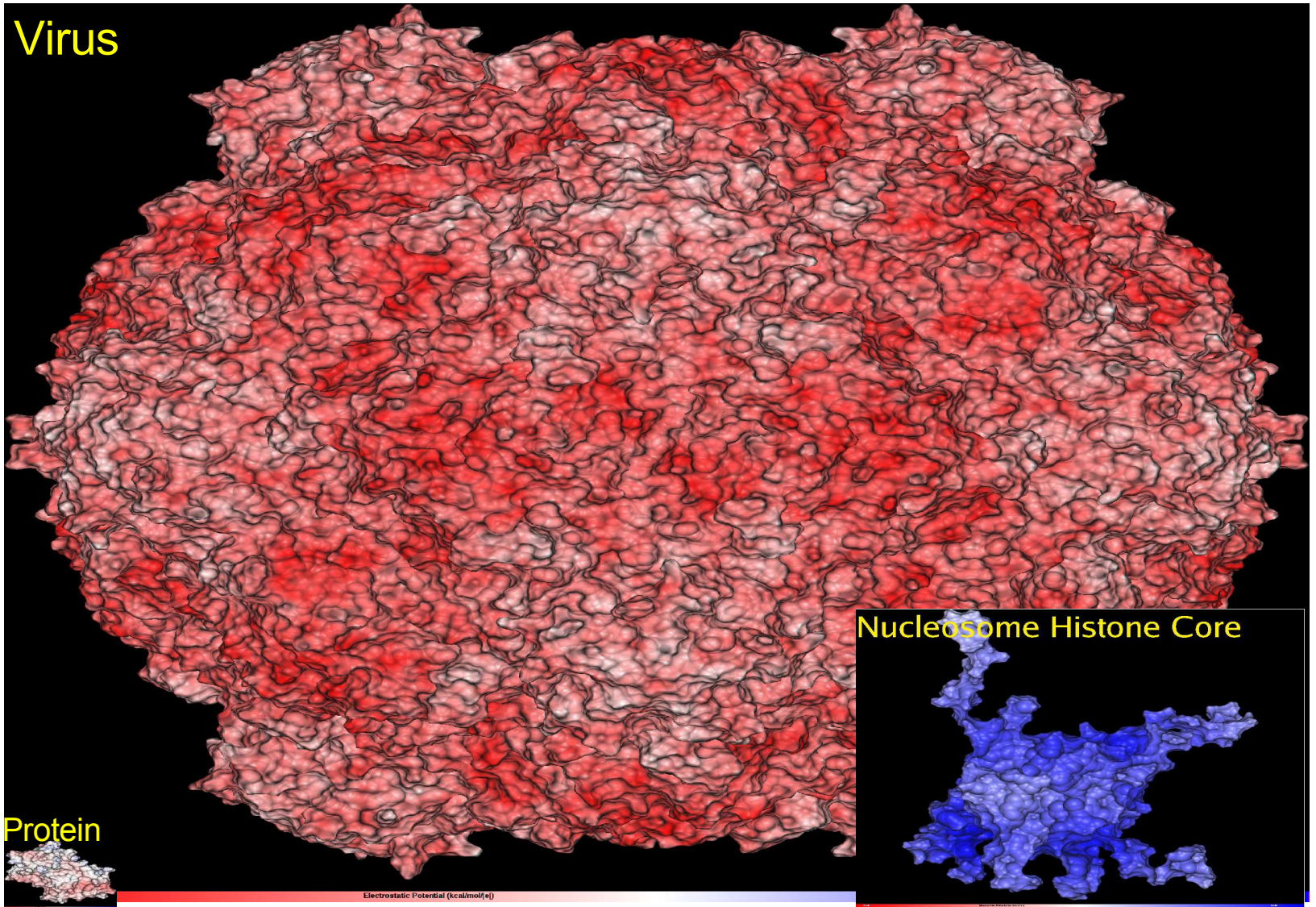
Run times on a 2 GHz PC:

Time required to compute and display the potential across the surface of a molecule from the test set using DelPhi:
~ 10 minutes.

Time required to compute and display the potential across the surface of a molecule from the test set using our program GEM based on
ALPB:
~ 15 seconds.

Almost as accurate, 40x faster

The possibilities are...



1nm

Supercomputer vs. PC
or Numerical vs. Analytical

"Electrostatics of nanosystems: Application to microtubules and the ribosome",
PNAS 98, 10037 (2001)
N.A. Baker,
D. Sept,
S. Joseph,
M.J. Holst,
and J.A. McCammon

finite element algorithm (APBS)
343 CPUs of the NPACI Blue Horizon
Ribosomal complex (100,000 atoms)
0.41 Angstrom resolution.
Execution time: ???
Memory ???

"Analytical electrostatics for biomolecules",
to be submitted (2005)
J.C. Gordon,
A.T. Fenley,
and A. Onufriev

ALPB
single PC
virus capsid (500,000 atoms)
.5 Angstrom resolution
Execution time: overnight
Memory: < 100 MB

Agenda

- Memory management
  - Stack management
  - Heap management
- Implementing mathematical formulae, factoring and computational complexity

Memory

- Static variables aside, there are two types of variables, those that are dynamically allocated and those that are always allocated when a function is put on the stack. Local variables are stored on the **stack** with the executable code, and dynamically allocated variables are stored in the **heap**.

Stack overview

- Stack memory is allocated on entry of a function and released on return.
- Stack memory is all writable, meaning that even executable code is writable when on the stack.
- A function is organized in the following order on the stack: variables, function return address, function label, code.

Buffer Overflow Errors

- Because data is organized before executable code, if one were to overstep a local array in a function and write binary instructions in where execution should be, those instructions will be executed.
- Most common buffer overflow: fixed length character arrays to handle user input
- Recognizing buffer overflow: in *nixes, buffer overflow commonly results in stack corruption, as evidenced by “In ()??” in debuggers or by the inability to access the pointer to the function that is currently executing.

Heap overview

- Memory that is allocated and deallocated dynamically is stored on the heap.
- Heap variables are not structurally related to the functions in which they are allocated or deallocated, the heap itself can be seen as a large contiguous block of available memory situated in a separate address space from the stack.

Common heap related bugs

- Memory leaks – result in unavailable memory when it is needed later.
- Dereferencing pointers to chaos or the void (typically uninitialized)
- Assuming memory will be available and not checking allocation.
- Using dynamic structures such as “vector” without considering the underlying methods for resizing the array.

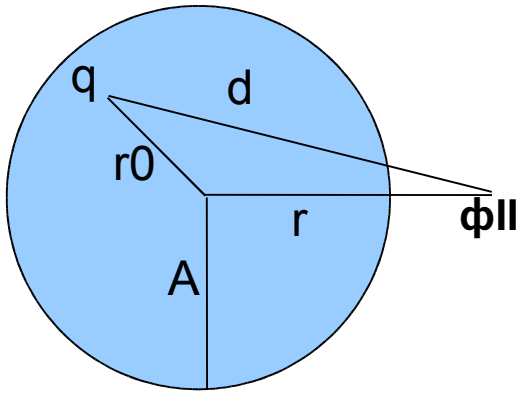
Implementing Mathematical Formulae for Performance

- Prior to implementation, mathematical functions can be factored with an eye for computational complexity to improve performance. (e.g. $O(\sum 3 * k) > O(3 \sum k)$).
- One should consider the computational complexity of an operation when using it, and optimize for performance (e.g. $O(2*k) > O(k+k)$).
- Reuse intermediate calculations whenever possible.

Example Problem:

- Setup for problem: solve for the potential at the surface of a generic molecular shape.

- The ***exact*** solution for a sphere:



$$\Phi_{II} = \frac{q}{(\varepsilon_i r)} \sum_{l=0}^{\infty} \left[\left(\frac{r_0}{r} \right)^l P_l(\cos\Theta) \right] - \frac{q}{r} \left(\frac{1}{\varepsilon_i} - \frac{1}{\varepsilon_o} \right) \sum_{l=0}^{\infty} \left[\left(\frac{r_0}{r} \right)^l \frac{1}{1 + \frac{l}{l+1} \frac{\varepsilon_i}{\varepsilon_o}} P_l(\cos\Theta) \right]$$

Infinite sum is useless in practice. Keeping just few terms will not work since when (r0/r) -> 1 (charges close to the surface) the sum converges very slowly.

- Apply the following sum trick:

$$\sum_{l=0}^{\infty}\left[\left(\frac{r_0}{r}\right)^{l}\frac{1}{1+\frac{l}{l+1}\beta}P_l(\cos\Theta)\right]\approx\frac{1}{(1+\alpha\beta)}\left[\alpha\beta+\sum_{l=0}^{\infty}\left(\frac{r_0}{r}\right)^{l}P_l(\cos\Theta)\right]$$

- Key approximation: $\alpha = constant = \frac{l}{l+1},\ 0.5 < \alpha < 1.0$

Note: $\beta = \frac{\varepsilon_i}{\varepsilon_o}$

Approximate Solution

$$\Phi_{II} = \frac{q}{\varepsilon_o}\left(\frac{1+\alpha}{d} - \frac{\alpha(1-\beta)}{r}\right)$$

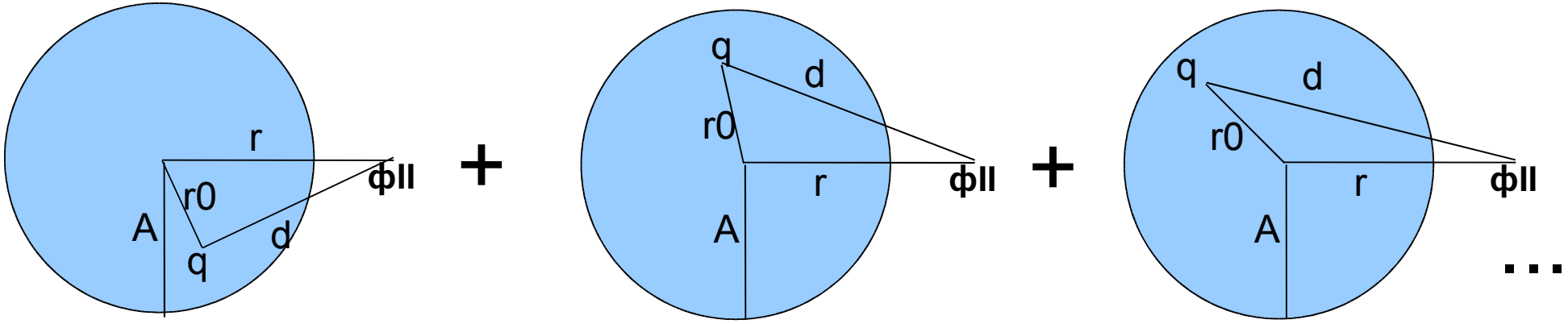
- Where α is determined from minimizing maximum error L_∞ when compared with the exact solution on a sphere: $\alpha \approx 0.755$
- The distance, *r*, to a point of interest, *p*, is greater than or equal to A, the radius of the sphere.

The prior function was for only 1 contribution to ϕII

$$\Phi_{II} = \frac{q}{\varepsilon_o}\left(\frac{1+\alpha}{d} - \frac{\alpha(1-\beta)}{r}\right)$$

If we leave the sum in front of the whole function, we are obviously missing out on a lot of possible optimization.

How do we go about factoring variables out of this function assuming it is in a sum?



Current Implementation of Prev.

$$(1+ \alpha)\sum q/d + \alpha(1 - \beta)/r \sum q$$