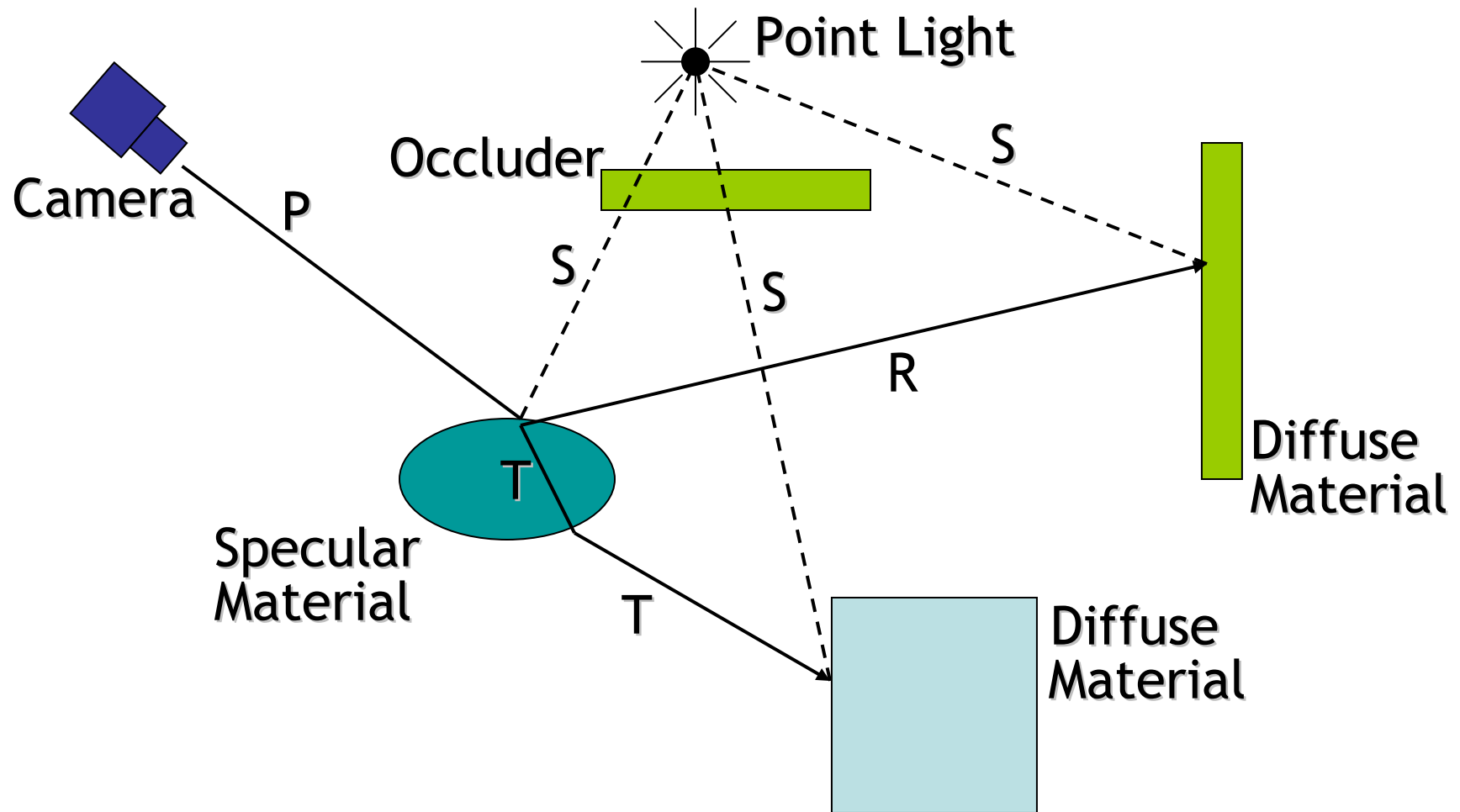

Ray Tracing on GPUs



Ian Buck

Graphics Lab
Stanford University

Ray Tracing

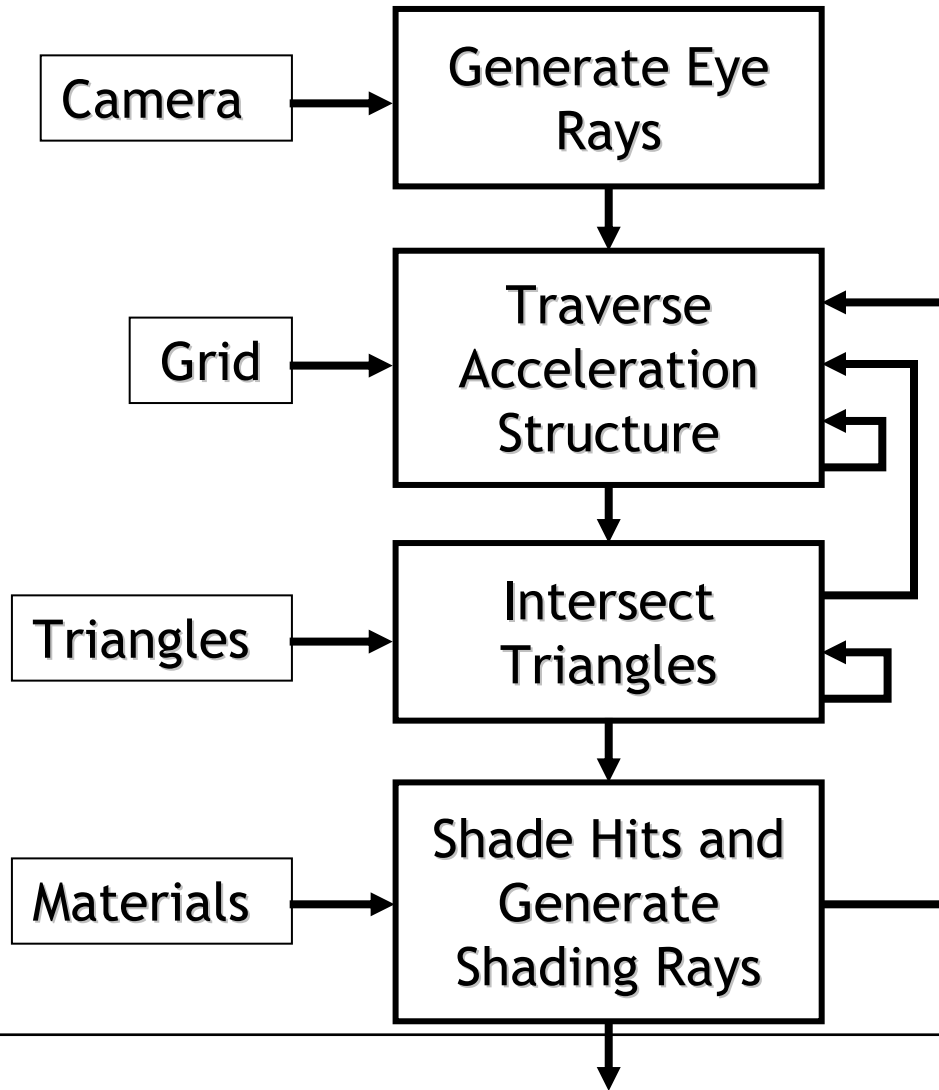


Implementation Options

- GPU as a ray-triangle intersection engine [Carr et al. 2002]
 - Rays and geometry streamed to GPU
 - Intersection calculation results read back
 - Acceleration structure traversal done on host CPU
- GPU as a ray tracing engine [Purcell et al. 2002]
 - Scene geometry and acceleration structure stored on GPU
 - GPU performs ray generation, acceleration structure traversal, intersection, and shading
 - Host provides camera info



Streaming Ray Tracer

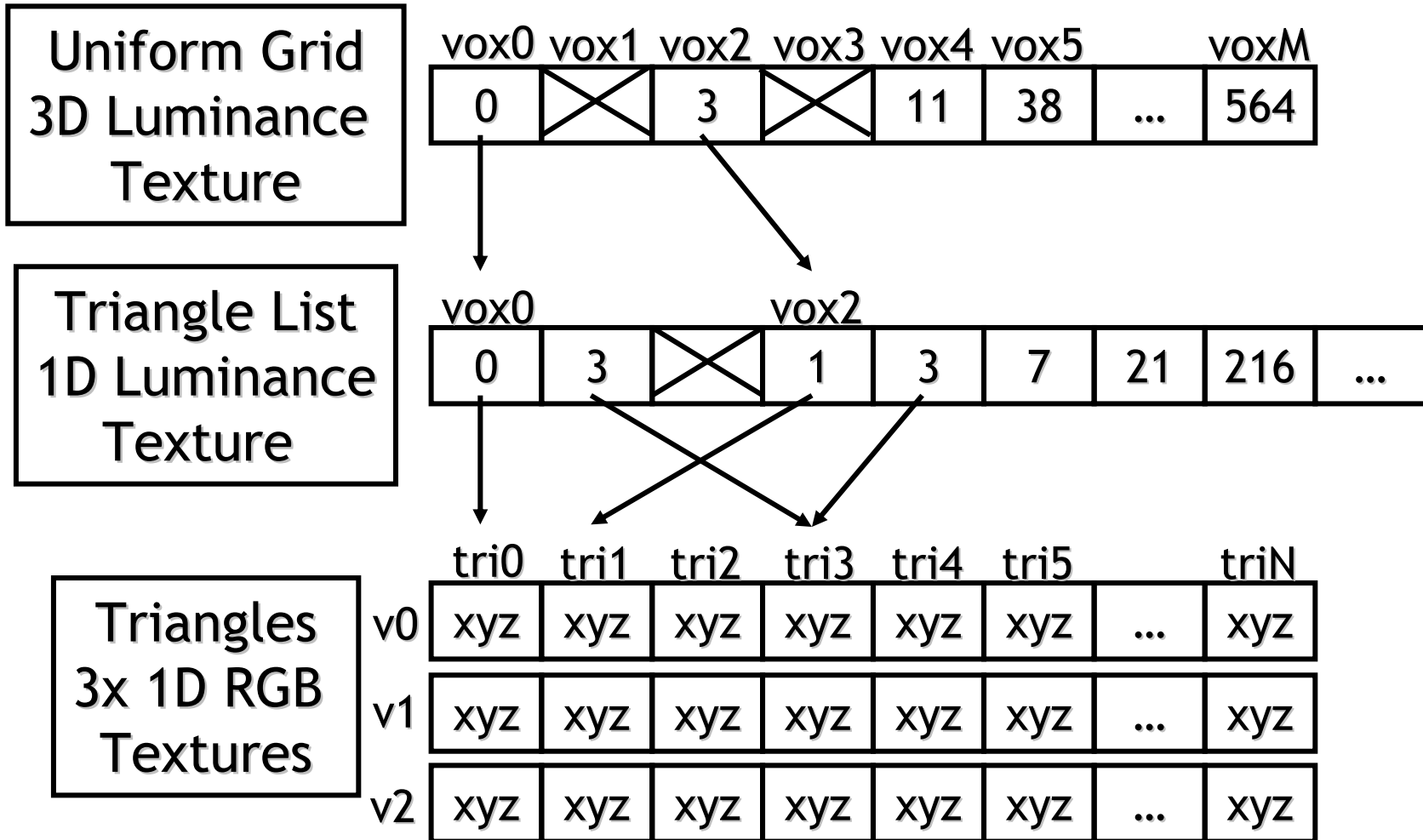


Techniques Used

- Data structure navigation
 - Texture memory stores data structures
 - Dependent texture fetches walk through data
- Flow control
 - Kernel binding based on occlusion query results
 - Efficient selective execution of kernels using early-z occlusion culling
 - Difficulty in flow control disappearing with newest graphics cards
 - PS 3.0

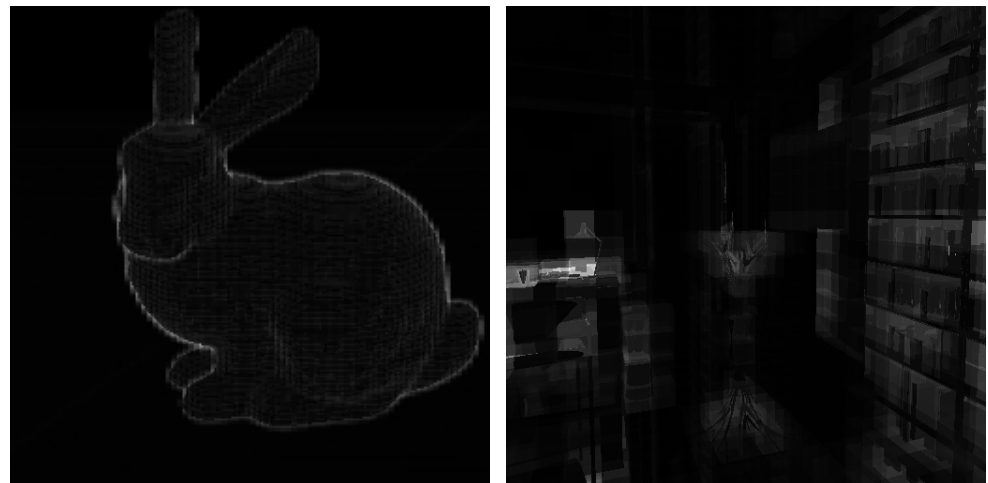
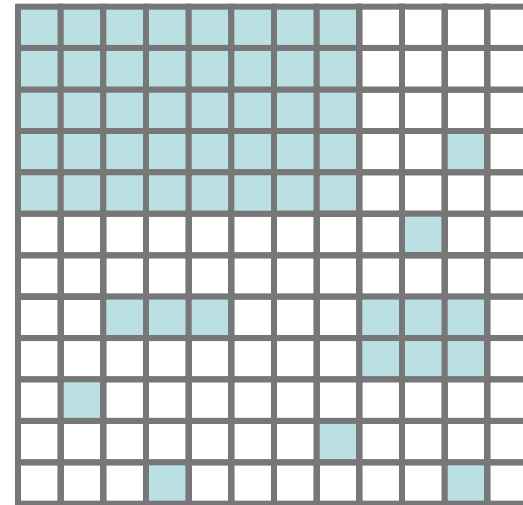


Texture Memory Organization

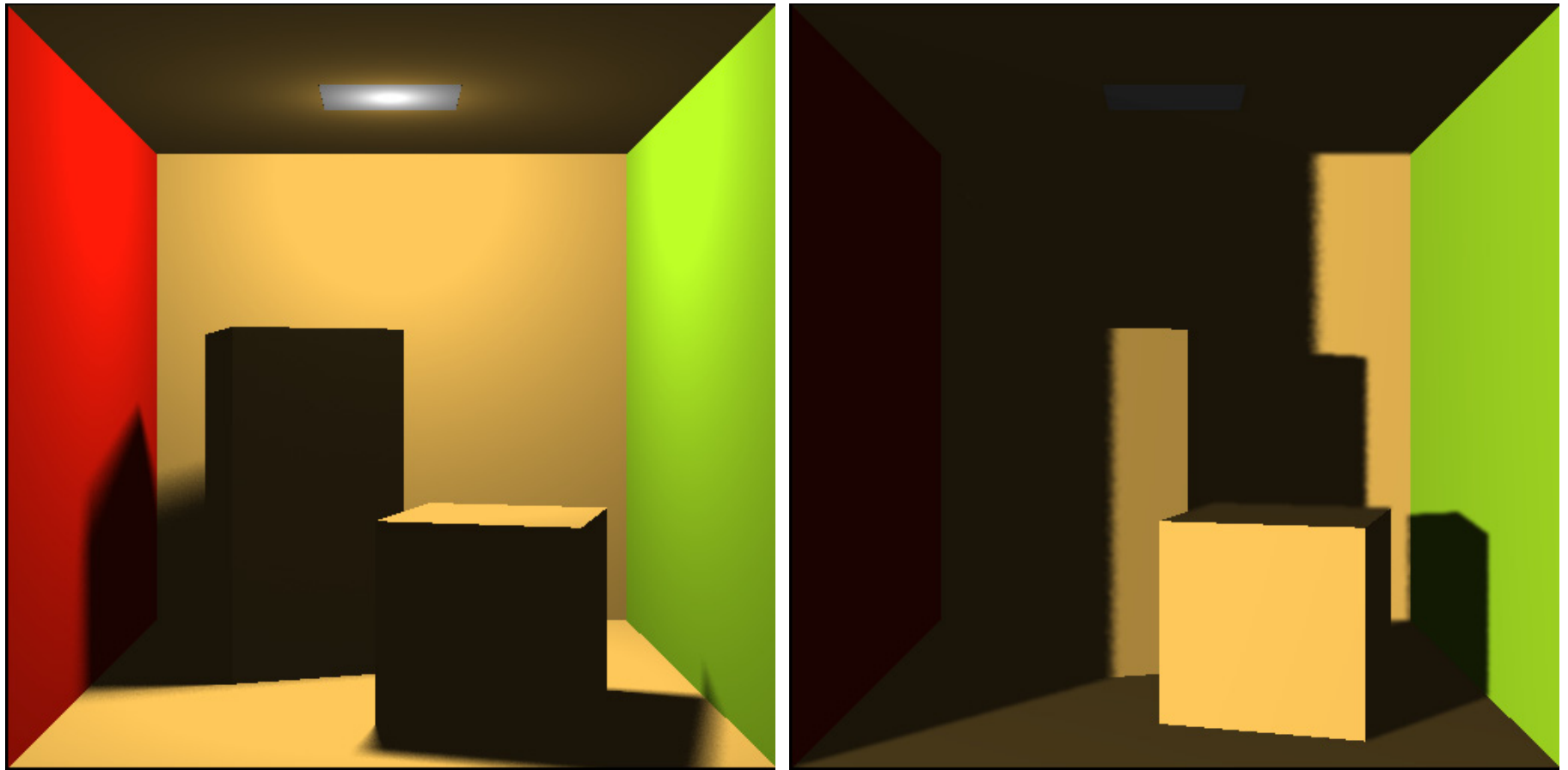


Efficient Selective Execution

- Rendering giant screen filling quad not ideal
 - Not all pixels need to process every rendering pass
- Use early fragment kill
 - Computation mask
 - Controllable early-Z occlusion culling
- Trade computation for bandwidth



Cornell Box – Ray Traced Shadows



Rendered using a Radeon 9700 Pro

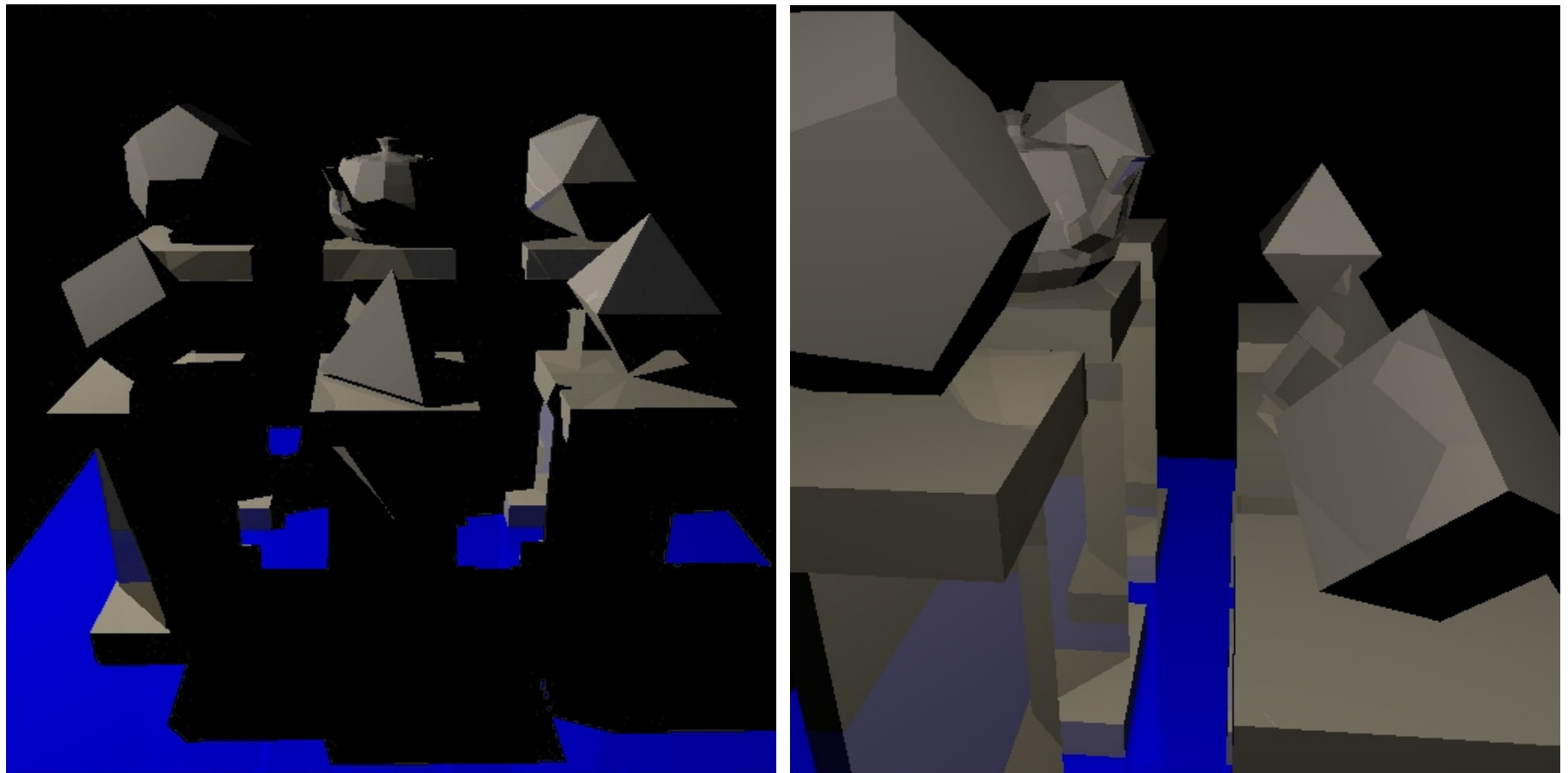


GPGPU
Ian Buck
Stanford University

L8

20
VIS04

Teapotahedron



Rendered using a Radeon 9700 Pro

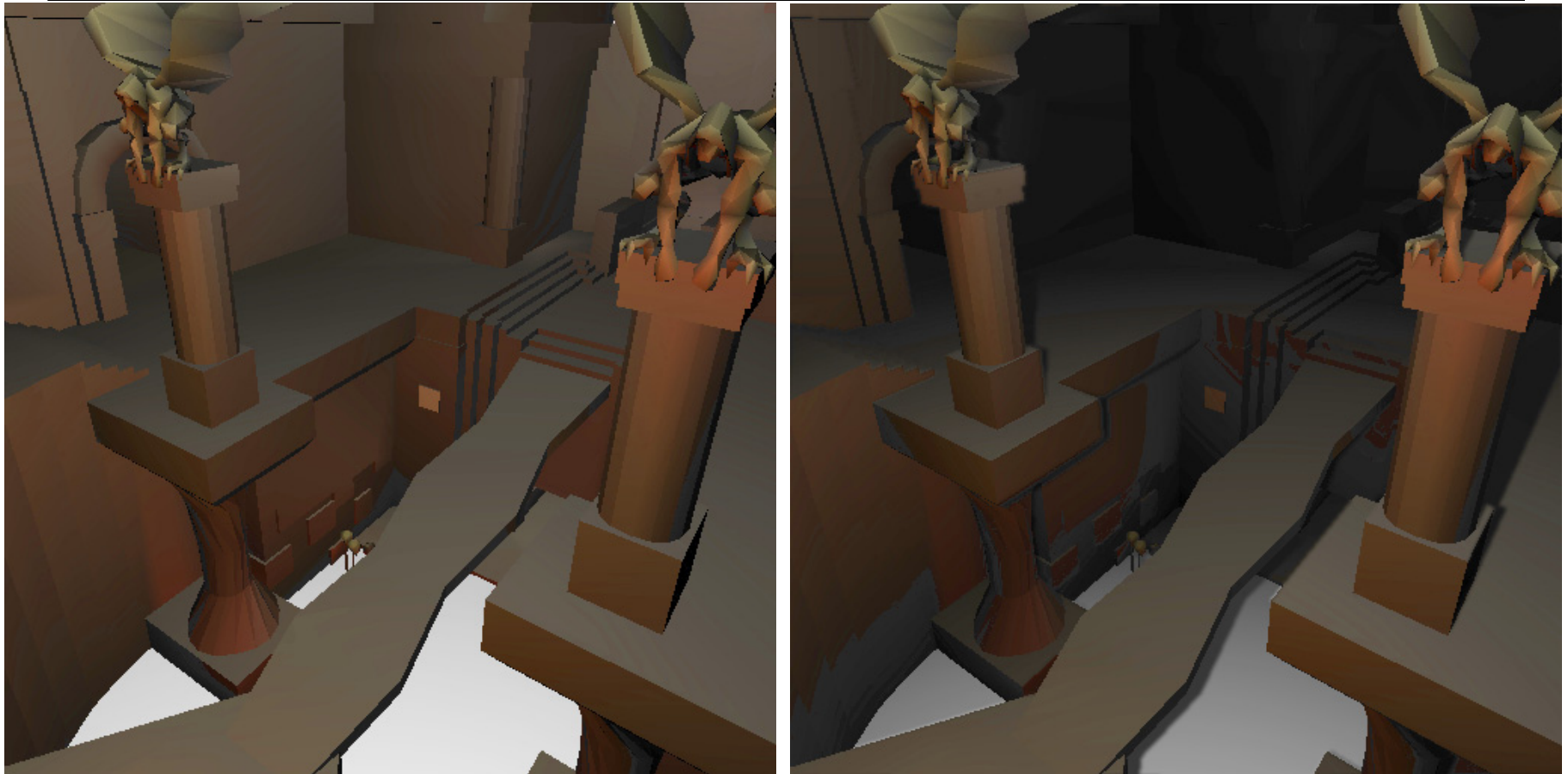


GPGPU
Ian Buck
Stanford University

L9

20
VIS04

Quake 3 – Ray Traced Shadows



Rendered using a Radeon 9700 Pro



GPGPU
Ian Buck
Stanford University

L10

20
VIS04

Performance Results

- Radeon 9700 Pro
 - 100M ray-triangle intersections/s
 - 300K to 4.0M rays/s
 - Between 3 – 12 fps @ 256x256 pixels
- Radeon X800
 - Brook Version: 187M ray-triangle intersections/s
- CPU implementation
 - 20M intersections/s P3 800 MHz [Wald et al. 2001]
 - 800K to 7.1M ray/s 2.5 GHz P4 [Wald et al. 2003]
 - With simple shading: 1.8M to 2.3M rays/s



Molecular Dynamics on Graphics Hardware



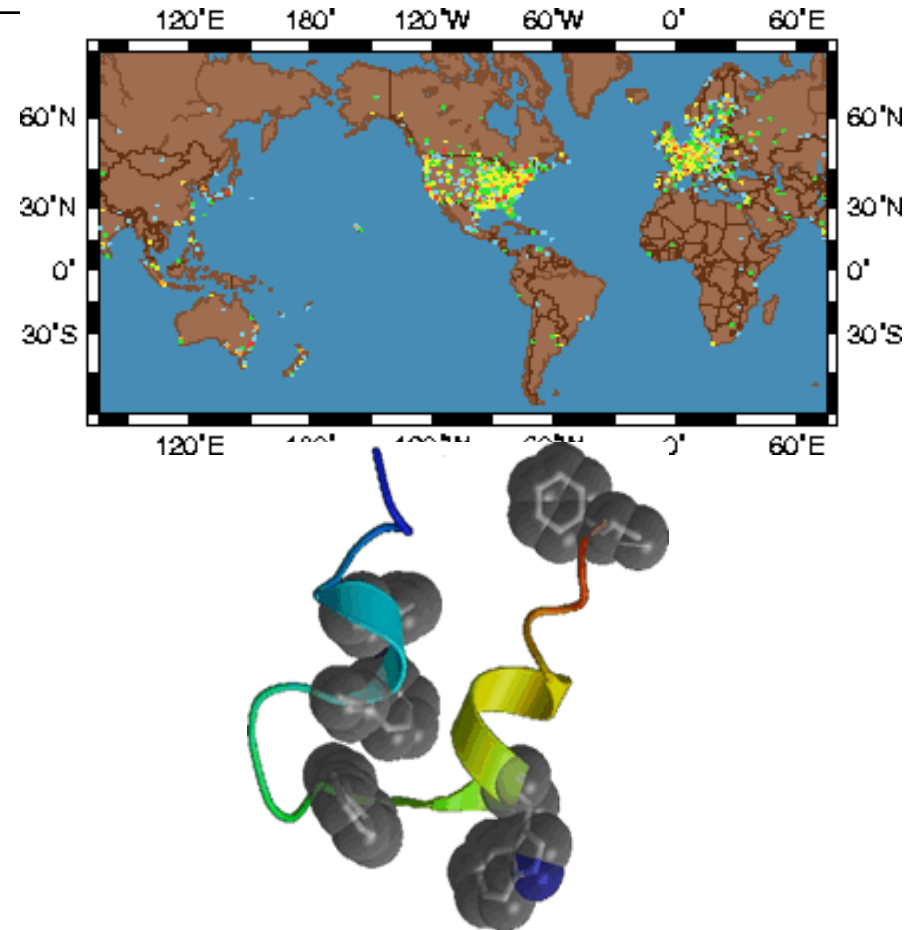
Ian Buck

Graphics Lab
Stanford University

Folding@home: Vijay Pande

What does Folding@Home do?

Folding@Home is a distributed computing project which studies [protein folding](#), misfolding, aggregation, and [related diseases](#). We use novel computational methods and **large scale distributed computing**, to simulate timescales thousands to millions of times longer than previously achieved. This has allowed us to simulate folding for the first time, and to now direct our approach to examine folding related disease.

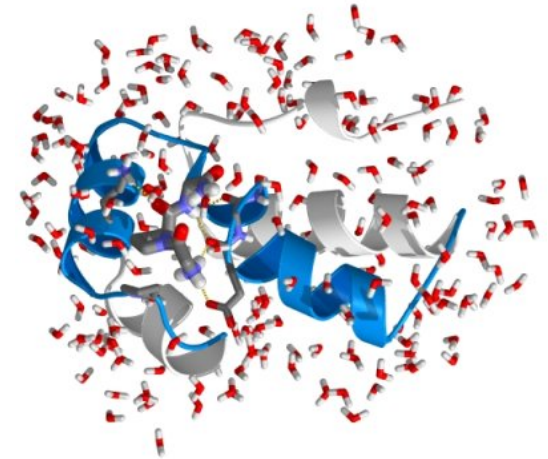


Results from Folding@Home simulations of villin



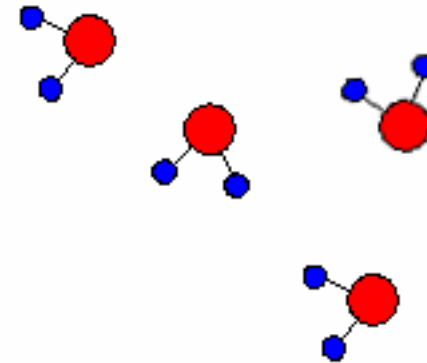
GROMACS: Erik Lindahl

- GROMACS provides *extremely high performance* compared to all other programs.
- Lot of algorithmic optimizations:
 - Own software routines to calculate the inverse square root.
 - Inner loops optimized to remove all conditionals.
 - Loops use SSE and 3DNow! multimedia instructions for x86 processors
 - For Power PC G4 and later processors: AltiVec instructions provided
- normally 3-10 times faster than any other program.



Nonbonded forces

- Accounts for 80% of the runtime in C/Fortran code
- Most common form:



$$V_{nb} = \sum_{i,j} \left[\frac{1}{4\pi\epsilon_0} \frac{q_i q_j}{r_{ij}} + \left(\frac{C_{12}}{r_{ij}^{12}} - \frac{C_6}{r_{ij}^6} \right) \right]$$

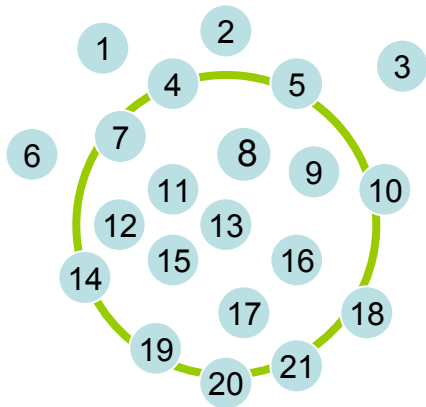
Electrostatics

Lennard-Jones



Using cutoffs & neighbor lists

- Neighbor list constructed every 10 steps.
- In practice: 10,000-100,000 atoms, with 100-200 neighbors in each list



Neighbor list for atom 13 =
{ 8, 9, 11, 12, 15, 16, 17 }



What we do in the inner loop?

For each i atom {
 fetch atom i data
 i_force = 0;
 For each j atom in our neighborlist {
 fetch atom j data
 Calculate vectorial distance; $\mathbf{dr} = \mathbf{r}_i - \mathbf{r}_j$
 Calculate $r^2 = dx^2 + dy^2 + dz^2$, and $1/r = 1/\sqrt{r^2}$
 Calculate potential and vectorial force
 Subtract the force from the j atom force
 i_force += force;
 }
 Store i_force;
}

Inner loop



What we do in the inner loop?

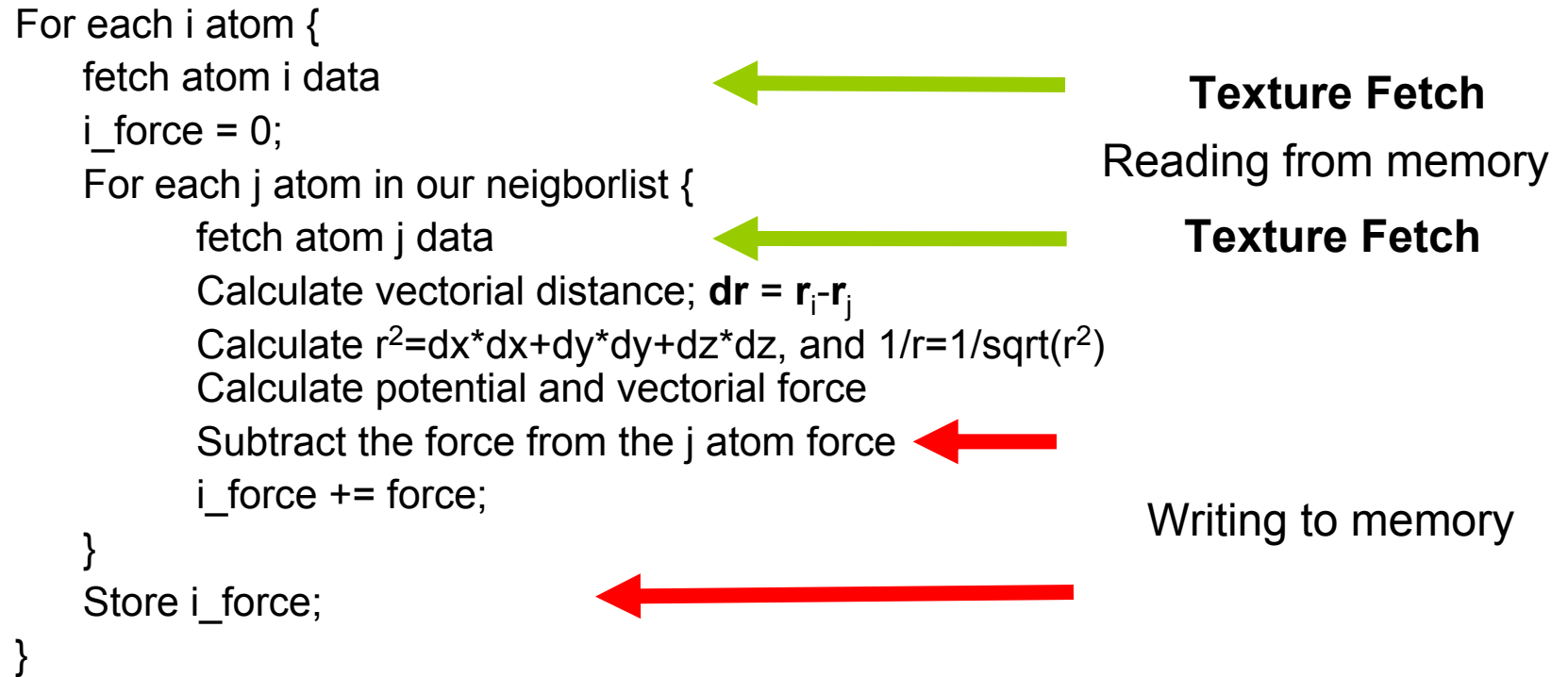
```
For each i atom {  
  fetch atom i data  
  i_force = 0;  
  For each j atom in our neighborlist {  
    fetch atom j data  
    Calculate vectorial distance;  $\mathbf{dr} = \mathbf{r}_i - \mathbf{r}_j$   
    Calculate  $r^2 = dx^2 + dy^2 + dz^2$ , and  $1/r = 1/\sqrt{r^2}$   
    Calculate potential and vectorial force  
    Subtract the force from the j atom force  
    i_force += force;  
  }  
  Store i_force;  
}
```

Reading from memory

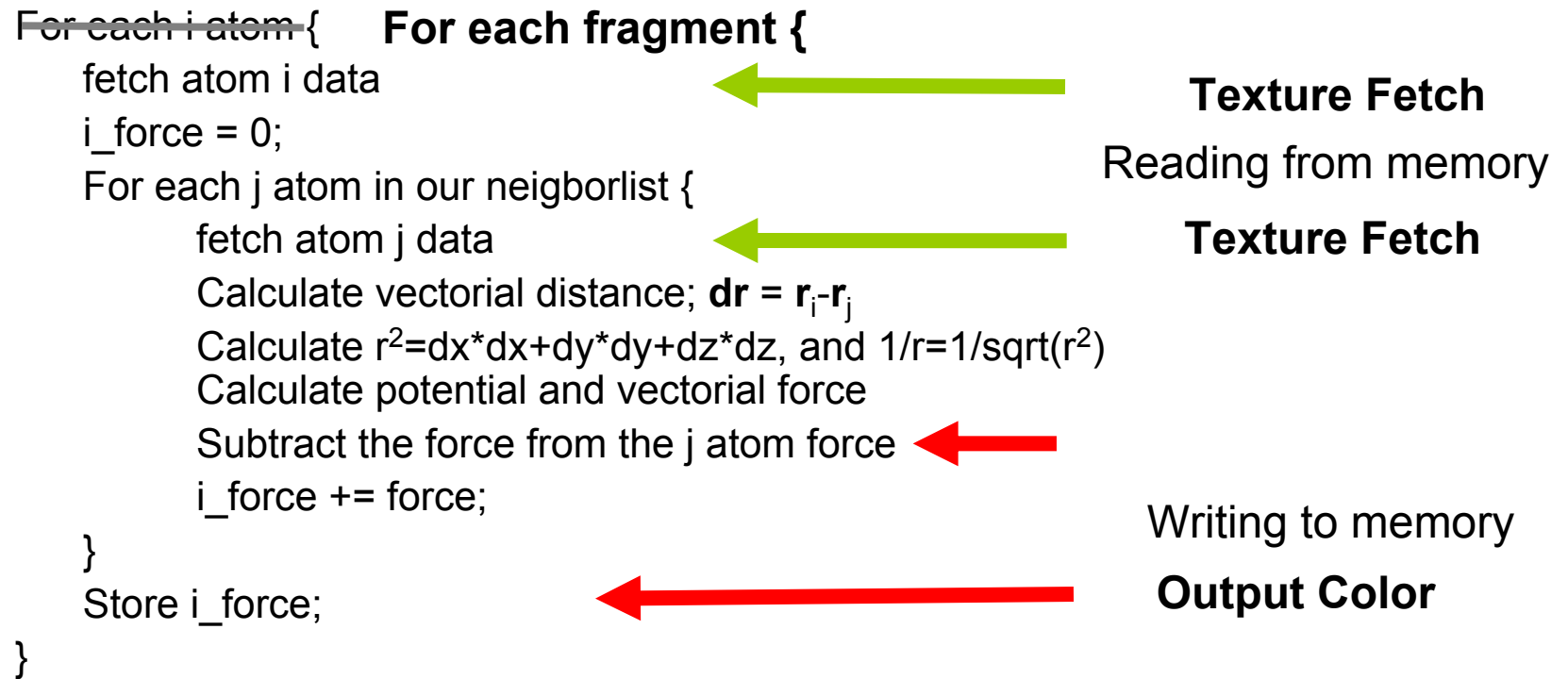
Writing to memory



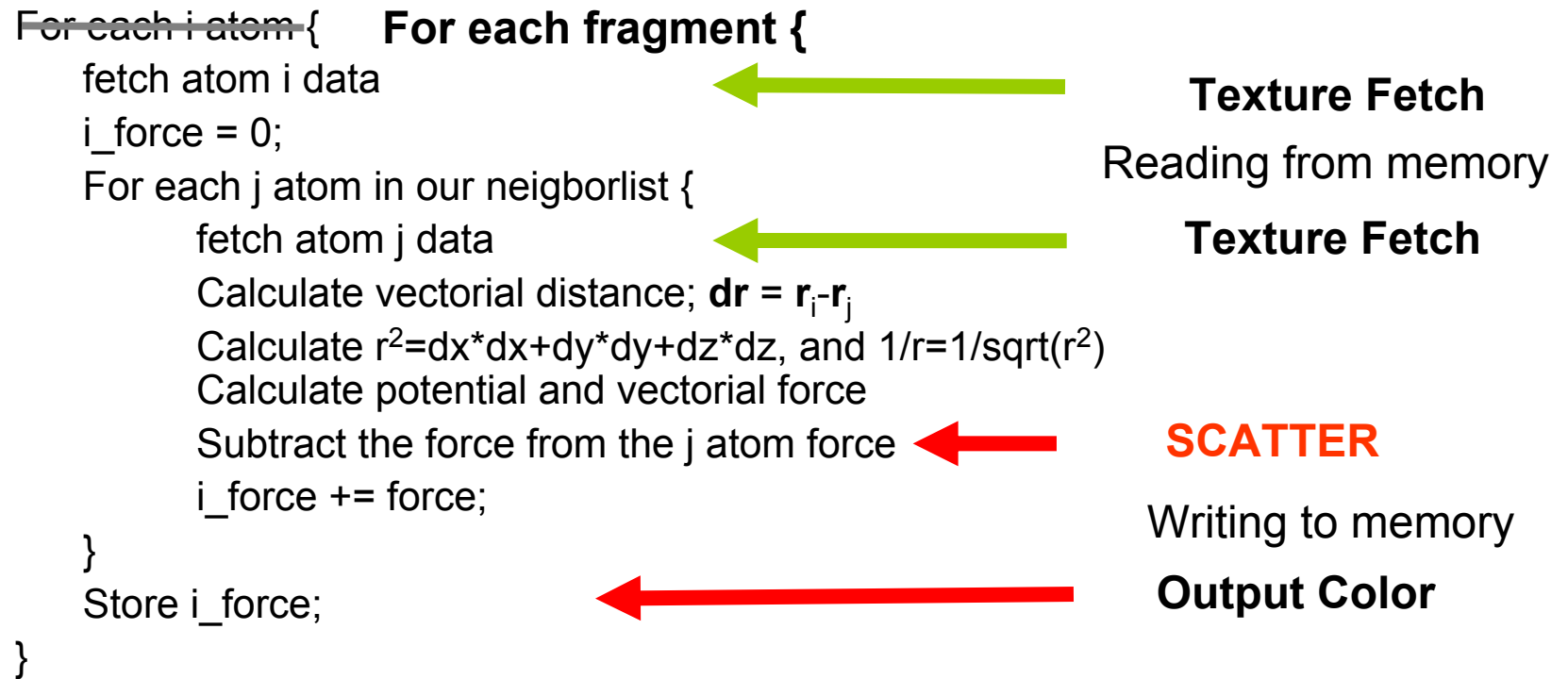
What we do in the inner loop?



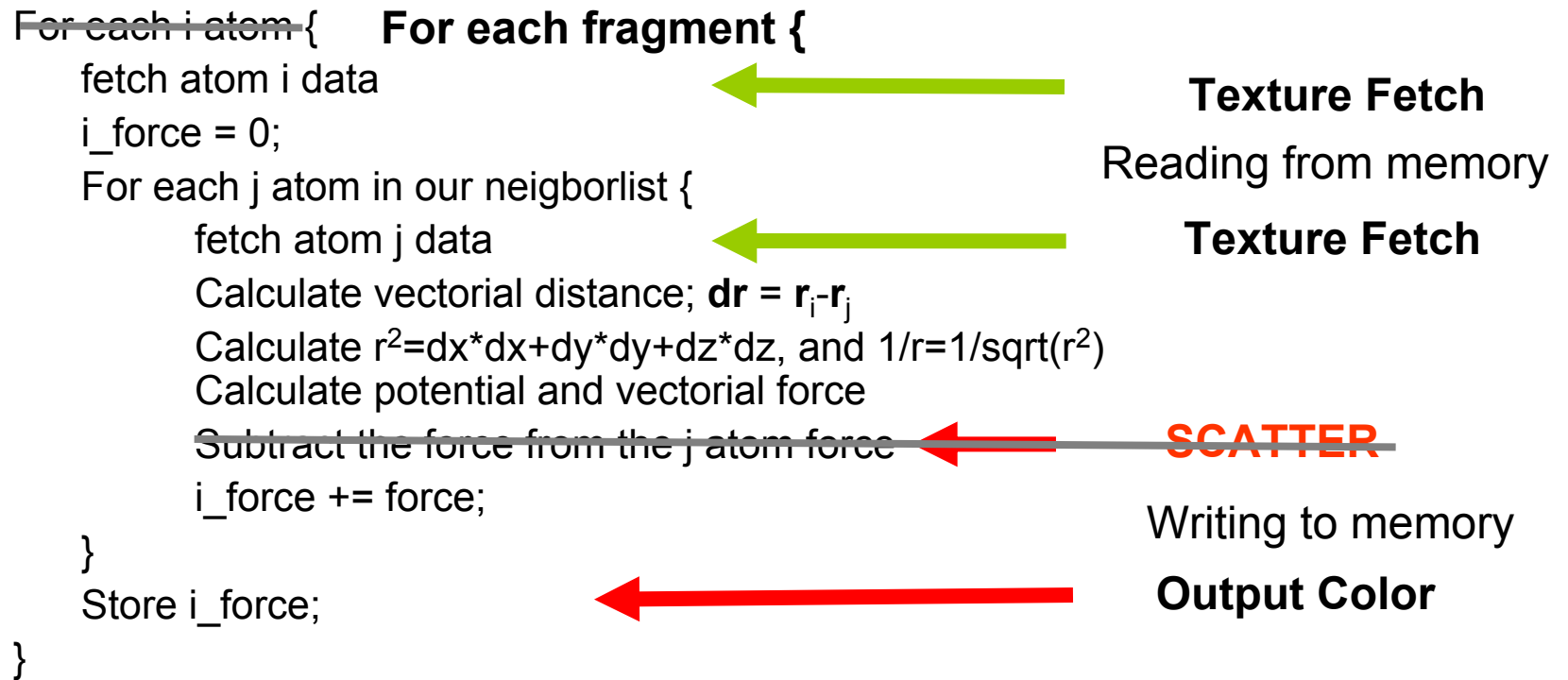
What we do in the inner loop?



What we do in the inner loop?



What we do in the inner loop?



Perform each force computation twice



Inner loop

C Version

```
jnr = jjnr[k];
j3 = 3*jnr;
jx = pos[j3];
jy = pos[j3+1];
jz = pos[j3+2];
dx = ix - jx;
dy = iy - jy;
dz = iz - jz;
rsq = dx*dx+dy*dy+dz*dz;
rinv = 1.0/sqrt(rsq);
rinvsg = rinv*rinv;
rinvsix = rinvsg*rinvsg*rinvsg;
tjA = ntiA+2*type[jnr];
vnb6 = rinvsix*nbfj[tjA];
vnb12 = rinvsix*rinvsix*nbfj[tjA+1];
vnbtot = vnbtot + vnb12-vnb6;
qq = iqA*charge[jnr];
vcoul = qq*rinv;
fs = (twelve*vnb12-
      six*vnb6+vcoul)*rinvsg;
vctot = vctot + vcoul;
tx = dx*fs;
ty = dy*fs;
tz = dz*fs;
fix = fix + tx;
fiy = fiy + ty;
fiz = fiz + tz;
```

Cg Version

```
jnr = fltex1D (jjnr, k);
j = f3tex1D(pos, jnr);

d = i - j;

rsq = dot(d, d);
rinv = rsqrt(rsq);
rinvsg = rinv*rinv;
rinvsix = rinvsg*rinvsg*rinvsg;
tjA = ntiA+2*fltex1D(type, jnr);
vnb6 = rinvsix*fltex1D(nbfj, tjA);
vnb12 = rinvsix*rinvsix*fltex1D(nbfj, tjA+1);
vnbtot = vnbtot+vnb12-vnb6;
qq = iqA * fltex1D(charge, jnr);
vcoul = qq*rinv;
fs = (twelve*vnb12-
      six*vnb6+vcoul)*rinvsg;
vctot = vctot + vcoul;
t = d * fs;

fi = t;
```



Challenges

- Scalar inner loop code
 - Solution: Perform 4 force calc per loop iteration
- Duplicate force calculations
 - Bad: 2x computation than CPU
 - Good: Much less bandwidth!!!
 - Don't have to output partial forces
 - Overall bandwidth much more expensive than compute on GPUs
- Inner loop unrolling
 - 20 interactions before instruction limit

