Recitation 2.1

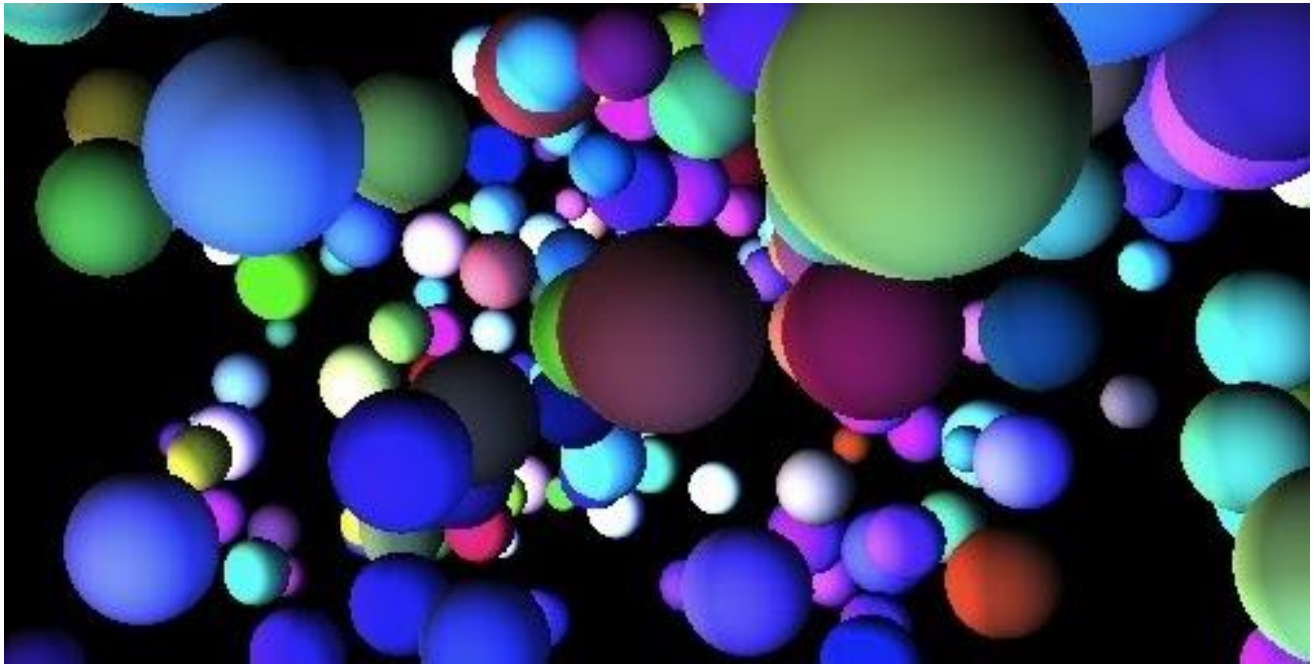
March 12, 2025

Check off:

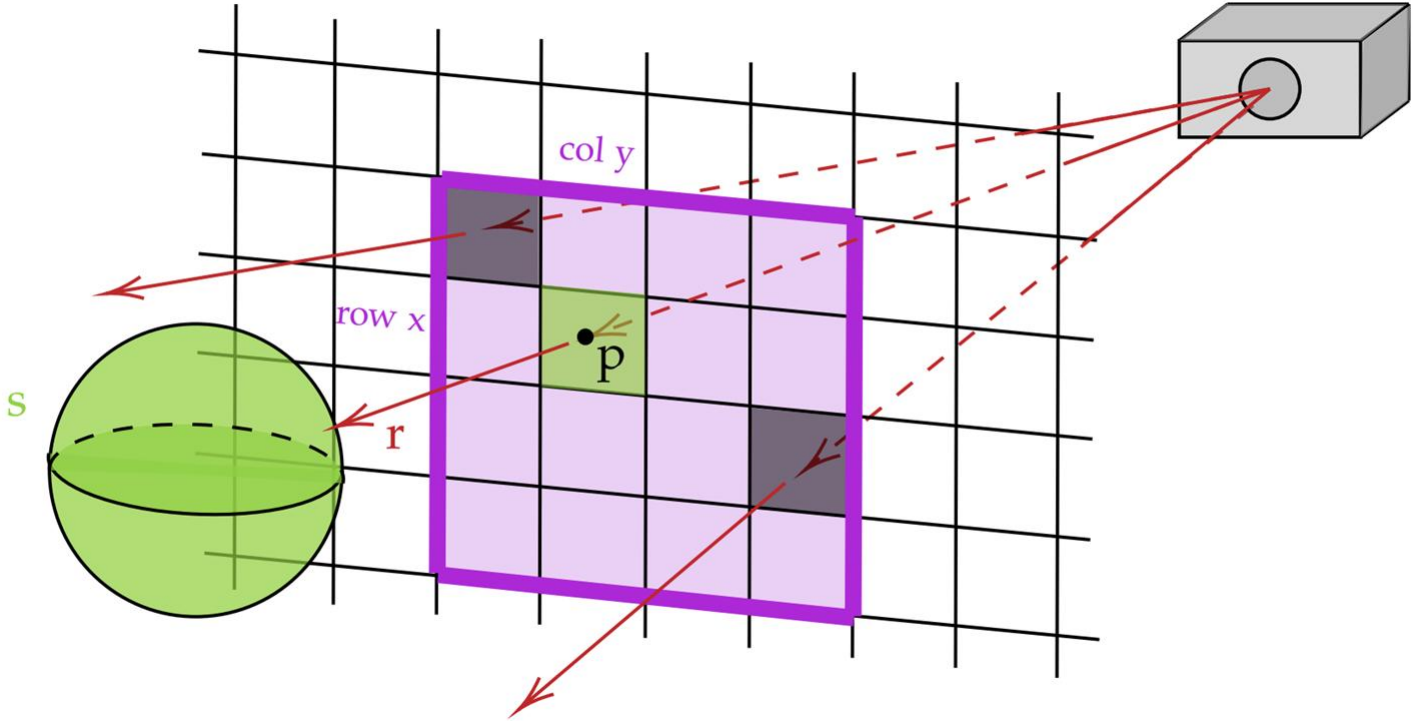
Clone your project and Run :

```
make clean; make
./bin/ref-test tiers/1/s tiers/1/r
./bin/find-tier
```

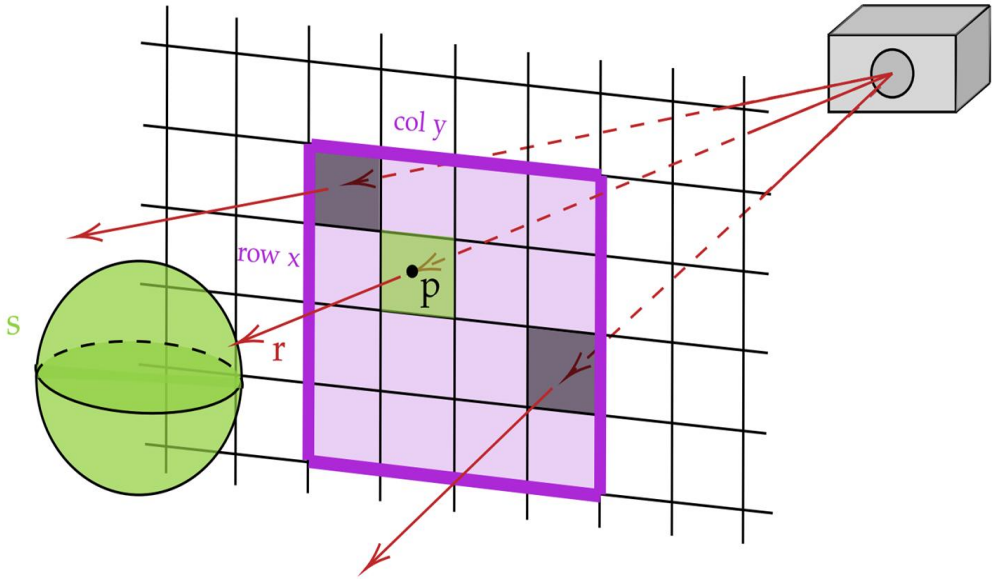
Project 2 : N body simulation



Ray Tracing

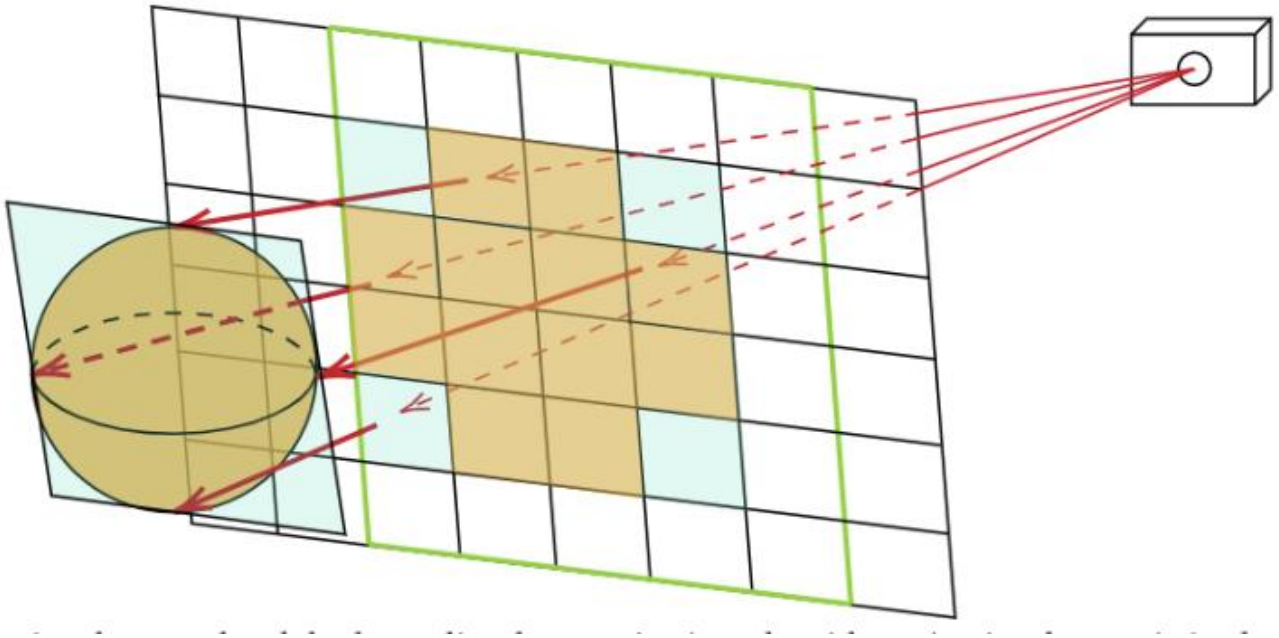


Elements



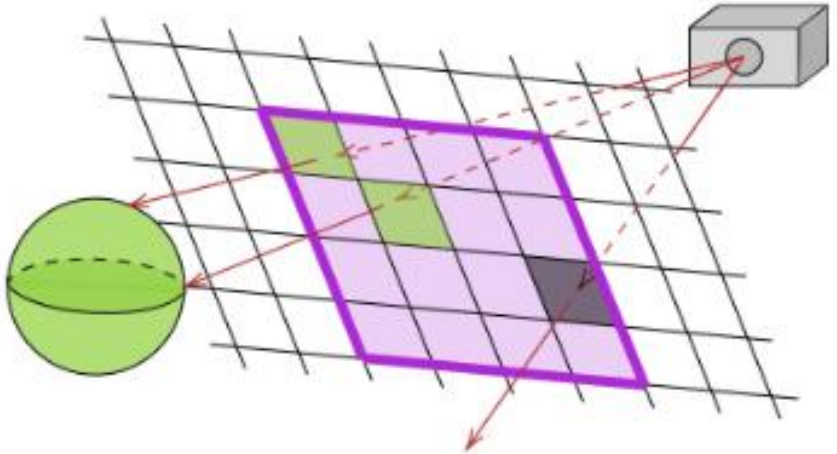
1. Origin (O)
2. Eye location (e)
3. Image Plane (can be inclined to the eye)
4. Plane orthonormal vectors (u and v)
5. Sphere : radius and center
6. Output Image is a resolution x resolution (square) image of pixels
7. Pixels have a fixed width, given by viewport/resolution

Bounding Boxes



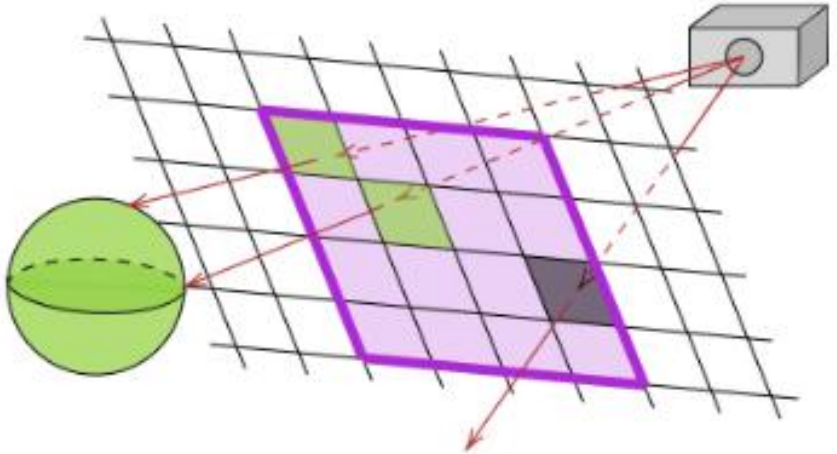
3 coordinates (2 spaces)

- X, Y, Z of the spheres
- X, Y, Z of the coordinates occurring on the plane (same units as spheres)
- I, J of the image (labeled x, y in the code)



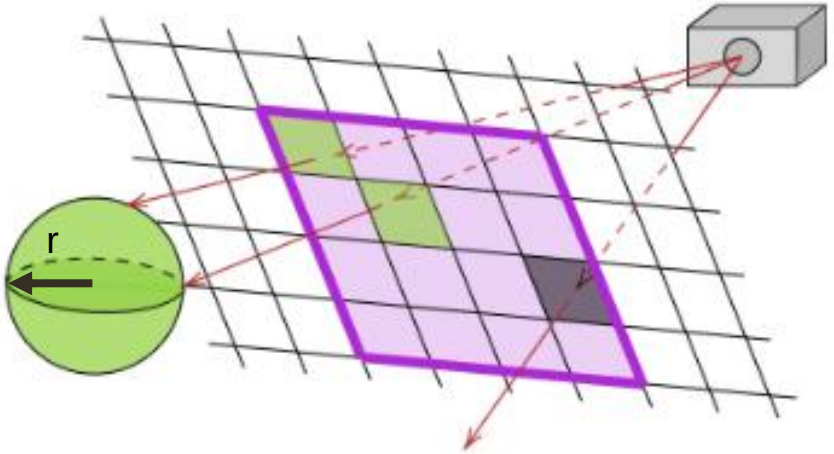
How do we get the pixel value at I J? (initially)

- Given an image pixel location (I, J)
- Find the point in the 3D plane that corresponds (calculate ray from the eye)
- Determine if the ray hits a sphere
    - Complicated math



A different way (better?)

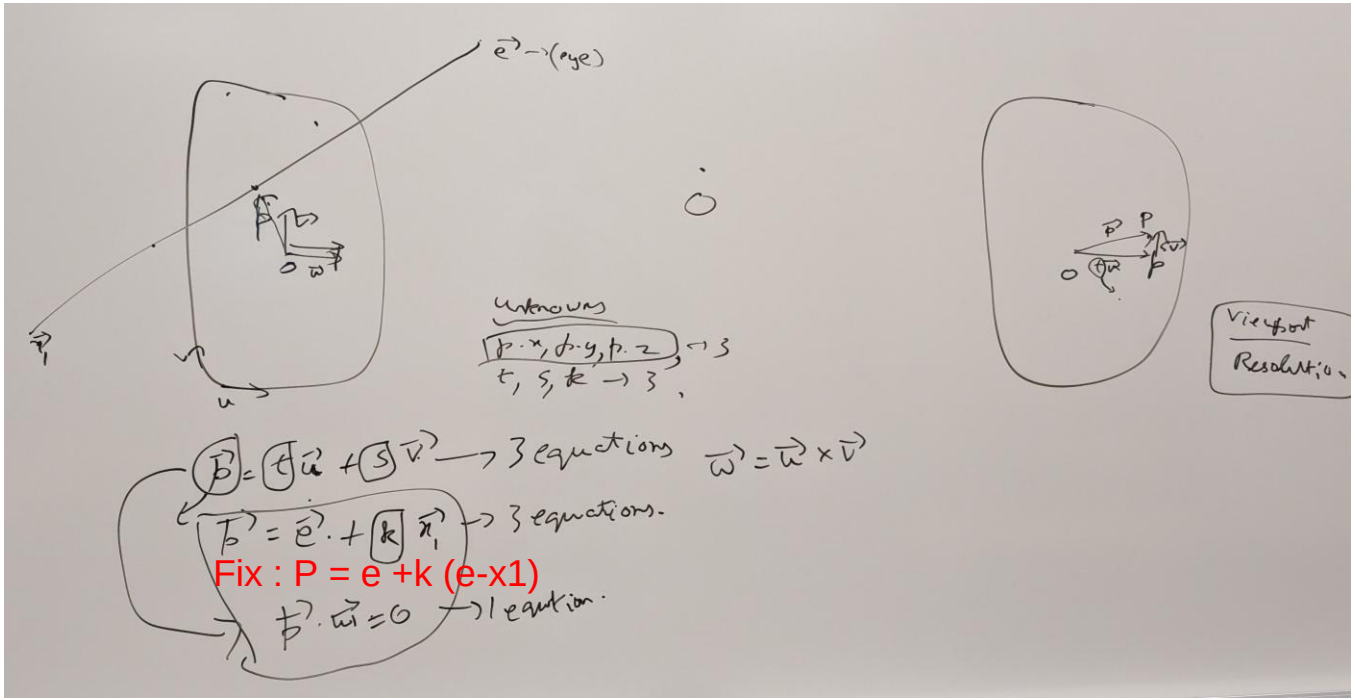
- Given a sphere location, get 8 points making a bounding cube
  - Ex: Given r and (x, y, z) some corners are (x+r, y+r, z+r), (x-r, y-r, z-r), (x+r, y-r, z-r)
- Project those points onto the plane
- For the bounding box of the cube in the image, check if the points intersect the sphere
  - If so, color the pixel



Project the cube onto the plane

- Given points a, b find where they intersect a plane through the origin with orthonormal vectors u, v
- $\overleftrightarrow{ab} = \vec{b} + c * \left(\vec{a} - \vec{b}\right)$ for any number c
- Plane = $t\vec{u} + s\vec{v}$ for any numbers t, s
  - Also given by $p \cdot \vec{w} = 0$ where $\vec{w} = \vec{u} \times \vec{v}$
- 3 unknowns, 3 equations (one each for x, y, z components)
- Solve to get the 8 points and get a bounding box of those 8 points
- You can then have a bigger bounding box with just 4 points

Vector Algebra for Rendering Explained



Other Possible Render Optimizations

- Avoid redundancy in light diffusion on material calculations (some part is fixed for each sphere)
- Use quicksort or other efficient sorting algorithms to sort spheres
- Pre-compute origin_to_pixel values. (Identifying large amount of work happening on adjacent values can then be vectorized)

Some Possible Simulate Optimizations

- Redundant collision checks and force of gravity computations
- Try having a bounding box for collision of a sphere over an interval – If nothing intersects this box, the sphere does not collide with anyother sphere for the current iteration of simulation

Parallelization

- Render – Easy to parallelize pixel wise computation (the majority of work)
- Simulate – Can have a race condition!

Force calculation with redundancy

```
for (int i = 0; i < N; i++) {
    for {int j = 0; j < N; j++) {
        if (i != j) force[i] += f(i, j);
    }
}
```

Force calculation without redundancy

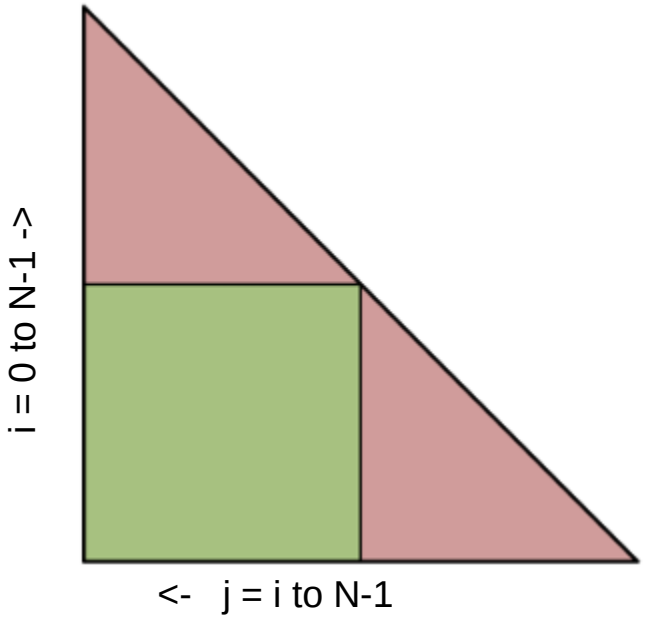
```
for (int i = 0; i < N; i++) {
    for (int j = i+1; j < N; j++) {
        force = f(i, j);
        force[i] += force;
        force[j] -= force;
    }
}
```

Parallelization of Simulate

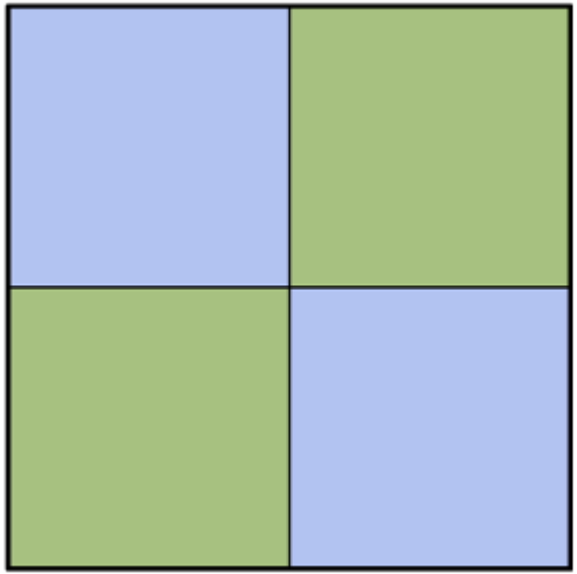
- Iteration space of :

```
for (int i = 0; i < N; i++) {
    for (int j = i+1; j < N; j++) {
        // ...
    }
}
```

Triangles can be run in parallel.
Rectangle should be run
before/after the triangles.



Parallelization of Simulate - 2



When executing for the rectangle, it can further be parallelized (diagonally opposite ones execute together)

Project 2 Guidelines

- **Tier for Guaranteed B in Beta : 47**
- Your code must match the behavior of the original code exactly!
  - Beware of the lack of floating point associativity!
  - No -ffast-math
- No determinacy races (including benign races)
  - Cilksan should report no races
- Intrinsics for atomic operations and locks are not allowed
  - Ex. the compare-and-swap intrinsic

Check off:

Clone your project and Run :

```
make clean; make
./bin/ref-test tiers/1/s tiers/1/r
./bin/find-tier
```