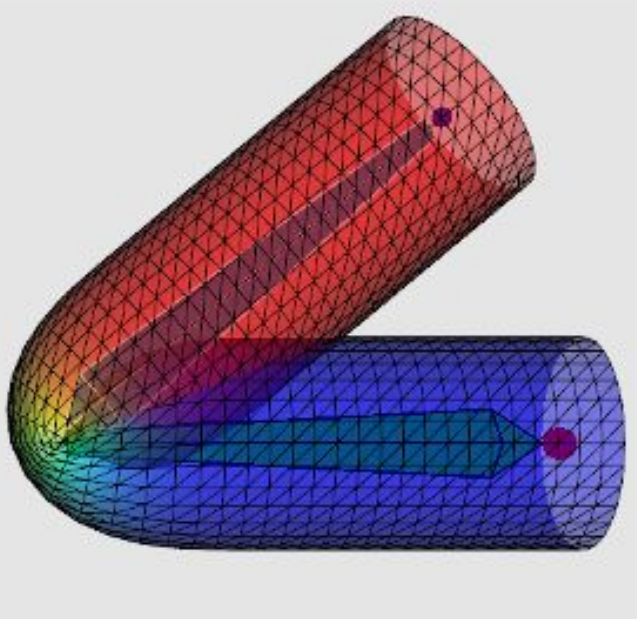
A4.5: Skinning & Particles

- Linear Blend Skinning
- Particle Simulation

- Linear Blend Skinning
- Particle Simulation

Linear Blend Skinning

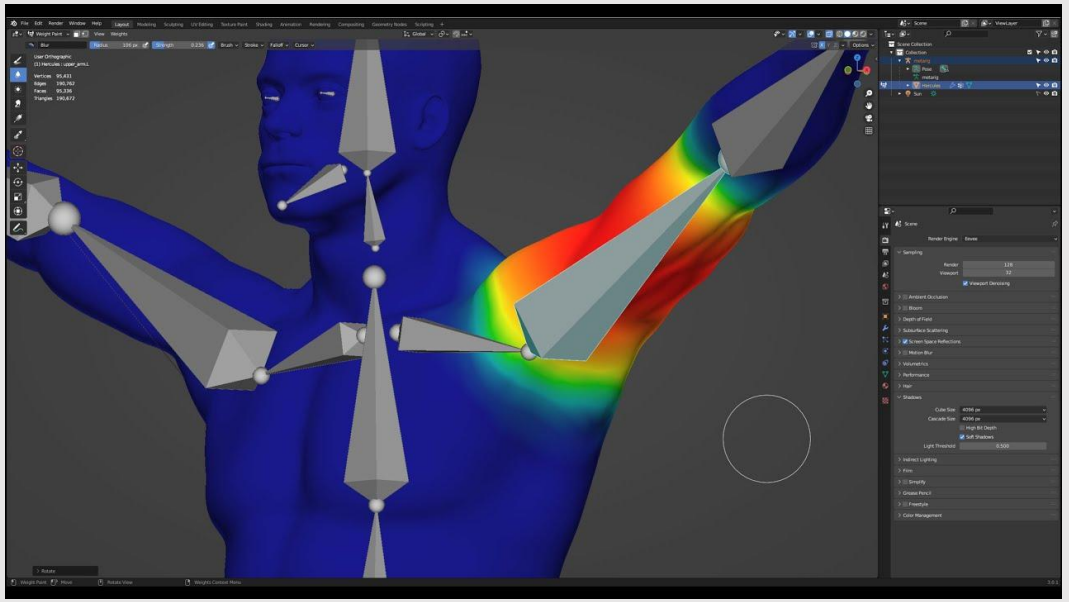
Motivation: We control how much the mesh geometry moves as the bones rotate by assigning each vertex a "weight" per bone



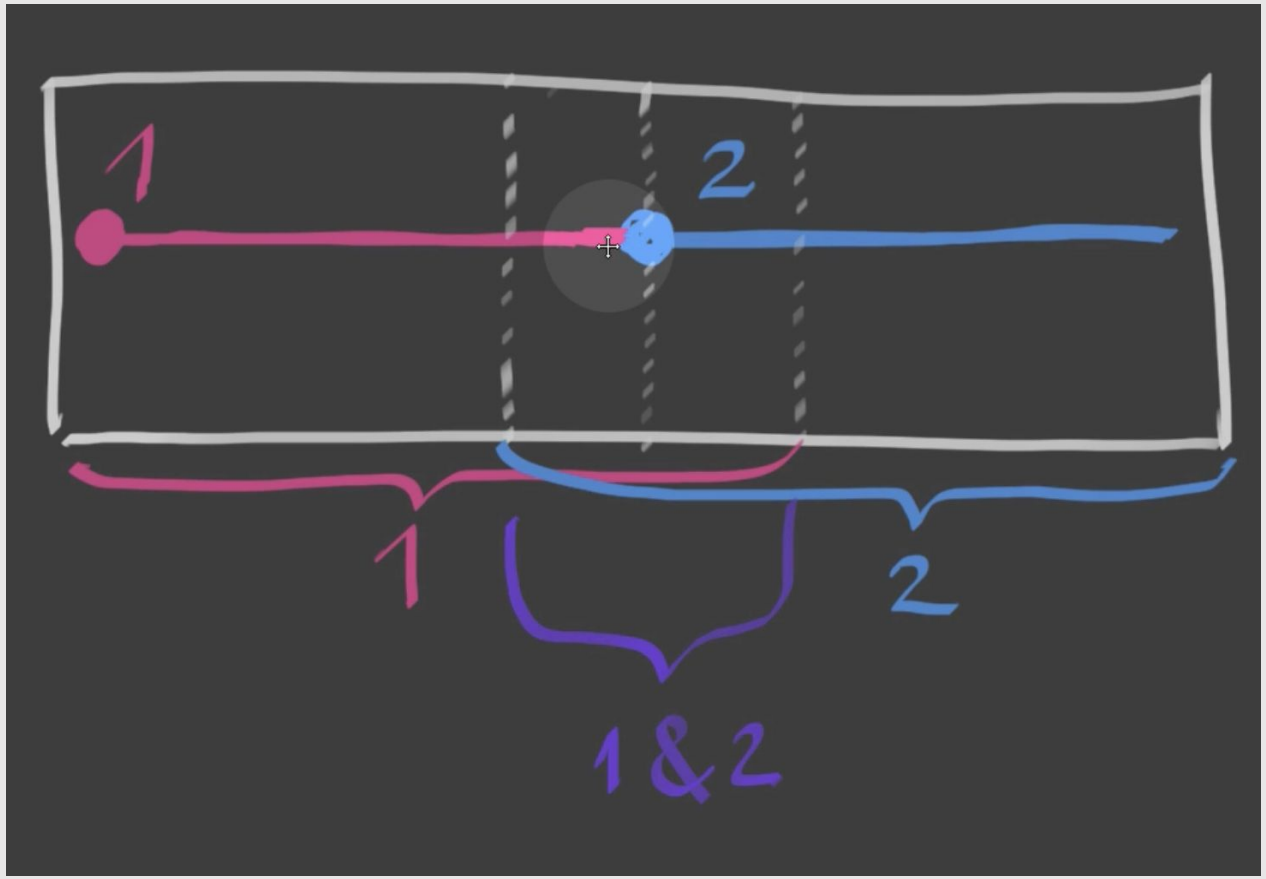
Linear Blend Skinning

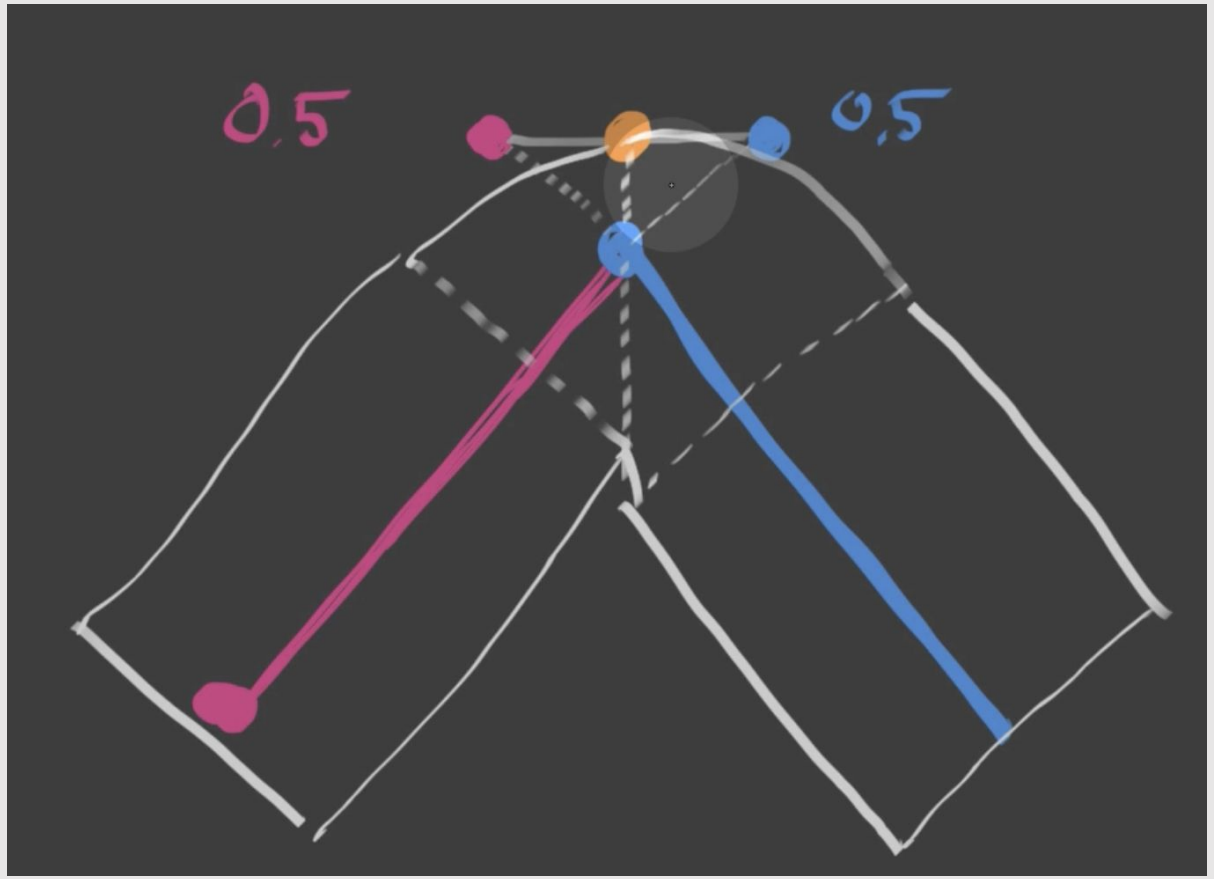
There are many ways one can assign these weights

- Manually - assign weights by having an artist “paint” the weights with a 3D program
- Automatically/Algorithmically
  - One method is Linear Blend Skinning, where we assign weights inversely proportional to the distance between the vertex and the bones



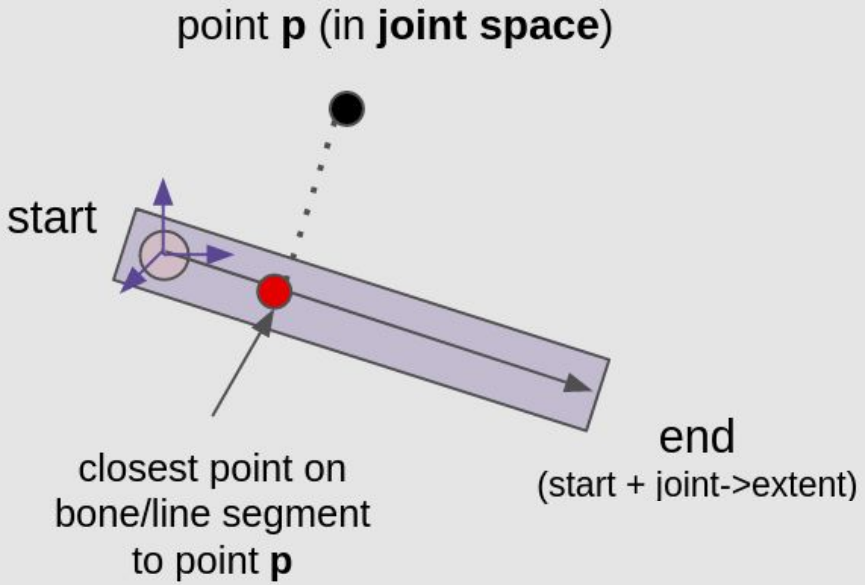
Linear Blend Skinning





Linear Blend Skinning

Weights assigned via inverse distance from vertex to the bone (represented by a line segment) up to a max distance define by bone::radius



For weight of vertex i with bone j,

and distance between the two given by d:

$$\hat{w}_{ij} \equiv \frac{\max(0, r - d_{ij})}{r}$$

Note : need to normalize per vertex so all weights add to one!

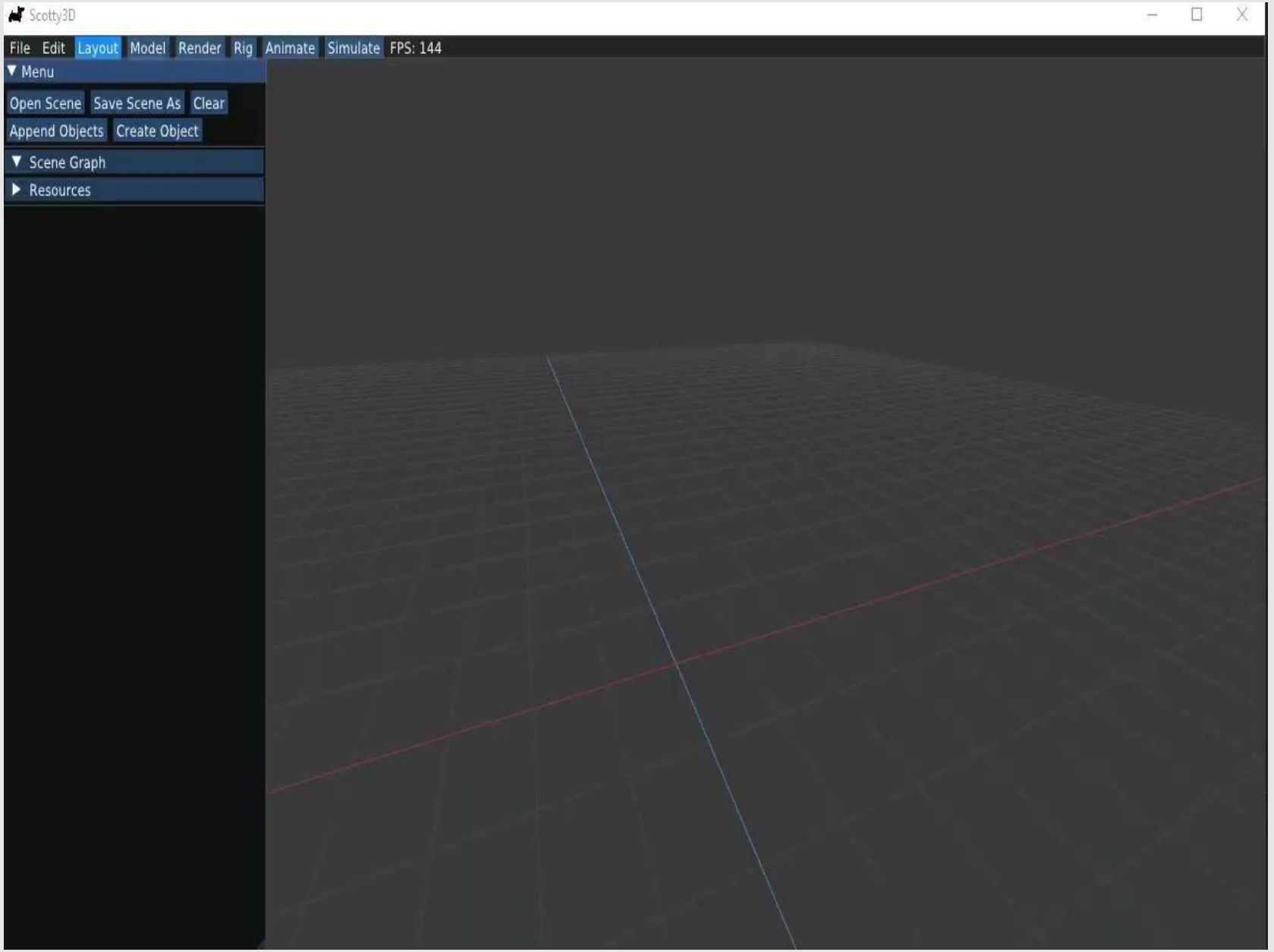
Linear Blend Skinning

New vertex positions are thus a weighted sum of transformations under the bone transformations

- Transformations are from bind space (B) to pose space (P)

$$v_i' = \left(\sum_j w_{ij} P_j B_j^{-1}\right) v_i$$

- Linear Blend Skinning
- Particle Simulation


Scotty3D
File Edit Layout Model Render Rig Animate Simulate FPS: 144
Menu
Open Scene Save Scene As Clear
Append Objects Create Object
Scene Graph
Resources

Particles in Scotty3D

- Non-self-interacting, physics-simulated, spherical particles that interact with the rest of the scene
- Can use basic physics to simulate where a particle will be in the scene at a given time

$$F = ma$$

$$a = \frac{dv}{dt}$$

$$\frac{dv}{dt} = \frac{F}{m}$$

$$\frac{dx}{dt} = v$$

Particles in Scotty3D

- Much easier to update position/velocity at small time-steps then continuously over a large time period
- Forward Euler
  - Can be unstable and does not conserve energy in a system – but that's ok for now
  - Means we evaluate function at our current configuration

$$x_{t+\Delta t} = x_t + v_t \cdot \Delta T$$

$$v_{t+\Delta t} = v_t + a \cdot \Delta T$$

Particle Collisions

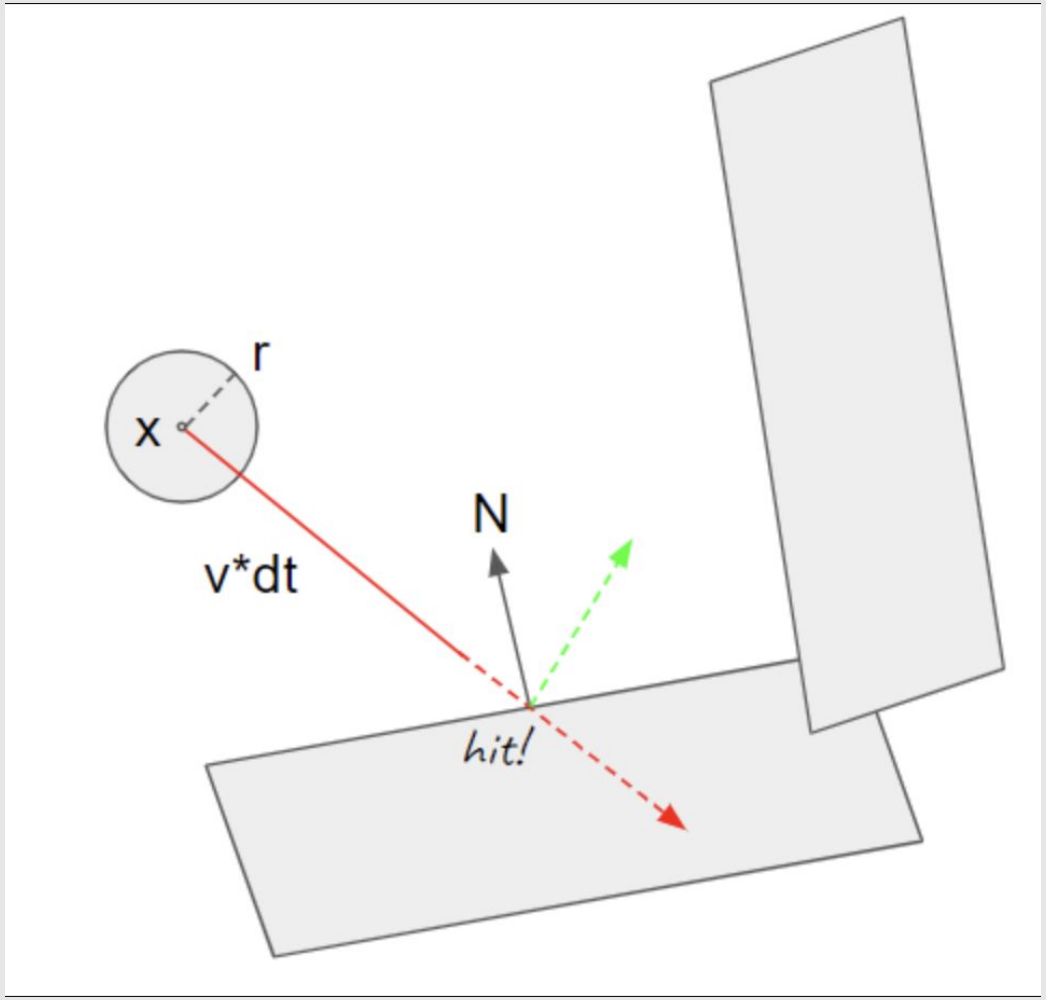
- Can use Scotty3D's ray tracing (that you implemented!) to detect if particle collides with the scene within a timestep
  - Assume all particles collide elastically (i.e. a particle's velocity should be the same before and after a collision, with its direction reflected based on the surface normal)

  - Create a ray based on the particle's position and velocity
  - If the particles hits the scene within the current timestep
    - Reflect the velocity
  - Update velocity and position based on current timestep
  - Repeat until entire timestep is consumed

Particle Collisions

- **If the particles hits the scene within the current timestep**
  - Reflect the velocity

- Note this is not as simple as checking if the ray hits something in the scene



Pseudocode

```
void update_ball(Scene scene, Sphere *ball, float timestep) {

    Ray ray(ball->position, ball->velocity);
    bool hit = scene.hit(ray);

    if (hit)
    {
      bool hit_within_step = //TODO: does the hit happen within the timestep?
      float hit_time =       //TODO: how much time does the hit take?

      if(hit_within_step)
      {
        ball->position += hit_time * ball->velocity;
        ball->velocity = // TODO: what will the new velocity be?
        // We may have another collision, so check again
        return update_ball(scene, ball, timestep - hit_time);
      }
    }
    // Update with forward Euler
    ball->position += timestep * ball->velocity;
    ball->velocity += timestep * scene.gravity;
}
```

Note on Grading/Test Cases

- We only give you a few test cases
- In the past these tests have been flaky
- It’s possible your implementation is 100% correct but a test fails
- Assess for correctness based on tests AND visually via the GUI
  - view expected behavior on Scotty3D docs/writeup
- We will grade your code visually as well
  - come to OH if you want to verify correctness
- Watch out for:
  - particles falling through walls
  - particles intersecting walls before bouncing

Notes

- See cmu-graphics.github.io/Scotty3D-docs/guide/animate_mode/ (Scotty3d docs) for instructions on how to make an animation
- Only ONE LATE DAY allowed for A4.5
  - regardless of how many late days you have left
- Final exam will be cumulative
  - Same format as midterm
  - Final review EITHER next recitation or in class Wednesday (TBD)
- All of you are awesome … please apply to TA 15-362 next sem!!!
  - We have a specific form posted on Piazza (we don't look at applications from the CMU CS TA website)