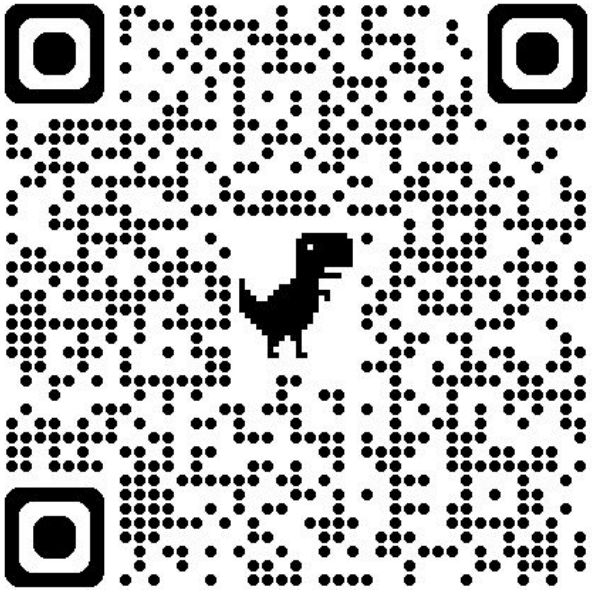
Software Performance Engineering


# Recitation 2.5

Sophia Sun
Tuesday, December 2, 2025

# Attendance!



Good luck on the final!

# Some due dates

- Homework 6 was due Monday, December 1 (yesterday).

- Project 2 Final is due Thursday, December 4.
- Project 3A and 3B are due Monday, December 8.
  - Let me know ASAP if you need any help on these!

- Also, don’t forget there’s a survey in HW6.

# PROJECT 2 FINAL

# Basic Collision Check

```
for (int i = 0; i < state->s_spec.n_spheres; i++) {
  for (int j = i + 1; j < state->s_spec.n_spheres; j++) {
    if (check_for_collision(state->spheres, i, j, &minCollisionTime)) {
      indexCollider1 = i;
      indexCollider2 = j;
    }}}
```

Each `check_for_collision()` call involves computation :

- Euclidean Distance between two spheres (has sqrt)
- Relative velocity of two spheres
- Use relative velocity to find if collision will happen (another sqrt)

**Expensive!**

# How can we improve this?

For each sphere,

- Find the ending position of the sphere for the timestep of the simulation
- Use its starting and ending position, to have a bounding box for the sphere for potential collisions
  - We've talked about how to build bounding boxes

# How can we improve this?

Before collision checks,

- Check if the box can collide with another box by comparing the individual coordinates of the boxes (no square roots and only uses 3 comparisons)
- The above filter provides false positives for collisions but no false negatives – Double check with the original `check_for_collision()` on false positives.

# How to further improve this?

- Notice how the comparisons in the previous technique is such that if checks on even 1 dimension fails, then collision cannot happen.
  - Eg: if not colliding on x-axis, then no need to check for y and z.
- Hence, we can reduce the initial “filter” to 1-dimension checks and check the other dimensions as and when necessary.

# The Sweep & Prune Algorithm

- A good reference: Sort, sweep, and prune: Collision detection algorithms
- The algorithm has the following three parts:
- Sort: Sort the box coordinates based on 1 dimension (say X axis)
- Sweep: Filter out impossible collisions by inexpensive box comparisons
- Prune: Double-check to ensure correctness (by removing false positives)

# The Sort

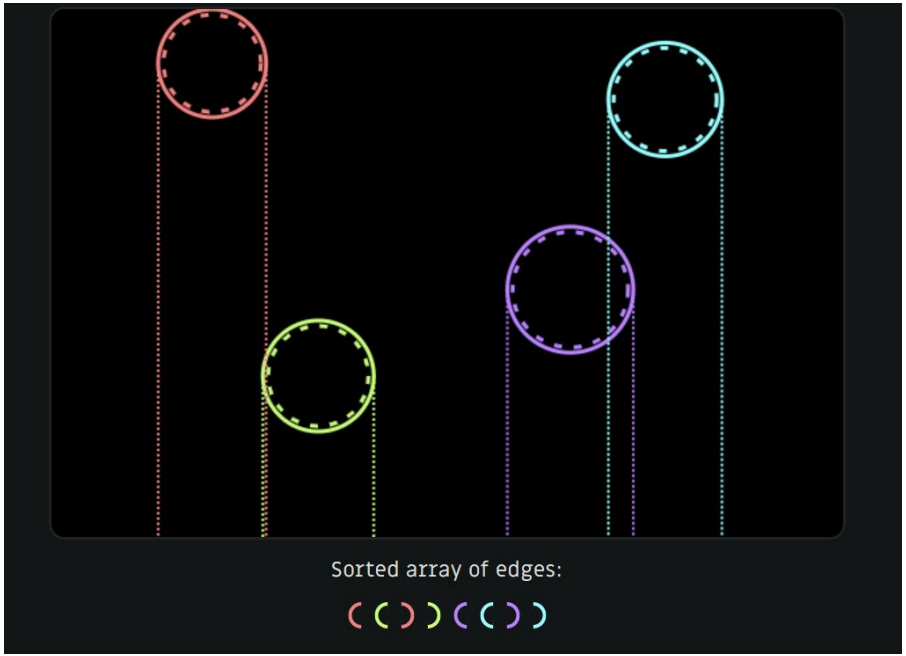



Image from : https://leanrada.com/notes/sweep-and-prune/

- Sort the spheres using a box that entails the spheres possible locations within the timestep.
- (Figure shows a sphere but think of them as boxes for our project)

# The Sort

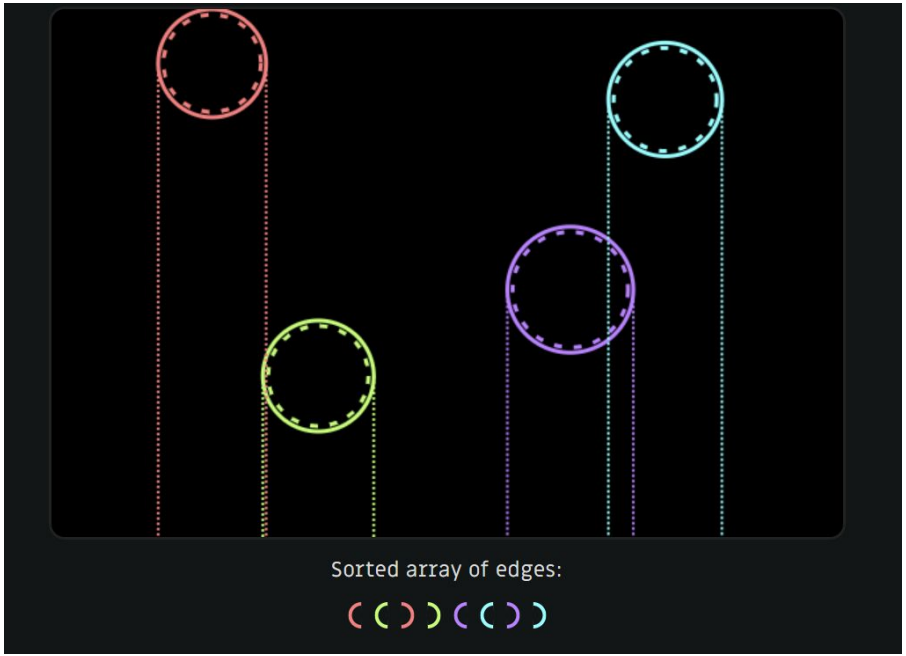



Image from : https://leanrada.com/notes/sweep-and-prune/

- For each sphere, eliminate collision checks with coordinates further away from the last check that cannot collide.
- Example: Red box cannot collide with purple, hence we don't have to check collision with blue)

# The Prune

- Now that 1 dimension has filtered out a lot of checks,
- Check for the other dimensions with the box comparison.
- Those that pass, maybe false positives (may not actually be colliding) – Use the default check only on these.

# Why is it that it can be faster?

- The worst case maybe the same – every box collides with every other box
- But when the balls are well scattered (happens to many pairs in this project), it reduces the number of checks involving square root operations significantly.
- I suggest reading the reference that beautifully explains the algorithm with better visualization (Don’t miss its second page!) : https://leanrada.com/notes/sweep-and-prune/

# Next steps?

- Beyond this optimization, run perf reports to see which of the two (render, simulate) is worth focusing your optimizations on.
- If only the recitation optimizations have been performed, it is very likely to be render.
- Use perf reports to guide you on what could be improved.
- Vectorization can be very powerful in many steps involving the image computation.
- Aim to reach the bonus tier!

# Some technique worth trying

- Improve Render by using what you've learnt from proj1 and homeworks.
  - Vectorization
- Bentley Rules (feel free to try out anything helpful)
  - Precomputation
  - Subexpression Elimination
  - Replace expensive algebraic expressions
  - Hoisting
- Be guided by perf reports for further optimizations.

# CODING TIME!