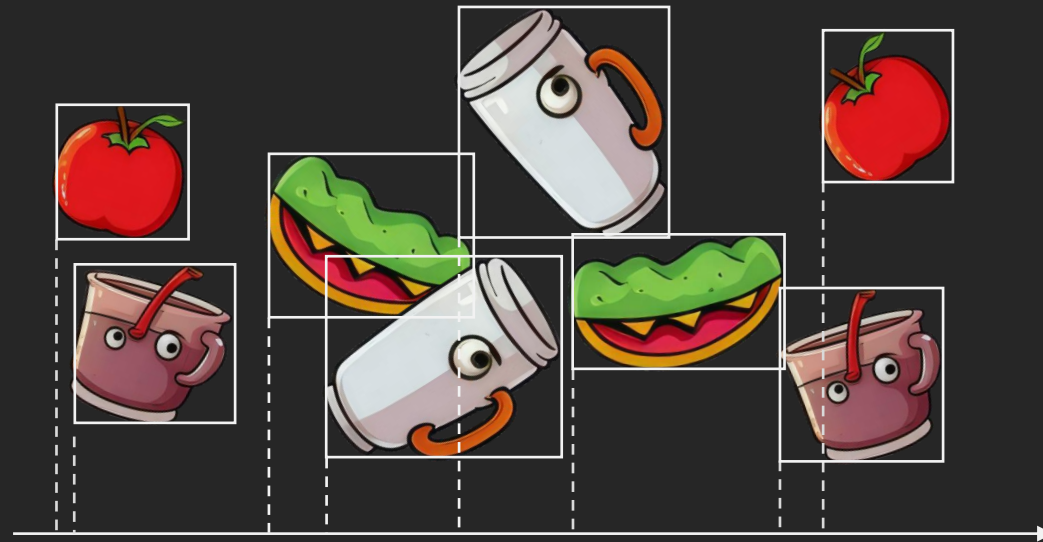


# Sweep and Prune



Matthias Müller, Ten Minute Physics

[matthiasmueller.info/tenMinutePhysics](http://matthiasmueller.info/tenMinutePhysics)



# Problem

- Find the collisions between objects

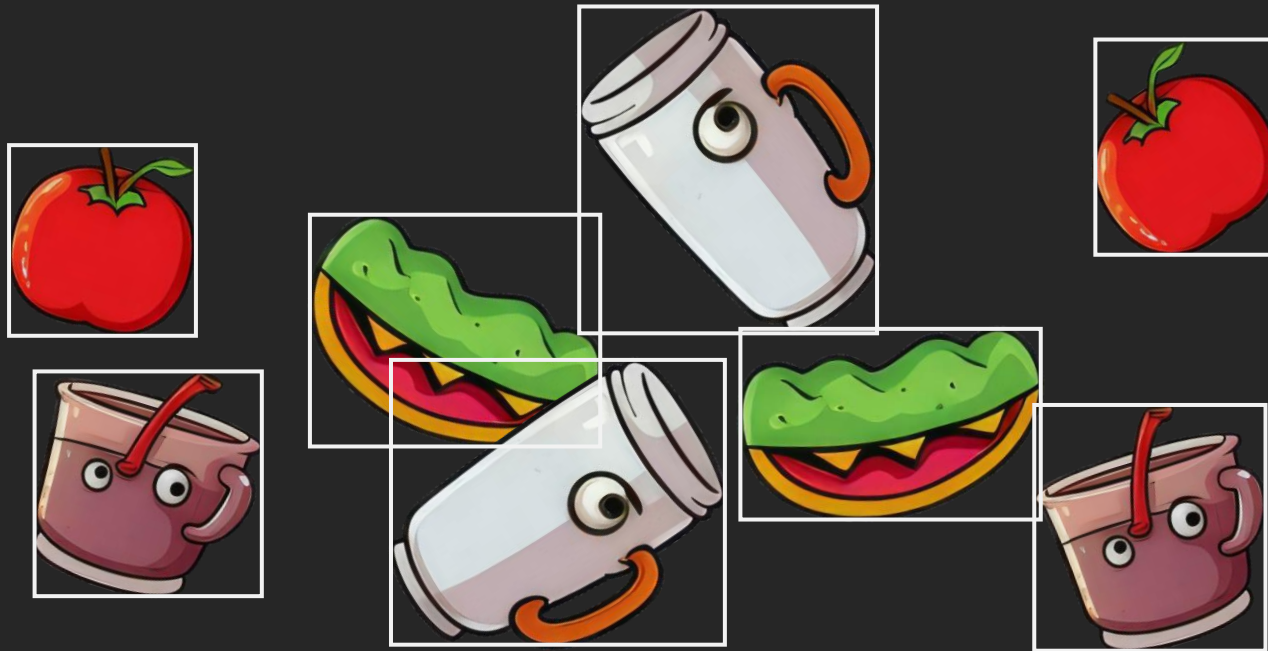


- Expensive for objects with detailed surface



# Idea

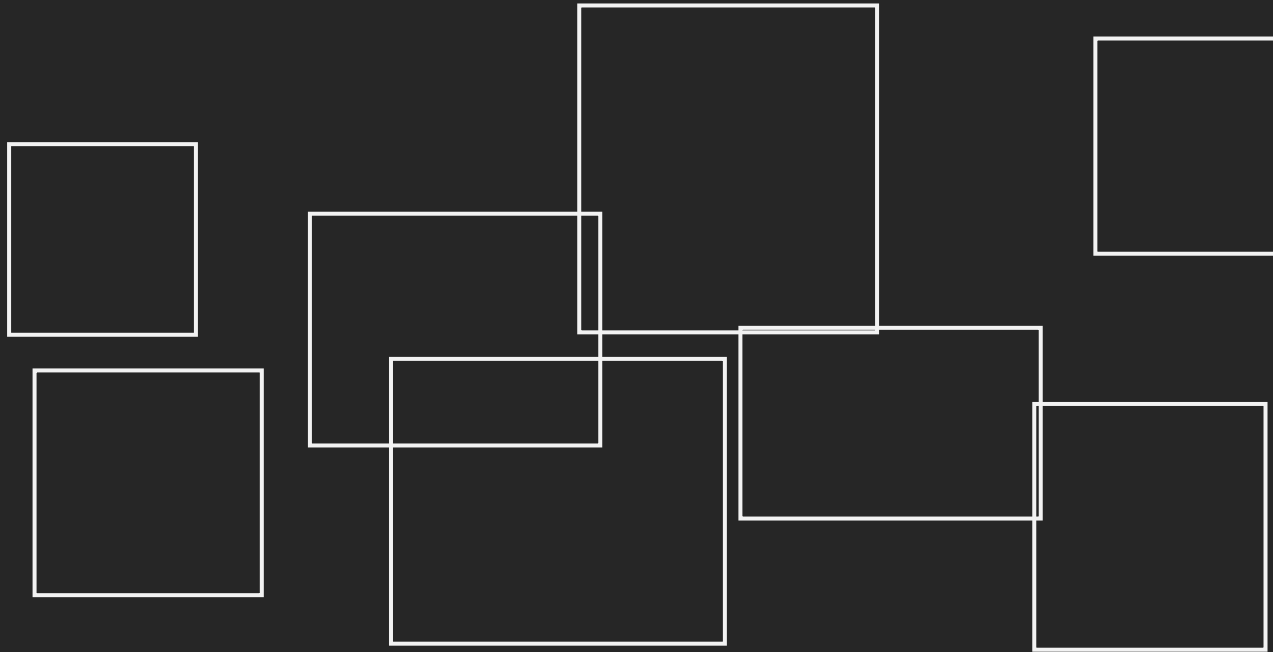
- Broad phase: Replace objects by their bounding boxes



- Narrow phase: Only test pairs with overlapping bounding boxes



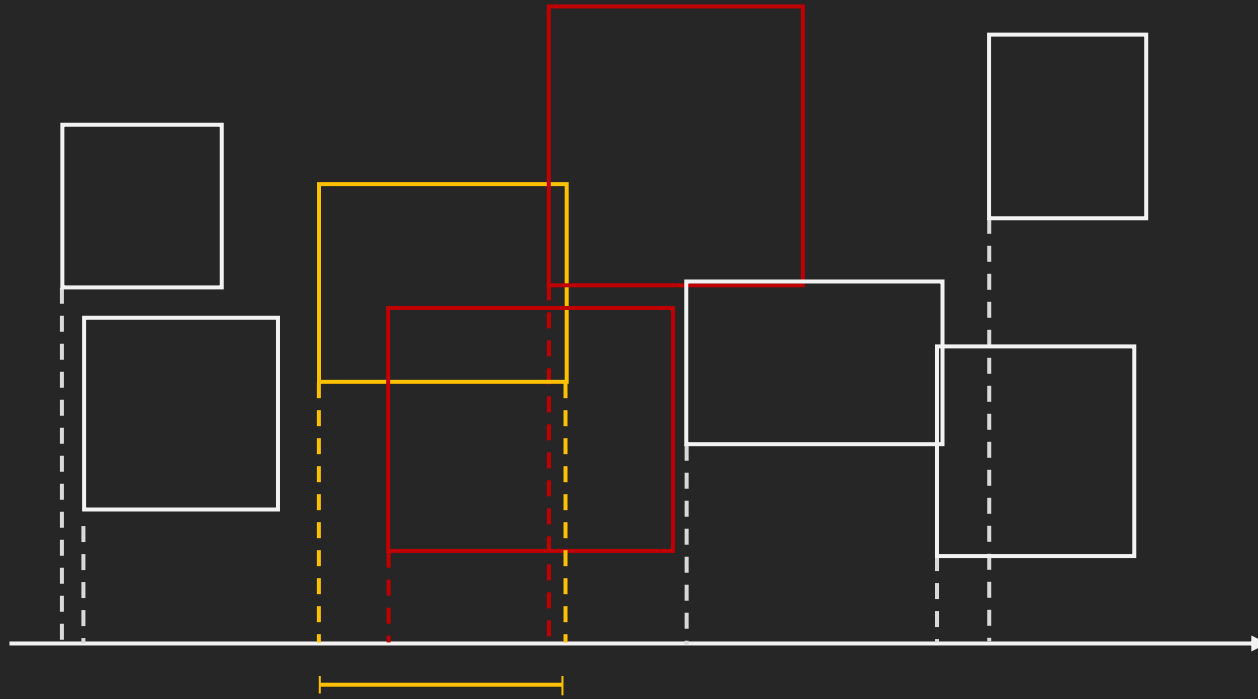
# Brute Force



- Test all pairs
- For 10,000 boxes: 100,000,000 / 2 tests!



# Sweep and Prune



- Choose an axis
- Sort boxes by their left edge
- Each **box** only tests **boxes in front** whose **left edge** is smaller than its **right edge**



# Ten Lines of Code!

```
function sweepAndPruneCollisions(bodies)
{
  const sortedBodies = bodies.sort((a, b) => a.left - b.left);

  for (let i = 0; i < sortedBodies.length; i++) {
    const body1 = sortedBodies[i];

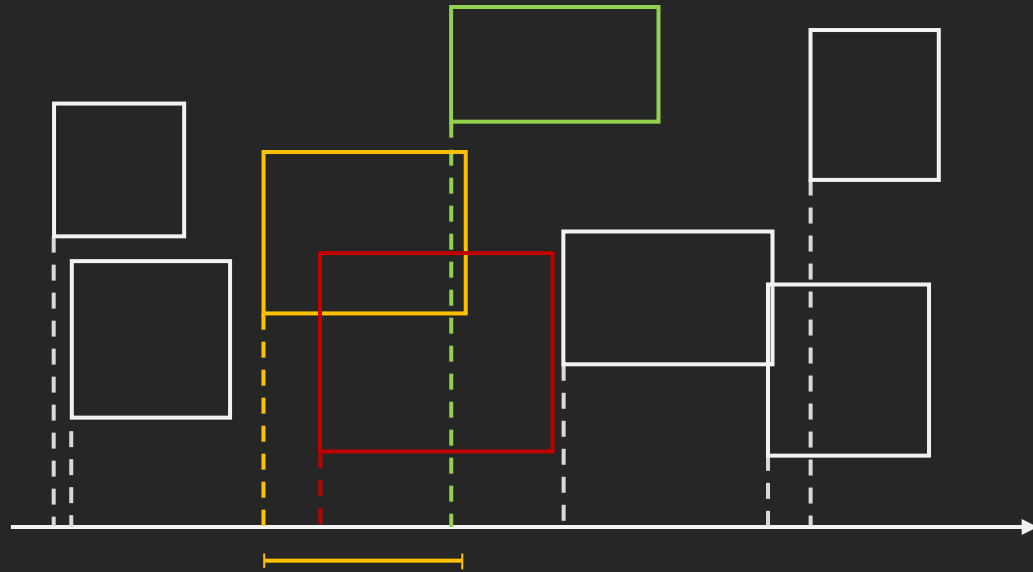
    for (let j = i + 1; j < sortedBodies.length; j++) {
      const body2 = sortedBodies[j];

      if (body2.left > body1.right)
        break;

      if (body2.bounds.overlap(body1.bounds))
        solveCollision(body1, body2);
    }
  }
}
```



# Remarks



- Axis overlap is conservative: Additional full bounding box test is needed
- Choose the longest axis of the bounding box of the scene
- Sort complexity  $O(n \log(n))$ ,  $n = 10,000 \rightarrow n \log(n) \approx 100,000$
- Actual number of tests depends on the configuration of the objects



# Demo



Written to 100% by Claude!





# Conversation with Claude



**MM** I have another implementation of sweep and prune without an active list. You iterate through all spheres as you do. Then for each sphere you iterate through the spheres after the current sphere until their left position is larger than the right position of the current sphere. Is this correct?

Yes, that's absolutely correct! That's actually a simpler and equally valid implementation of sweep and prune. Let me modify the code to show this version:



**Falling Spheres Simulation**

Click to open website

This version is more elegant because:

1. It eliminates the need for an active list or temporary storage of potential collisions
2. It uses a simple early exit condition: if a sphere's left edge is beyond the current sphere's right edge, we can stop checking
3. It maintains the  $O(n \log n)$  complexity from the sort while being more straightforward to understand
4. Each sphere only needs to check forward in the sorted list, as any previous collisions would have been handled already



**Thanks for watching!**

**See you in the next tutorial...**

