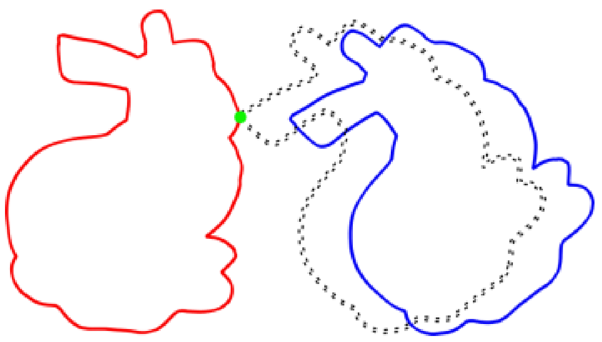
Collisions and spheres

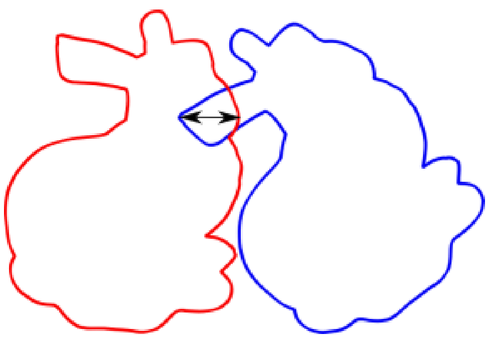
The main problem

when we represent objects in computer graphic the actual representations are seen as a set of polygones, like triangles for meshes. the big problem arrives when we want to compute the interception between two objects, maybe also in movement. there are 2 main problemns related to this computation:

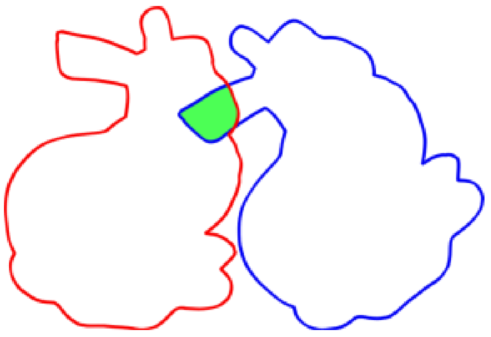
- the objects moves discretely, this means that we cannot compute a “single point collision” as soon as we obtain the collision, but we could find ourselves with a collision when it’s already too late



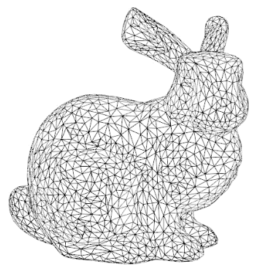
Continuous collision detection

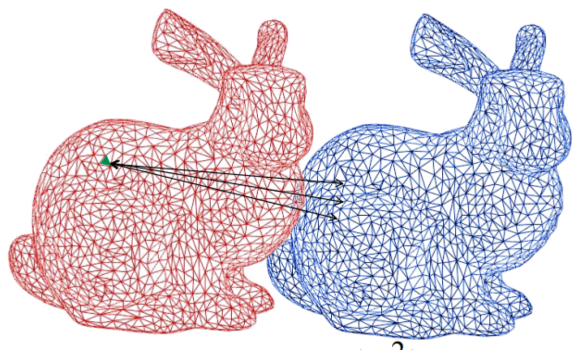


Translational penetration depth



- to compute a collision we have to manually check every triangle, bringing us to a complexity of $O(\prod_{n_i=\#\text{tri of obj i}} n_i)$, in the simple case of two objects with n triangles each $O(n^2)$(simple yes, but always not good)

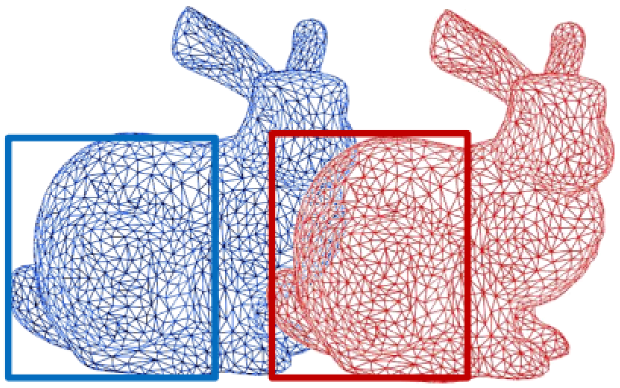


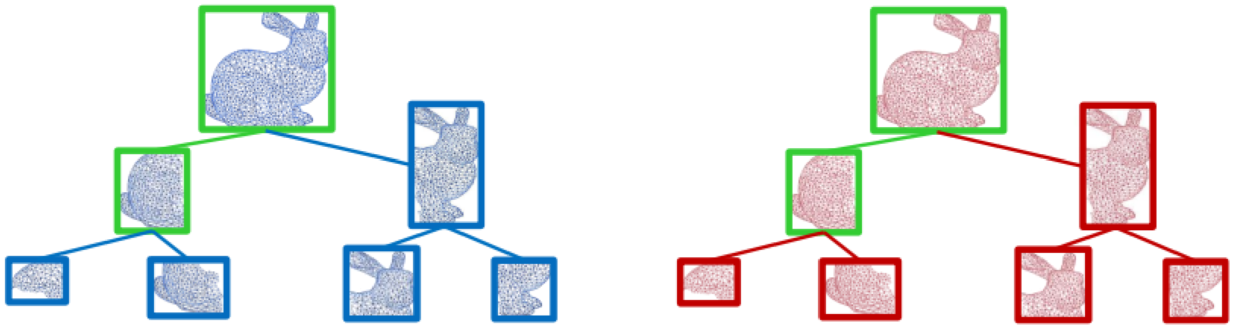


Bounding volumes hierarchies

Acceleration data structures

one possible solution is to use bounding boxes, in this way we can compute a “pre-collision test” between the bboxes of the objects, and only proceed in a tree-like structure until we reach a single mesh or we discover we are in a false positive





```
bool checkCollision( BV A, BV B )
  if A and B are Leaves then
    return checkPolygons(Polygon of A, Polygon of B)
  else
    for all the Children Ai of A do
      for all the Children Bi of B do
        if( overlap (Ai , Bi))
          return checkCollision(Ai , Bi)
  return false
```

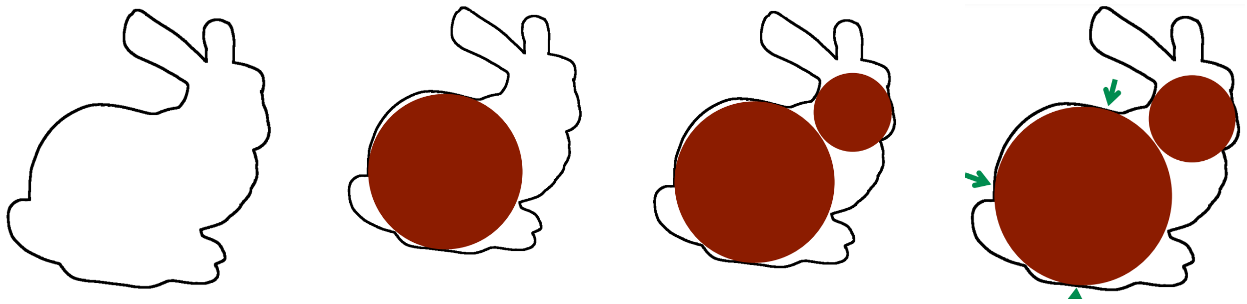
Inner spheres trees

The main idea behind inner spheres is to build a tree of spheres inside the object, instead of approaching from the outside with the Bboxes

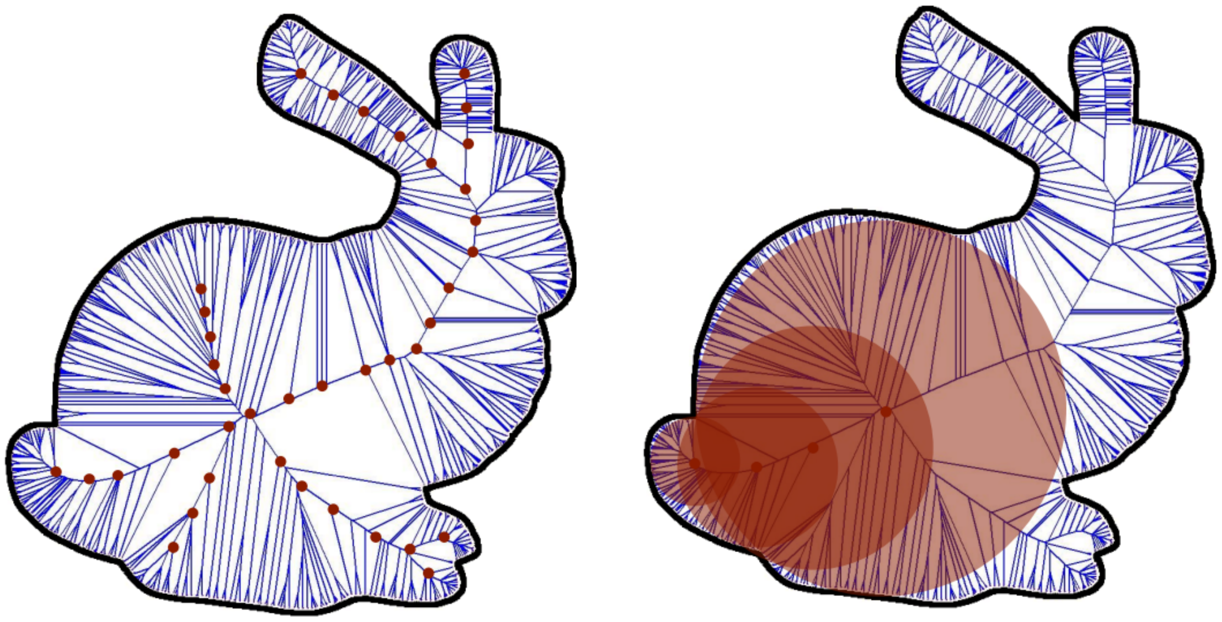
now the problem is only: **How do we pack the spheres?** they need to take the biggest possible space into the object, be all inside it and not overlap, neiderthenless, the computation must be as fast as possible, efficient and light*(no, we are not asking too mutch ndr)*

Basic idea

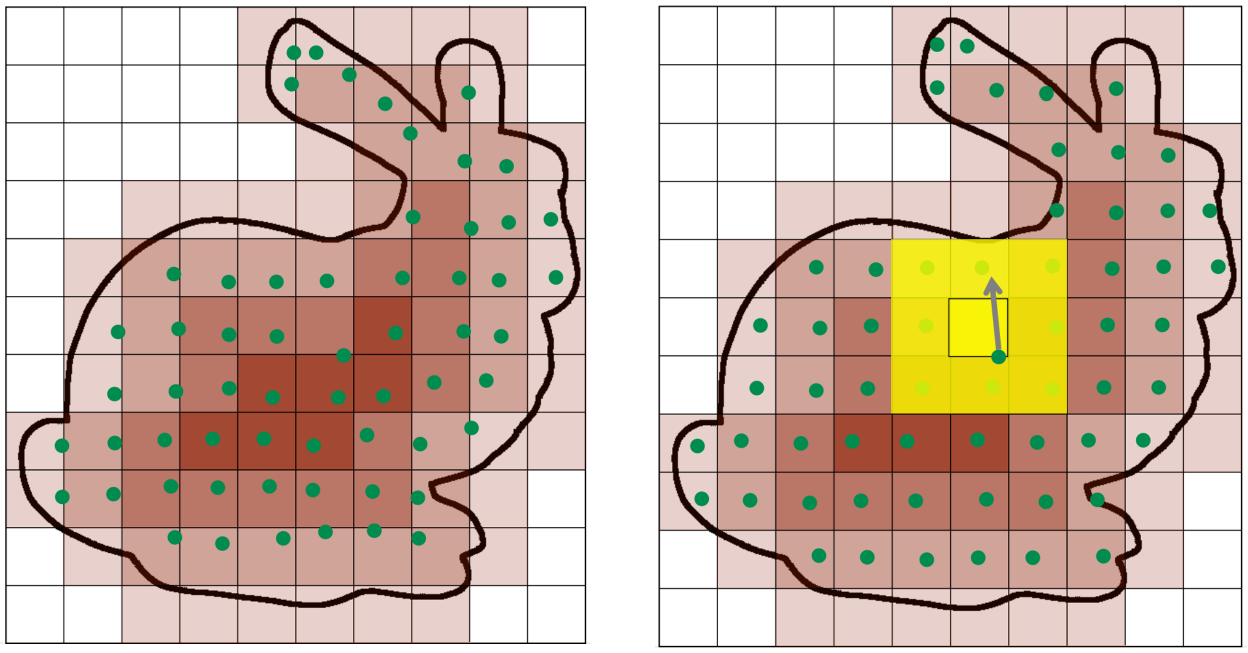
the first idea is to take a point inside the object, and start creating a sphere until it touches an edge, then move it if possible until it's not possible anymore to enlarge the radius, at that point we move to the next sphere and so on, until we are satisfied



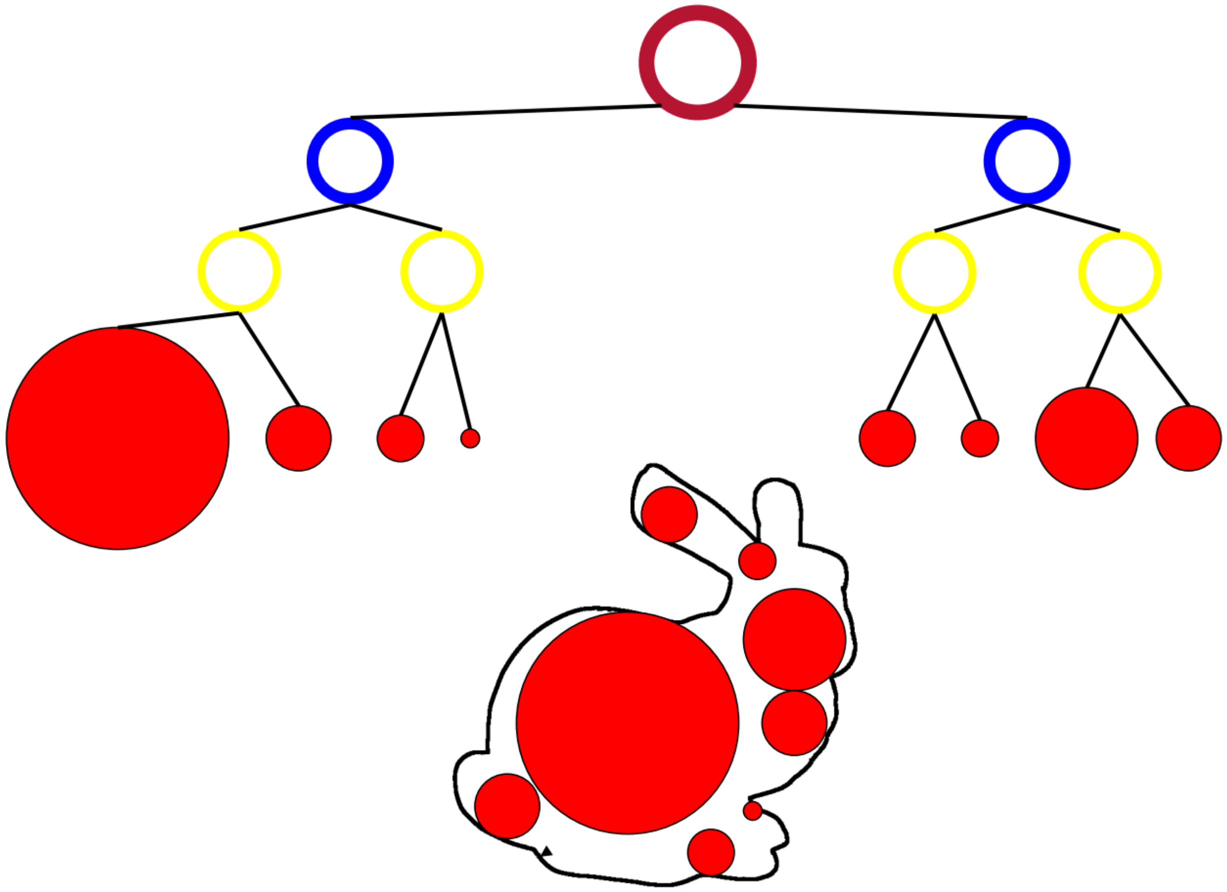
Problem: where do we need to place our centers? it's not immediate, since we could take any point, but we cannot be sure that the distribution is optimal. a solution is to create a voronoi diagram of the inside with the verteces, then plce the spheres on the verteces of the voronoi with the highest nodes count



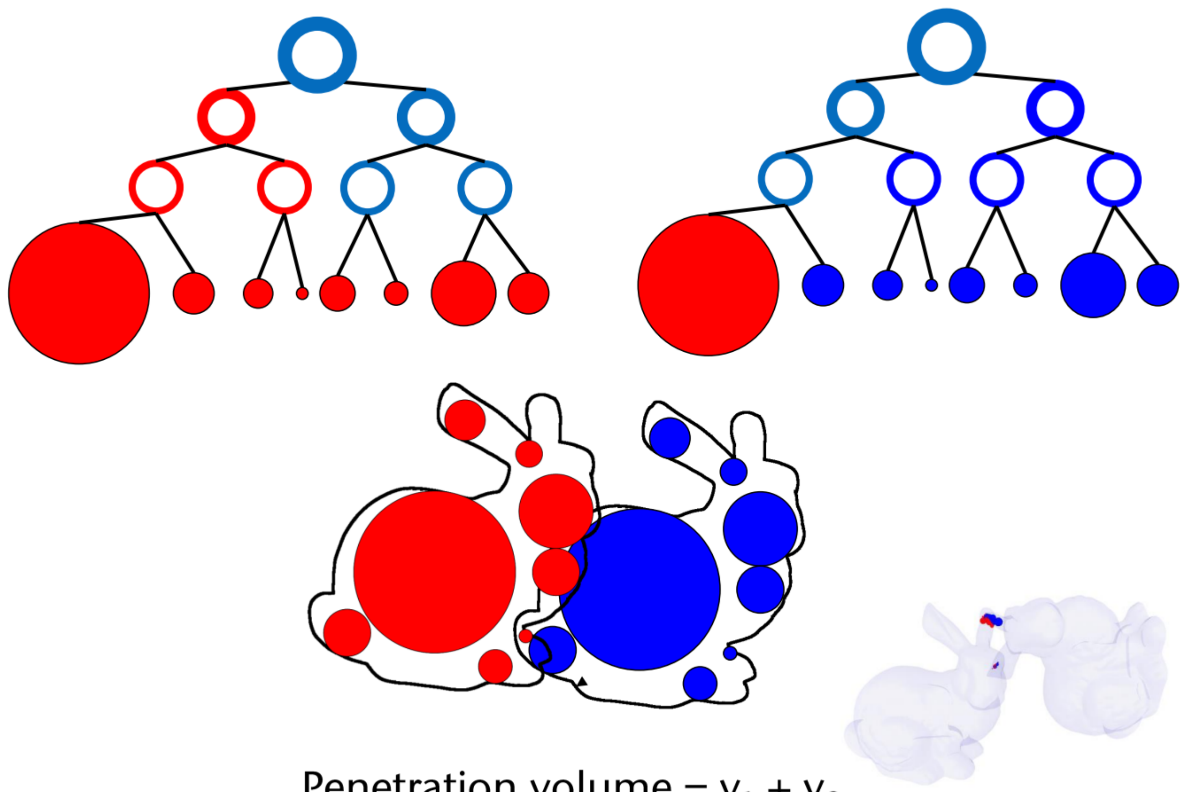
the algorithm can also be parallelized , by subdividing the geometry space into a grid and then taking random start points from inside each cell



after creating the sphere tree for the objects, all we need to do is creating a hierarchy for storing them as a tree. the idea here is to create outer speres that includes the inner one(conceptual) and then other spheres that contains the inner ones

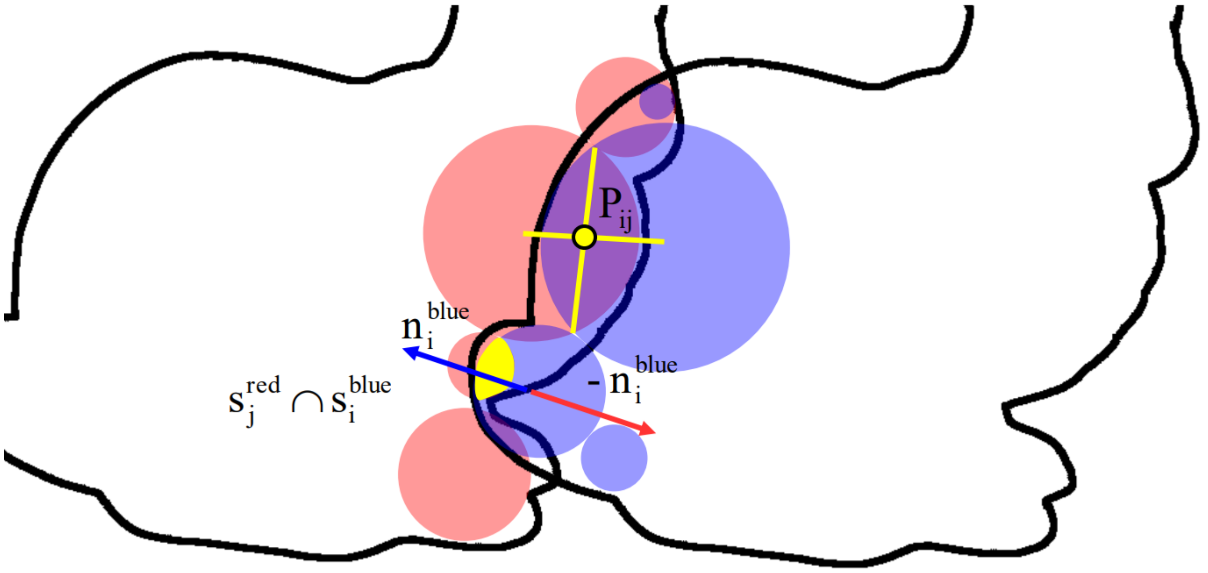


Use for collision detection



Penetration volume = $v_1 + v_2$

all we need to do now is to traverse the sphere trees to check for collisions, like for the bounding boxes and apply forces and torques to the objects in order to repel the collision



$$f_{ij}^{blue} = (s_j^{red} \cap s_i^{blue})(-n_i^{blue}) \qquad \tau_{ij}^{blue} = (P_{ij} - C_m) \times (f_{ij}^{blue})$$

$$f_{total}^{blue} = \sum f_{ij}^{blue} \qquad \tau_{total}^{blue} = \sum \tau_{ij}^{blue}$$

lemma

A single sphere s intersects a **constant** number of disjoint spheres A with at least the same radius

Theorem

The maximum number of intersecting pairs of spheres of two polydisperse sphere packings A and B with n spheres is $O(n)$

```
for each sphere $s \in B$:
  Compute hierarchy level $l$
  for all levels $l \leq li \leq lmax$:
    for all cells $cj$ in level $li$ overlapped by $s$
      for all spheres $sk \in cj$
        Compute overlap volume for $s$ and $sk$
```

the maximum complexity is $O(n)$, where n is the number of spheres, all the rest is $O(1)$. if the algorithm is parallelyzed the complexity becomes $O(1)$