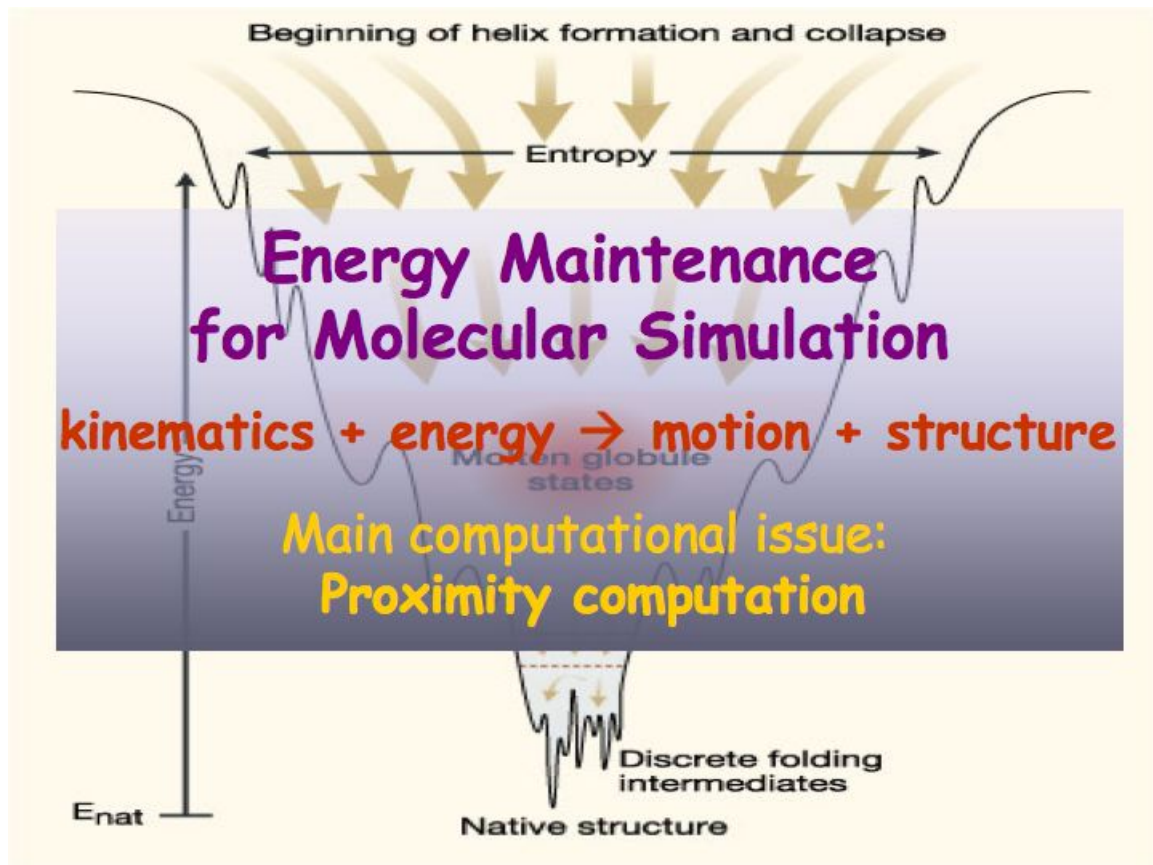


CS 273
Prof. Serafim Batzoglou
Prof. Jean-Claude Latombe
Spring 2006

Lecture 12 : Energy maintenance (1)

Lecturer: Prof. J.C. Latombe

Scribe: Neda Nategh



How do you update the energy function during the simulation process?

During the simulation process conformation changes so energy function changes and we want to have the value of the energy function.

Kinematics describes the possible motion of the mechanical systems independent of causes that create this motion.

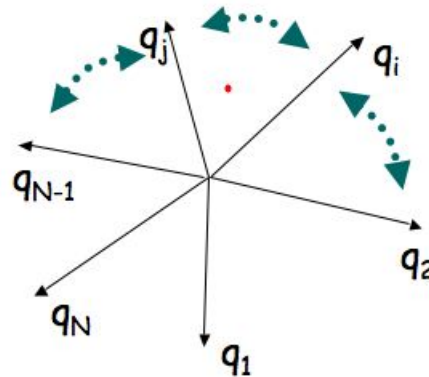
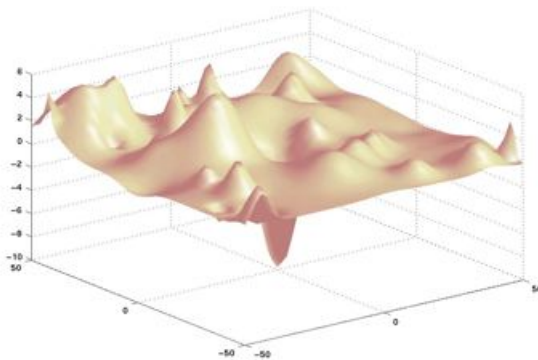
Energy codes for the motion of the protein and tries to go to the local minimum by changing the structure to achieve this local minimum. So

Kinematics + Energy $\rightarrow$ Motion + structure

We do not focus on the energy function. Energy function is the task of biologists to determine which energy function should be used. We talk about how to maintain the value of the energy function during the motion when conformation changes.

Energy

Function defined over large dimensional conformation space



2

energy function defines the conformation space of the molecule.

Energy function consists of a lot of terms; each of them is the sum of many other terms.

Bonded terms:

In the following figure you can see the components of the bonded terms.

E_s is the energy pair associated with the energy of two atoms that are bonded together.

The value of this energy depends on the length of the atoms. We have E_s terms as many as pairs of bonded atoms.

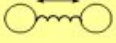
K_s parameter depends on the temperature and the type of two atoms and etc.

θ corresponds to the angles between successive bonds.

Energy Function

$$E = E_S + E_\Theta + E_{S\Theta} + E_{\text{Tor}} + E_{\text{vdW}} + E_{\text{dipole}}$$

bonded terms
(in linear number)

 $E_s = 143.88 \frac{k}{2} (l - l_0)^2 (1 - 2(l - l_0))$
 $E_\Theta = 0.21914 k_\Theta (\Theta - \Theta_0)^2 (1 + 7 \times 10^{-5} (\Theta - \Theta_0)^4)$ $E_{s_\Theta} = 2.51124 k_{s_\Theta} (\Theta - \Theta_0) [(l - l_0)_s + (l - l_0)_b]$
 $E_{\text{Tor}} = \frac{V_1}{2} (1 + \cos \omega) + \frac{V_2}{2} (1 + \cos 2\omega) + \frac{V_3}{2} (1 + \cos 3\omega)$

3

Non-bonded terms:

For any to atoms we can calculate the Van der Waals energy (EvdW). It is low when two atoms are very far apart and it is large when they are very close. Beyond the vdM energy conformation of two molecules would not happen. vdW energy tells two atoms that are closing to each other when they cross each other.

If vdW energy is low most likely the conformation of two atoms is not feasible.

Another non-bonded energy is Edipole that is the energy associated with the dielectrics.

Role of vdW terms:

vdW terms are a maze in conformational space. Other terms steer the molecule in this maze.

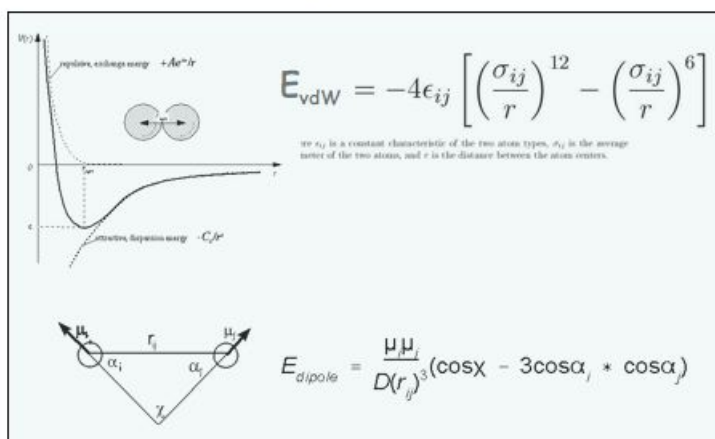
Another energy term is associated with the interaction with solvent that is important in some simulations. There is two ways of interaction with solvent.

1-**Explicit solvent models:** 100s or 1000s of discrete solvent molecules. That is very expensive.

2-**Implicit solvent models:** solvent as continuous medium, interface is **solvent-accessible surface**. This model only consider the interaction between solvent and protein and define the energy associated with this interface that is called solvent accessible surface.

Energy Function

$$E = \underbrace{E_S + E_\Theta + E_{S\Theta} + E_{\text{Tor}}}_{\text{bonded terms (in linear number)}} + \underbrace{E_{\text{vdW}} + E_{\text{dipole}}}_{\text{non-bonded terms (in quadratic number)}}$$



4

Energy Function

$$E = \Sigma \text{ bonded terms} \\ + \Sigma \text{ non-bonded terms} \\ + \Sigma \text{ solvation terms}$$

Bonded terms are relatively few and there are linear number of them.

Non-bonded terms depend on the distances between pairs of atoms and there are quadratic number of them and so it is expensive to compute them.

Solvation terms may require computing molecular surfaces if we consider the implicit model.

The focus would be on non-bonded terms.

Uses of energy function

- Generate energetically plausible conformations: sample (at random), minimize, cluster
- Generate meaningful distributions (e.g. Boltzman) of conformations: Monte Carlo simulation
- Generate motion pathways to study molecular kinetics: molecular dynamics, MC simulation

Monte Carlo Simulation (MCS)

It is the first technique that is widely used to study thermodynamic and kinetic properties of proteins.

Algorithm:

- Random walk through conformation space
- At each cycle :
 - Perturb current conformation at random
 - Accept step with probability:

$$P(\text{accept}) = \min \{ 1, \exp(-\Delta E/KT) \}$$

(Metropolis acceptance criterion)
- The conformations generated by an arbitrary long MCS are Boltzman distributed, i.e., #conformations in $V \sim \int \exp(-E/KT) dV$

We start with an initial conformation and intuitively through some degrees of freedom we generate new conformation at random in small neighborhood of current conformation. Then we accept the new conformation with criterion probability, $P(\text{accept})$, that determines the quality of acceptance. $P(\text{accept})$ depends on the temperature and ΔE that is the energy of new conformation minus the energy of previous conformation. If energy decreases in the new conformation $P(\text{accept})$ would be 1 i.e. we accept the new conformation. If E goes up the process stops quickly.

If we take sufficiently long MCS, we get a set of sample conformation that satisfies Boltzman distribution. It means if we take any volume V in the conformation space the number of sample is proportional to $\int \exp(-E/KT) dV$ that means it satisfies the Boltzman distribution.

One issue in common is that energy must be evaluated frequently e.g. MD and MC simulation runs may consist of millions of steps.

The problem is :

How to efficiently compute and update energy during minimization and simulation?

The idea of computer science is that there might not be too much difference between conformation of two consecutive steps of simulation; so we might have energy terms that are the same and what we want to do is to take advantage of certain amount of consistent between energy function.

Non-bonded Energy terms

We want to compute any pairs of atoms that are molecules. When we look at the energy terms in most models we see they go to zero or some constant when the distance between two atoms increases; so we can set a cut-off distance between two atoms. When they are further apart they do not contribute significantly to the energy.

For a given volume due to vdW force there is a constant number of atoms at most that we can put them in that volume. Cut-off distance between two atoms defines a fixed volume.

So each atom can interact at most with a certain constant number of atoms so in total there is only linear number of interacting pairs of atoms.

Problems:

How can we find the interacting pairs without enumerating all atom pairs?

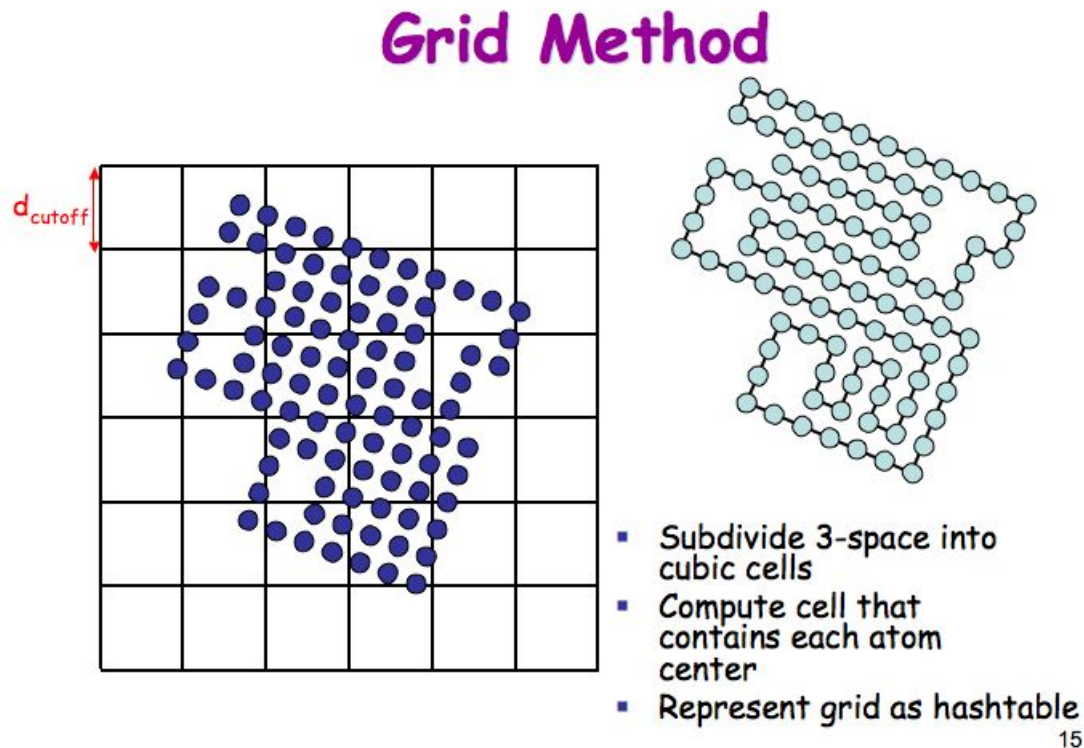
We compute the distance between all the atom pairs, check if they are apart beyond the cut-off distance or not so we can find the interacting pairs of atoms. It has quadratic cost.

How can we detect atomic clashes quickly?

It means vdW energy is beyond the threshold that is associated with the distance threshold of atoms

Main computational issue is Proximity computation.

Grid Method:



15

The method that is widely used in biology for this kind of issues is the Grid method. We want to compute all the pairs of atoms that are interacting. We index the atoms into a 3-D grid of cubes, all have the same size and the side length of energy cube is cut-off distance. We get the maximum cut-off distance of all the pairs. In each step for each atom in each cube it takes linear time to compute. Once we have computed this grid we want to know all the atoms that are possibly interacting with a certain atom. We will open a window that contains the corresponding cube of that certain atom and consider all the

cubes which are around it; then we retrieve from this cube what are the atoms which are indexed in that cube and we know this atom only interacts with atoms around this cube. So we will find all the interacting pairs of atoms including this atom in a linear time.

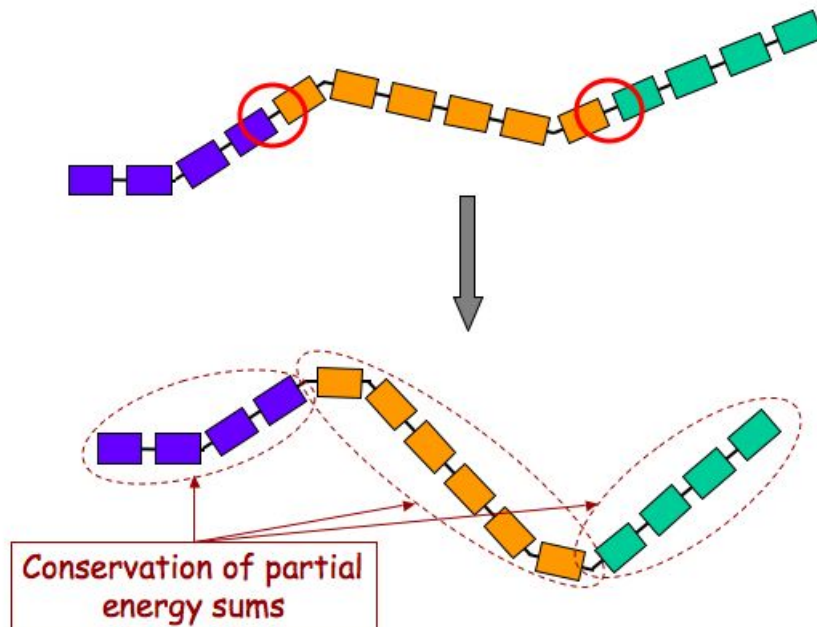
So,

- $O(n)$ time to build grid
- $O(1)$ time to find interactive pairs for each atom
- $\Theta(n)$ to find all interactive pairs of atoms
- It is asymptotically optimal in worst-case

This is asymptotically optimal in the worst case. Actually there is a linear number of interacting pairs of atoms. To compute all of them we have to pay linear cost. It does not mean it is always optimal because in many cases there might not be a linear number of interacting pairs of atoms but we still have to always pay a linear cost.

How to update the energy?

- Compare the interacting pairs at new step with those at previous step
- For every pair that has disappeared, subtract the corresponding energy term from energy value
- For every new pair, add the corresponding energy term to energy value
- Takes $\Theta(n)$ time, even if very few pairs have changed



The grid method is unable to recognize and re-use such partial sums

18

For example if we take a protein like one in the figure above and if we have two degrees of freedom and if we restrict the conformation like the next figure the grid method is totally unable to realize that there are number of partial sums of energy terms which are not changed; so the partial sum remains the same meaning that do not recompute it but check interacting atoms in new step to previous step to in order to see they have not changed. But we still have linear cost to recognize that they have not changed.

The problem is:

How to reuse partial sums?

Another issue is how to detect atomic clashes very quickly?

The advantage of the Grid method is that it is very general but the other techniques are not as general as Grid method.

We can use the Grid method to compute the molecular surface. Using the Grid method we compute the solvation term of the energy in linear time.

- Each sphere intersects $O(1)$ spheres
 - Computing each atom's contribution to molecular surface takes $O(1)$ time
 - Computation of molecular surface takes $\Theta(n)$ time
- Implicit solvation term in $\Theta(n)$ time

General problem:

Molecules form geometrically complex objects that deform and move relative to each other:

- (self-)collision detection
- Distance computation

There are several computational approaches to solve these problems:

- Space occupancy: grid, octree
In grid method you have a coordinate system that is fixed and you encode your objects like proteins in this grid. But in octree technique we will represent the objects with the coordinate attached to it.
- Tracking pairs of closest features
The idea is that if you have objects moving toward one another and if you consider slow increment of motion, at one stage you compute all the pairs of closest point between two objects; if you move this object slightly many those closest pairs remain the same. It does not mean the distance between the pairs remain the same but the closest pairs remain the same.
- Polynomial equation
This is a general technique that is used in robotics, graphics and etc.
- Bounding-volume hierarchies (BVH)
- Spanners
This technique is still at the level of research. The idea is as the following:
Suppose we have a backbone of protein. What we want to do is to try to find a kind of graph such that every edge of the graph as a length represent the distance

between two points in the backbone of protein. If we take a point in this backbone and another point in another backbone then we try to find the distance between the nearest points in the spanner and then compute the minimum distance between those two points in the spanner, we get the conformation of the distance between those two points. The idea is to simplify the representation of the protein that allows you to compute approximate distance very efficiently.

Bounding Volume Hierarchies (BVHs)

<http://www.cs.unc.edu/~geom/collide>

Basic problem

Given the geometric models and relative positions of two objects, determine whether they overlap.

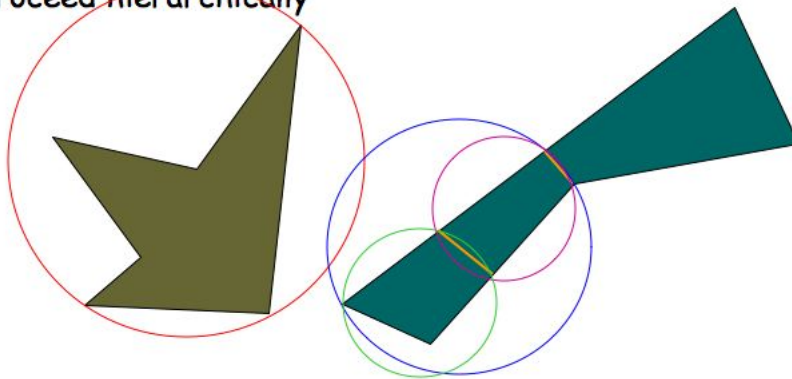
The basic problem in distance computation of collision detection is that we have two objects and we want to know if the distance between them is positive. Distance zero means two objects have collided.

Applications

- Computer graphics and simulation
- Robotics
- Haptics

Basic Idea of Solution

- Enclose objects into bounding volumes (spheres or boxes)
- Check the bounding volumes first
- Decompose an object into two
- Proceed hierarchically



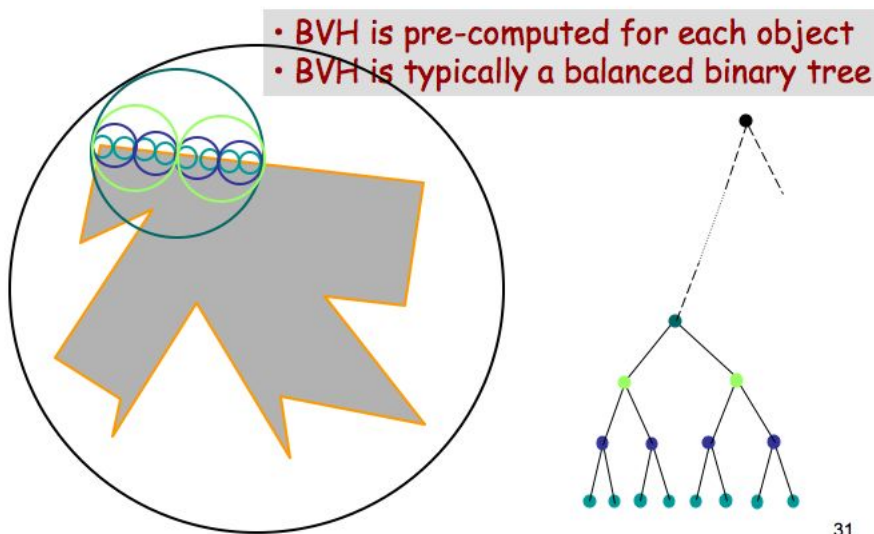
29

The basic idea of solution is to enclose objects into bounding volumes (spheres or boxes) and check the bounding volumes first to see if two bounding volumes intersect. The test

is easy ; we compute the distance between centers of spheres and compare it to the radius of them. Then we decompose an object into two and proceed hierarchically.

The problem is that every time for collision detection we bound the object with an enclosed bounding volume. We waste the time on computing BV just checking whether two objects collide or not. So we can precompute the BV hierarchically for rigid objects. For deformable objects precomputation of bounding volumes means we set small samples of small spheres around the boundary and we start those spheres as leaves of a tree then we bound them by a bigger sphere until eventually we bound the entire object. Typically BVH is going to be a binary tree.

Bounding Volume Hierarchy (BVH)

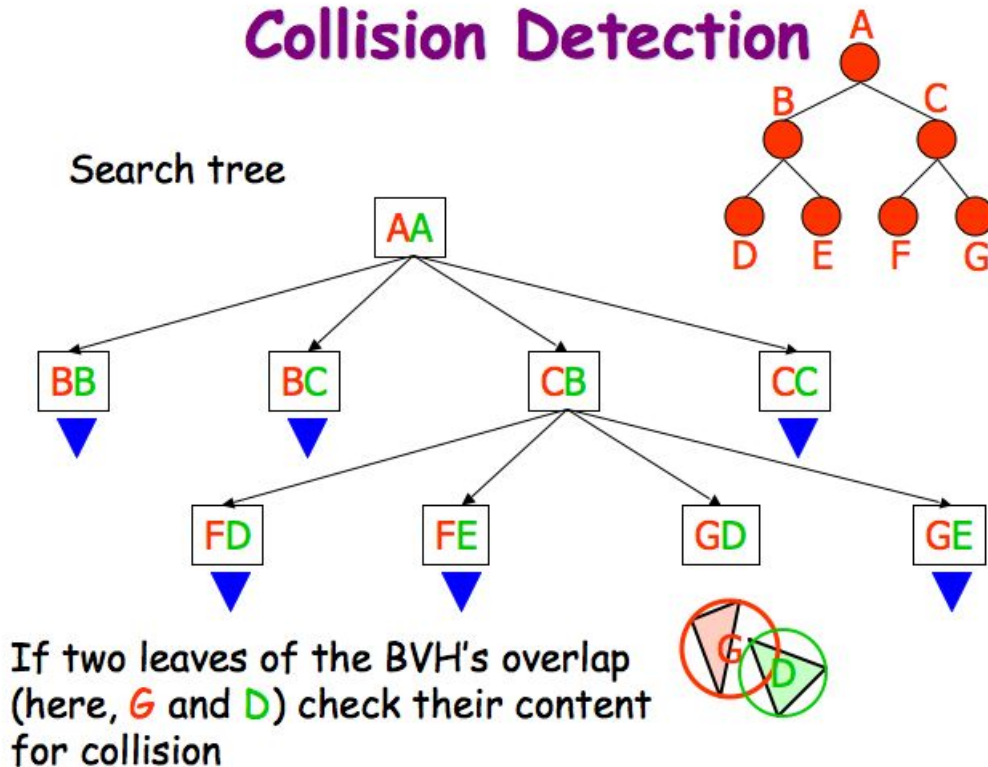


31

Collision Detection:

Suppose we have two rigid objects described by their precomputed BVHs, red and green objects with different sizes. We start at the root. Two red and green “A” spheres do not overlap. Then we divide these two spheres into two spheres to find the BVH. We have 4 spheres and then compare **BB**, **BC**, **CB** and **CC**. We continue the procedure to search the entire tree.

Hopefully large number of these spheres do not collide.



37

There are lots of variants in the literature but they are relatively the same and the principle is the precomputation of hierarchy.

Collision Detection:

- Pruning discards subsets of the two objects that are separated by the BVs
- Each path is followed until pruning or until two leaves overlap
- When two leaves overlap, their contents are tested for overlap

The power of hierarchy:

If you have an object described by a million triangles and check it against another object with a million triangles, you try to find pairs of triangles intersect there could be quadratic number of triangles but here for two complicated objects even if they come close to overlap slightly still we have less collision sets which are far apart so BVH allows you to roll out any possible collision.

Search strategy and heuristics:

- If there is no collision, all paths must eventually be followed down to pruning or a leaf node
- But if there is collision, it is desirable to detect it as quickly as possible
- Greedy best-first search strategy with $f(N) = d / (r_x + r_x)$

We want to minimize $f(N)$. If it is small it means that the two spheres are relatively large. If two spheres are relatively large they are more likely to have collision. We can do heuristic search to do collision detection.

Recursive (Depth-First) Collision Detection Algorithm

Test (A, B)

1. If A and B do not overlap, then return 1
2. If A and B are both leaves, then return 0, if their contents overlap and 1 otherwise
3. Switch A and B if A is a leaf or if B is bigger and not a leaf
4. Set A1 and A2 to be A's children
5. If **Test** (A1, B) = 1 then return **Test** (A2, B) else return 0

This algorithm is very easy to be implemented.

Performance:

Several thousands collision checks per second for 2 three-dimensional objects each described by 500,000 triangles, on a 1-GHz PC

Greedy Distance Computation

The recursion is the same as collision detection. The idea is that we have two bounded volumes to compute the distance between them. We need this distance to approximate the distance between two objects. This distance is called greedy distance. If we run this algorithm many times the distance computed by this algorithm we get on the average %50 better approximation; which is very nice.

Greedy-Distance (A, B)

1. If $\text{dist}(A, B) > 0$, then return $\text{dist}(A, B)$
2. If A and B are both leaves, then return distance between their contents
3. Switch A and B if A is a leaf, or if B is bigger and not a leaf
4. Set A1 and A2 to be A's children
5. $d1 \leftarrow \text{Greedy-Distance}(A1, B)$
6. If $d1 > 0$ then
 - a. $d2 \leftarrow \text{Greedy-Distance}(A2, B)$
 - b. If $d2 > 0$ then return $\text{Min}(d1, d2)$
7. Return 0

Exact Distance Computation

To do the exact distance computation it takes more time and it is significantly more expensive than just collision detection.

M (upper bound on distance) is initialized to very large number.

Distance (A, B)

1. If $\text{dist}(A, B) > M$, then return M
2. If A and B are both leaves, then
 - a. $d \leftarrow \text{distance between their contents}$

- b. Return Min (d, M)
3. Switch A and B if A is a leaf, or if B is bigger and not a leaf
4. Set A1 and A2 to be A's children
5. $M \leftarrow \text{Distance}(A1, B)$
6. If $M > 0$ then return $\text{Distance}(A2, B)$
7. Else return 0

Approximate Distance Computation

It is the variant of the previous algorithm. We define a parameter α . The distance we get by this algorithm comprises between the actual distance between two objects and the distance between two objects multiplied by α .

M (upper bound on distance) is initialized to very large number.

Approx-Distance (A, B) [$\rightarrow da : da \leq de$ and $de - da \leq \alpha * de$]

1. If $\text{dist}(A, B) > M$, then return M
2. If A and B are both leaves, then
 - a. $d \leftarrow \text{distance between their contents}$
 - b. If $d < M$ then return $(1 - \alpha) * d$ else return M
3. Switch A and B if A is a leaf, or if B is bigger and not a leaf
4. Set A1 and A2 to be A's children
5. $M \leftarrow \text{Approx-Distance}(A1, B)$
6. If $M > 0$ then return $\text{Approx-Distance}(A2, B)$
7. Return 0

Guaranteed to return an approximate distance between $(1 - \alpha)d$ and d

Performance comparison:

Collision detection is slightly faster than Greedy distance computation and also slightly faster than 0.5 Approximate distance computation and much faster than Exact distance computation.

Desirable properties of BVs and BVHs

In order to make this algorithm as fast as possible and practical there are some properties to achieve:

BVs:

- Tightness
- Efficient testing
- Invariance

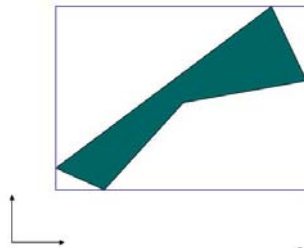
BVHs:

- Separation
- Balanced tree

For BV spheres are invariant and efficient to test but not tight.

Other BVs:

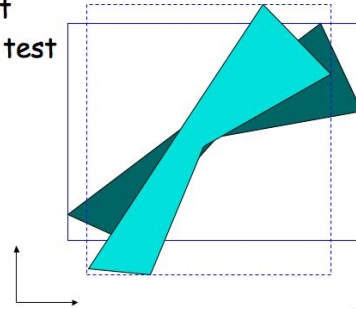
Axis-Aligned Bounding Box (AABB)



51

Axis-Aligned Bounding Box (AABB)

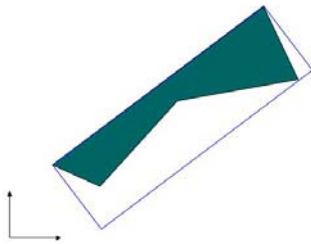
- Not invariant
- Efficient to test
- Not tight



52

Oriented Bounding Box (OBB)

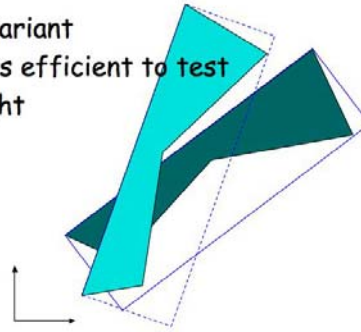
[Gottschalk, Lin, and Manocha, 96]



53

Oriented Bounding Box (OBB)

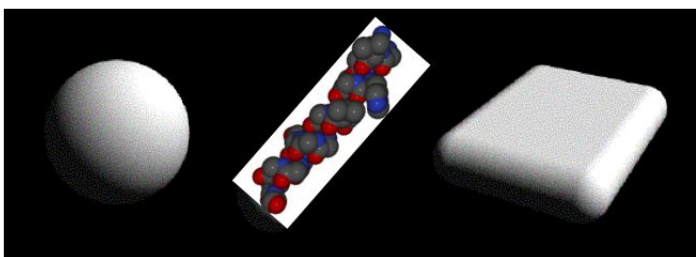
- Invariant
- Less efficient to test
- Tight



54

Rectangle Swept Spheres (RSS)

Similar to OBBs



→ Efficient distance computation

55

There is not one type of bounding volume that is optimal. The rectangle and sphere tend to be the best bounding boxes.