Resolve the excluded volume constraints with the help of game engine

Pascal Carrivain*

* Univ Lyon, Ens de Lyon, Univ Claude Bernard, CNRS, Laboratoire de Physique, F-69342 Lyon, France

June 26, 2018

Outline for section 1

Game engine : many tools in one

Figure: Starcraft. Courtesy of Blizzard

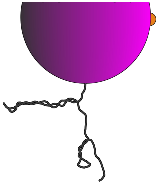
GameMaker Studio
Gamebryo
Defold
Lumberyard
HERO ENGINE
COCOS2D-X
CRYENGINE
unity
XENKO
GODOT Game engine
GameSalad
ShiVa
UNREAL ENGINE
JMONKEYENGINE
LÖVE
libGDX
PLAYCANVAS
MONOGAME
TORQUE 3D
Corona Labs
Figure: List of game engines. Courtesy of Mo King

Figure: Crysis. Courtesy of Crytek

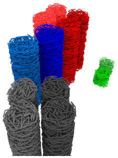
1. space partitionning : hash table, quadtree, octree ...
2. collision detection between complex geometries
3. mechanical joints to solve articulated systems under collision constraints

Game engines have been already used to ...

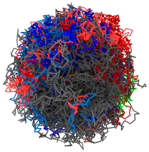
1. simulate the *Yeast* nucleus as a polymer brush.
2. simulate DNA under torsionnal constraint : plectonomes.



3. simulate chromosomes decondensation (*drosophila* nucleus) while keeping the topological state :

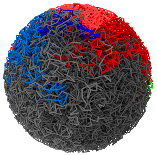


initial conformation



no resolving of the excluded volume constraint

movie



resolving of the excluded volume constraint

Outline for section 2

Resolve the excluded volume constraints with game engines or polymer simulations

Game engines : rigid body dynamics (described by the position, body frame, linear and angular velocities) under mechanical constraints.

Polymer systems : a set of connected points (spring, distance constraints) described by position and velocity that can be viewed as segments.

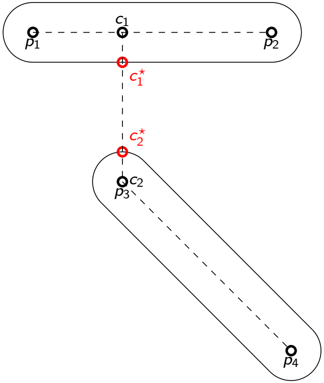
$\rightarrow$mapping between the two formalisms.

They share the need of resolving excluded volume constraints :

1. minimal distance between two segments (game engines and polymer systems).
2. resolve the excluded volume constraint :
   - **game engines** : mechanical contact joint between two bodies
   - **polymer systems** : repulsive forces between two points

First step : minimal distance between two segments

$\boldsymbol{c}$'s are the contact points along the segment, $\boldsymbol{c}^{\star}$'s are the contact points on the capsule surface



surface to surface minimal distance $d\left(\boldsymbol{c}_1^{\star}, \boldsymbol{c}_2^{\star}\right) <$ diameter of the capsule : overlap.

Minimal distance between two segments : four algorithms

two segments :

$$\begin{aligned} \boldsymbol{s}_1(a) &= \boldsymbol{p}_1 + a(\boldsymbol{p}_2 - \boldsymbol{p}_1) \\ \boldsymbol{s}_2(b) &= \boldsymbol{p}_3 + b(\boldsymbol{p}_4 - \boldsymbol{p}_3) \end{aligned}$$

minimal distance d^2 : $\min_{a,b}(\boldsymbol{s}_2(b) - \boldsymbol{s}_1(a))^2$

To solve the problem :

1. Kumar & *al* 2001, Goujon & *al* 2008, Sirk & *al* 2012 : solving $\frac{\partial d^2}{\partial a} = \frac{\partial d^2}{\partial b} = 0$ that is **"one line" algorithm** !
2. Distance enumeration between points lying on the first segment $\boldsymbol{s}_1(a)$ and points lying on the second segment $\boldsymbol{s}_2(b)$ with $a, b \in [0, 1]$: **really slow** !
3. `https://www.geometrictools.com/Documentation/DistanceLine3Line3.pdf` by David Eberly, Geometric Tools (GPU version of the algorithm exists) : **many cases to assess**.
4. `www.ode.org` by Russell Smith (based on Voronoi Clipping Rules) : **many cases to assess**.

Minimal distance between two segments : four algorithms

two segments :

$$\begin{aligned} \boldsymbol{s}_1(a) &= \boldsymbol{p}_1 + a(\boldsymbol{p}_2 - \boldsymbol{p}_1) \\ \boldsymbol{s}_2(b) &= \boldsymbol{p}_3 + b(\boldsymbol{p}_4 - \boldsymbol{p}_3) \end{aligned}$$

minimal distance d^2 : $\min\limits_{a,b} (\boldsymbol{s}_2(b) - \boldsymbol{s}_1(a))^2$

To solve the problem :

1. Kumar & *al* 2001, Goujon & *al* 2008, Sirk & *al* 2012 : solving $\frac{\partial d^2}{\partial a} = \frac{\partial d^2}{\partial b} = 0$ that is **"one line" algorithm** !
2. Distance enumeration between points lying on the first segment $\boldsymbol{s}_1(a)$ and points lying on the second segment $\boldsymbol{s}_2(b)$ with $a, b \in [0, 1]$: really slow !
3. `https://www.geometrictools.com/Documentation/DistanceLine3Line3.pdf` by David Eberly, Geometric Tools (GPU version of the algorithm exists) : many cases to assess.
4. `www.ode.org` by Russell Smith (based on Voronoi Clipping Rules) : many cases to assess.

Minimal distance between two segments : four algorithms

two segments :

$$\begin{aligned} \boldsymbol{s}_1(a) &= \boldsymbol{p}_1 + a(\boldsymbol{p}_2 - \boldsymbol{p}_1) \\ \boldsymbol{s}_2(b) &= \boldsymbol{p}_3 + b(\boldsymbol{p}_4 - \boldsymbol{p}_3) \end{aligned}$$

minimal distance d^2 : $\min\limits_{a,b} (\boldsymbol{s}_2(b) - \boldsymbol{s}_1(a))^2$

To solve the problem :

1. Kumar & *al* 2001, Goujon & *al* 2008, Sirk & *al* 2012 : solving $\frac{\partial d^2}{\partial a} = \frac{\partial d^2}{\partial b} = 0$ that is **"one line" algorithm** !
2. Distance enumeration between points lying on the first segment $\boldsymbol{s}_1(a)$ and points lying on the second segment $\boldsymbol{s}_2(b)$ with $a, b \in [0, 1]$: **really slow** !
3. `https://www.geometrictools.com/Documentation/DistanceLine3Line3.pdf` by David Eberly, Geometric Tools (GPU version of the algorithm exists) : **many cases to assess**.
4. `www.ode.org` by Russell Smith (based on Voronoi Clipping Rules) : **many cases to assess**.

Minimal distance between two segments : four algorithms

two segments :

$$\begin{aligned} \boldsymbol{s}_1(a) &= \boldsymbol{p}_1 + a(\boldsymbol{p}_2 - \boldsymbol{p}_1) \\ \boldsymbol{s}_2(b) &= \boldsymbol{p}_3 + b(\boldsymbol{p}_4 - \boldsymbol{p}_3) \end{aligned}$$

minimal distance d^2 : $\min\limits_{a,b} (\boldsymbol{s}_2(b) - \boldsymbol{s}_1(a))^2$

To solve the problem :

1. Kumar & *al* 2001, Goujon & *al* 2008, Sirk & *al* 2012 : solving $\frac{\partial d^2}{\partial a} = \frac{\partial d^2}{\partial b} = 0$ that is **"one line" algorithm** !
2. Distance enumeration between points lying on the first segment $\boldsymbol{s}_1(a)$ and points lying on the second segment $\boldsymbol{s}_2(b)$ with $a, b \in [0, 1]$: **really slow** !
3. `https://www.geometrictools.com/Documentation/DistanceLine3Line3.pdf` by David Eberly, Geometric Tools (GPU version of the algorithm exists) : **many cases to assess**.
4. `www.ode.org` by Russell Smith (based on Voronoi Clipping Rules) : **many cases to assess**.

Minimal distance between two segments : four algorithms

two segments :

$$\begin{aligned} \boldsymbol{s}_1(a) &= \boldsymbol{p}_1 + a(\boldsymbol{p}_2 - \boldsymbol{p}_1) \\ \boldsymbol{s}_2(b) &= \boldsymbol{p}_3 + b(\boldsymbol{p}_4 - \boldsymbol{p}_3) \end{aligned}$$

minimal distance d^2 : $\min_{a,b} (\boldsymbol{s}_2(b) - \boldsymbol{s}_1(a))^2$

To solve the problem :

1. Kumar & *al* 2001, Goujon & *al* 2008, Sirk & *al* 2012 : solving $\frac{\partial d^2}{\partial a} = \frac{\partial d^2}{\partial b} = 0$ that is **"one line" algorithm** !
2. Distance enumeration between points lying on the first segment $\boldsymbol{s}_1(a)$ and points lying on the second segment $\boldsymbol{s}_2(b)$ with $a, b \in [0, 1]$: **really slow** !
3. `https://www.geometrictools.com/Documentation/DistanceLine3Line3.pdf` by David Eberly, Geometric Tools (GPU version of the algorithm exists) : **many cases to assess**.
4. `www.ode.org` by Russell Smith (based on Voronoi Clipping Rules) : **many cases to assess**.

Minimal distance between two segments : comparison of the four algorithms

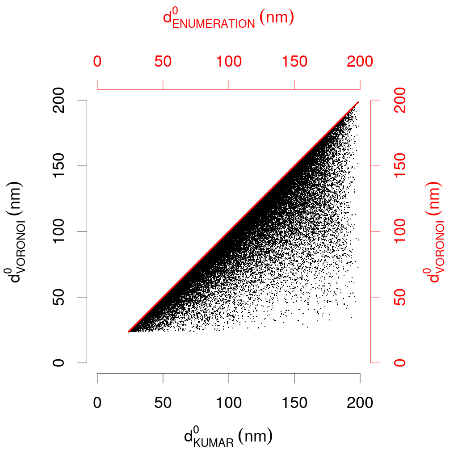


Figure: (Black point) : The scatter-plot of the minimal distance between two segments with the “Voronoi Clipping Rule” and method from Kumar & *al* 2001. (Red point) : The scatter-plot of the minimal distance between two segments with the “Voronoi Clipping Rule” and “exact enumeration”.

15 % **of the black points with error greater than** 10 %.

Second step : once the minimal distance has been found ...

1. Repulsive forces between the two segments (soft approach), the "Segmentive-Repulsive-Potential" method (SRP)
   1.1 The two segments start to feel each others.
   1.2 The repulsive force decrease the acceleration to zero ...
   1.3 ... and then increase the acceleration to push away the two segments.
2. Contact joint at the contact point (if the two objects overlap)
   2.1 The segments follow their moves ...
   2.2 ... until the collision detection algorithm says there is a contact point.
   2.3 a contact joint is set up ...
   2.4 ... to vanish the overlap over the time step.

Second step : once the minimal distance has been found ...

1. Repulsive forces between the two segments (soft approach), the "Segmentive-Repulsive-Potential" method (SRP)
    1.1 The two segments start to feel each others.
    1.2 The repulsive force decrease the acceleration to zero ...
    1.3 ... and then increase the acceleration to push away the two segments.
2. Contact joint at the contact point (if the two objects overlap)
    2.1 The segments follow their moves ...
    2.2 ... until the collision detection algorithm says there is a contact point.
    2.3 a contact joint is set up ...
    2.4 ... to vanish the overlap over the time step.

Solving a collision by mean of game engine (1)

what do we need ?

$\boldsymbol{u}_i = (1 - \|\boldsymbol{c}_i - \boldsymbol{p}_i\|)\, \boldsymbol{v}_i + \|\boldsymbol{c}_i - \boldsymbol{p}_i\| \boldsymbol{v}_{i+1}$: velocity at the contact point $\boldsymbol{c}_i$ on the segment i.

$\|\boldsymbol{c}_i - \boldsymbol{p}_i\|$: 1d position of the contact point.

$\boldsymbol{n}$: normal to the contact.

game engines ask for the contact relative velocity $\boldsymbol{u}_i - \boldsymbol{u}_j$ (along the normal $\boldsymbol{n}$) to be reversed in one time step :

$$\begin{aligned} (\boldsymbol{u}_i - \boldsymbol{u}_j)^T \boldsymbol{n} &< 0 \quad \text{at time step } t \\ (\boldsymbol{u}_i - \boldsymbol{u}_j)^T \boldsymbol{n} &> 0 \quad \text{at time step } t + \Delta t \end{aligned}$$

For many collisions we have a system of equations :

$$\begin{aligned} \boldsymbol{\mathcal{J}}(t)\, \boldsymbol{\mathcal{V}}(t) &< 0 \quad \text{at time step } t \\ \boldsymbol{\mathcal{J}}(t)\, \boldsymbol{\mathcal{V}}(t + \Delta t) &> 0 \quad \text{at time step } t + \Delta t \end{aligned}$$

with $\boldsymbol{\mathcal{J}}$ geometric informations, $\boldsymbol{\mathcal{V}}$ generalized velocity vector.

Solving a collision by mean of game engine (2)

We can play with the constraints :

$$\begin{aligned} \mathcal{J}(t)\,\mathcal{V}(t) &= \boldsymbol{\Delta} && \text{at time step } t, \textbf{ given} \\ \mathcal{J}(t)\,\mathcal{V}(t+\Delta t) &= \boldsymbol{X} && \text{at time step } t+\Delta t, \textbf{ what you want} \end{aligned}$$

where $\boldsymbol{\Delta}$ is given by the conformation at time-step t and $\boldsymbol{X}$ is what you want at time-step $t + \Delta t$.

Examples :

$\boldsymbol{X} = -\frac{\delta}{\Delta t}$ is a term that tells the velocities to correct the overlap δ over one time-step Δt.

$\boldsymbol{X} = -\boldsymbol{\Delta}$ is a term that tells the relative velocity along the normal to the contact to be reversed over one time-step Δt.

You can ask for various contact joint responses !

Solving a collision by mean of game engine (3)

We already came up with the system of equations $\boldsymbol{\mathcal{J}}\boldsymbol{\mathcal{V}}$ describing the excluded volume constraints but what next ?

These non-holonomic constraints correspond to Lagrange multipliers $\boldsymbol{\lambda} > 0$ and forces $\boldsymbol{\mathcal{J}}^T\boldsymbol{\lambda}$.

We made the following approximation $\boldsymbol{\mathcal{V}}(t+\Delta t) \simeq \boldsymbol{\mathcal{V}}(t) + \boldsymbol{\mathcal{M}}^{-1}\boldsymbol{\mathcal{J}}^T\boldsymbol{\lambda}$ that's enough to solve for $\boldsymbol{\lambda}$.

Finding $\boldsymbol{\lambda}$ is then solving a "Linear-Complementarity-Problem" $\boldsymbol{\mathcal{A}}\boldsymbol{\lambda} = \boldsymbol{b} \mid \lambda > 0$.

We can split the matrix $\boldsymbol{\mathcal{A}}$ in as many parts as independent collision graphs to solve in parallel.

Resolve the excluded volume constraints : summary

1. The “Segmentive-Repulsive-Potential” (SRP) method (Sirk & *al* 2014) acts on the position and leave unchanged the velocities.
2. The contact joint acts on the velocities to correct both velocities and positions in one time-step.
3. The contact joint method has no associated time-scale.
4. The contact joint correction is easily coupled to the SHAKE algorithm correction step for the velocities.

The number of crossing depends on the potential parameters with the “SRP” method.

No crossing has been detected so far with the contact joint method.

Outline for section 3

Resolve the excluded volume constraints for more complex geometries

Complex geometries can be described with a set of triangle meshes.

Example : icosphere.

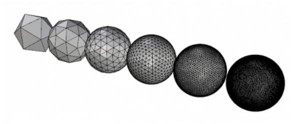
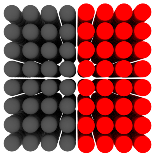


Figure: Icosphere with more and more triangle meshes. Courtesy of food4Rhino

Cylinder can be described in term of triangle meshes too.

Collision detection between triangle meshes, contact joint between two triangles

Example : mix of cylinders and icospheres built as triangle meshes.



Outline for section 4

Conclusions

1. We used a game-engine-based method to resolve the excluded volume constraints :
   - mapping of the rigid body description to point-based polymer systems.
   - minimal distance between two segments.
   - contact joint to resolve the overlap.
2. We showed a proof of example of a system of complex geometries using the trimesh formalism

Perspectives

For a linear or circular polymer :

1. Implementation of the segment to segment minimal distance presented here : **easy**
2. Implementation of contact joint : **easy**
3. Implementation of contact joint solver : **medium**

For a system of complex shapes :

1. Implementation of complex shape/trimesh : **medium**
2. Implementation of space partitioning for complex geometries : **medium**
3. Implementation of detection collision between complex geometries/trimesh : **hard**
4. Implementation of libccd (library by Daniel Fiser) for collision between convex shapes (open dynamics engine, bullet3 and FCL already did it)
5. Implementation of a parallel solver for the independent collision graphs