Penetration-Free Deformation Field Computation

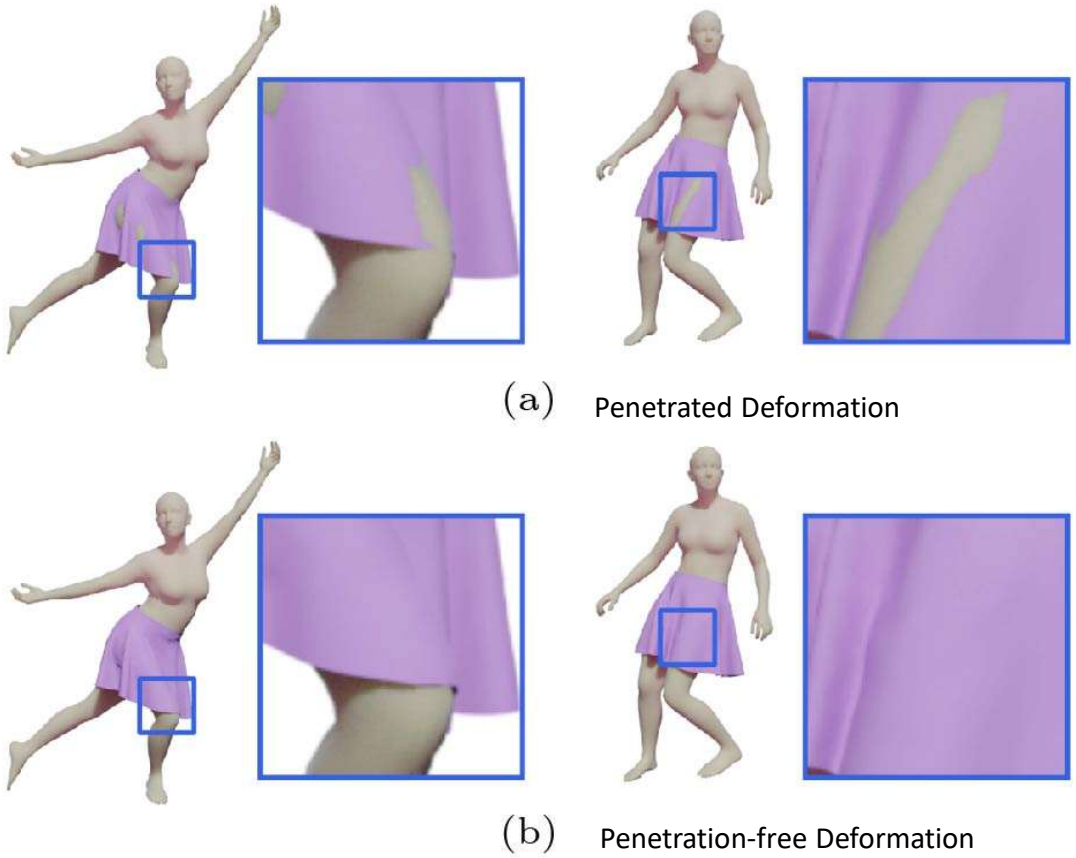
Anka Chen[1], Jerry Hsu[2]

1: NVIDIA 2: University of Utah

Penetration Free Deformation

Shape 1: X

Deformation:
$X = \Phi(X_0)$



Shape 2: X_0

Why is Penetration-Free Deformation So Important in Simulation

Penetrations can:

- Cause Visual Artifacts
- Cause Numerical Problems
- Non-physical in most of the times
- Require a lot of effort in parameter & setting tuning
- For co-dimensional object (e.g., hair, cloth), it is very hard to recover from penetration after it happens

How Simulation Works

$$\mathbf{x}^{n+1} = \mathbf{x}^n + \mathrm{h}\mathbf{v}^{n+1}$$
$$\mathbf{v}^{n+1} = \mathbf{v}^n + hM^{-1}\mathbf{f}(\mathbf{x}^{n+1})$$

Backward Euler

$$\underset{X}{\operatorname{argmin}} \frac{1}{2h^2}|\mathbf{x}-\mathbf{y}|_M^2 + E_e(\mathbf{x}) + E_c(\mathbf{x}) + \cdots$$

where, $\mathbf{y} = \mathbf{x}_{prev} + \mathbf{v}_n + \frac{h^2}{2}M^{-1}\mathbf{f}_{ext}$

Variational Formulation: an implicit form

How Simulation Works

$$\mathbf{x}^{n+1} = \mathbf{x}^n + \mathrm{h}\mathbf{v}^{n+1}$$
$$\mathbf{v}^{n+1} = \mathbf{v}^n + hM^{-1}\mathbf{f}(\mathbf{x}^{n+1})$$

Backward Euler

$$\underset{X}{\operatorname{argmin}}\, G(\mathbf{x}) = \frac{1}{2h^2}|\mathbf{x}-\mathbf{y}|_M^2 + E_e(\mathbf{x}) + E_c(\mathbf{x}) + \cdots$$

where, $\mathbf{y} = \mathbf{x}_{prev} + \mathbf{v}_n + \frac{h^2}{2}M^{-1}\mathbf{f}_{ext}$

Variational Formulation: an implicit form

How Simulation Works

Pipeline

- For t in time steps
  - Apply initial guess
  - For i in max_iterations:
    - Collision detection
    - Evaluate $E_e(\mathbf{x}) + E_c(\mathbf{x}) + \cdots$ and their gradient (force)
    - Find a descent direction$\Delta\mathbf{x}$
    - **x=x+Δx**
    - break if optimization has converged

$$\underset{X}{\operatorname{argmin}}\, G(\mathbf{x}) = \frac{1}{2h^2}|\mathbf{x}-\mathbf{y}|_M^2 + E_e(\mathbf{x}) + E_c(\mathbf{x}) + \cdots$$

where, $\mathbf{y} = \mathbf{x}_{prev} + \mathrm{h}\mathbf{v}_n + \frac{h^2}{2}M^{-1}\mathbf{f}_{ext}$

Question: where can penetration happen?

How Simulation Works

Pipeline

- For t in time steps
  - Apply initial guess
  - For i in max_iterations:
    - Collision detection
    - Evaluate $E_e(\mathbf{x}) + E_c(\mathbf{x}) + \cdots$ and their gradient (force)
    - Find a descent direction $\Delta\mathbf{x}$
    - $\mathbf{x}=\mathbf{x}+\Delta\mathbf{x}$
    - break if optimization has converged

Essentially all just apply a deformation:

$\mathbf{x}=\mathbf{x}+\Delta\mathbf{x}$

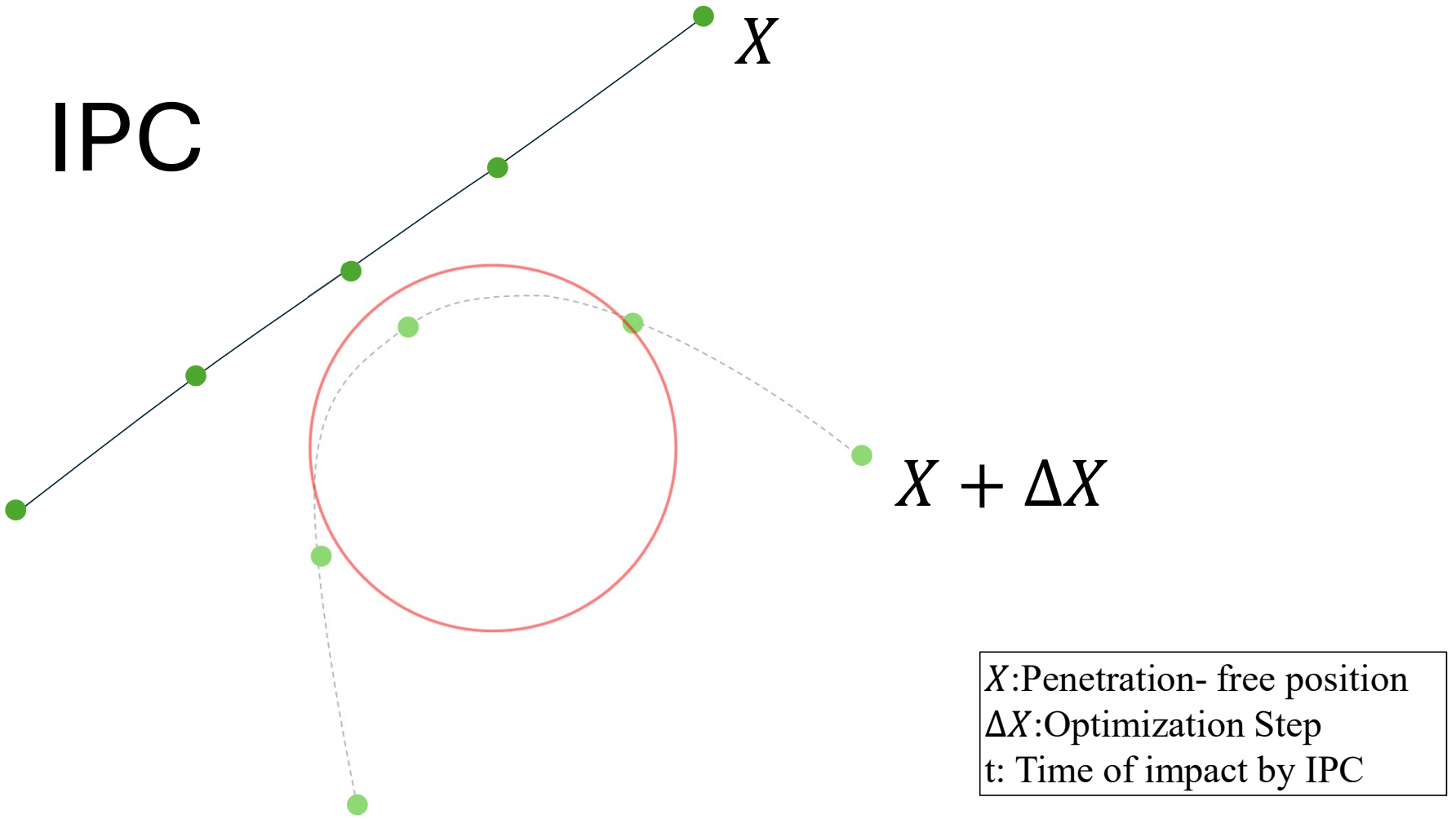
and $\nabla\mathbf{x}$ may cause penetration

Question: where can penetration happen?

How to Achieve Penetration Free Simulation

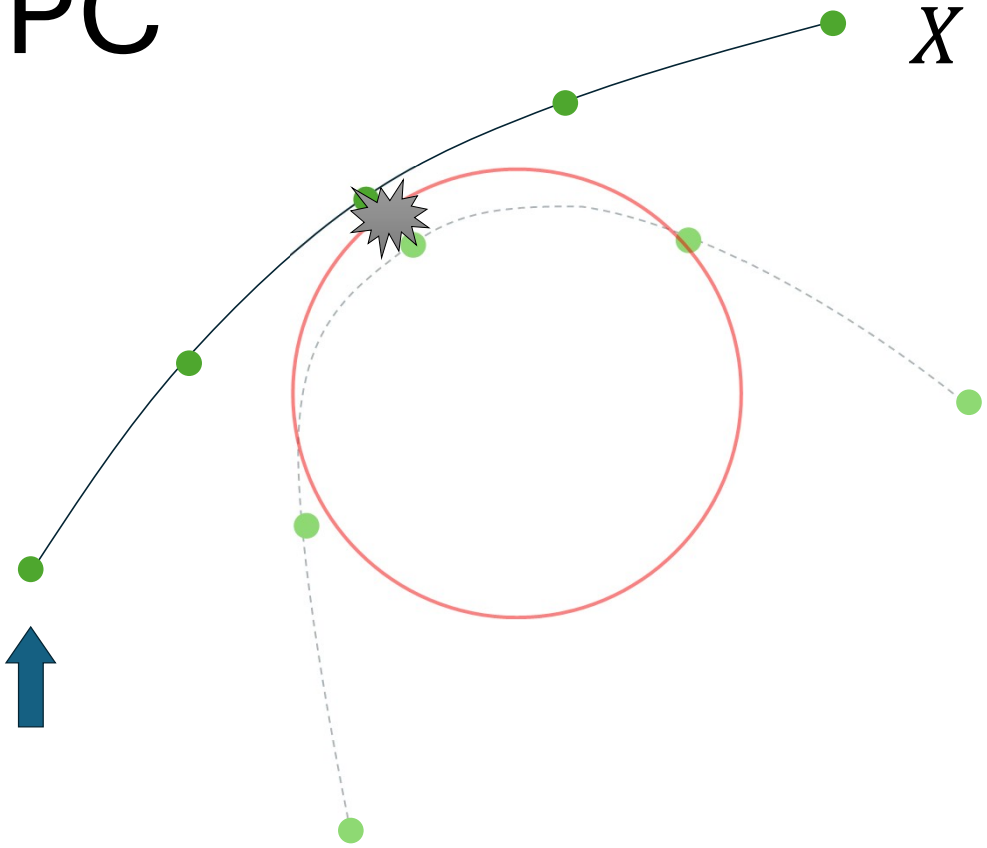
Modify $\nabla \mathbf{x}$ so that it does not cause penetration

- CCD collision culling
- Penetration-free conservative deformation bounds



X:Penetration- free position
ΔX:Optimization Step
t: Time of impact by IPC

IPC

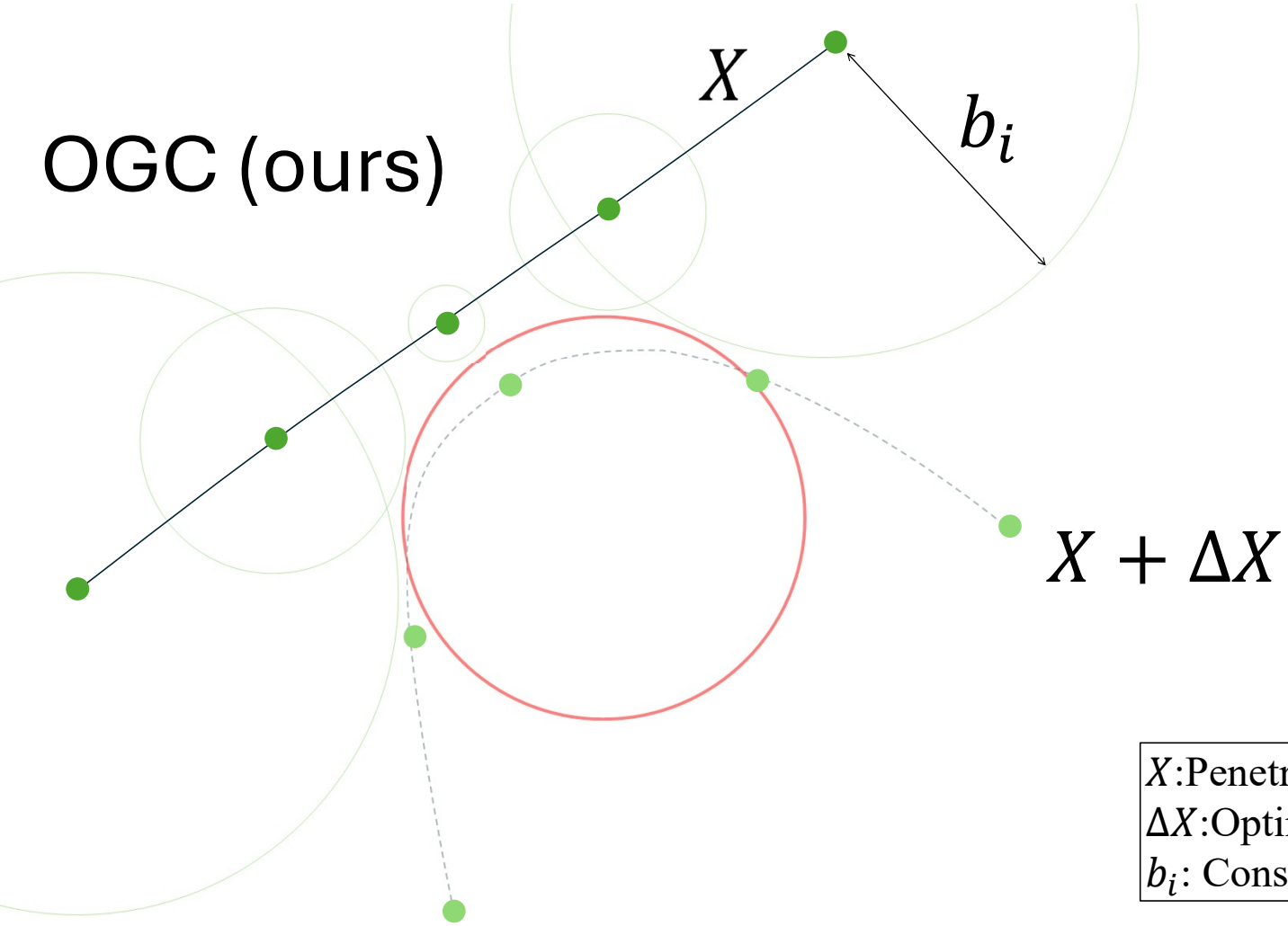


$X + t\Delta X$

$X + \Delta X$

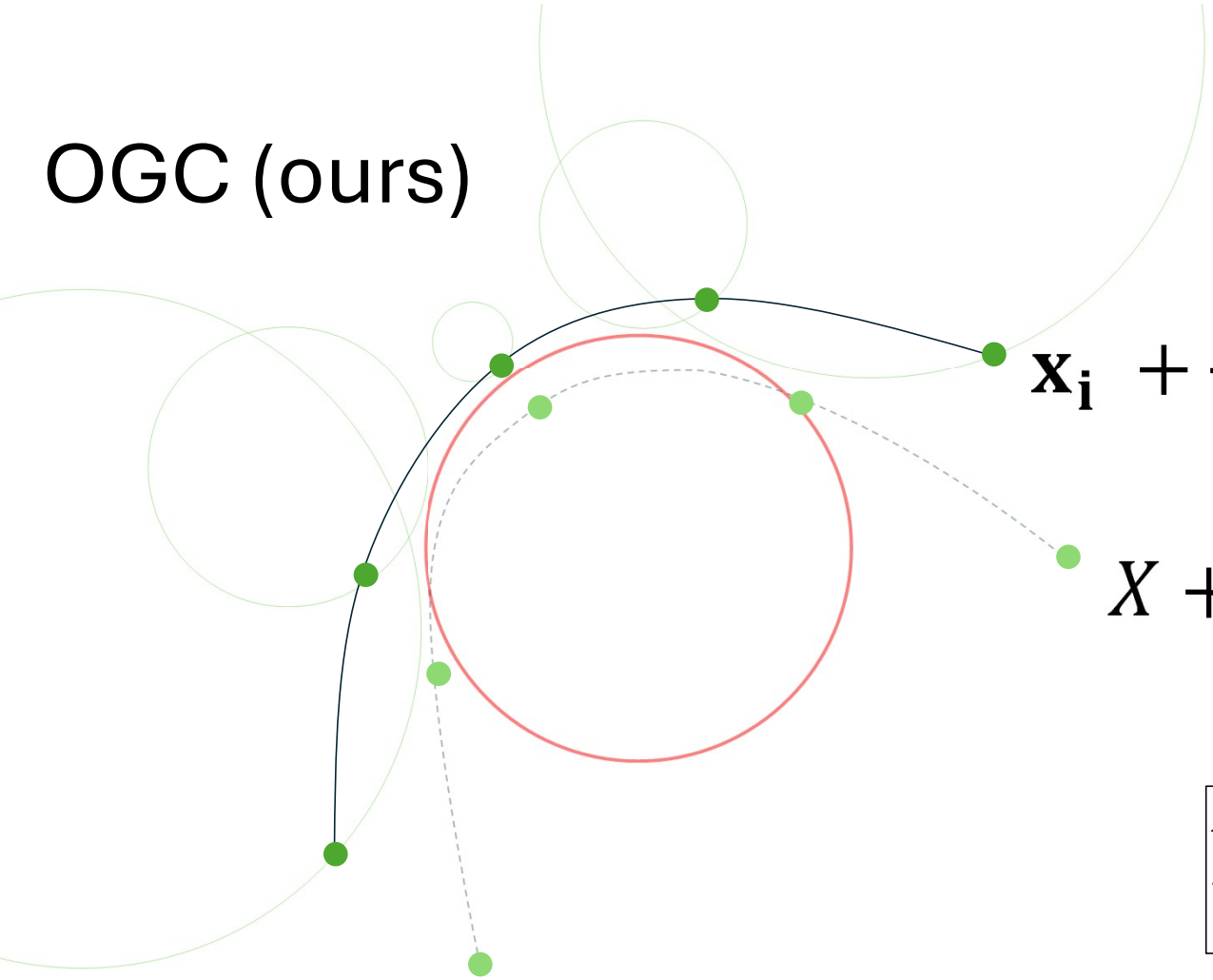
X:Penetration- free position
ΔX:Optimization Step
t: Time of impact by IPC

OGC (ours)



X:Penetration- free position
ΔX:Optimization Step
b_i: Conservative bound for vertex i

OGC (ours)

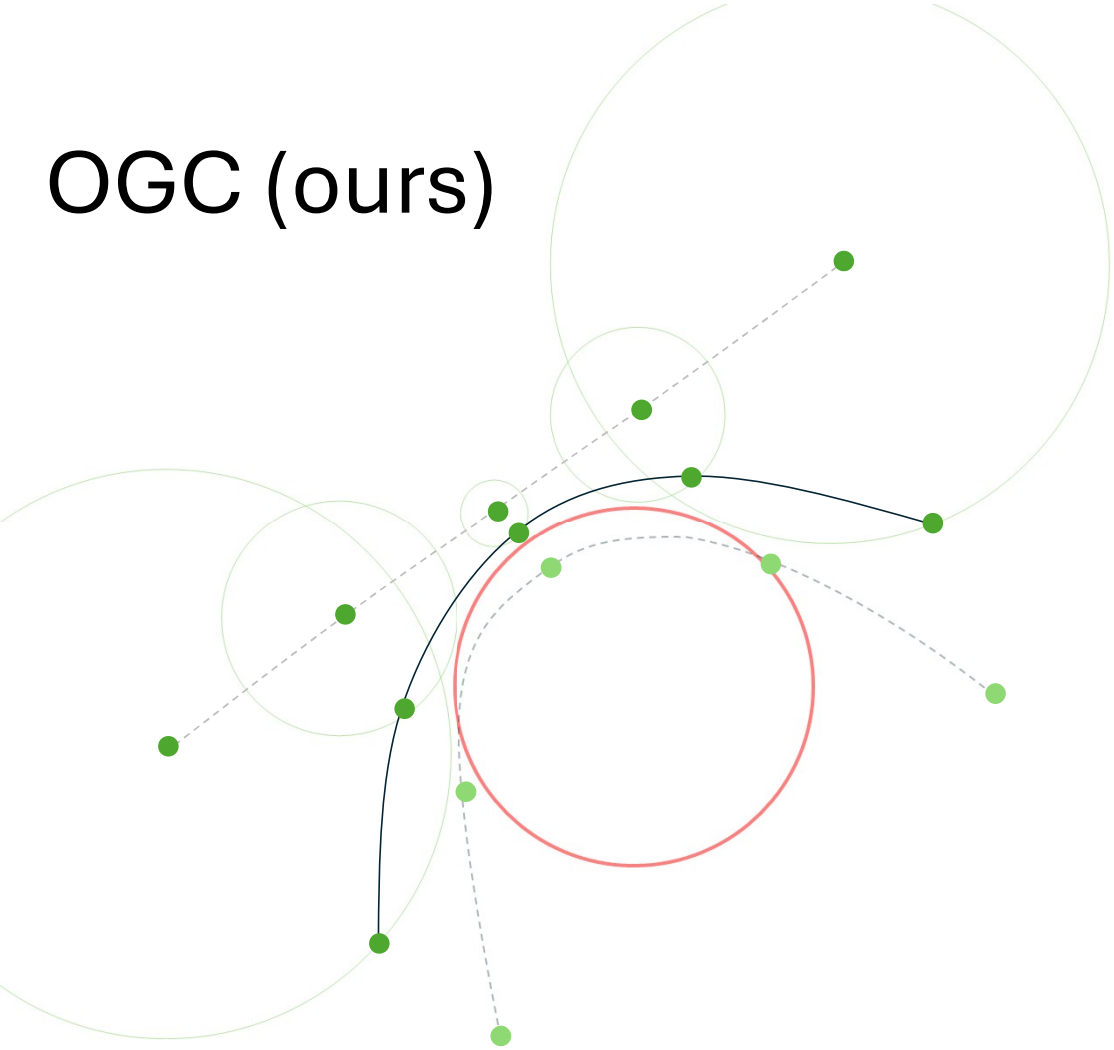


$$\mathbf{x_i} + \frac{b_i}{\|\mathbf{\Delta x_i}\|}\mathbf{\Delta x_i}$$

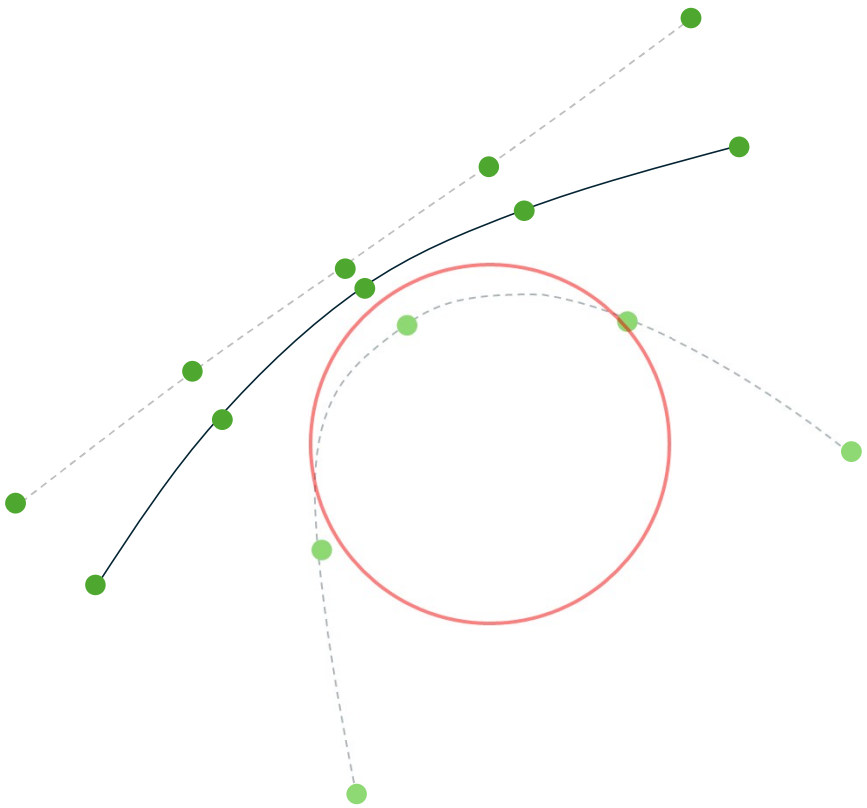
$$X + \Delta X$$

X:Penetration- free position
ΔX:Optimization Step
b_i: Conservative bound for vertex i

OGC (ours)



IPC



Conservative Bound

conservative bound for each vertex v:

$$b_v = \gamma_p \min(d_{\min,v}, d^E_{\min,v}, d^T_{\min,v}), \tag{21}$$

where $0 < \gamma_p < 0.5$ is a relaxation parameter and $d_{\min,v}$ is v's minimal distance to all the facets that do not include v:

$$d_{\min,v} = \min_{t \in \mathcal{T}, v \notin t} dis(\mathbf{x}_v, t), \tag{22}$$

and $d^E_{\min,v}$ is the minimal value of v's neighbor edges' minimal distances to all other edges:

$$d^E_{\min,v} = \min_{e \in \mathcal{E}_v} d_{\min,e}, \tag{23}$$

$$d_{\min,e} = \min_{e' \in \mathcal{E}, e \cap e' = \emptyset} dis(e, e'), \tag{24}$$

and $d^T_{\min,v}$ is the minimal value of v's neighbor facets' minimal distances to all other vertices:

$$d^T_{\min,v} = \min_{t \in \mathcal{T}_v} d_{\min,t}, \tag{25}$$

$$d_{\min,t} = \min_{v' \in \mathcal{V}, v' \notin t} dis(v', t), \tag{26}$$

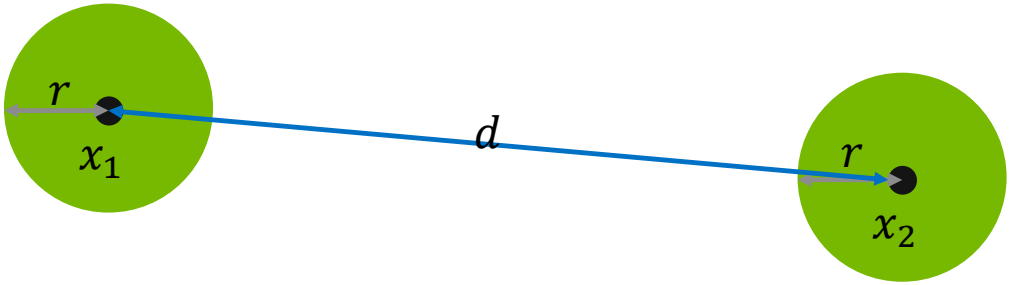
where $\mathcal{E}_v$ and $\mathcal{T}_v$ represents v's neighbor edges and facets respectively.

If the model starts in an intersection-free state X^{prev}, it will remain intersection-free in state if each $\mathbf{x}_v$ satisfies:

$$||\mathbf{x}_v - \mathbf{x}_v^{\text{prev}}|| \leq b_v, \forall v \in \mathcal{V}. \tag{27}$$

Intimidating!

Conservative Bound: a Simpler Case

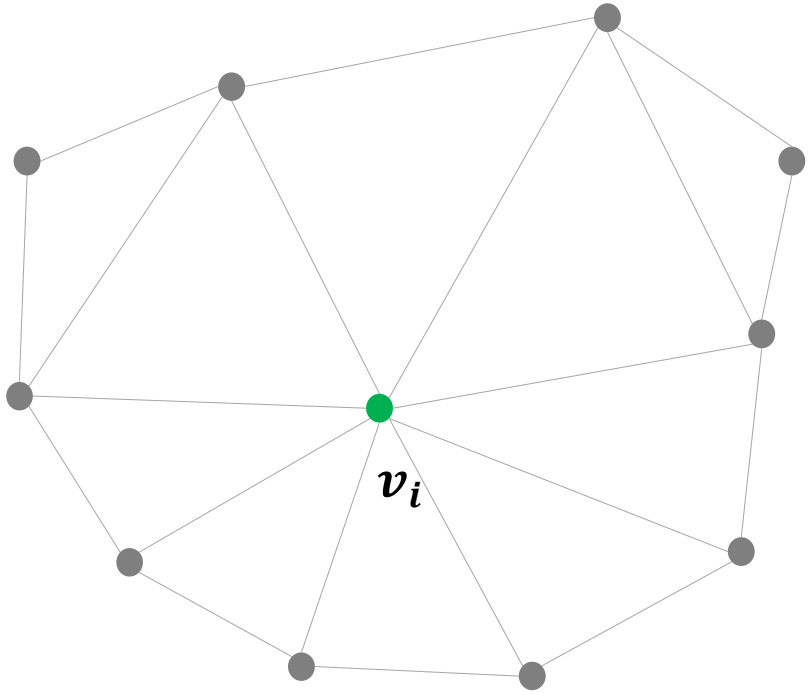


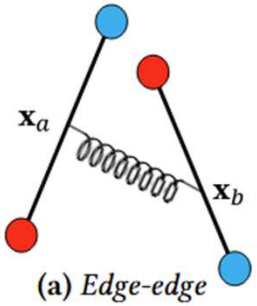
Sphere 1 and 2 will be penetration free if:

$$|\Delta x_1| < \frac{1}{2}d - r$$

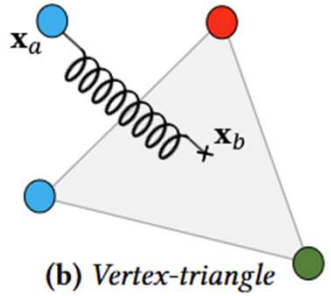
$$|\Delta x_2| < \frac{1}{2}d - r$$

Conservative Bound: for Triangular Mesh





(a) *Edge-edge*



(b) *Vertex-triangle*

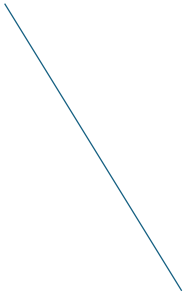
Need to prevent edge-edge and vertex-triangle penetration

Key idea: how does the translation of vertex $\boldsymbol{v_i}$ propagates to its neighbor edges, triangles?

Missions for the first stage

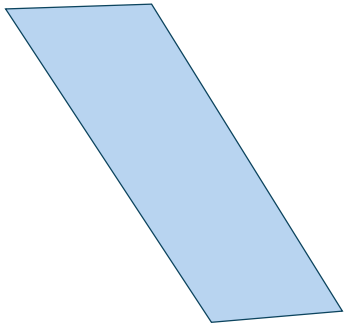
- Understand the material
- Get familiar with geometric processing with python + warp
- Implement the conservative bounds using TrimeshCollisionDetector in Warp, it already calculates: $d_{\min,v}, d^E_{\min,v}, d^T_{\min,v}$
  - I will provide the data as a series of meshes, you need to calculated the bounds of vertex of each mesh
- Optional：
  - Try to prove the effectiveness of the conservative bounds – for fellows who are most interested in the theoretical improvement
  - Try to implementation edge-edge and vertex-triangle CCD – for fellows who are more interested in improving the implementation and heuristic approaches
  - Start to think about how you can make this better!

How to Construct Complied Collision Constraint



a face

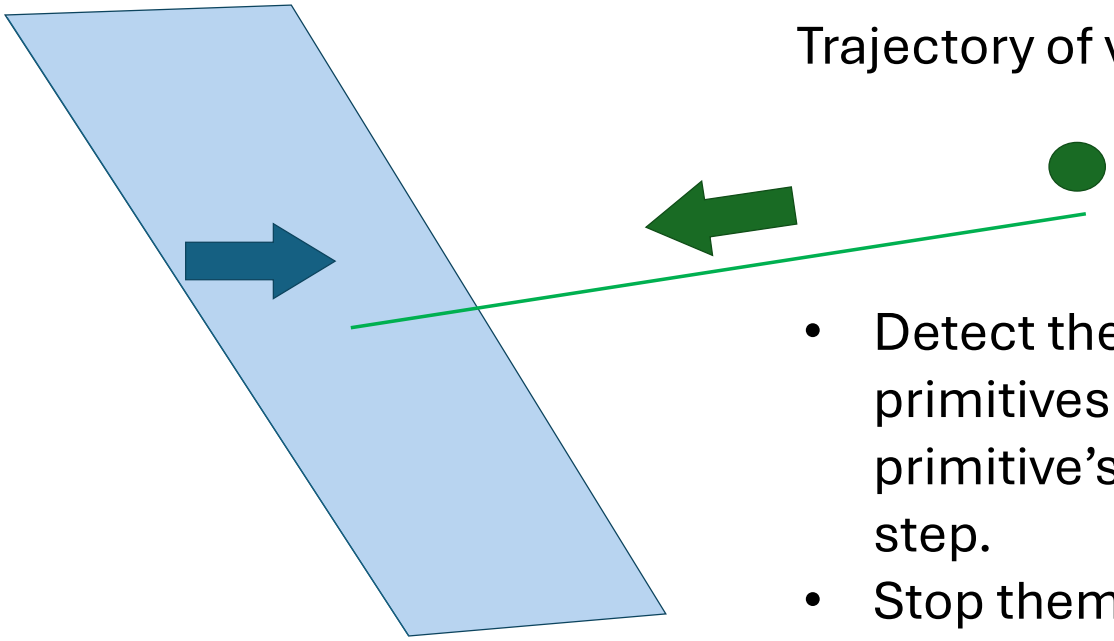
a vertex



Trajectory of a face

Trajectory of a vertex

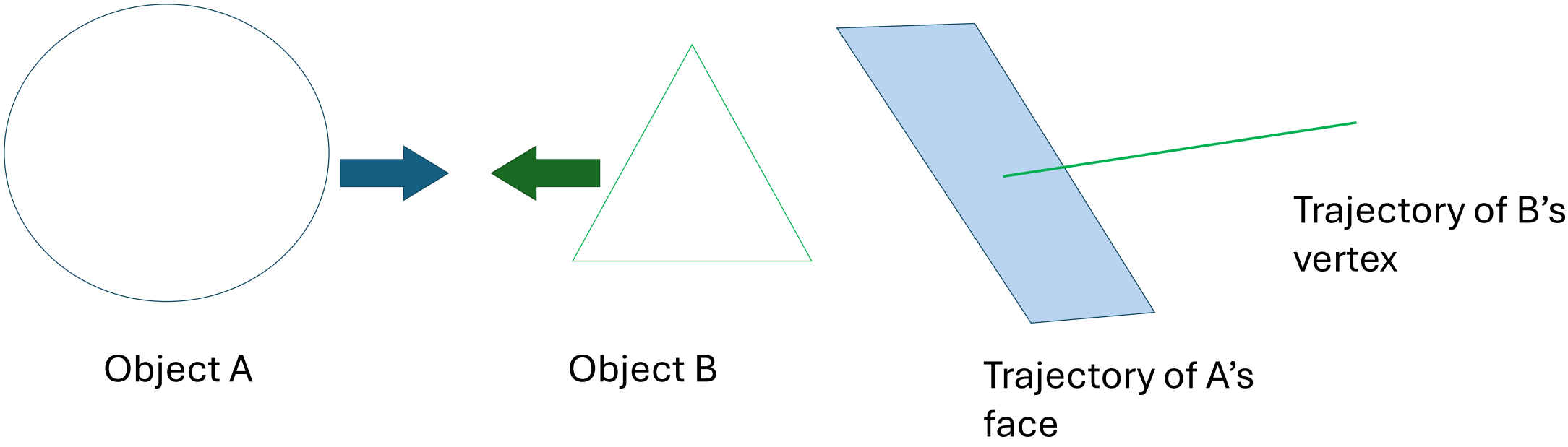
CCD



- Detect the collision using the primitives formed by each of their primitive's trajectory in one time step.
- Stop them at the time when they collide.

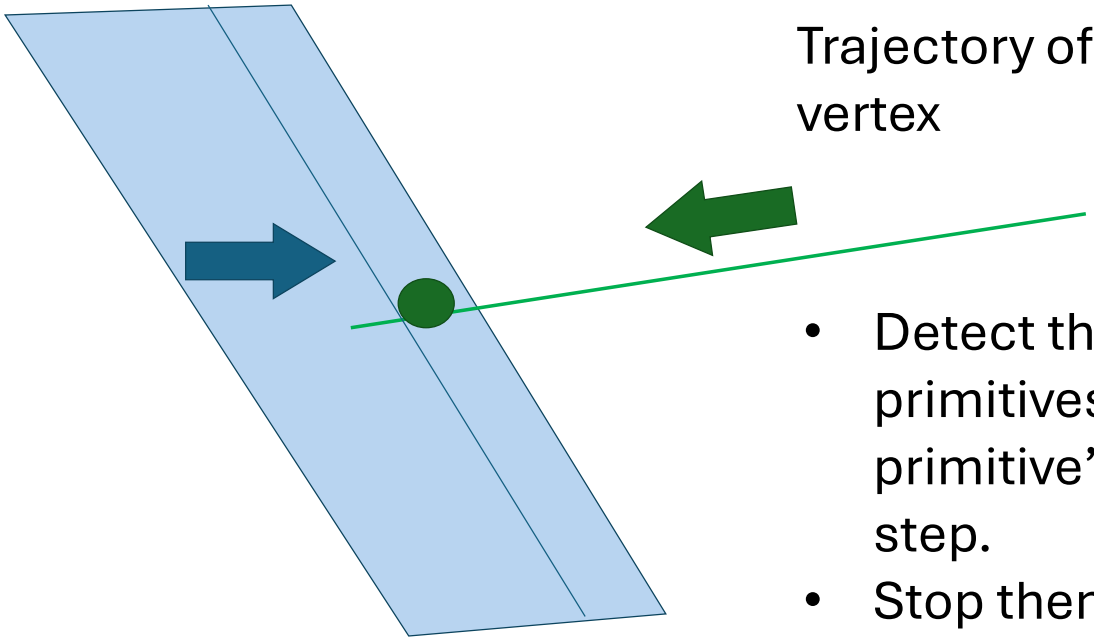
CCD vs DCD

CCD: continuous collision detection



CCD vs DCD

CCD



- Detect the collision using the primitives formed by each of their primitive's trajectory in one time step.
- Stop them at the time when they collide.