

# Nueva propuesta para detección de contacto entre poliedros a gran escala

Yerko Zec



**Universidad  
Andrés Bello®**  
Conectar • Innovar • Liderar

December 12, 2019

# Contenido

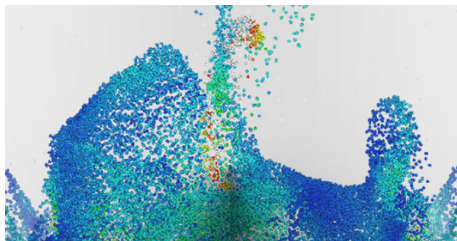
- 1 Motivación
- 2 Marco teorico
- 3 Problema
- 4 Objetivo
- 5 Metodología
- 6 Resultados
- 7 Conclusión
- 8 Referencias

# Motivación

- Detección de contacto entre poliedros.
- Simulación de grandes movimientos de rocas propuesto por Cundall [?].
- Nueva propuesta para detección de contacto entre poliedros.
- Problema de detección aún no se encuentra resuelto en su totalidad.

# Marco teorico

- En 1971 Cundall [?] elaboró un método llamado Discrete Element Method (DEM).
- DEM está dividido en 3 fases: neighbor search, contact detection y force resolution.
- Esta propuesta simula grandes movimiento de partículas.
- Hasta la fecha existen más de 15 algoritmos para la detección de contacto entre partículas.

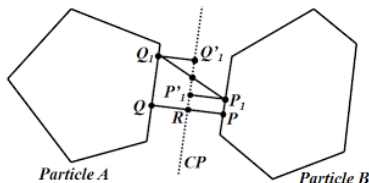


# Algunos algoritmos

- En 1988 Cundall[?] propuso un algoritmo llamado Common-Plane (CP)
- Mencionando algunos algoritmos utilizados a lo largo de la historia son: CP, FCP, MR, SLM, Multi-grid y MSC entre otros.
- Estos algoritmos empezaron desde la misma base que ocupa el algoritmo Common-Plane.

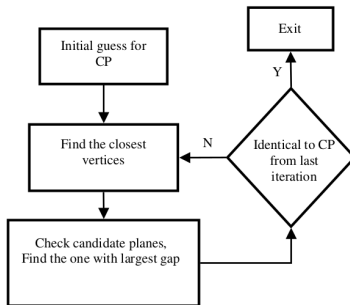
# Common-Plane

- Common-Plane(CP), fue en primera instancia creado por Cundall en 1988.
- Principio del algoritmo: Localizar un plano entre dos figuras e ir calculando la distancias.



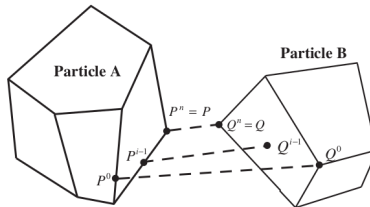
# Fast Common-Plane

- En 2004 Nezami [?] propuso una nueva propuesta para el cálculo del CP y se llamó Fast Common-Plane (FCP).
- Con esta nueva propuesta mejoró el orden del algoritmo por ello la eficiencia.



# Shortest Link Method

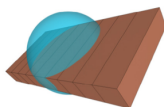
- Nezami en 2006 [?] propuso otro algoritmo ocupando como base FCP.
- En base a resultados expuestos por Nezami [?], el SLM es 17 veces más rápido que otros algoritmos convencionales.



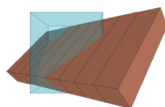


# Multi-shell Contact Detection

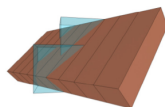
- Fue desarrollado por Zhuang el 2014 [?].
- La detección de contacto y la búsqueda de vecindario.
- MSC como método tanto de detección de vecindario como detección de contacto es uno de los más eficiente, según concluye Zhuang.



(d) 3D NBS



(e) 3D DESS



(f) 3D MSC

# Problema

- Detección de contacto computacionalmente costoso.
- Análisis de distintos tipos de colisiones.

# Objetivo

## Objetivo General

- Desarrollar una nueva propuesta que detecte colisiones entre poliedros.

## Objetivo Específico

- Desarrollar una representación geométrica para los cuerpos rígidos.
- Desarrollar un algoritmo de detección de contactos para vértices y aristas.
- Desarrollar un algoritmo de detección de contacto entre cuerpos rígidos de orden no mayor a  $\mathcal{O}(N)$ , donde  $N$  es la cantidad de cuerpos de la simulación.





# Metodología

- Se realizó un estudio sobre el tema propuesto, donde se recopilaron los datos pertinentes.
- Se generó una línea de tiempo donde se señalaron los trabajos más relevantes al tema.
- Desarrollo de algoritmo en 2-D.
- Upgrade del algoritmo a 3-D.
- Optimización de código, con el fin de probar un set de prueba **Cambiar el ultimo punto para que se entienda mejor.**

- Principales framework de trabajo son github, pycharm, kile y overleaf.



- Para el control de versiones del código se utilizó un repositorio en github.

 <b>yerkozec</b> modified: diapos reuniones/hito1/hito1.tex ...		Latest commit 2d61ebe 10 hours ago
 ptgit	modified: diapos reuniones/hito1/hito1.tex	10 hours ago
 .gitignore	Initial commit	13 hours ago
 README.md	Update README.md	13 hours ago

# Resultados

## Código 2-D

```
# -----generador de figura geometricas 2d-----  
  
def genfig():  
    vectorx = random.sample(range(30), 2)  
    vectory = random.sample(range(30), 2)  
    vectorz = random.sample(range(30), 2)  
    return (vectorx, vectory, vectorz)  
  
def genpunto():  
    punto = random.sample(range(30), 3)  
    return punto  
  
def genarista():  
    vectorx = random.sample(range(30), 2)  
    vectory = random.sample(range(30), 2)  
    vectorz = random.sample(range(30), 2)  
    return vectorx, vectory, vectorz
```

# Resultados

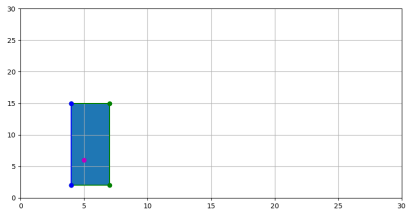
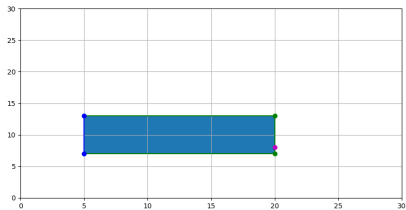
## Detección de punto

```
def pointdetection(figura , punto):  
    vertex = np.array(getvertex(figura))  
    U0 = np.array(vertex[0])  
    U1 = np.array(vertex[1])  
    U2 = np.array(vertex[2])  
    U3 = np.array(vertex[3])  
    a = U1 - U0  
    c = U2 - U0  
    b = U3 - U0  
    B = getB(figura)  
    V0 = punto[0] - U0[0]  
    V1 = punto[1] - U0[1]  
    V2 = punto[2] - U0[2]  
    V = np.array([[V0], [V1], [V2]])  
  
    alpha = np.linalg.solve(B, V)  
  
    if (0 <= alpha[0] <= max(a)):  
        resultx = True  
    if (0 <= alpha[1] <= max(c)):  
        resulty = True  
    if (0 <= alpha[2] <= max(b)):  
        resultz = True  
    return resultx and resulty and resultz
```



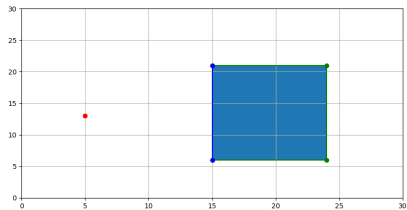
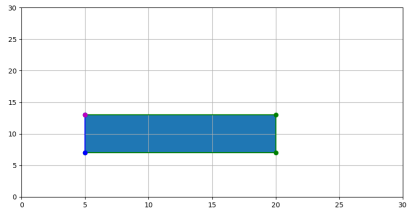
# Resultados

## Prueba 2-D



# Resultados

## Prueba 2-D



# Resultados

## Codigo 3-D

```
# ----- get input for the figure -----
U0 = input()
size = input()
point = input()
Vj = input()
Vk = input()

# ----- manage data -----
U0 = list(map(float, U0.strip().split()))
a, b, c = list(map(float, size.strip().split()))
point = list(map(float, point.strip().split()))
Vj = list(map(float, Vj.strip().split()))
Vk = list(map(float, Vk.strip().split()))

# ----- Config Plano catersiano -----

ax = plt.axes(projection='3d')
ax.set_xlim3d(0, 30)
ax.set_ylim3d(0, 30)
ax.set_zlim3d(0, 30)
ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
ax.set_zlabel('Z axis')
```

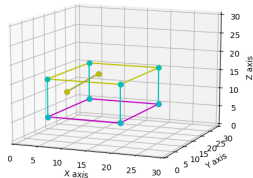
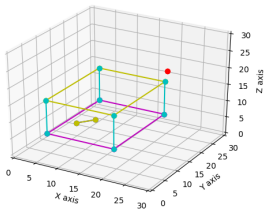
# Resultados

## Codigo 3-D

```
def aristadetection(U0, a, b, c, Vj, Vk):
    S = np.zeros([3, 2])
    U0 = np.asarray(U0)
    Vj = np.asarray(Vj)
    Vk = np.asarray(Vk)
    Vjprima = Vj - U0
    Vkprima = Vk - U0
    B = getB()
    Ro = np.array([[0, a],[0, b],[0, c]])
    W = Vkprima - Vjprima
    for i in range(0, 3):
        S[i] = (Ro[i] - Vjprima[i]) / W[i]
    S = np.sort(S)
    inside = insidedetection(S)
    if (inside):
        for i in range(0, 3):
            if (S[i][0] <= 0):
                S[i][0] = 0
            if (S[i][1] >= 1):
                S[i][1] = 1
        Si = np.max(S[:, 0])
        Sf = np.min(S[:, 1])
        Si = (Vjprima + Si * (Vkprima - Vjprima)) + U0
        Sf = (Vjprima + Sf * (Vkprima - Vjprima)) + U0
        return Si, Sf, True
    else:
        return Vj, Vk, False
```

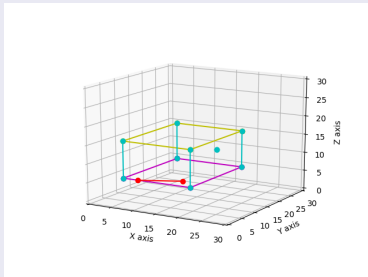
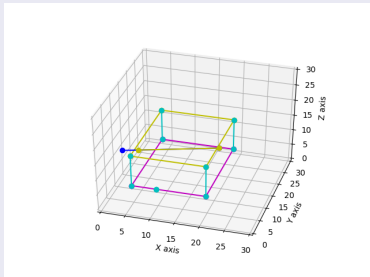
# Resultados

## Prueba 3-D



# Resultados

## Prueba 3-D



# Conclusión

- 
- En esta fase del desarrollo de la propuesta se logró detectar si un vértice y una arista están en contacto con el poliedro.

# Referencias



# Nueva propuesta para detección de contacto entre poliedros a gran escala

Yerko Zec



**Universidad  
Andrés Bello®**  
Conectar • Innovar • Liderar

December 12, 2019