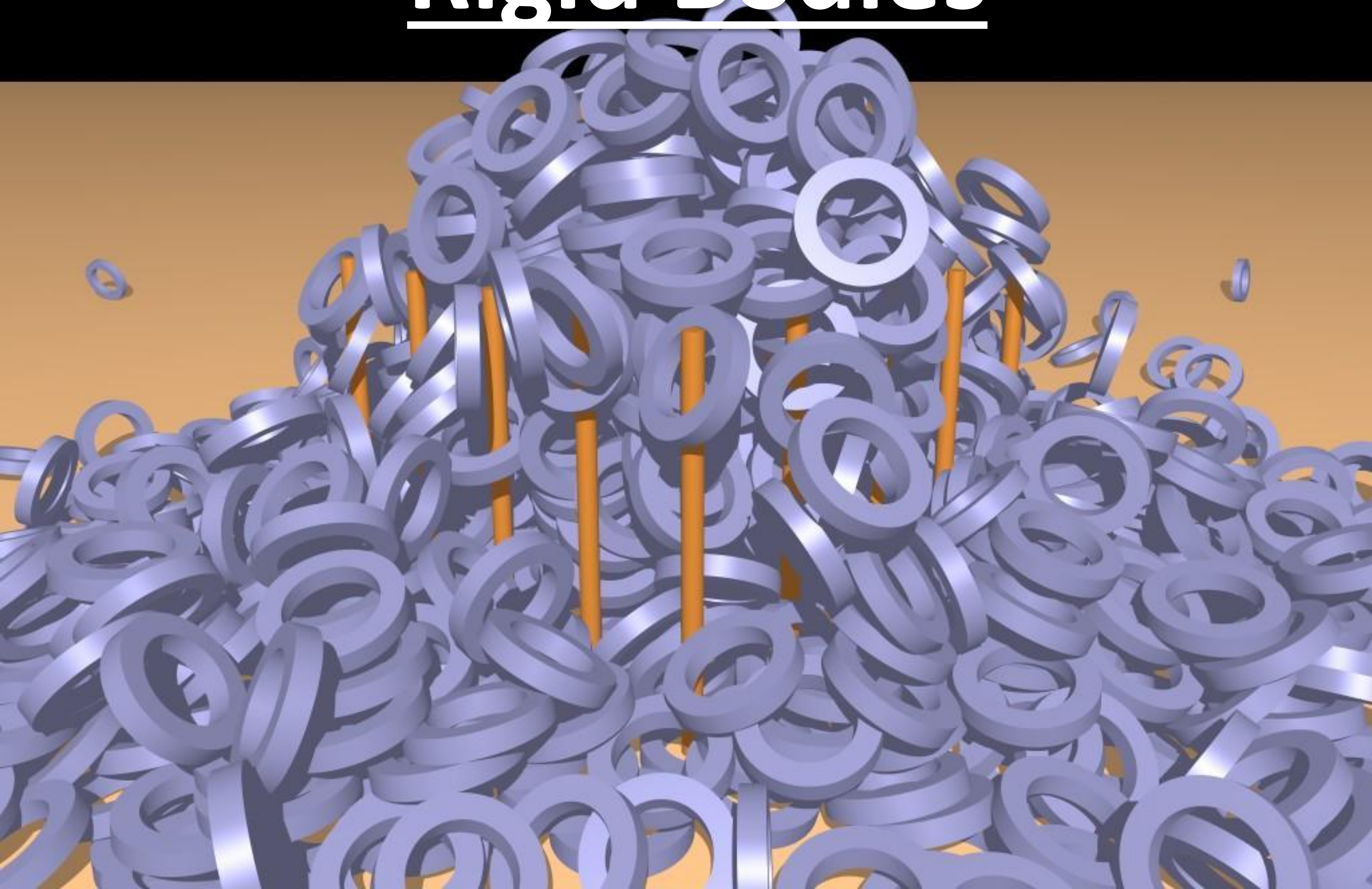
# Rigid Bodies



## Simulation Homework

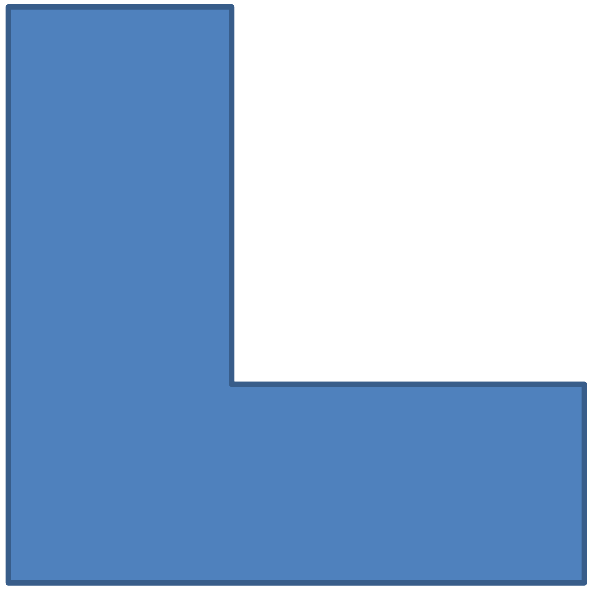
- Build a particle system based either on F=ma or procedural simulation
  - Examples: Smoke, Fire, Water, Wind, Leaves, Cloth, Magnets, Flocks, Fish, Insects, Crowds, etc.
- Simulate a Rigid Body
  - Examples: Angry birds, Bodies tumbling, bouncing, moving around in a room and colliding, Explosions & Fracture, Drop the camera, Etc.

**Thursday**

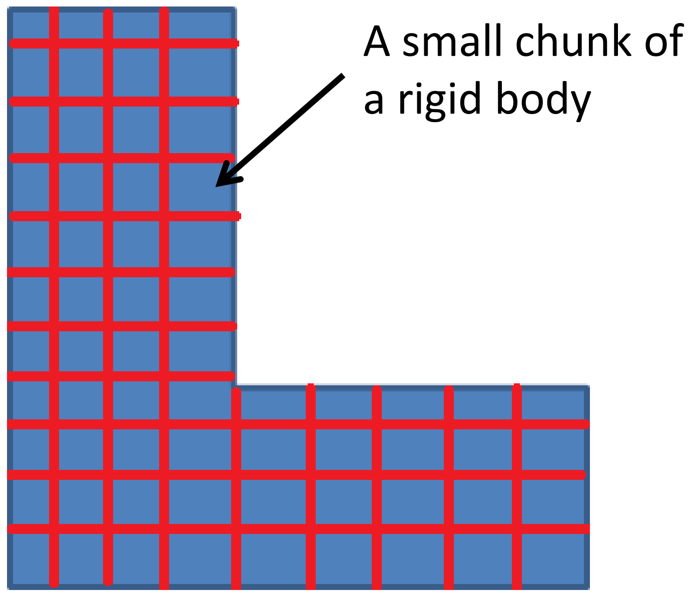
**Simulation &**

**Unity**

# Rigid Body as “Particle Chunks”



Consider a rigid body



It can be broken up into chunks (or elements), and each chunk can be treated as a single particle if it is small enough

# Center of Mass

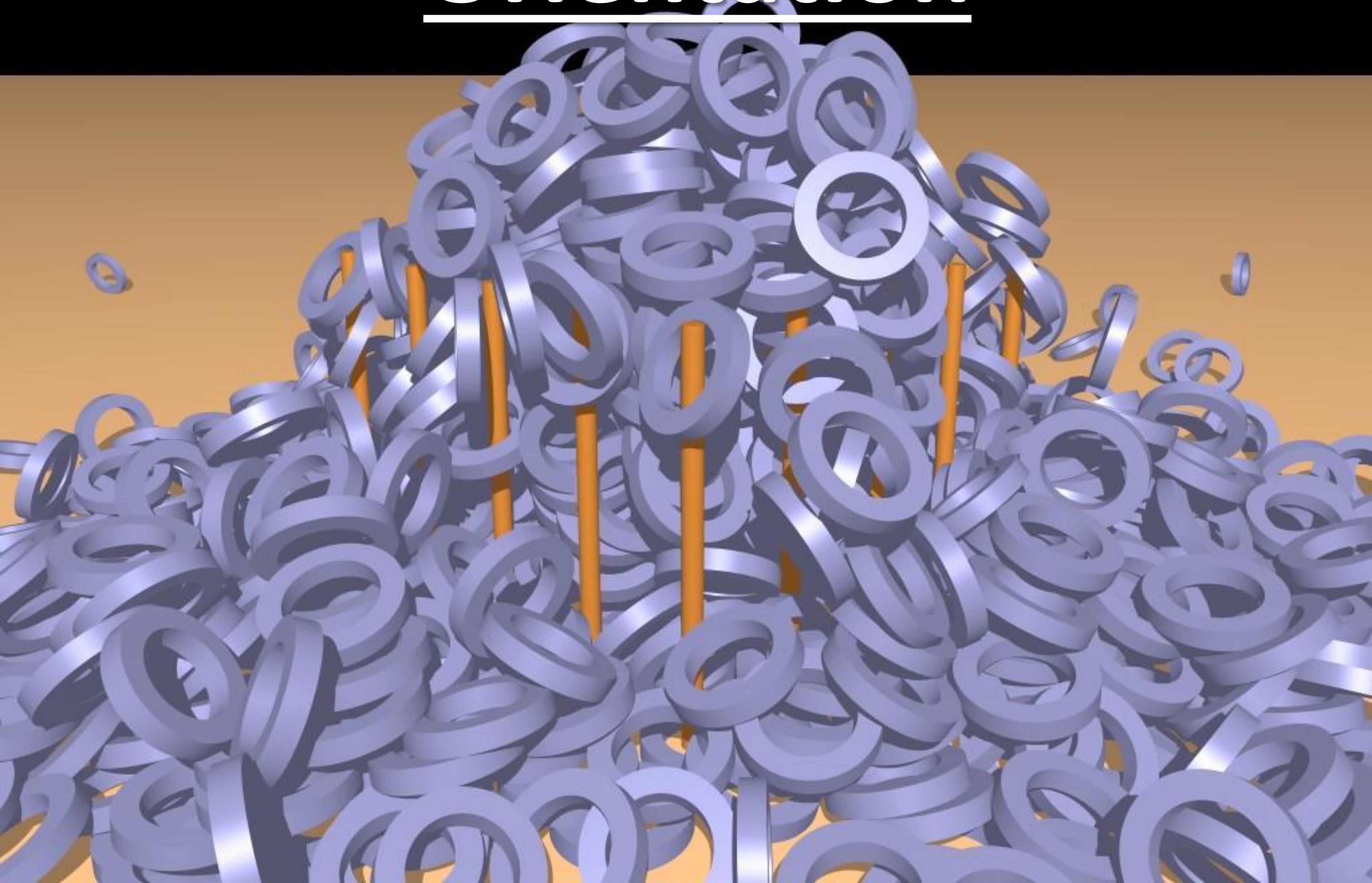
- A body composed of $n$ particle "chunks" with masses $m_i$ has total mass

$$M = \sum_{i=1}^{n} m_i$$

- If the particles have positions $\vec{x}_i$, the center of mass is

$$\bar{x} = \frac{\sum_{i=1}^{n} m_i \vec{x}_i}{M}$$

- The center of mass for a rigid body has position $\bar{x}$ and translational velocity $\bar{v}$ similar to a single particle
- When we refer to the position and velocity of a rigid body, we are referring to the position and translational velocity of its center of mass
- The center of mass obeys the same ODEs for position and velocity as a particle does, but using the TOTAL mass and the NET force



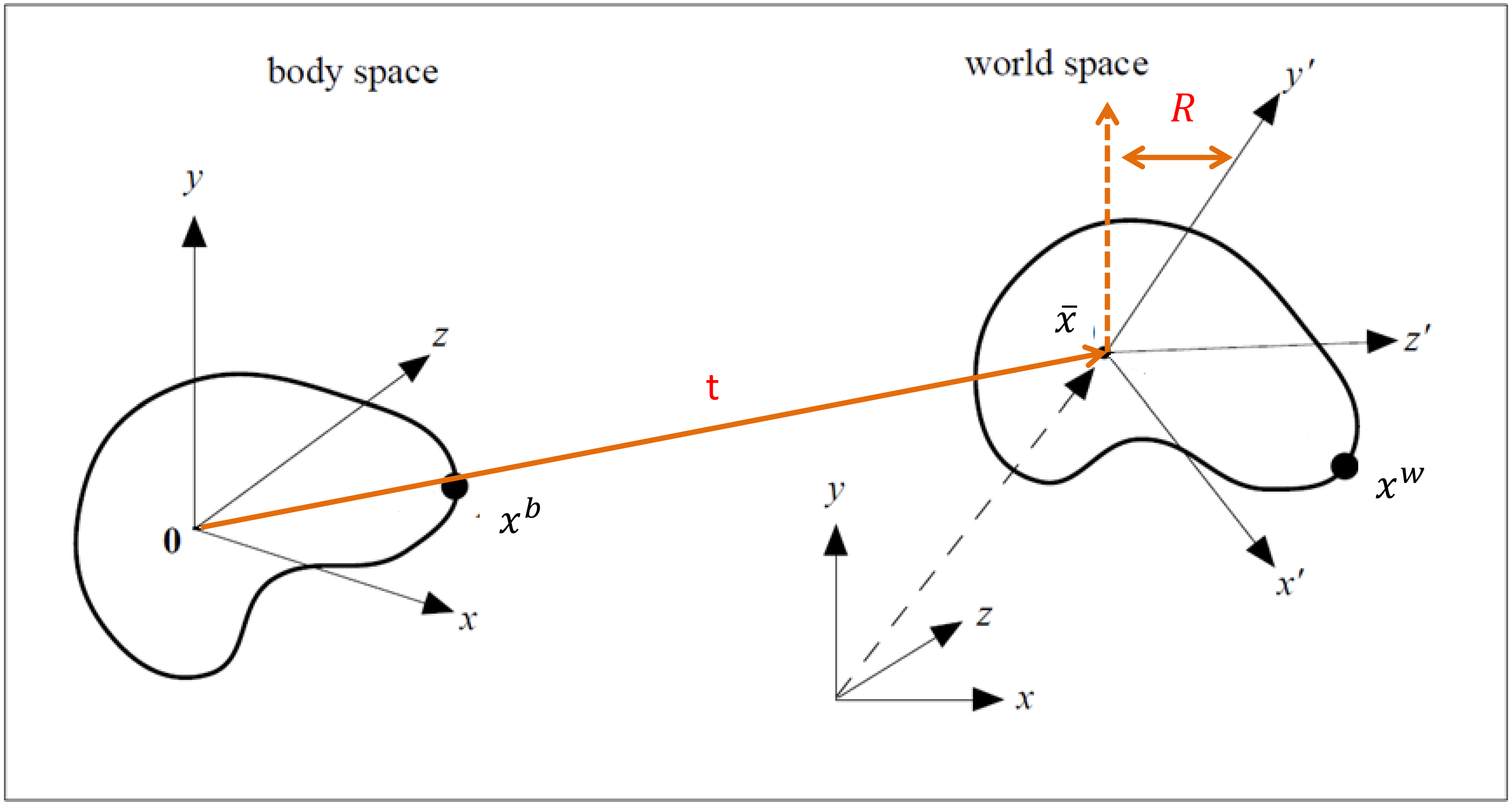
## Orientation

# Orientation

- A rigid body can rotate or change its orientation while its center of mass is stationary
- Different ways to keep track of the rotation $R$:
  - 3x3 Matrix, 3 Euler angles, 1 Quaternion
- Place a coordinate system at the center of mass in object space
- The rotation $R$ rotates the rigid body (and the object space coordinate system) into its world space orientation
- Recall: the columns of $R$ are the three object space axes in their world space orientations

# Combining Position and Orientation

- The rigid body has an intrinsic coordinate system in its object space, with its center of mass at the origin
- It's put into world space with a translation and a rotation



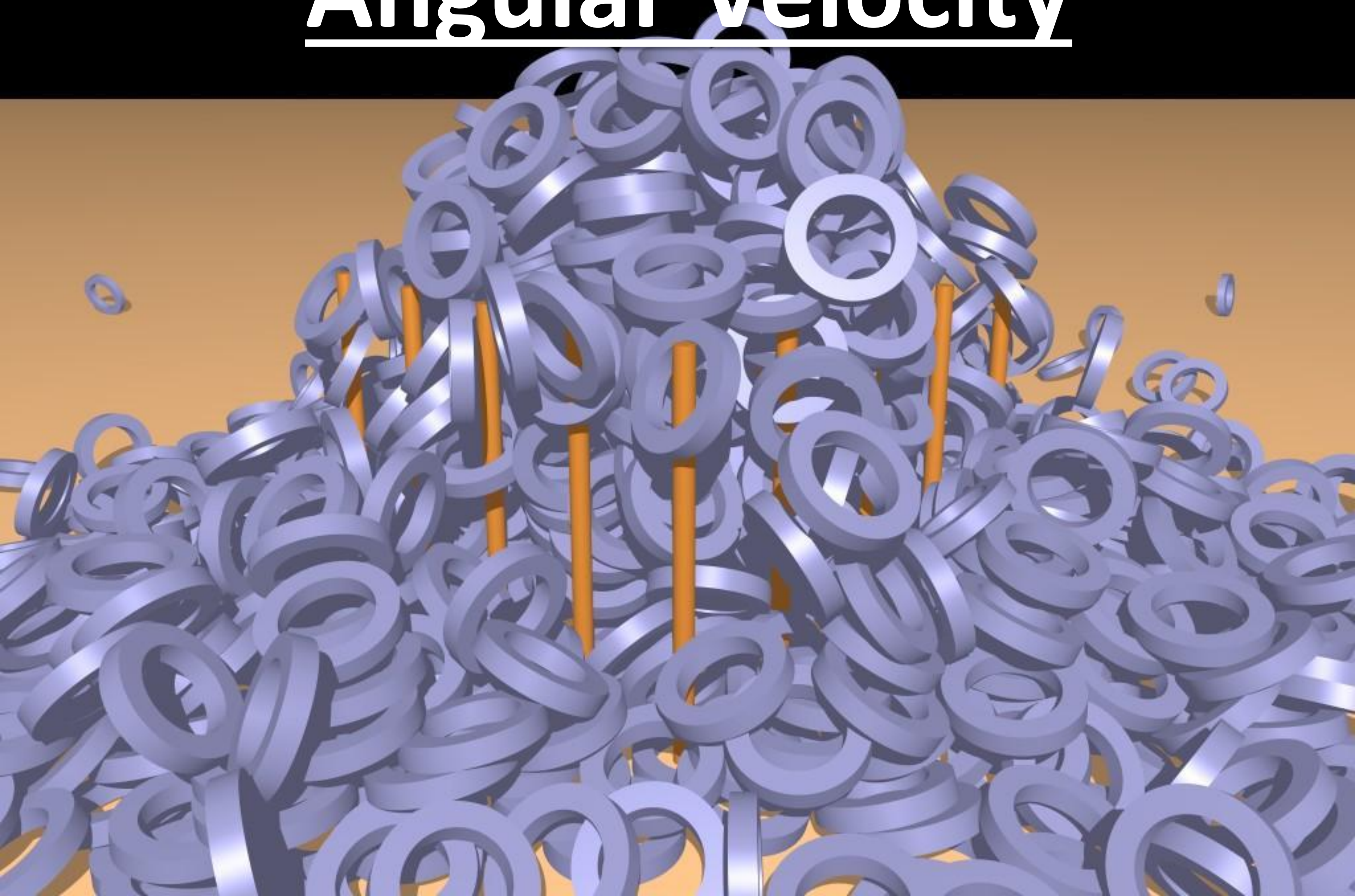
## Combining Position and Orientation

- The translation of the origin of the object space coordinate system is given by the position of the center of mass $\bar{x}$
- The world space orientation of the object space coordinate system is given by $R$
  - Assume $R$ is a matrix, equivalently expressed as a unit quaternion
- A point $x^o$ in object space has a world space location

$$x^w = \bar{x} + Rx^o$$

  - Notice that the center of mass (at the origin) maps to $\bar{x}$
  - Notice that (1,0,0), (0,1,0), (0,0,1) are all rotated by $R$ before being translated by $\bar{x}$

# Angular Velocity



# Angular Velocity

- Both $\bar{x}$ and $R$ are functions of time
- The rate of change of the position of the center of mass $\bar{x}$ with respect to time is the translational velocity of the center of mass $\bar{v}$
- (From our quaternion discussion…) The orientation of the body is changing as it is rotated about some axis $\hat{n}$ emanating from the center of mass
- The rate of change of the orientation $R$ is given by the world space angular velocity $\omega$
  - its direction is the axis of rotation, $\hat{n}$
  - Its magnitude is the speed of rotation
- The pointwise velocity of any point $\mathrm{x}$ on the rigid body is given by

$$v_p = \bar{v} + \omega \times (x - \bar{x}) = \bar{v} + \omega \times r$$

where r is the moment arm and $\times$ is the cross-product

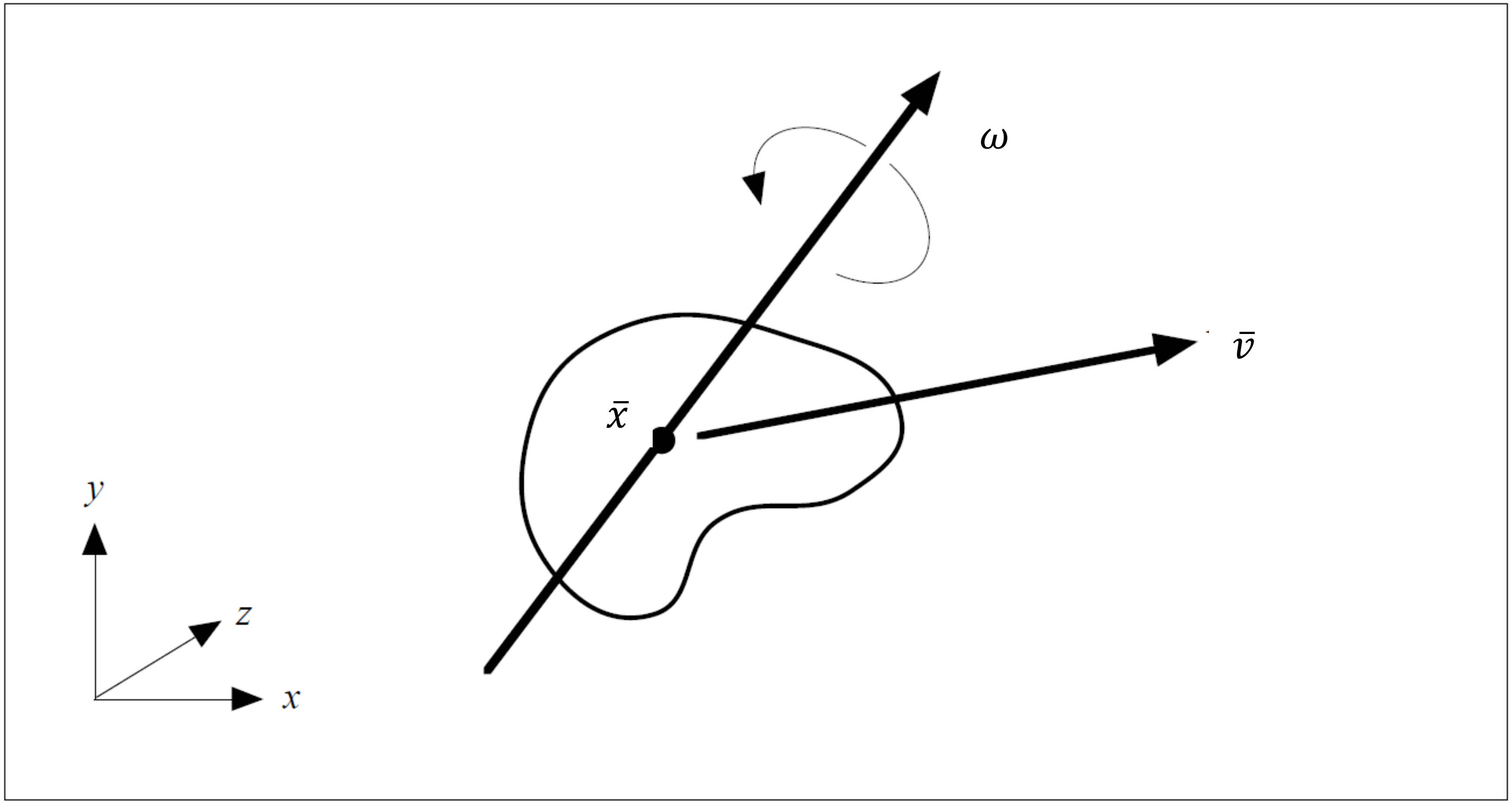
## Aside: Cross Product Matrix

- Given vectors $a = [a_1, a_2, a_3]$ and $b = [b_1, b_2, b_3]$, their cross product $a \times b$ can be written as matrix multiplication $a^* b$ by converting $a$ to a cross product matrix
- $a^* = \begin{pmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{pmatrix}$
- ${a^*}^T = -a^*$
- $a^* b = -b^* a$

- Using this new notation, the pointwise velocity of any point x on the object is then given by:

$$v_p = \bar{v} + \omega^* r$$

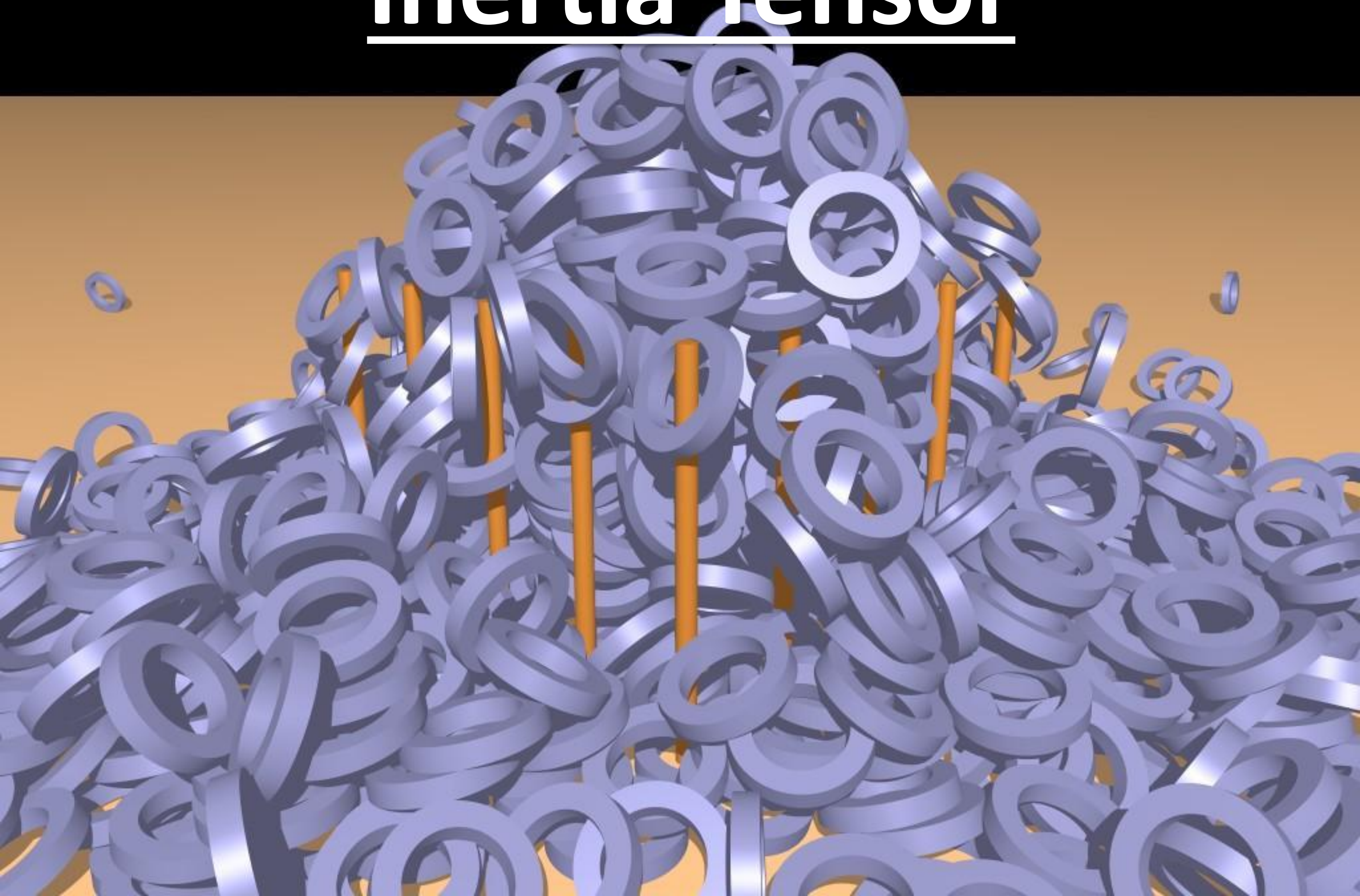
# Linear and Angular Velocity



# ODE for Orientation

- The ODE for orientation (angular position) is given by
$$\dot{R} = \omega^* R$$
- That is, $\dot{R} = \omega \times R$ where the cross product is applied independently to each of the three columns of $R$
- Writing the 3x3 matrix $\omega^*$ and using matrix multiplication $\omega^* R$ automatically performs these 3 cross products

# Inertia Tensor



# Inertia Tensor

- Linear momentum is defined as the product of the mass times the translational velocity
  - Mass is something that resists change in velocity
- Angular momentum $L$ is defined as an "angular mass" times the angular velocity $\omega$
- The "angular mass" is called the moment of inertia (or inertia tensor) $I$ of the rigid body
- If you spin in your chair while extending your legs, and then suddenly pull your legs closer the chair spins faster
  - $I$ reduces when you pull your legs closer.
  - Hence $\omega$ has to increase to keep angular momentum $L = I\omega$ constant

# Inertia Tensor

- For a system of n particles, the angular momentum is

$$L = \sum_i (x_i^w)^* m_i v_i = \sum_i (\bar{x} + r_i)^* m_i v_i = \bar{x}^* M \bar{v} + \sum_i r_i^* m_i v_i$$

  - where $r_i$ is the moment arm of the $i^{th}$particle

- $v_i$ can be written as $\bar{v} + \omega^* r_i$ so

$$L = \bar{x}^* M \bar{v} + \sum_i r_i^* m_i \bar{v} + \sum_i r_i^* m_i \omega^* r_i$$

- $\sum_i r_i^* m_i \bar{v} = (\bar{v}^*)^T \sum_i m_i r_i = 0$ so

$$L = \bar{x}^* M \bar{v} + \sum_i m_i (r_i^*)^T r_i^* \omega = \bar{x}^* M \bar{v} + I\omega$$

  - where $I = \sum_i m_i (r_i^*)^T r_i^*$

# Object Space Inertia Tensor

- We can pre-compute the <u>object space</u> inertia tensor as a 3x3 matrix $I^o$
- Then, the world space inertia tensor is given by the 3x3 matrix

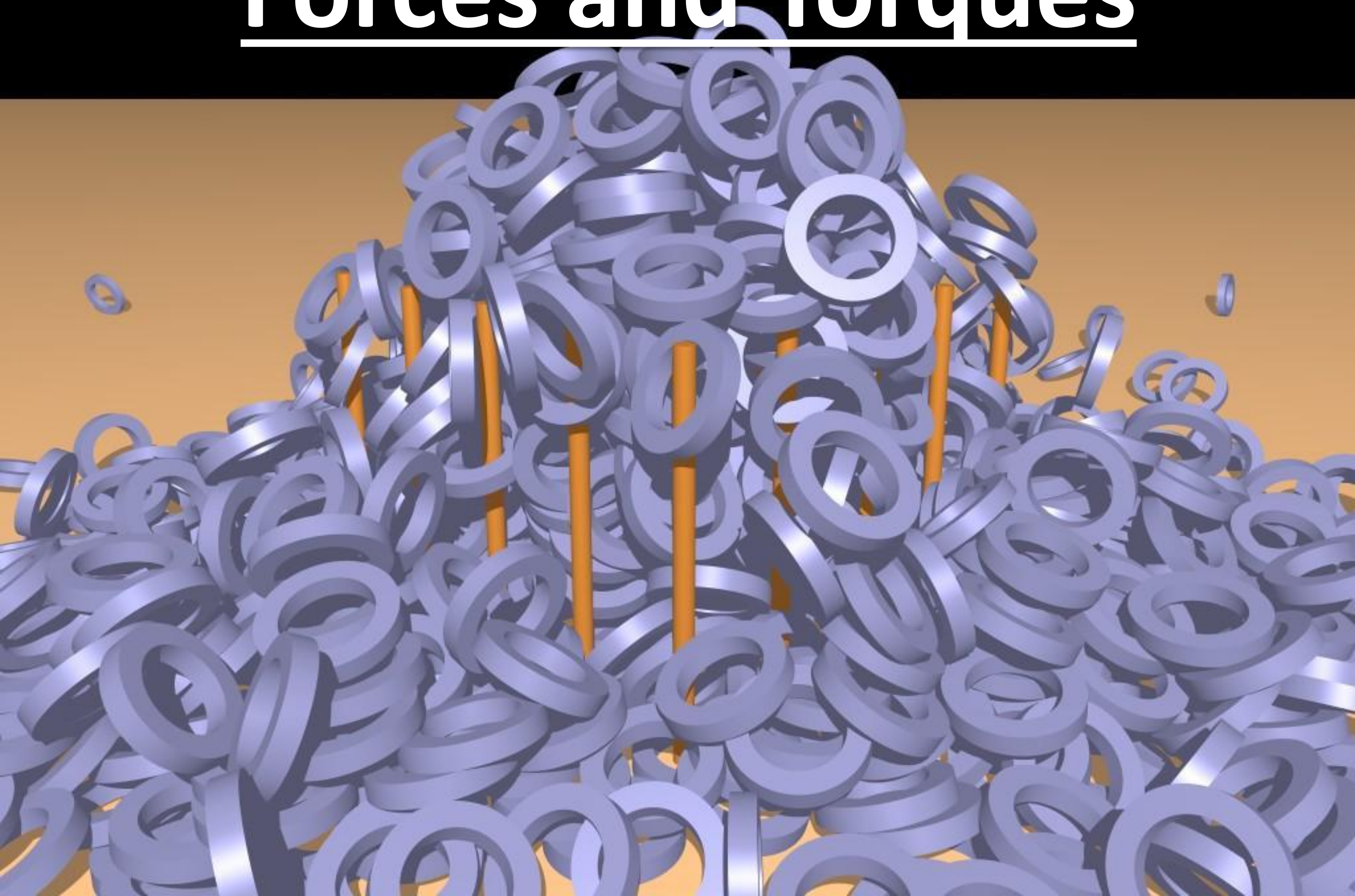
$$I = RI^oR^T$$

- One can compute the SVD of the symmetric 3x3 matrix $I^o$ to obtain $I^o = UDU^T$ where $D$ is a 3x3 diagonal matrix of 3 singular values
- Then rotating the object space rest state of the rigid body by $U^{-1}$ gives a new object space inertia tensor of

$$U^{-1}I^oU^{-T} = U^{-1}UDU^TU^{-T} = D$$

- That is, properly orienting the rest pose of a rigid body in object space gives a <u>diagonal</u> object space inertia tensor $I^o$

# Forces and Torques



# Forces and Torques

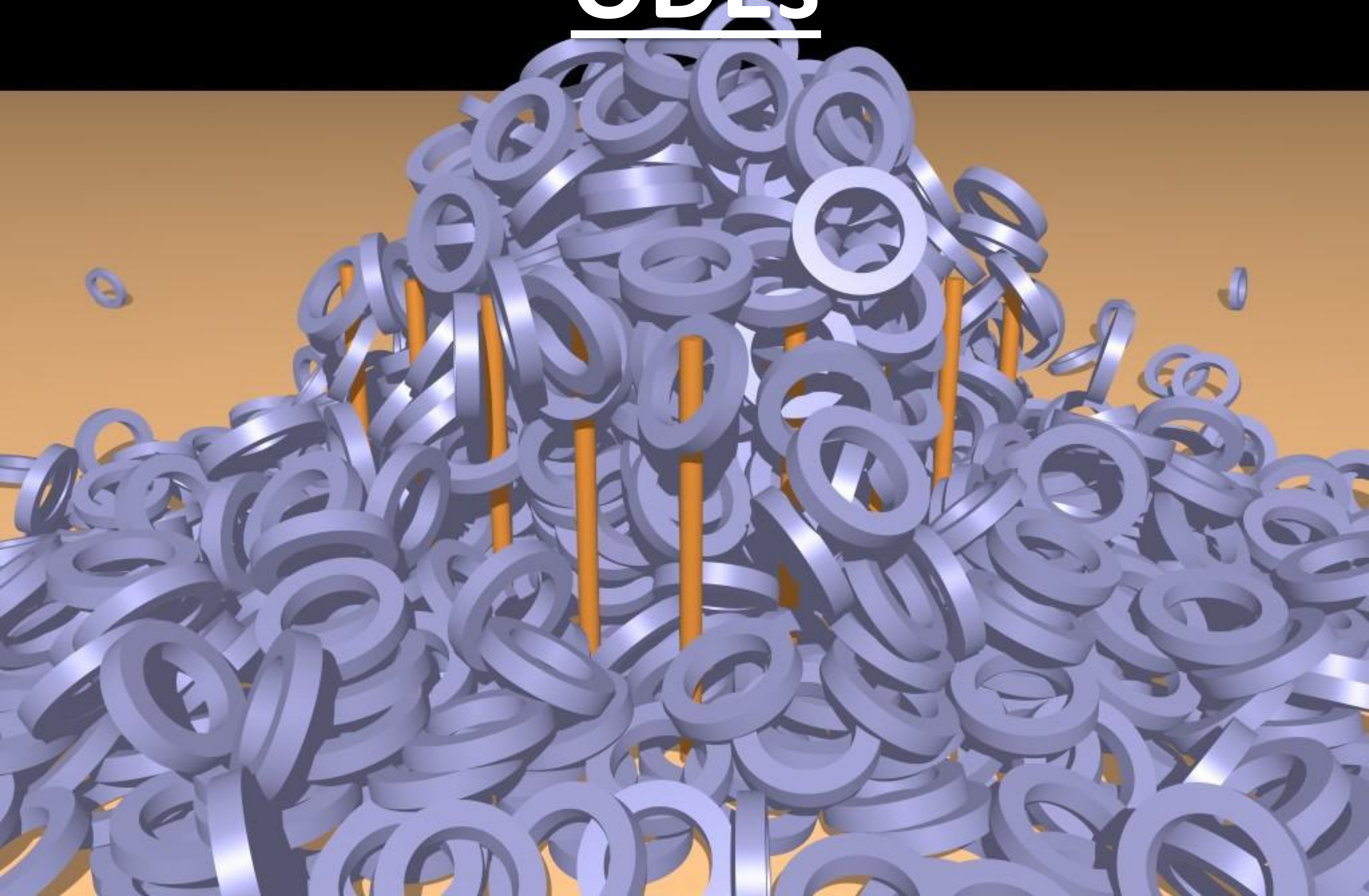
- Newton’s second law for angular quantities
- A force $F$ changes both the linear momentum $M\bar{v}$ and the rotational (angular) momentum $L$
- The change in linear momentum is independent of the point on the rigid body $x$ where the force is applied
- The change in angular momentum does depend on the point $x$ where the force is applied
- The torque is defined as

$$\tau = (x - \bar{x}) \times F = r \times F$$

- The net change in angular momentum is given by the sum of all the external torques

$$\dot{L} = \tau$$

## ODEs



# Rigid Body: Equations of Motion

- State vector for a rigid body

$$X = \begin{pmatrix} \bar{x} \\ R \\ M\bar{v} \\ L \end{pmatrix}$$

- Equations of motion

$$\frac{d}{dt}X = \frac{d}{dt}\begin{pmatrix} \bar{x} \\ R \\ M\bar{v} \\ L \end{pmatrix} = \begin{pmatrix} \bar{v} \\ \omega^{*}R \\ F \\ \tau \end{pmatrix}$$

# Rigid Body: Equations of Motion

- State vector for a rigid body

$$X = \begin{pmatrix} \bar{x} \\ R \\ \bar{v} \\ L \end{pmatrix}$$

- Equations of motion

$$\frac{d}{dt}X = \frac{d}{dt}\begin{pmatrix} \bar{x} \\ R \\ \bar{v} \\ L \end{pmatrix} = \begin{pmatrix} \bar{v} \\ \omega^{*}R \\ F/M \\ \tau \end{pmatrix}$$

Equations of motion for a particle at the center of mass

## Forward Euler Update

$$X^{n+1} = X^n + \Delta t \begin{pmatrix} \bar{v} \\ \omega^* R \\ F/M \\ \tau \end{pmatrix}^n$$

- Newmark for better accuracy and stability, etc…
- Better results are obtained on the second equation by rotating the columns of $R$ directly using the vector $\Delta t \omega$
- Need to periodically <u>re-orthonormalize</u> the columns of $R$ to keep it a rotation matrix

# Rigid Body: Equations of Motion

- State vector for a rigid body

$$X = \begin{pmatrix} \bar{x} \\ q \\ \bar{v} \\ L \end{pmatrix}$$

- Equations of motion

$$\frac{d}{dt}X = \frac{d}{dt}\begin{pmatrix} \bar{x} \\ q \\ \bar{v} \\ L \end{pmatrix} = \begin{pmatrix} \bar{v} \\ \frac{1}{2}\omega^* q \\ F/M \\ \tau \end{pmatrix}$$

- Once again, preferable to rotate q by $\Delta t \omega$
- Renormalize q using a square root

## Question #1

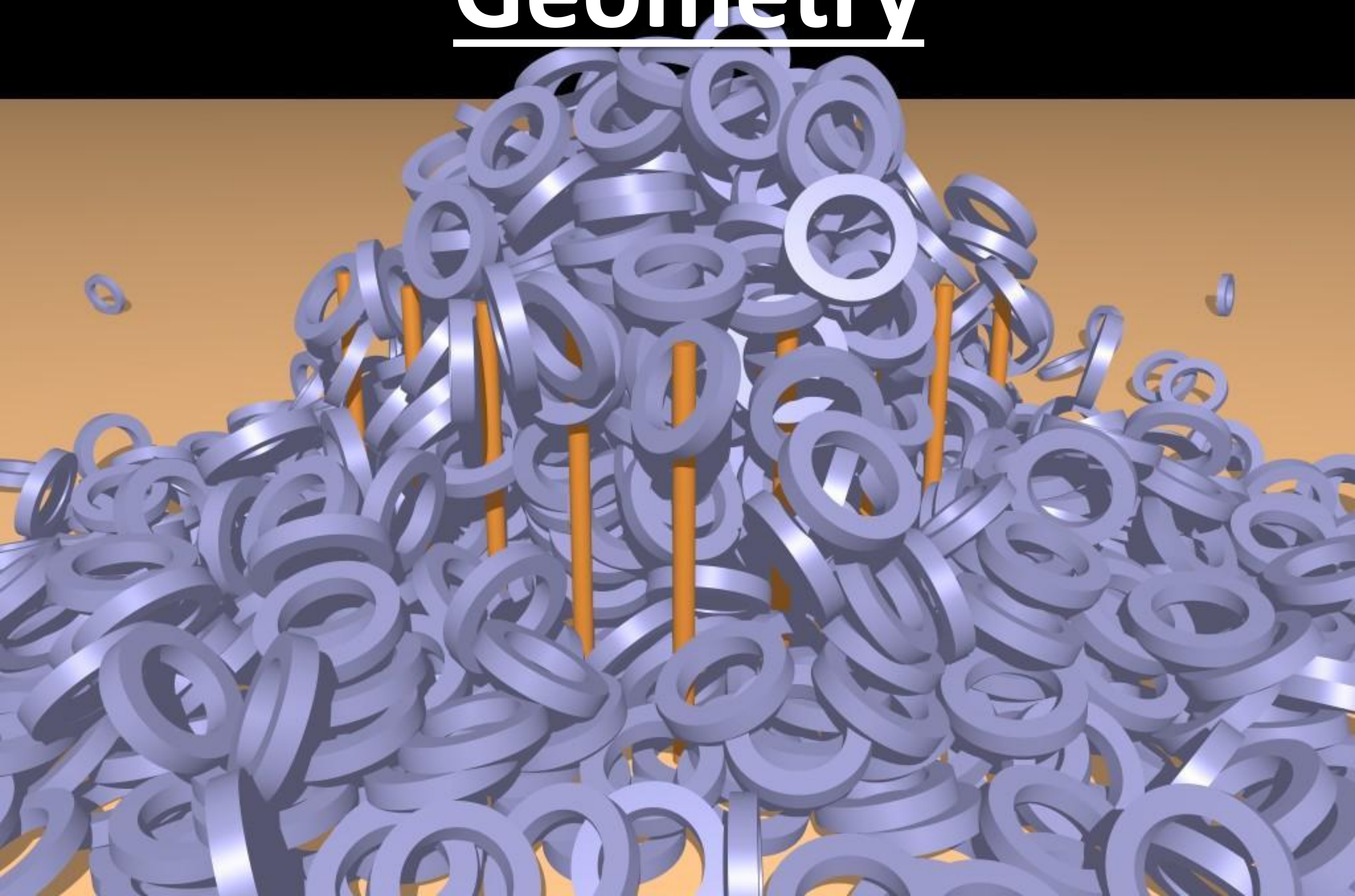
**LONG FORM:**

- Summarize rigid body simulation.
- Identify 10 rigid bodies in a typical room.

**SHORT FORM:**

- Identify 10 rigid bodies in this room.

# Geometry

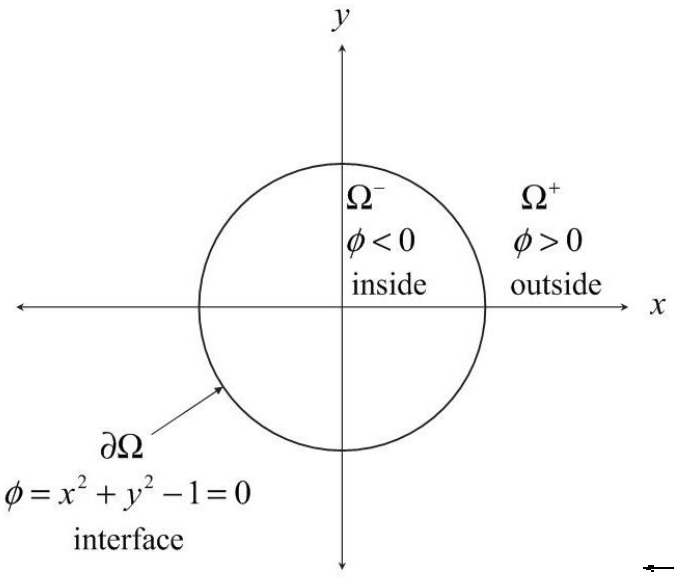


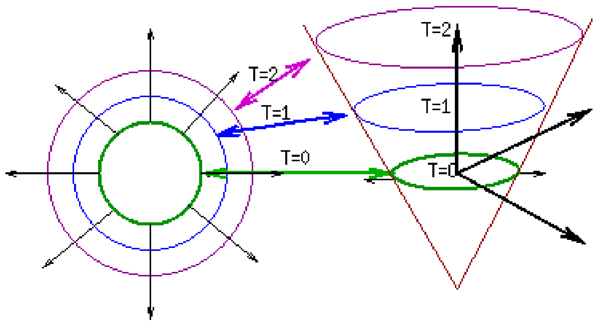
# Rigid Body Modeling

- Store an object space <u>triangulated surface</u> to represent the <u>surface</u> of the rigid body
- Store an object space <u>implicit surface</u> to represent the <u>interior volume</u> of the rigid body
- <u>Collision detection</u> between two rigid bodies can then be carried out by checking the surface of one body against the interior volume of another
- Implicit surfaces can be used to model the interior volume of kinematic and static objects as well
- Implicit surface representations of interior volumes can also be used for collisions with particles and particle systems

# Recall: Implicit Surfaces

- Implicit surfaces represent a surface with a function $\phi(x)$ defined over the whole 3D space
- The inside region $\Omega^{-}$, the outside region is $\Omega^{+}$, and the surface $\partial\Omega$ are all defined by the function $\phi(x)$
  - $\phi(x) < 0$ inside
  - $\phi(x) > 0$ outside
  - $\phi(x) = 0$ surface

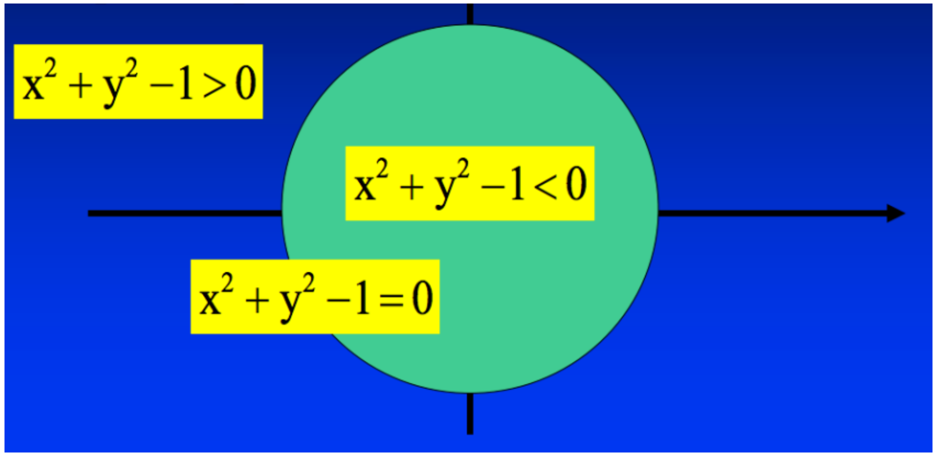




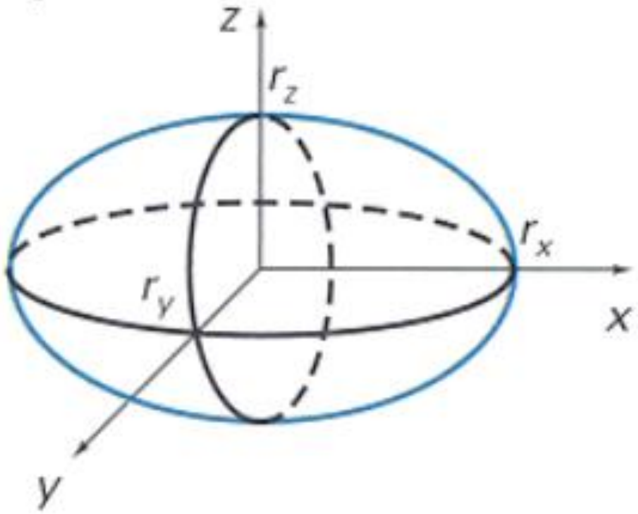
- Easy to check if a point is inside an object
- Efficient to make topology changes to an object
- Efficient boolean operations: Union, Difference, Intersection

# Analytic Implicit Surfaces

- For simple functions, write down the function and analytically evaluate $\phi(x)$ to see if $\phi(x) < 0$ and thus $x$ is inside the object
- 2D circle
- 3D ellipsoid



$$\left(\frac{x}{r_x}\right)^2 + \left(\frac{y}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 - 1 = 0$$



# Discrete Implicit Surfaces

- Lay down a grid that spans the space you are trying to represent, e.g. a padded bounding box
- Store values of the function at grid points
- Then for arbitrary locations in space, interpolate from the nearby values on the grid to see if $\phi(x) < 0$
  - use trilinear interpolation, like 3D textures
- Signed Distance Functions are implicit surfaces where the magnitude of the function gives the distance to the closet point on the surface

# Constructing Signed Distance Functions

- Start with a triangulated surface
- Place a grid inside a slightly padded bounding box of the object
  - This grid will contain point samples for the signed distance function
  - The resolution of the grid is based on heuristics
- Place a sphere at every grid point and find all intersecting triangles
  - the radius of the sphere only needs to be a few grid cells wide, because we only care about grid points near the triangulated surface
  - If the sphere does not intersect any triangles, then the grid point is not near the triangulated surface
  - (An acceleration structure is useful, e.g. bounding box hierarchy)
- For each nearby triangle, find the closest point on that triangle
  - see this document for details
- Take the minimum of all such distances as the magnitude of $\phi$
- Could initialize all grid points this way, but it is expensive for points farther from the surface where one has to check many more (potentially all) triangles

# Fast Marching Method

- Similar to Dijkstra's algorithm
- Walk outwards from the previously initialized points to fill the rest of the domain

**Initialization**

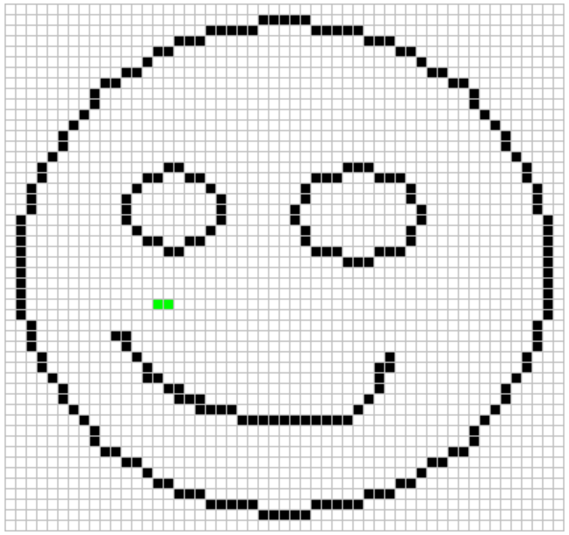
- Mark all the previously computed points nearby the triangulated surface as Black
- Mark the rest of the points White

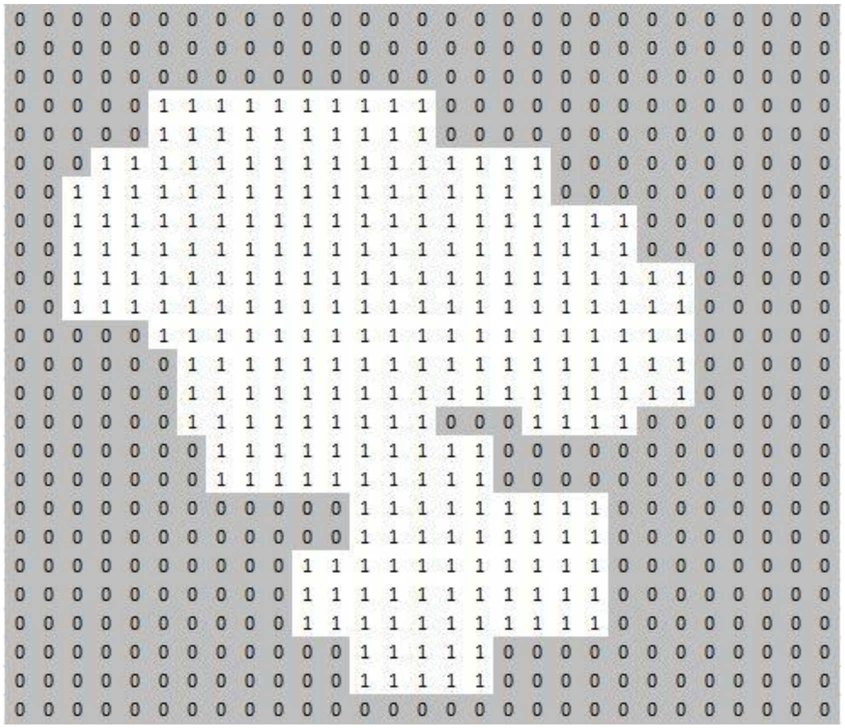
**Iteration**

- White points adjacent to Black points are re-labeled as Red
- Estimate the distance value for all Red points using only their Black neighbors
  - by solving a quadratic equation for distance
- The Red point with the smallest distance value is found and labeled Black
  - a heap data structure is ideal, and then the fast marching method runs in $\mathcal{O}(NlogN)$ where $N$ is the number of grid points
- Labeling this point Black turns some White points into Red
  - and also changes the value of some of the previously computed Red points, since there is a new Black point to use in their distance computation

# Sign of $\phi$

- Perform a flood fill on the grid
  - Start from a random grid point
  - Put it on a stack; mark it as 0
  - Pop it off; put its connected non-occluded neighbors on the stack; mark them as 0; repeat
  - When there are no more cells on the stack, find a random uncolored cell, mark it as 1, and repeat
- The region (0 or 1) that touches the grid boundary is marked as outside
- The other region is marked as inside
- Could have more than two regions in some cases





| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

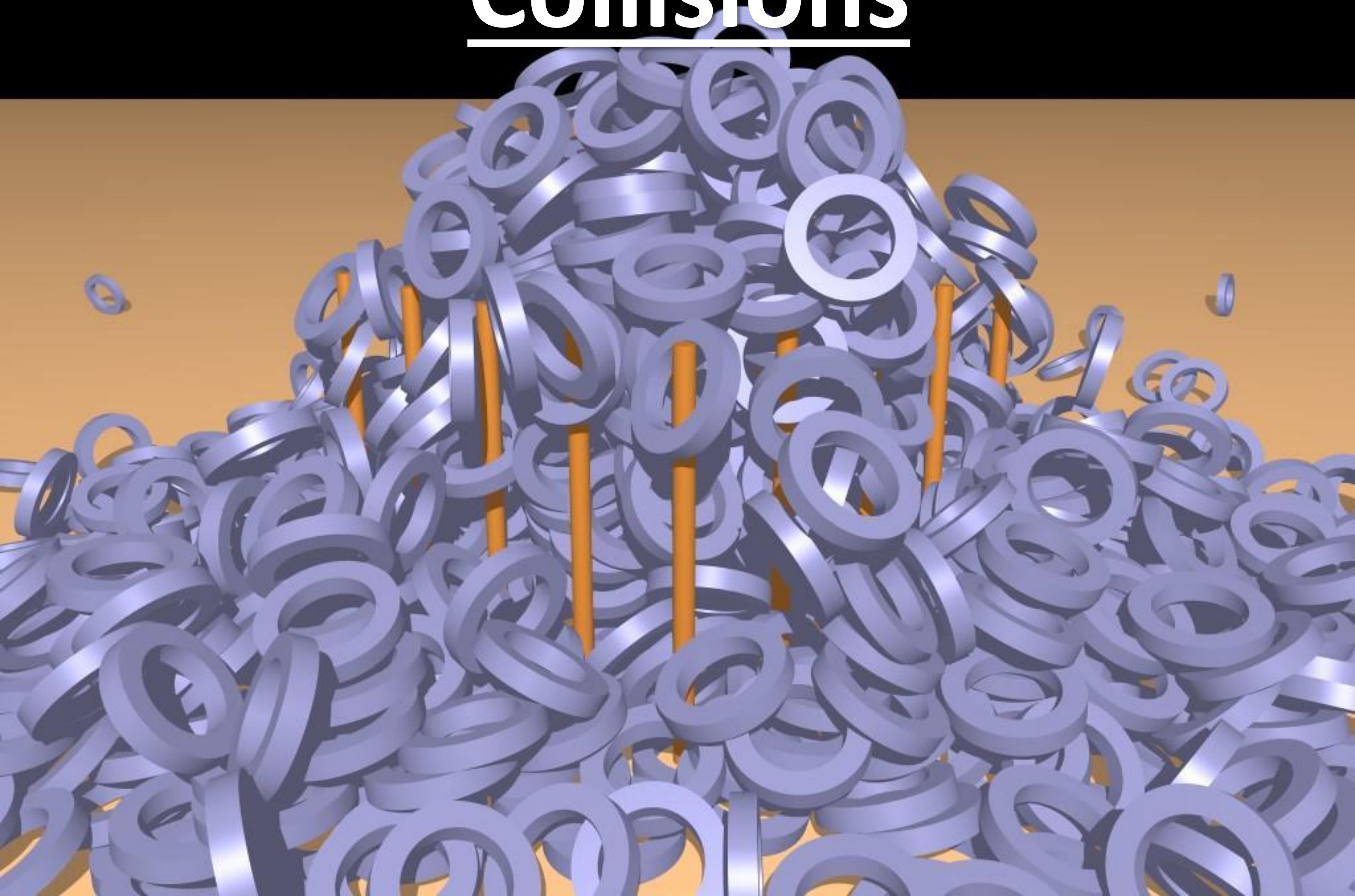
## Question #2

**LONG FORM:**

- Summarize rigid body geometric modeling.
- Answer short form question below

**SHORT FORM:**

- Give an example of a rigid body with an <u>interesting shape</u> that could be used in a game. How would it be used?



## Collisions

# Collision Detection

- Test all the triangulated surface points of one body against the implicit surface volume of the other (and vice versa)
  - A world space point is put into object space to check against an implicit surface using

$$x^o = R^{-1}(x^w - \bar{x})$$

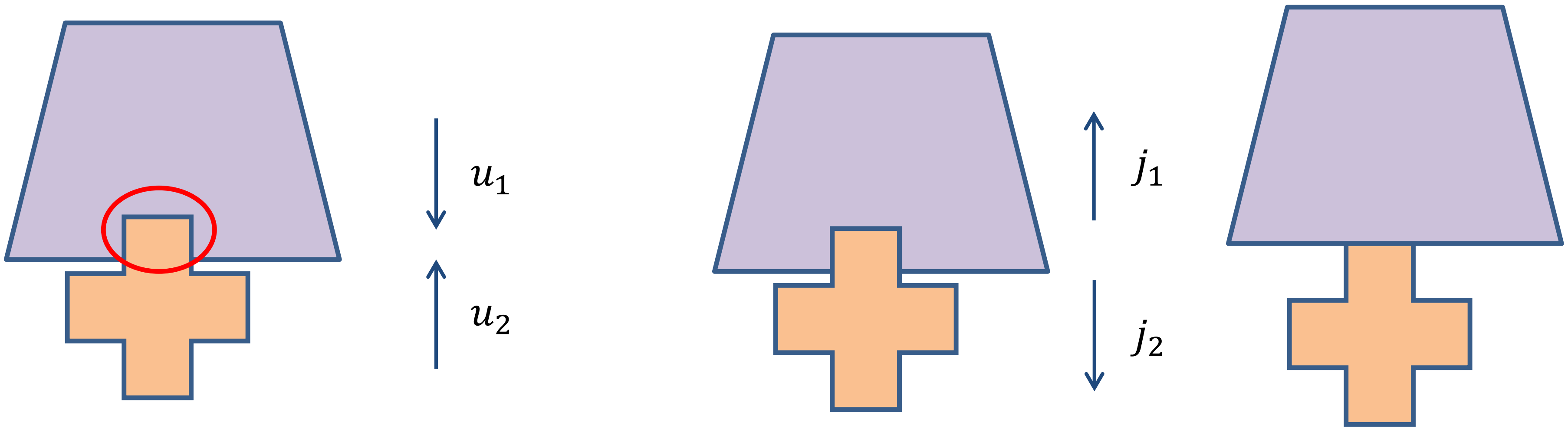
- Partial derivatives are used to compute the normal

$$n(x_0, y_0, z_0) = \left(\frac{d\phi}{dx}, \frac{d\phi}{dy}, \frac{d\phi}{dz}\right)\bigg|_{x_0, y_0, z_0}$$

$$Normal(x_0, y_0, z_0) = \frac{n(x_0, y_0, z_0)}{|n(x_0, y_0, z_0)|}$$

- Note: A particle can be moved to the surface of the implicit surface by tracing a ray in the normal direction and looking for the intersection with the $\phi = 0$ isocontour (see CS148)

# Rigid Body Collisions



Collision detection

Compute the initial relative velocity

The final relative velocity is calculated using the coefficient of restitution.

Calculate and apply the collision impulse

After evolving the bodies in time, they eventually separate

# Collision Response

Equations for applying an impulse to one body with collision location $r_p$ with respect to its center of mass:

- $M\bar{v}^{new} = M\bar{v} + j$
- $I\omega^{new} = I\omega + r_p^* j$ (note $I$ doesn't change)
- And then, in terms of the pointwise velocity…
- $u_p^{new} = \bar{v}^{new} + \omega^{new*} r_p = \bar{v}^{new} + r_p^{*T} \omega^{new}$
- $u_p^{new} = \bar{v} + \frac{j}{M} + r_p^{*T} \left(\omega + I^{-1} r_p^* j\right)$
- $u_p^{new} = u_p + \left(\frac{1}{M} I_{3x3} + r_p^{*T} I^{-1} r_p^*\right) j = u_p + Kj$
- Infinite mass kinematic/static objects (e.g. ground plane) are treated by setting the impulse factor $K = 0$

# Collision Response

- Equal and opposite impulse applied to each body:
  $u_1^{new} = u_1 + K_1 j$ and $u_2^{new} = u_2 - K_2 j$
- Calculate the relative velocity $u_{rel} = u_1 - u_2$ at the point of collision
  - Relative normal velocity is $u_{rel,N} = u_{rel} \cdot N$
  - Only collide when $u_{rel,N} < 0$, i.e. bodies not already separating
- Define a total impulse factor $K_T = K_1 + K_2$, then
$$u_{rel}^{new} = u_{rel} + K_T j$$
$$u_{rel,N}^{new} = u_{rel,N} + N^T K_T j$$
- Since the collision impulse should be in the normal direction, we can write $j = N j_n$, hence
$$u_{rel,N}^{new} = u_{rel,N} + N^T K_T N j_n$$
- Given $u_{rel,N}^{new} = -c_R u_{rel,N}$, we solve for $j_n$ and apply $j = N j_n$

# Friction

- Relative tangential velocity is $u_{rel,T} = u_{rel} - u_{rel,N}N$
- First assume static friction, i.e. $u_{rel,T}^{new} = 0$, so that
$$u_{rel}^{new} = -c_R u_{rel,N}N$$
- Solve for a full 3D impulse $j$ using $u_{rel}^{new} = u_{rel} + K_T j$, by inverting the 3x3 matrix $K_T$
- If this impulse is in the friction cone, i.e. if $|j - (j \cdot N)N| \leq \mu_s(j \cdot N)$, then the assumption of sticking due to static friction was correct
- Otherwise we start over using kinetic friction instead ($\mu_k \leq \mu_s$)
  - With tangential direction $T = \frac{u_{rel,T}}{|u_{rel,T}|}$, the kinetic friction impulse is $j = j_n N - \mu_k j_n T$
  - And we can solve $-c_R u_{rel,N} = u_{rel,N} + N^T K_T(N - \mu_k T)j_n$ to find $j_n$ before applying $j = (N - \mu_k T)j_n$

## Question #3

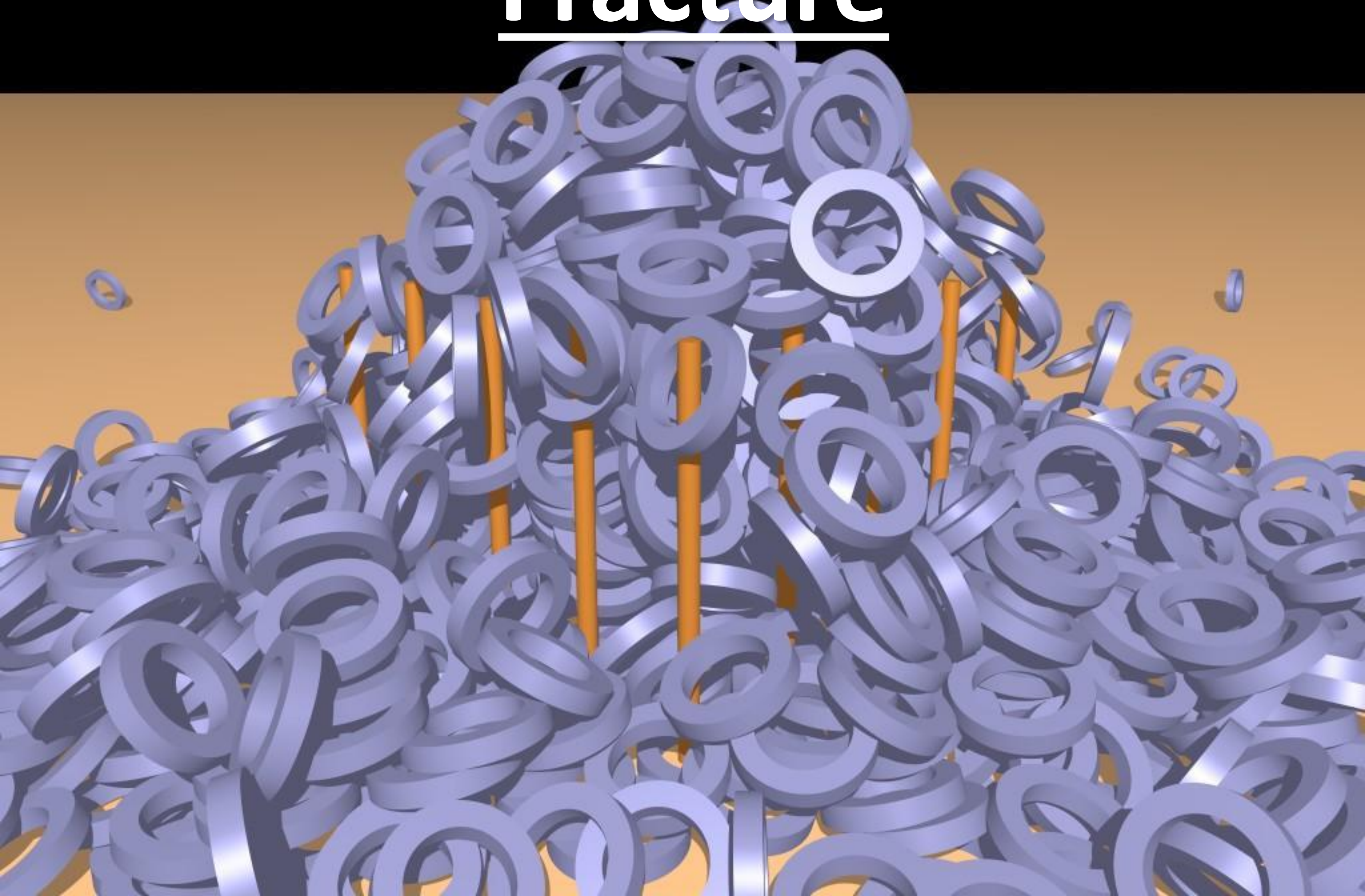
**LONG FORM:**

- Summarize rigid body collision handling.
- Answer short form question below.

**SHORT FORM:**

- Give an example of using collisions between rigid bodies for a game.

# Fracture



# Fracture

# Fracture

- Suppose a rigid body fractures into n pieces with masses $m_1, , m_2 \ \dots m_n$, velocities $v_1, v_2 \dots . v_n$, inertia tensors $I_1, I_2, \dots I_n$ and angular velocities $\omega_1, \omega_2, \dots \omega_n$
- The mass and inertia tensor of each new piece can be computed based on the geometry
- What can we say about the fractured pieces?
  - M$v = \sum m_i v_i$
  - $I\omega = \sum(r_i^* m_i v_i + I_i \omega_i)$
- To ensure this:
  - Assign each rigid body the velocity its newly created center of mass had before fracturing i.e. $v_i = v \ + \omega \times r_i$
    - where $r_i$ points from the center of mass of the original rigid body to the center of mass of the i-th child
  - Angular momentum is then conserved by setting $\omega_i = \omega$