COMPSCI 715 Part 2

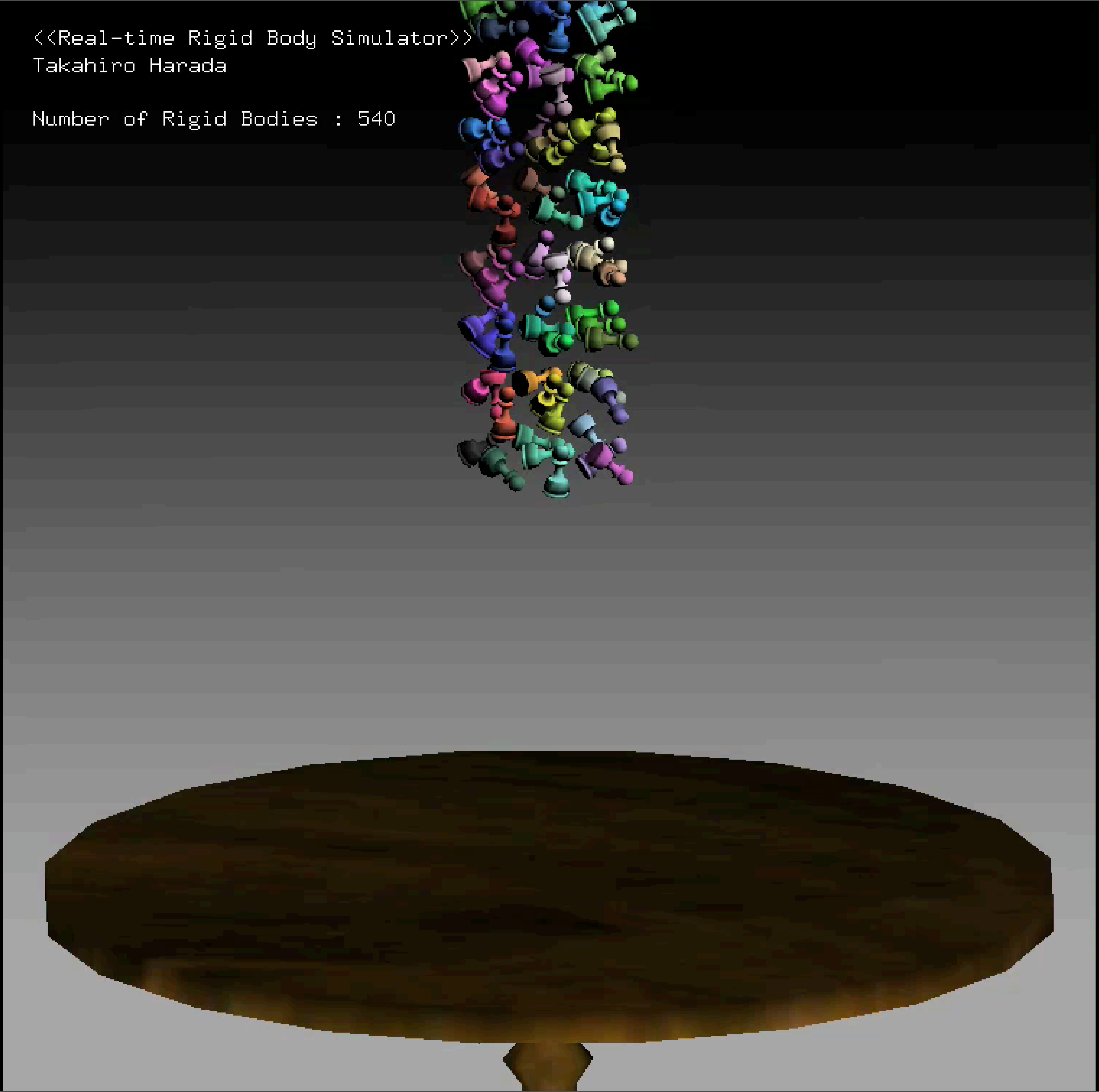
Lecture 6 - Rigid Body Mechanics

Rigid Body

- In physics, a rigid body is an idealization of a solid body of finite dimension in which deformation is neglected. In other words, the distance between any two given points of a rigid body remains constant regardless of external forces exerted on it.

Rigid Body Dynamics

- Classical Mechanics / robotics
- Game physics == rigid body mechanics
  - Collision Detection and response
  - Solving system constraints
  - Arbitrary shapes, interactions and dependencies



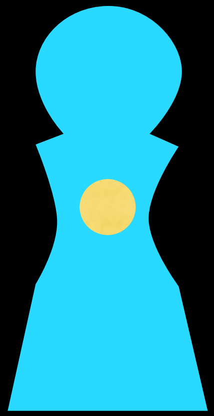
16,384 pieces
44 fps

Rigid Body Position

- Expressed as combination of translation and rotation from fixed reference
- Thus position has both linear and ‘orientation’ component

Frame of Reference

- We choose the center of mass:
  - $Mr_c = \sum m_i r_i$
  - linear momentum is independent of rotational momentum
  - angular momentum is the same regardless of translation and is always $\omega \times I$
  - simplifies possible motion (no forces) to constant translation and constant velocity

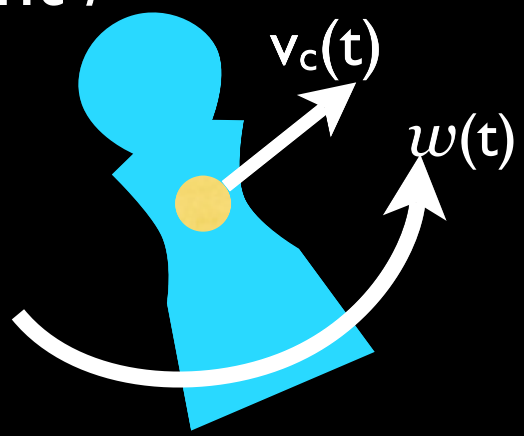


Position (of a point)

- $r(t, r_0) = r_c(t) + \Omega(t) r_0$
  - r_0 is position w.r.t. reference
  - $r(t, r_0)$ is position at time t
  - r_c is the reference position
  - $\Omega(t)$ is the orientation matrix

Velocity (of a point)

- $v(t,r_0) = v_c(t) + w(t) \times \Omega(t) r_0$
  - $v(t,r_0)$ is the total velocity of the point / particle
  - $v_c(t)$ is translational velocity
  - $w(t)$ is the angular velocity
- Remember: angular velocity is $\partial\Omega/\partial t$



Linear Momentum

- Momentum of the body is the sum of the momentum on each point
  - $\mathbf{p}_T = \sum m_i v_i$ OR
  - $\mathbf{p}_T = M v_c$

Angular Momentum

- Represented as a vector
  - direction is axis of spin, magnitude is speed
  - $\mathbf{L} = \sum(r_i \times m_i\mathbf{p}_i)$
- Torque
  - $T=(r_i-r_c)XF$
  - $T=\partial\mathbf{L}/\partial t = I\dot{\omega}$
  - $T=\sum((r_i-r_c) \times \sum f_i)$

Inertia

- A tensor describing how hard it is to change the rotation
- Written: *I*
- $I_{body} = \sum m_i((r_{0i}{}^T r_{0i}) 1 - r_{0i} r_{0i}{}^T)$
- $I(t) = \Omega(r) I_{body} \Omega(t)^T$

Acceleration

- Linear:
  - $a=\sum F_i/M$
- Angular:
  - $\acute{\omega}=\Upsilon/I$

Algorithm

1. Determine center of mass
2. Set initial postion, orientation etc
3. Find sum of all forces / total mass
4. For each force find related effect on torque
5. Divide torque by interia
6. Use ODE Solver to update position, velocity, orientation, and angular velocity

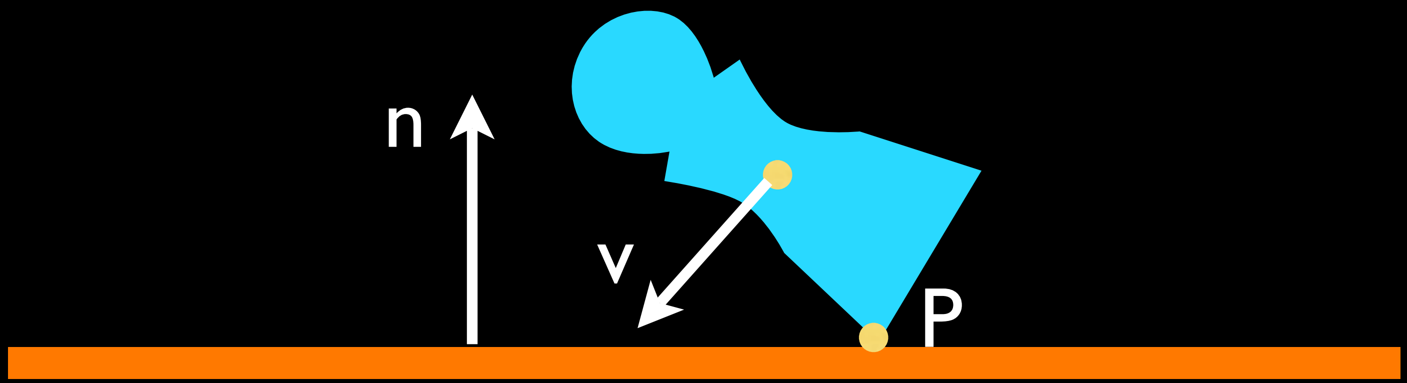
Example Forces

- Gravity (or any constant field)
  - Does not create torque
- Spring on a point
  - Calculate torque as described

Example Forces

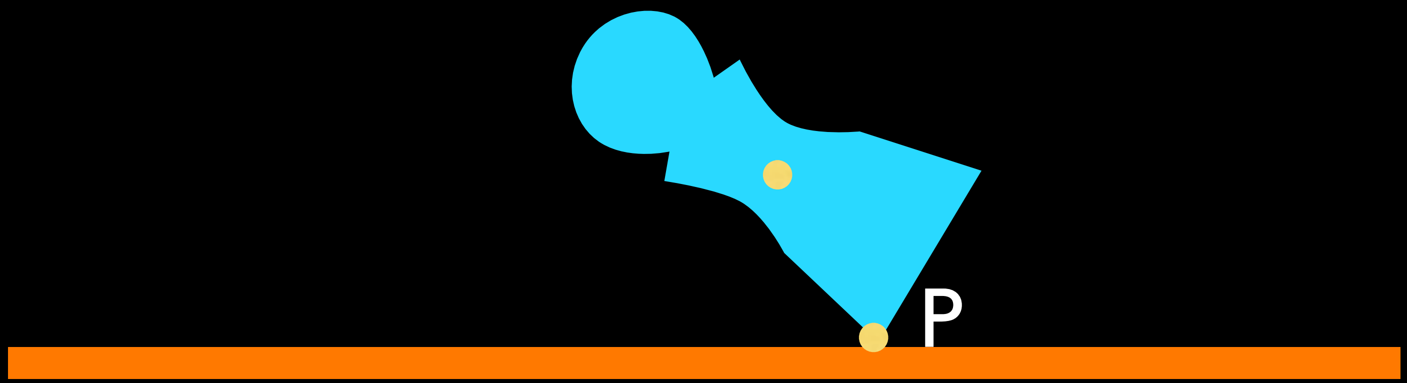
- Drag / Friction
  - Find which points of the surface would be affected and apply where relevant
- Attractor/Repulser
  - Either treat as a field or (if attenuated) act as force on sample points around COM

Collisions



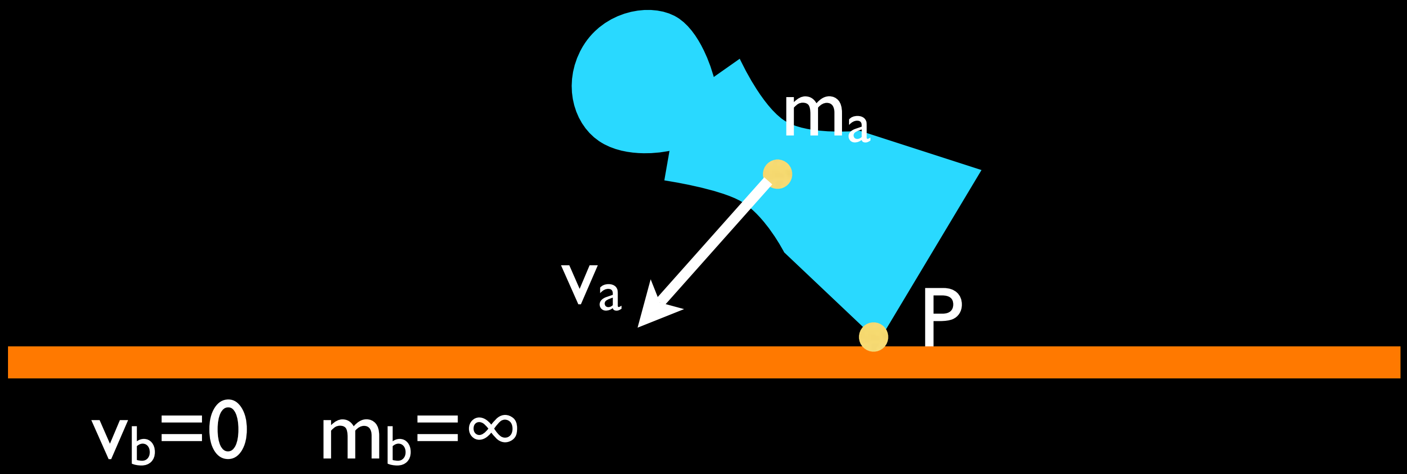
- A collision occurs when a point on one body touches a point on another body with a negative relative normal velocity
- i.e. $(v_a - v_b) \cdot n < 0$
    - (n must be chosen carefully)

Collision Detection



- Question: Is position inside or outside of an object?
  - Can be difficult for complex shapes
  - Need to bring system back to time of collision

Collision Resolution



- Respond to collision based on physics of the two objects
    - Spin and translate based on momentum of the two objects

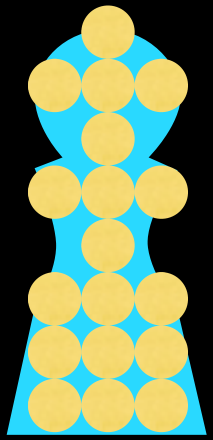
Simple Collision Resolution

- Can’t change velocities instantaneously so use ‘impulse’
- Use ‘Law of Restitution for Instantaneous Collisions with No Friction’
  - Only collision forces apply

Collision Resolution

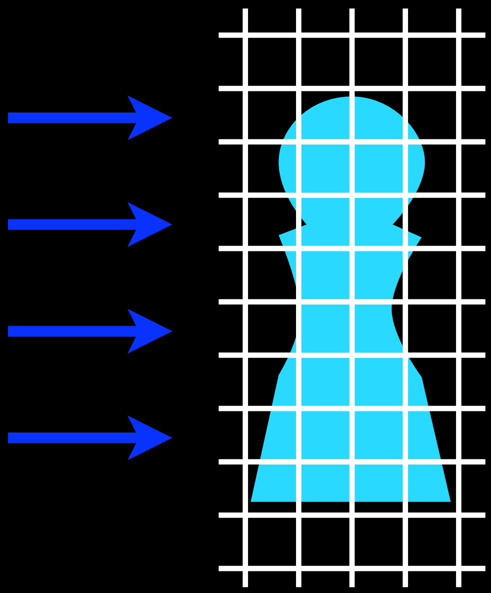
- Velocity:
  - $v_{a1} = v_{a0} + (\epsilon/M)n$
- Angular Velocity
  - $\omega_{a1} = \omega_{a0} + (\Omega \bullet \epsilon n)/I$
- Where ϵ is the coefficient of restitution

Rigid Body Implementation



- Need way of creating rigid body representation of arbitrary objects
- Can use particles
  - Simple way to calculate COM
  - Very easy to approximate forces at points and collision detection

Filling an object (Depth Peeling)



- Object projected onto axis, depth is first intersection point, second depth image is second intersection and so on
- Render to 3D voxels and determine if between an odd and even depth image
- Center of each voxel represents particle

Rigid Body Implementation

- 3 phases in each timestep
  - Integrate positions / velocities (GPU)
  - Detect collisions (CPU)
  - Resolve collisions (Depends)
  - Communicate between the two

Rigid Body Implementation

- Integration of positions / velocities
  - Each force is ‘applied’ to nearest particle and accumulated
  - Equations solved as described before

Rigid Body Implementation

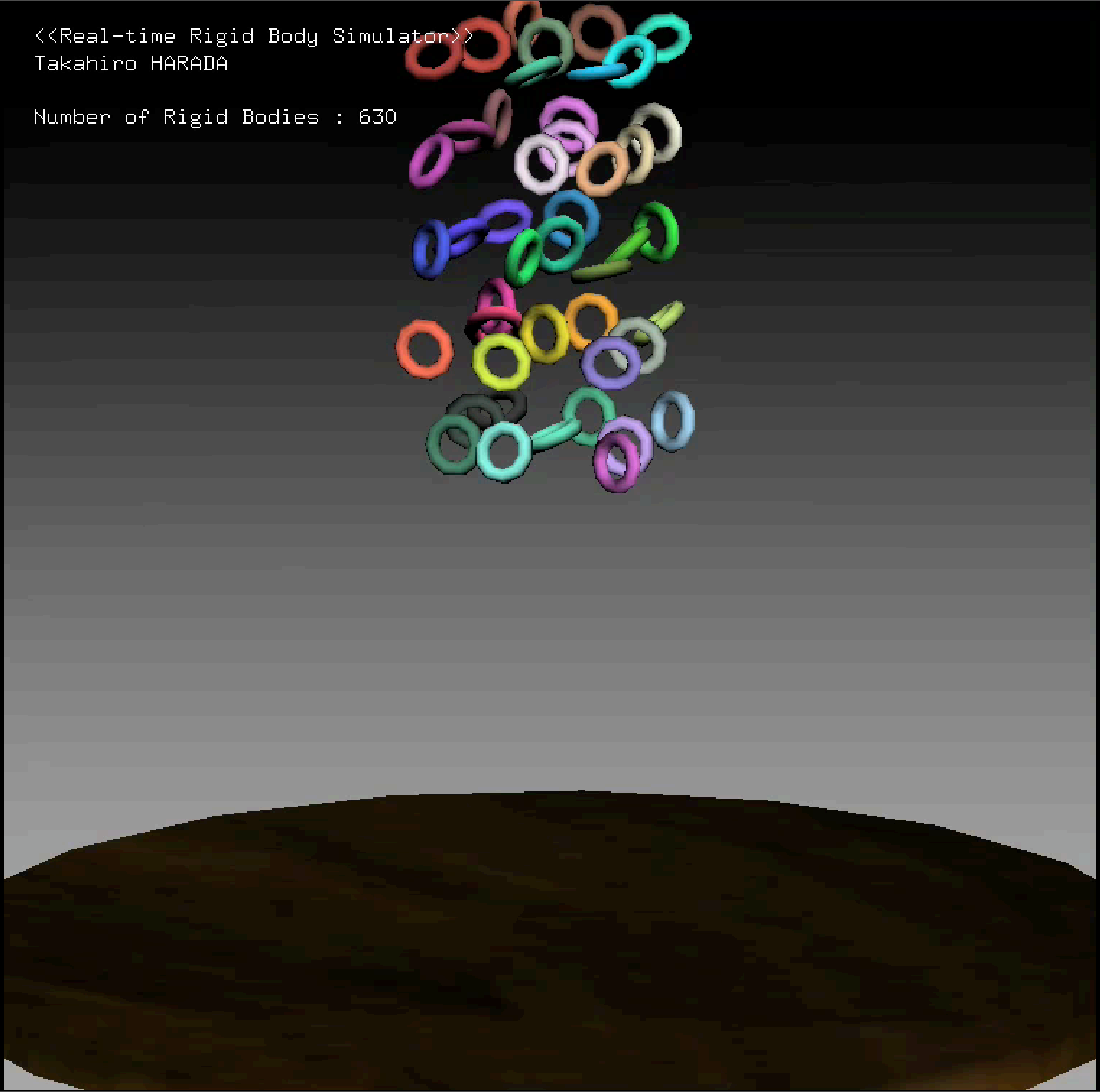
- Collision Detection
  - Calculate collisions between particles, not objects
  - Sphere<->Sphere is easy based on distance
  - Use space subdivision techniques to lower computational complexity
    - i.e. Uniform Grid

Rigid Body Implementation

- Collision Reaction
  - Discrete Element Method
    - Repulsive force modeled by spring and dampening force
    - $f_s = -(k(d-|r_{ij}|)r_{ij})/|r_{ij}|$
    - $f_d = \eta v_{ij}$

Rigid Body Implementation

- This technique:
  - Allows for multiple resolutions (large particles means faster rendering)



10922 Tori
68fps

Sources

- Baraff, D (2001) Physically Based Modeling: Rigid Body Simulation. SIGGRAPH Course notes
- Harada, T (2007). Real-Time Rigid Body Simulation on GPUs. GPU Gems 3
- Hecker, C (1995). Physics, The Next Frontier. Game Developer Magazine