Building a 2D Physics Engine for MASON

Christian Thompson

Goals

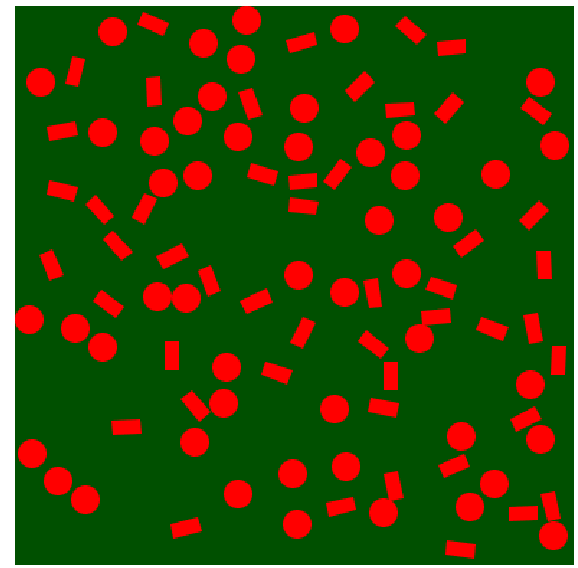
- Provide generic framework for physically realistic MASON simulations
- Small
- Easy to use
- Fast

Features

- Numerical Integrator to move objects in a physically realistic way based on the equations of motion
- Collision detection and response
- Joints (constraints)
- Constrained collision response

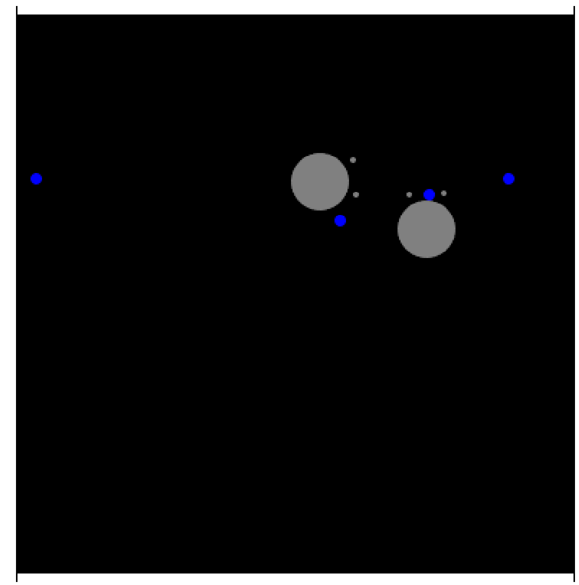
Collisions Simulation

- Motion
- Collision Detection and Response



Robots Simulation

- Constraints
  - Effector
  - Robot carries can with pin joint
- Collision detection and response
- Forces and torques
  - Friction (different for cans and robots)
  - PD Controller



Physically Realistic Motion

- Equations
  - Kinematics
  - Kinetics
- Numerical Integration
- State representation

Kinematics

- Study of motion of bodies in the absence of forces and torques
- Calculating new poses

$$x(t+dt) = x(t) + \dot{x}(t)dt$$

$$\theta(t+dt) = \theta(t) + \dot{\theta}(t)dt$$

- Calculating new velocities

$$\dot{x}(t+dt) = \dot{x}(t) + \ddot{x}(t)dt$$

$$\dot{\theta}(t+dt) = \dot{\theta}(t) + \ddot{\theta}(t)dt$$

Kinetics

- Study of motion of bodies due to forces and torques
- Equations of Motion used to calculate linear and angular accelerations

$$f = m\ddot{x}$$

$$\tau = I\ddot{\theta}$$

Numerical Integration

- Kinematic equations calculate new positions and velocities after infinitesimally small time period dt.
- Simulator must integrate the equations over each time step
- The Numerical Integrator approximates answers to the kinematic equations

Numerical Integration

- Kinematic Equations can be expanded using Taylor Series

$$\dot{x}(t+\Delta t) = \dot{x}(t) + \ddot{x}(t)\Delta t + \frac{\dddot{x}(t)\Delta t^2}{2!} + \dots$$

$$x(t+\Delta t) = x(t) + \dot{x}(t)\Delta t + \frac{\ddot{x}(t)\Delta t^2}{2!} + \dots$$

Numerical Integration

- The Euler integrator takes the first two terms of the Taylor Series expansion

$$\dot{x}(t+\Delta t) = \dot{x}(t) + \ddot{x}(t)\Delta t$$

$$x(t+\Delta t) = x(t) + \dot{x}(t)\Delta t$$

- This approximation is often too rough

Numerical Integration

- The Runge-Kutta integrator approximates the first four terms of the Taylor Series expansion for much greater accuracy

$$k1 = \Delta t \dot{x}(x(t))$$

$$k2 = \Delta t \dot{x}\left(x(t) + \frac{k1}{2} \right)$$

$$k3 = \Delta t \dot{x}\left(x(t) + \frac{k2}{2} \right)$$

$$k4 = \Delta t \dot{x}(x(t) + k3)$$

$$x(t + \Delta t) = x(t) + \frac{(k1 + 2k2 + 2k3 + k4)}{6}$$

State Representation

- Global state vectors and matrices for state of all objects
  - Pose vector
  - Velocity vector
  - External forces vector
  - Mass inverse matrix
- Objects’ data stored in blocks according to their indices

State Representation

- Pose Vector

$$\begin{bmatrix} object \quad 1\begin{cases} x_1 \\ y_1 \\ \theta_1 \end{cases} \\ object \quad 2\begin{cases} x_2 \\ y_2 \\ \theta_2 \end{cases} \end{bmatrix}$$

- Mass Inverse Matrix

$$\begin{matrix} \frac{1}{m_1} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{m_1} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{I_1} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{m_2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{m_2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{I_2} \end{matrix}$$

State Representation

- Allows numerical integrator to solve for positions and velocities of all objects in one step
- For example, the Euler equations are:

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

$$\dot{\mathbf{x}} = \dot{\mathbf{x}} + \mathbf{WF}$$

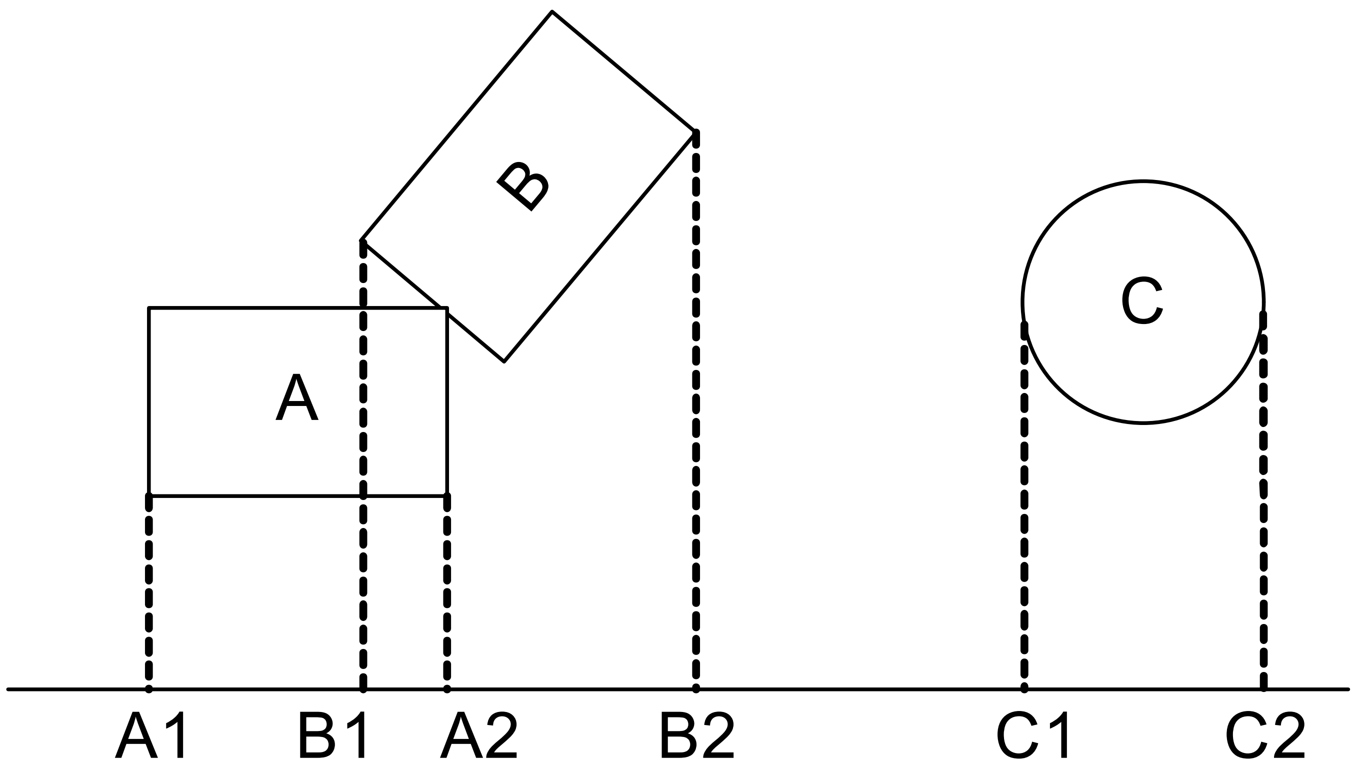
Collision Detection

- Naïve method
  - Compare all pairs of objects
  - n^2 – to slow for big simulations
- Two phase
  - Rough but fast “broad phase” returns objects that could be colliding
  - Exact “narrow phase”

Broad Phase Collision Detection

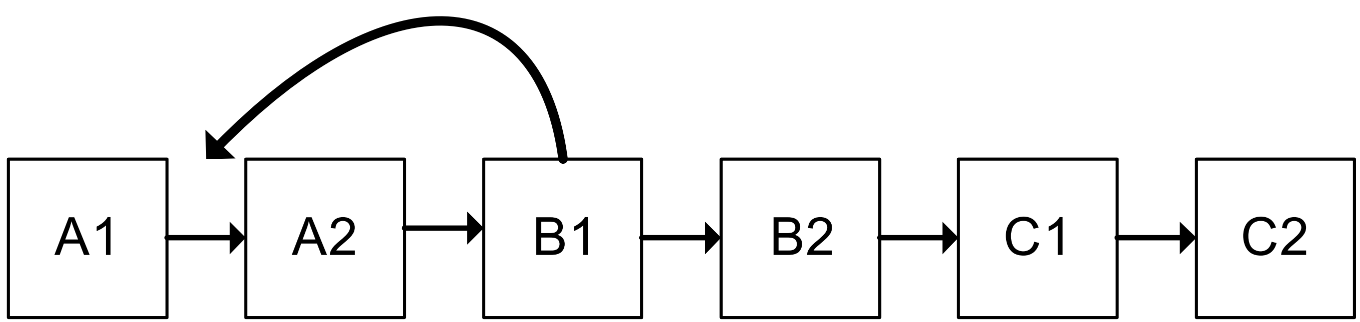
- I chose to use the dimension reduction strategy used in the I-Collide collision detection system
- Consider each dimension (x and y) one at a time
- Determine if objects’ endpoints are overlapping
- If overlapping in both dimensions, objects might be colliding

Dimension Reduction



Dimension Reduction Implementation

- Keep lists of endpoints in each dimension
- Sort the lists in each time step using the insertion sort
- As endpoints are moved into place, track overlaps



Dimension Reduction Efficiency

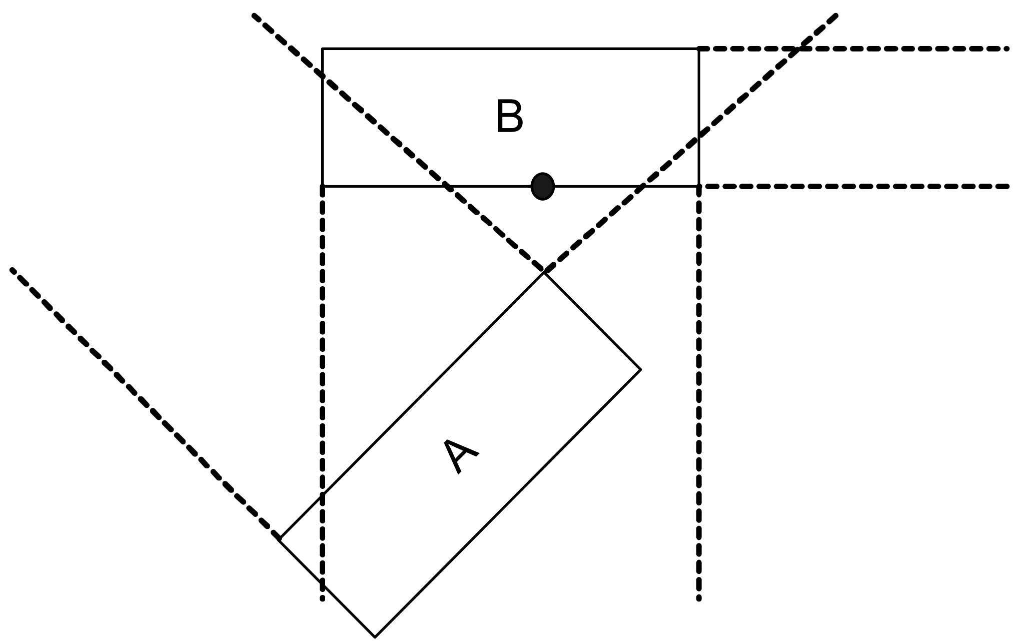
- Objects move relatively little over 1 time step
- Endpoint lists remain almost sorted
- Insertion sort completes in an expected $O(n)$ time

Narrow Phase Collision Detection

- Algorithm based on techniques from the Lin-Canny algorithm for closest feature (edge or vertex) detection
- Use Voronoi regions to determine if a pair of features is closest

Closest Feature Pair

- Only pair for which closest points on the features fall in other feature's Voronoi region



Algorithm

- Loop through pairs of features and test if closest
  - If closest, test if length between closest point below threshold
  - If not below threshold, record pair to use as a starting point for the next time step
- If no closest feature pair, objects have penetrated
  - Search back in time over last time step to find penetration point

Collision Response

- Prevent interpenetration of colliding objects with instantaneous velocity change
- Apply “impulses” - infinite force over infinitesimally short period of time

Calculating Impulses (point masses)

- Newton's Law of Restitution (1 equation)

$$\mathbf{v}^{rel}_{post} \cdot \mathbf{n} = -e\mathbf{v}^{rel}_{pre} \cdot \mathbf{n}$$

- Impulses change objects' momentums (4 equations)

$$m\dot{\mathbf{x}}_{post} = m\dot{\mathbf{x}}_{pre} + \mathbf{R}$$

- Impulses must be along collision normal (1 equation)

$$\mathbf{R}_{\perp} \cdot \mathbf{n} = 0$$

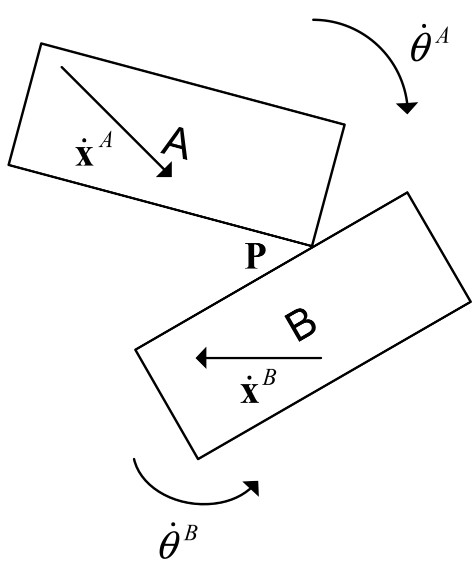
Calculating Impulses (rigid bodies)

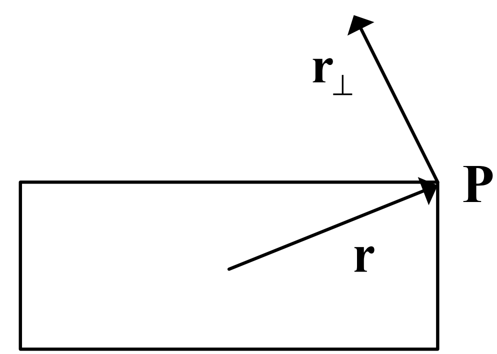
- Account for rotation of object
- Affects relative velocities in Newton’s Law of Restitution
- Requires additional equations for change in angular momentums

Velocity Point on Rigid Body

- Includes linear and angular components

$$\dot{\mathbf{x}}^{\mathbf{P}} = \dot{\mathbf{x}} + \dot{\theta}\mathbf{r}_{\perp}$$





Colliding Rigid Body Equations

- Updated Newton's Law of Restitution

$$\left((\dot{\mathbf{x}}^{A}_{post} + \dot{\theta}^{A}_{post}\mathbf{r}^{A}_{\perp}) - (\dot{\mathbf{x}}^{B}_{post} + \dot{\theta}^{B}_{post}\mathbf{r}^{B}_{\perp})\right)\cdot \mathbf{n} = -e\left((\dot{\mathbf{x}}^{A}_{pre} + \dot{\theta}^{A}_{pre}\mathbf{r}^{A}_{\perp}) - (\dot{\mathbf{x}}^{B}_{pre} + \dot{\theta}^{B}_{pre}\mathbf{r}^{B}_{\perp})\right)\cdot \mathbf{n}$$

- Change in angular momentum

$$I\dot{\theta}_{post} = I\dot{\theta}_{pre} + \mathbf{r}_{\perp} \cdot \mathbf{R}$$

Collision Response Equations

- Eight equations and eight unknowns

$$m^A \dot{x}^A_{post} = m^A \dot{x}^A_{pre} + \mathbf{R}_x$$

$$m^A \dot{y}^A_{post} = m^A \dot{y}^A_{pre} + \mathbf{R}_y$$

$$I\dot{\theta}^A_{post} = I\dot{\theta}^A_{pre} + \mathbf{r}_\perp \cdot \mathbf{R}$$

$$m^B \dot{x}^B_{post} = m^B \dot{x}^B_{pre} - \mathbf{R}_x$$

$$m^B \dot{y}^B_{post} = m^B \dot{y}^B_{pre} - \mathbf{R}_y$$

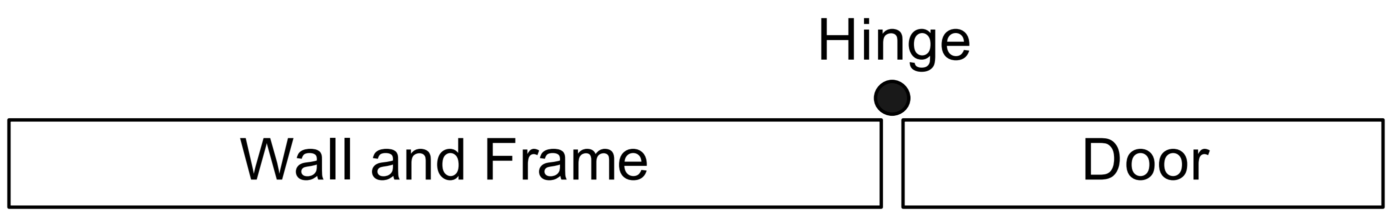
$$I\dot{\theta}^B_{post} = I\dot{\theta}^B_{pre} - \mathbf{r}_\perp \cdot \mathbf{R}$$

$$\left((\dot{\mathbf{x}}^A_{post} + \dot{\theta}^A_{post}\mathbf{r}^A_\perp) - (\dot{\mathbf{x}}^B_{post} + \dot{\theta}^B_{post}\mathbf{r}^B_\perp)\right)\cdot \mathbf{n} = -e\left((\dot{\mathbf{x}}^A_{pre} + \dot{\theta}^A_{pre}\mathbf{r}^A_\perp) - (\dot{\mathbf{x}}^B_{pre} + \dot{\theta}^B_{pre}\mathbf{r}^B_\perp)\right)\cdot$$

$$\mathbf{R}_\perp \cdot \mathbf{n} = 0$$

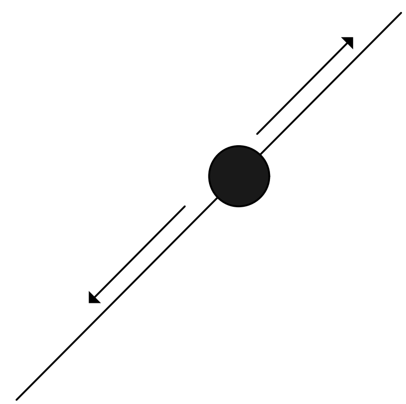
Constrained Dynamics

- Physical world joints restrict the motions of the connected objects



- Constraints simulate joints by restricting motions of simulated objects

Example: Bead on a Wire



- Bead sliding on a wire that lies on the line

$$y = x$$

- To simulate, physics engine applies “constraint forces” to the simulated bead so its position satisfies the equation:

$$C(\mathbf{x}) = y - x = 0$$

Example: Bead on a Wire

- To solve for constraint forces, need equation containing accelerations – take two derivatives of C($\mathbf{x}$)

$$\dot{C}(\mathbf{x}) = \dot{y} - \dot{x} = 0$$

$$\ddot{C}(\mathbf{x}) = \ddot{y} - \ddot{x} = 0$$

- Plug in Newton's Second Law (f = ma)

$$\ddot{C}(\mathbf{x}) = (f_y - f_x)/m = 0$$

Example: Bead on a Wire

- Separate total force into the sum of constraint force and external force and plug in

$$\ddot{C}(\mathbf{x}) = \left((f_y^{external} + f_y^{constraint}) - (f_x^{external} + f_x^{constraint}) \right) / m = 0$$

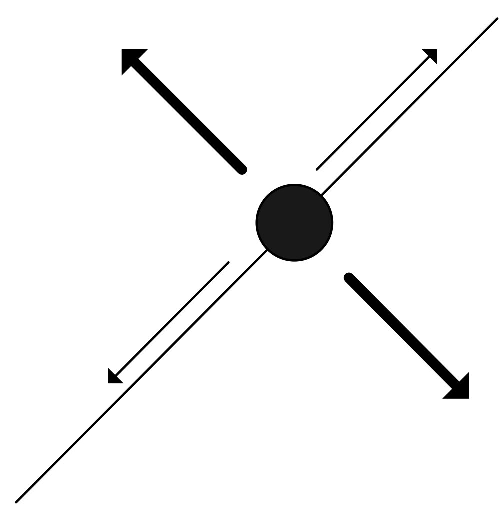
- Principal of virtual work gives second equation needed for the two unknowns

Example: Bead on a Wire

- Principal of Virtual Work: constraint forces must do no work in the system
- Work is amount of displacement caused by a force, so constraint forces can not cause displacements
- Constraint forces must be orthogonal to legal displacements

Example: Bead on a Wire

- Legal displacements are along
  $y = x$
  so constraint forces must be along
  $y = -x$



Example: Bead on a Wire

- Can also say constraint forces must be orthogonal to legal velocities

$$\mathbf{F}^{constraint} = \lambda \begin{bmatrix} -\dot{y} \\ \dot{x} \end{bmatrix}$$

- Plugging into legal acceleration equation and solving for λ gives

$$\lambda = (f_x^{external} - f_y^{external}) / (\dot{y} + \dot{x})$$

Example: Bead on a Wire

- After solving for λ, solve for the components of the constraint force and apply to the object
- Demo

General Constraint Equations

- Need vector and matrix version of the equations to allow any number and combination of constraints
- First step: define constraint vector (1 row for each constraint equation)

$$\mathbf{C} = 0$$

General Constraint Equations

- Take two derivatives:

$$\dot{\mathbf{C}} = \frac{\partial \mathbf{C}}{\partial \mathbf{x}} \dot{\mathbf{x}} = 0$$

$$\dot{\mathbf{C}} = \mathbf{J}\dot{\mathbf{x}} = 0$$

$$\ddot{\mathbf{C}} = \mathbf{J}\ddot{\mathbf{x}} + \dot{\mathbf{J}}\dot{\mathbf{x}} = 0$$

- Plug in equations of motion ($\ddot{\mathbf{x}} = \mathbf{W}(\mathbf{F} + \mathbf{F}^{constraint})$) and rearrange

$$\mathbf{JWF}^{constraint} = -\mathbf{JWF} - \dot{\mathbf{J}}\dot{\mathbf{x}}$$

Constraint Equations (Principal of Virtual Work)

- Need constraint forces to be orthogonal to all legal displacements
- $\frac{\partial \mathbf{C}}{\partial \mathbf{x}}$ is in the direction of illegal displacements (since C is not allowed to change)
- $\frac{\partial \mathbf{C}^T}{\partial \mathbf{x}}$ $(\mathbf{J}^T)$ contains the set of vectors orthogonal to the legal displacements
- Ensure constraint forces falls into the set spanned by illegal displacement vectors

$$\mathbf{F}^{constraint} = \mathbf{J}^T \boldsymbol{\lambda}$$

Constraint Equations

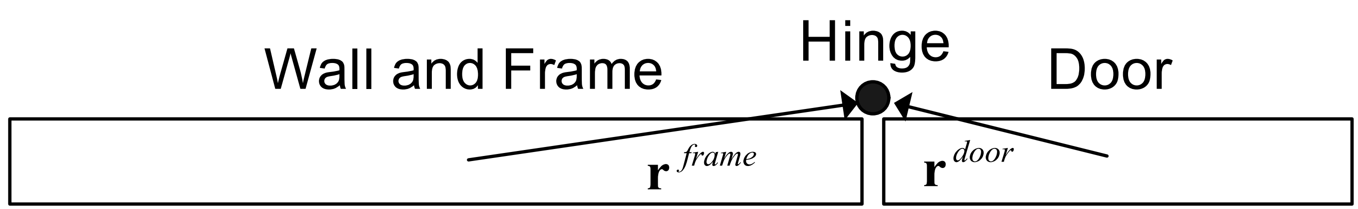
- Plugging virtual work equation in gives

$$\mathbf{J}\mathbf{W}\mathbf{J}^{T}\boldsymbol{\lambda} = -\mathbf{J}\mathbf{W}\mathbf{F} - \dot{\mathbf{J}}\dot{\mathbf{x}}$$

which can be solved for $\boldsymbol{\lambda}$ and the constraint force

Example: Pin Joint

- Unknowns in constraint equation are $\mathbf{J}$ and $\dot{\mathbf{J}}$
- Define C for door hinge example



- Constraint: frame's and door's local coordinate of hinge must convert to same global coordinate

$$\mathbf{R}_{\theta^{frame}}\mathbf{r}^{frame} + \mathbf{x}^{frame} - (\mathbf{R}_{\theta^{door}}\mathbf{r}^{door} + \mathbf{x}^{door}) = 0$$

Example: Pin Joint

- Multiplying by rotation matrices gives **C**:

$$\begin{bmatrix} \cos\theta^{frame}\mathbf{r}_x^{frame} - \sin\theta^{frame}\mathbf{r}_y^{frame} + \mathbf{x}^{frame} - \left(\cos\theta^{door}\mathbf{r}_x^{door} - \sin\theta^{door}\mathbf{r}_y^{door} + \mathbf{x}^{door}\right) \\ \sin\theta^{frame}\mathbf{r}_x^{frame} + \cos\theta^{frame}\mathbf{r}_y^{frame} + \mathbf{y}^{frame} - \left(\sin\theta^{door}\mathbf{r}_x^{door} + \cos\theta^{door}\mathbf{r}_y^{door} + \mathbf{y}^{door}\right) \end{bmatrix}$$

- Partial derivative of **C** with respect to **x** gives:

$$\mathbf{J} = \begin{bmatrix} 1 & 0 & -\mathbf{r}_x^{frame}\sin\theta^{frame} - \mathbf{r}_y^{frame}\cos\theta^{frame} & -1 & 0 & \mathbf{r}_x^{door}\sin\theta^{door} + \mathbf{r}_y^{door}\cos\theta^{door} \\ 0 & 1 & \mathbf{r}_x^{frame}\cos\theta^{frame} - \mathbf{r}_y^{frame}\sin\theta^{frame} & 0 & -1 & -\mathbf{r}_x^{door}\cos\theta^{door} + \mathbf{r}_y^{door}\sin\theta^{door} \end{bmatrix}$$

Example: Pin Joint

- Multiply **J** by $\dot{\mathbf{x}}$ to get $\dot{\mathbf{C}}$:

$$\dot{\mathbf{J}} = \begin{bmatrix} 1 & 0 & -\mathbf{r}_x^{frame}\sin\theta^{frame} - \mathbf{r}_y^{frame}\cos\theta^{frame} & -1 & 0 & \mathbf{r}_x^{door}\sin\theta^{door} + \mathbf{r}_y^{door}\cos\theta^{door} \\ 0 & 1 & \mathbf{r}_x^{frame}\cos\theta^{frame} - \mathbf{r}_y^{frame}\sin\theta^{frame} & 0 & -1 & -\mathbf{r}_x^{door}\cos\theta^{door} + \mathbf{r}_y^{door}\sin\theta^{door} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{x}}^{frame} \\ \dot{\mathbf{y}}^{frame} \\ \dot{\boldsymbol{\theta}}^{frame} \\ \dot{\mathbf{x}}^{door} \\ \dot{\mathbf{y}}^{door} \\ \dot{\boldsymbol{\theta}}^{door} \end{bmatrix}$$

- Take partial derivative of $\dot{\mathbf{C}}$ with respect to **x** to get $\dot{\mathbf{J}}$

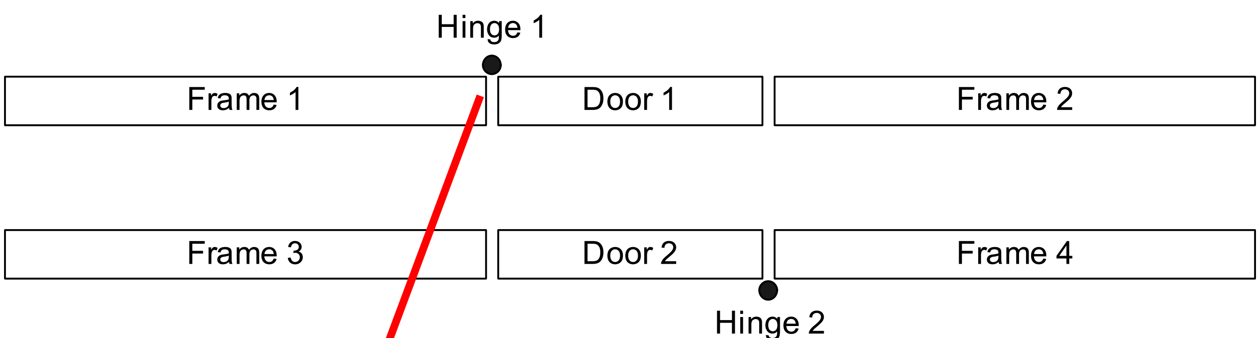
$$\dot{\mathbf{J}} = \begin{bmatrix} 0 & 0 & -\mathbf{r}_x^{frame}\cos\theta^{frame}\dot{\theta}^{frame} + \mathbf{r}_y^{frame}\sin\theta^{frame}\dot{\theta}^{frame} & 0 & 0 & \mathbf{r}_x^{door}\cos\theta^{door}\dot{\theta}^{door} - \mathbf{r}_y^{door}\sin\theta^{door}\dot{\theta}^{door} \\ 0 & 0 & -\mathbf{r}_x^{frame}\sin\theta^{frame}\dot{\theta}^{frame} - \mathbf{r}_y^{frame}\cos\theta^{frame}\dot{\theta}^{frame} & 0 & 0 & \mathbf{r}_x^{door}\sin\theta^{door}\dot{\theta}^{door} + \mathbf{r}_y^{door}\cos\theta^{door}\dot{\theta}^{door} \end{bmatrix}$$

- Plug into the constraint equation to get constraint forces

Constraint Implementation

- Each constraint represents a block in the global constraint matrices
- Constraints know their positions in the global constraint matrices and are responsible for updating their blocks

Example: Door Simulation

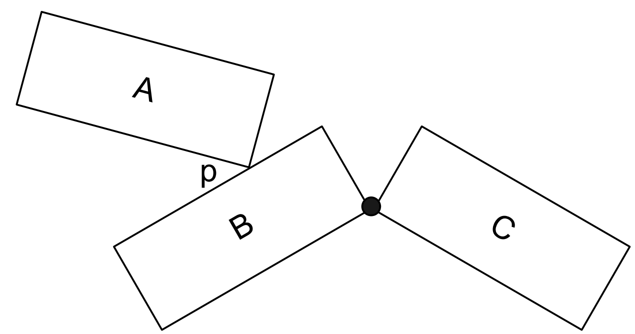


$$
\mathbf{J} = \begin{array}{c} \\ hinge\,1 \\ \\ hinge\,2 \\ \end{array}
\left[\begin{array}{cccccccccccccccccc}
\overbrace{x \quad x \quad x}^{frame\,1} & & & \overbrace{x \quad x \quad x}^{door\,1} & & & \overbrace{0 \quad 0 \quad 0}^{frame\,2} & & & \overbrace{0 \quad 0 \quad 0}^{frame\,3} & & & \overbrace{0 \quad 0 \quad 0}^{door\,2} & & & \overbrace{0 \quad 0 \quad 0}^{frame\,4} & & \\
x & x & x & x & x & x & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x & x & x & x & x & x \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x & x & x & x & x & x
\end{array}\right]
$$

Constrained Collision Response

- Constraints work under the assumption of legal positions and velocities
- Collision impulses change velocities of objects breaking this assumption
- Must solve for constraint impulses to maintain legal velocities after collisions
- Enhance collision response system of equations to account for constraints

Example: Pin Joint



- Add constraint impulse to momentum equations for object B
- Add momentum equations for object C
- Add Pin Joint equation that ensures post-collision velocities of B’s constrained point and C’s constraint point are equal

Example: Pin Joint

- Add constraint impulse to B's momentum equations

$$m^B \dot{x}^B_{post} = m^B \dot{x}^B_{pre} - R_x^{collision} + R_x^{joint}$$

$$m^B \dot{y}^B_{post} = m^B \dot{y}^B_{pre} - R_y^{collision} + R_y^{joint}$$

$$I^B \dot{\theta}^B_{post} = I^B \dot{\theta}^B_{pre} - \mathbf{r}_\perp^{collision} \cdot \mathbf{R}^{collision} + \mathbf{r}_\perp^{joint} \cdot \mathbf{R}^{joint}$$

Example: Pin Joint

- Add momentum equations for C

$$m^C \dot{x}^C_{post} = m^C \dot{x}^C_{pre} - R^{joint}_x$$

$$m^C \dot{y}^C_{post} = m^C \dot{y}^C_{pre} - R^{joint}_y$$

$$I^C \dot{\theta}^C_{post} = I^C \dot{\theta}^C_{pre} - \mathbf{r}^{joint}_{\perp} \cdot \mathbf{R}^{joint}$$

Example: Pin Joint

- Add Pin Joint equations

$$\dot{x}^B + x(\dot{\theta}^B \mathbf{r}_\perp^B) = \dot{x}^C + x(\dot{\theta}^C \mathbf{r}_\perp^C)$$
$$\dot{y}^B + y(\dot{\theta}^B \mathbf{r}_\perp^B) = \dot{y}^C + y(\dot{\theta}^C \mathbf{r}_\perp^C)$$

- System now has 13 equations and 13 unknowns – solve for and apply collision and constraint impulses
- Demo

Matrix Optimization

- After implementing constraints and constrained collision response, simulator was very slow
- According to profiler, matrix operations were biggest bottleneck by far
- The collision response matrix for 1000 rectangles “collisions” simulation is 3002x3002 and must be inverted for every collision – caused simulation to ground to a halt.

Sparse Matrices

- The collision response matrix has room for every object, but is only used to solve for a collision involving two objects
- Both collision response and constraint matrices hold very little useful data relative to its size – they are “sparse matrices”
- Both have a well-defined structure that can be exploited to increase efficiency

Collision Response Matrix

1	0	0	0	0	0	0	0	0	0.03	0.00
0	1	0	0	0	0	0	0	0	0.00	0.03
0	0	1	0	0	0	0	0	0	-0.02	0.01
0	0	0	1	0	0	0	0	0	-0.03	0.00
0	0	0	0	1	0	0	0	0	0.00	-0.03
0	0	0	0	0	1	0	0	0	-0.02	0.01
0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	1	0	0
-0.46	-0.89	-0.05	0.46	0.89	0.05	0	0	0	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0	0	0	-0.89	0.46

- Bordered Diagonal Identity Matrix for “collisions” simulation with 4 rectangles
- Only need to store and manipulate highlighted blocks (28 values vs. 121)

Constraint Matrices

- Constraint matrices are also very sparse

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	6	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	-12	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	-6	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	-12	0	0	0	0	1	0

- Blocks Sparse Matrix for “robots” simulation with 1 robot and 1 can (and 4 walls)

Efficient Sparse Matrix Operations

- Simulator exploits the structure of the sparse matrices to implement efficient matrix operations
  - Matrix time vector
  - Transpose matrix times vector
- The Biconjugate Gradient algorithm can be used to solve matrix equations involving any matrix that implements these two operations
- Enables the simulator to solve constraint and collision response equations must faster

Diagonal Matrix Operations

- The **W** matrix in the constraint equation is diagonal
- The values along the diagonal are stored as a 1-dimensional array
- “Times” method does element by element multiplication
- “Transpose Times” method is the same (since diagonal matrices are symmetric)

Block Sparse Operations

- Matrix stores a collection of blocks with their associated row and column offsets
- “Times” method multiplies a block by its corresponding section in the vector
- “Transpose Times” method is the same as times, but reverses row and column offsets and transposes block before multiplying

Block Sparse “Times” Example

- Multiply the following block sparse matrix with the vector
- Block sparse matrix contains 1 block at row 3 and column 2

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 4 & 0 & 0 \\ 0 & 5 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix}$$

Block Sparse "Times" Example

- Step 1: Create an answer vector (5 elements) and initialize to 0
- Step 2: Multiply the block with its corresponding section in the vector

$$\begin{bmatrix} 3 & 4 \\ 5 & 2 \end{bmatrix} * \begin{bmatrix} 4 \\ 5 \end{bmatrix} = \begin{bmatrix} 32 \\ 30 \end{bmatrix}$$

Block Sparse "Times" Example

- Step 3: Add resulting vector to appropriate section of answer vector

$$\begin{bmatrix} 0 \\ 0 \\ 32 \\ 30 \\ 0 \end{bmatrix}$$

- Step 4: Repeat for all blocks

Bordered Diagonal Identity Operations

- Section of vector corresponding to diagonal identity sub-matrix is copied into answer matrix (for both “times” and “transpose-times”)
- Uses same techniques as block sparse matrix for blocks in the borders

Optimization Results

- Matrix and other optimizations doubled the frame rate of “robots” simulation with 2 robots and 4 cans
- 1000 rectangle “collisions” simulation now functional

Feature Summary

- Numerical integrator to solve equations of motion and move objects in physically realistic way
- Collision detection and response
- Constraints
- Constrained Collision Response

Future Work

- Extend to 3D
- Resting contact
- Frictional collisions