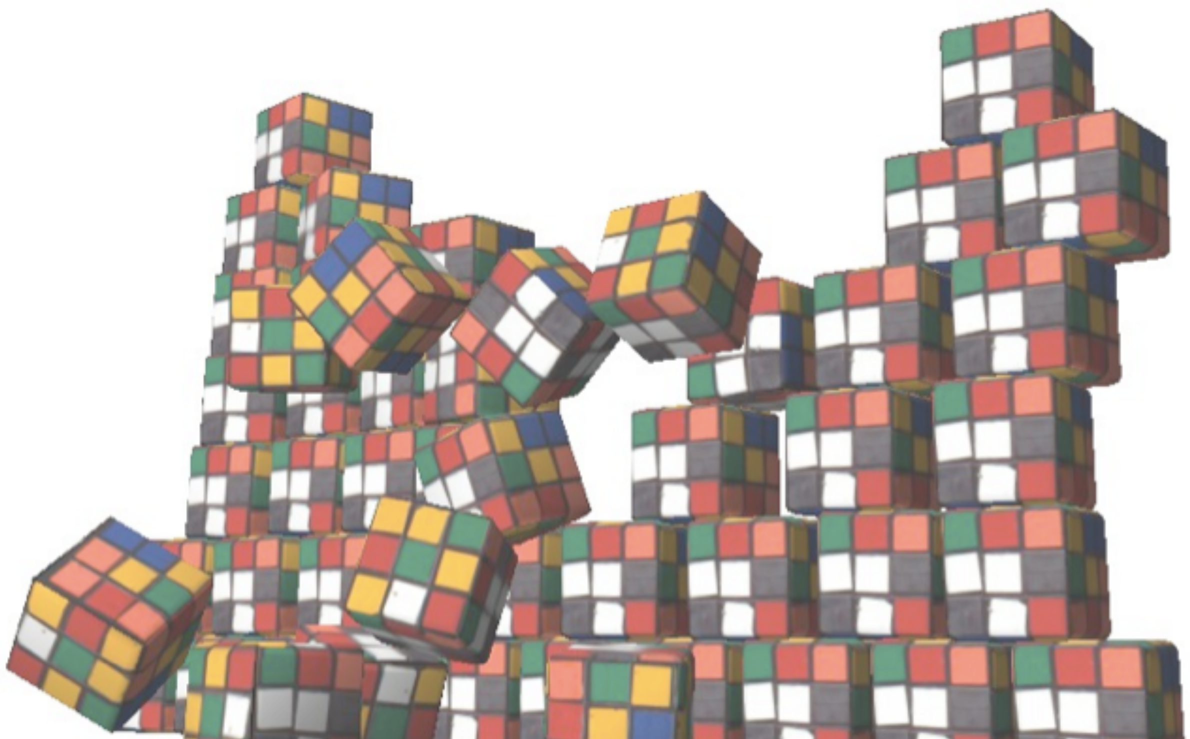
Rigid Body Simulations (3D)

Thuerey / Game Physics



Rigid Bodies - Moving to 3D

- Positions are easy: 1 new axis
  - Existing integration methods fully hold
- Orientations are quite different
  - use Quaternions

Angular Velocity in 3D

- So far, angular velocity was only the z component of the angular velocity vector
- In 3D, same principle for general vector ***w***
  - Along axis of rotation
  - Speed of rotation is given by length of ***w***
  - But now all three components are used...

Inertia Tensor

- 3x3 Values in 3D
- Tensors: generalization of vectors and matrices
  - Rank 1 tensor: vector
  - Rank 2 tensor: matrix
  - Rank 3 tensor: NxNxN block of values
  - And so on...
- Effectively: we just have a ”matrix” here!

Inertia Tensor in 3D

- Compute with mass-weighted co-variance matrix of body coordinate positions:

$$\mathbf{C} = \sum_n m_n \mathbf{x}_n \mathbf{x}_n^T$$

Tensor entries: $C_{j,k} = \sum_n m_n x_{n,j} x_{n,k}$

$$\text{trace}(\mathbf{A}) = a_{1,1} + a_{2,2} + a_{3,3}$$

$$\mathbf{I} = \mathbf{1}\ \text{trace}(\mathbf{C}) - \mathbf{C}$$

(Identity matrix denoted by $\mathbf{1}$)

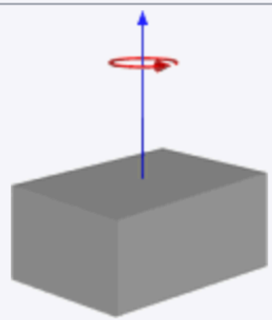
Beware: outer vector product $\mathbf{x}_n \mathbf{x}_n^T$ gives a 3x3 matrix here!

$$\mathbf{I} = \sum_n m_n \begin{pmatrix} x_2^2 + x_3^2 & -x_1 x_2 & -x_1 x_3 \\ -x_1 x_2 & x_1^2 + x_3^2 & -x_2 x_3 \\ -x_1 x_3 & -x_2 x_3 & x_1^2 + x_2^2 \end{pmatrix}$$

Note: *n* subscripts omitted here

Inertia Tensor in 3D for specific Shapes

- Use custom analytic expressions
- E.g., for rectangular box (for exercise 2) from wikipedia:

Solid cuboid of width *w*, height *h*, depth *d*, and mass *m*		$I = \begin{bmatrix} \frac{1}{12}m(h^2+d^2) & 0 & 0 \\ 0 & \frac{1}{12}m(w^2+h^2) & 0 \\ 0 & 0 & \frac{1}{12}m(w^2+d^2) \end{bmatrix}$

Updating the Inertia Tensor

- In 3D, the inertia tensor depends on the current orientation of the body!
- Luckily, we can compute this from the initial one

$$\mathbf{I}_{current} = \mathrm{Rot}_{\mathbf{r}}\ \mathbf{I}_0\ \mathrm{Rot}_{\mathbf{r}}^{-1} = \mathrm{Rot}_{\mathbf{r}}\ \mathbf{I}_0\ \mathrm{Rot}_{\mathbf{r}}^{T}$$

- Why? Used as $\mathbf{L} = \mathbf{I}\mathbf{w}$
  - > Transform angular velocity into initial orientation, multiply with inertia tensor, transform back
  - Same holds for inverse (used in practice)

Angular Motion in 3D

- In 3D: Euler step ok for velocity, but not correct for angular velocity!
- Important detail: it's not the angular velocity that is constant (without forces), but the angular momentum
- Thus: angular velocity can change *without* external forces and *without* temporal change of angular momentum
- Happens when:
  - Body has rotational velocity axis that is not a symmetry axis for body (i.e. angular momentum and angular velocity point in different directions)

Angular Motion in 3D - Example

Newton's 2nd Law for Rotations

- Angular momentum: $\mathbf{L} = \sum_i \mathbf{x}_i \times m_i \mathbf{v}_i = \mathbf{I}\mathbf{w}$
- Inertia tensor depends on current orientation ***r***
- Thus, compute with: $\mathbf{I}^{-1} = \mathrm{Rot}_{\mathbf{r}}\ \mathbf{I}_0^{-1}\ \mathrm{Rot}_{\mathbf{r}}^T$
- Torque $\mathbf{q} = \sum_i \mathbf{x}_i \times \mathbf{f}_i$
- Change of angular momentum: $\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{L} = \mathbf{q}$
- Integrate angular momentum, compute current angular velocity from it: $\mathbf{w} = \mathbf{I}^{-1}\ \mathbf{L}$

Newton's 2nd Law for Rotations

- Given forces we can now compute the change of angular velocity over time:

$$\mathbf{q}(t) = \sum_i \mathbf{x}_i \times \mathbf{f}_i$$

$$\mathbf{L}(t+h) = \mathbf{L}(t) + h\mathbf{q}$$

$$\mathbf{I}^{-1} = \mathrm{Rot}_{\mathbf{r}}\ \mathbf{I}_0^{-1}\ \mathrm{Rot}_{\mathbf{r}}^T$$

$$\mathbf{w}(t+h) = \mathbf{I}^{-1}\ \mathbf{L}(t+h)$$

Note: integrates angular momentum over time, not angular velocity!

Points vs. Rigid Bodies (3D)

- For particles:
  - …

- Dynamics:

$$\mathbf{v}(t) = \frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t}$$

$$\mathbf{a}(t) = \frac{\mathrm{d}\mathbf{v}(t)}{\mathrm{d}t}$$

- For a rigid body:
  - …

  - Angular dynamics:

$$\mathbf{q}(t) = \sum_i \mathbf{x}_i \times \mathbf{f}_i$$

$$\mathbf{L}(t+h) = \mathbf{L}(t) + h\mathbf{q}$$

$$\mathbf{w}(t+h) = \mathbf{I}^{-1}\ \mathbf{L}(t+h)$$

Integrating the Orientation

- Quaternion
  - General question - what is time derivative of orientation given as quaternion?
  - It turns out: $$\frac{\mathrm{d}\mathbf{r}}{\mathrm{d}t} = \frac{1}{2}\begin{pmatrix} 0 \\ \mathbf{w} \end{pmatrix}\mathbf{r}; \;\; \mathbf{r} = (s, xi, yj, zk)$$
  - Thus, integrate with:

$$\mathbf{r}' = \mathbf{r} + h/2\begin{pmatrix} 0 \\ \mathbf{w} \end{pmatrix}\mathbf{r}$$

Integrating the Orientation

- Rotation matrix (not recommended)
  - Recall - angular velocity given by "angular velocity cross relative position"
  - Express cross product as matrix, and apply to orientation matrix...
  - Integrate with:

$$R' = R + h \begin{pmatrix} 0 & -w_z & w_y \\ w_z & 0 & -w_x \\ -w_y & w_x & 0 \end{pmatrix} R$$

Simulation Algorithm 3D

Pre-compute:

$$M \leftarrow \sum_i m_i$$

$$\mathbf{x}'_{cm} \leftarrow \sum_i \mathbf{x}'_i m_i / M$$

$$\mathbf{x}_i \leftarrow \mathbf{x}'_i - \mathbf{x}'_{cm}$$

$$\mathbf{I}^{-1} \leftarrow \sum_i m_i \ \ldots$$

(Re-cap x' : original positions)

Initialize:

$$\mathbf{x}_{cm}, \mathbf{v}_{cm}, \mathbf{r}, \mathbf{L}$$

$$\mathbf{I}^{-1} \leftarrow \mathrm{Rot}_{\mathbf{r}} \ \mathbf{I}_0^{-1} \ \mathrm{Rot}_{\mathbf{r}}^T$$

$$\mathbf{w} \leftarrow \mathbf{I}^{-1}\mathbf{L}$$

$$\mathbf{F} \leftarrow \sum_i \mathbf{f}_i$$

External forces

$$\mathbf{q} \leftarrow \sum_i \mathbf{x}_i \times \mathbf{f}_i$$

$$\mathbf{x}_{cm} \leftarrow \mathbf{x}_{cm} + h\mathbf{v}_{cm}$$

Euler step

$$\mathbf{v}_{cm} \leftarrow \mathbf{v}_{cm} + h\mathbf{F}/M$$

$$\mathbf{r} \leftarrow \mathbf{r} + h/2 \begin{pmatrix} 0 \\ \mathbf{w} \end{pmatrix} \mathbf{r}$$

Quaternion!

$$\mathbf{L} \leftarrow \mathbf{L} + h\mathbf{q}$$

$$\mathbf{I}^{-1} \leftarrow \mathrm{Rot}_{\mathbf{r}} \ \mathbf{I}_0^{-1} \ \mathrm{Rot}_{\mathbf{r}}^T$$

$$\mathbf{w} \leftarrow \mathbf{I}^{-1} \ \mathbf{L}$$

$$\mathbf{x}_i^{world} \leftarrow \mathbf{x}_{cm} + \mathrm{Rot}_r \mathbf{x}_i$$

World position

$$\mathbf{v}_i^{world} \leftarrow \mathbf{v}_{cm} + \mathbf{w} \times \mathbf{x}_i$$

Euler vs. Leapfrog

- Why not use leapfrog for rigid bodies?

$$\mathbf{v}(t+h/2) = \mathbf{v}(t-h/2) + h\ \mathbf{a}(t)$$
$$\mathbf{x}(t+h) = \mathbf{x}(t) + h\ \mathbf{v}(t+h/2)$$

- Integration:

$$\mathbf{x}_{cm} \leftarrow \mathbf{x}_{cm} + h\mathbf{v}_{cm}$$
$$\mathbf{v}_{cm} \leftarrow \mathbf{v}_{cm} + h\mathbf{F}/M$$

- Note - no internal forces! $\mathbf{F}$ doesn't depend on position
- Higher order methods: collisions are expensive

Impulse works like before, mostly...

- Impulse in 2D was:

$$J = \frac{-(1+c)\mathbf{v}_{rel} \cdot \mathbf{n}}{\frac{1}{M_a} + \frac{1}{M_b} + (\mathbf{x}_a \times \mathbf{n})^2 / I_a + (\mathbf{x}_b \times \mathbf{n})^2 / I_b}$$

- Inertia tensor is matrix in 3D (less simplification possible):

$$J = \frac{-(1+c)\mathbf{v}_{rel} \cdot \mathbf{n}}{\frac{1}{M_a} + \frac{1}{M_b} + [(\mathbf{I}_a^{-1}(\mathbf{x}_a \times \mathbf{n})) \times \mathbf{x}_a + (\mathbf{I}_b^{-1}(\mathbf{x}_b \times \mathbf{n})) \times \mathbf{x}_b] \cdot \mathbf{n}}$$

Impulse in 3D

- Calculate with:

$$J = \frac{-(1+c)\mathbf{v}_{rel} \cdot \mathbf{n}}{\frac{1}{M_a} + \frac{1}{M_b} + [(\mathbf{I}_a^{-1}(\mathbf{x}_a \times \mathbf{n})) \times \mathbf{x}_a + (\mathbf{I}_b^{-1}(\mathbf{x}_b \times \mathbf{n})) \times \mathbf{x}_b] \cdot \mathbf{n}}$$

- Update:

$$\mathbf{v}'_a = \mathbf{v}_a + J\mathbf{n}/M_a$$
$$\mathbf{v}'_b = \mathbf{v}_b - J\mathbf{n}/M_b$$

$$\mathbf{L}'_a = \mathbf{L}_a + (\mathbf{x}_a \times J\mathbf{n})$$
$$\mathbf{L}'_b = \mathbf{L}_b - (\mathbf{x}_b \times J\mathbf{n})$$

Impulse in 3D - Fixed Bodies

- 3D Impulse:

$$J = \frac{-(1+c)\mathbf{v}_{rel} \cdot \mathbf{n}}{\frac{1}{M_a} + \frac{1}{M_b} + [(\mathbf{I}_a^{-1}(\mathbf{x}_a \times \mathbf{n})) \times \mathbf{x}_a + (\mathbf{I}_b^{-1}(\mathbf{x}_b \times \mathbf{n})) \times \mathbf{x}_b] \cdot \mathbf{n}}$$

- Heavy / fixed body: *M* approaches infinity
- Can be simulated by setting inverse mass and inertia tensor to zero
- Implementation only needs inverse values!

Video Examples

- ...

Contact Data

- Store collision point (world coordinates)
- Contact normal (body 2)
- Usually useful: penetration depth
- Possible implementation:

```
class Contact
{
  public:
    Vector3     position;
    Vector3     normal;
    float       depth;
    int         body1, body2;
    ...
}
```

What you should know…

- Rigid body representation
- Angular concepts of velocities and forces
- Calculate & implement examples of motion integration and applying forces
- Modeling collisions with impulses
- Calculate & implement simple rigid body collisions