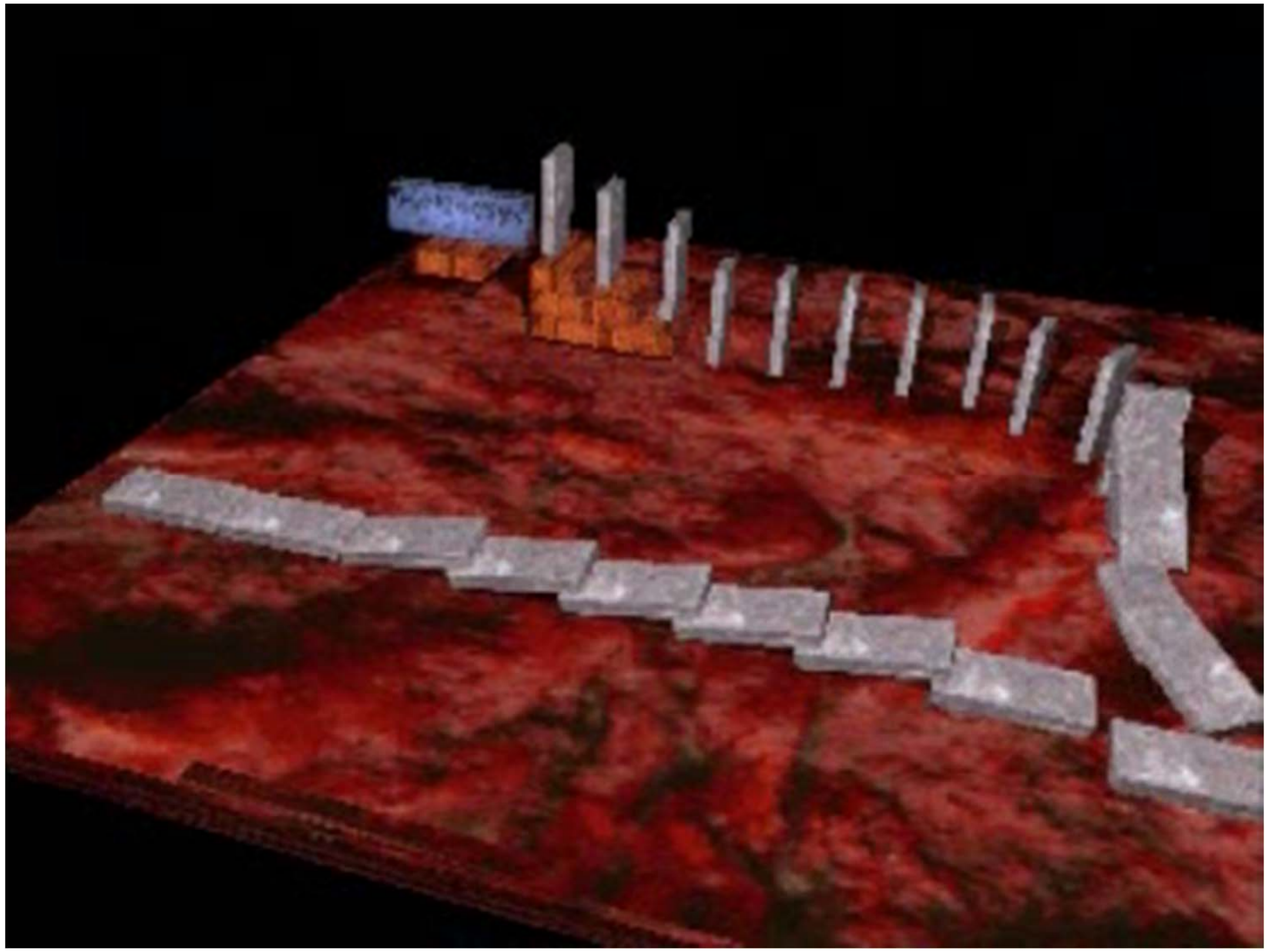
Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

ASSIGNMENT 2: RIGID BODY UNCONSTRAINED MOTION

Demo Due: 01/02/2015
Cumulative Report, Video and Source to be submitted 28/02/2015
20% penalty per day late for all deadlines

RIGID BODIES



© 1999 Telekinesys Research | Havok

ASSIGNMENT 2 (DEMO DUE 01/02/2016)

Implement and demonstrate simulation of the unconstrained motion of a rigid body to include:

- Rigid body state data structure
- Rigid body geometry Representation and Rendering (simple block or mesh)
- Rigid body state update – based on velocity and angular velocity
- Torque and Force; centre of mass and inertia tensor implementation

This assignment is worth ~10% of the final mark

- Expected ~8 hours to complete

You should demo your solution (~5 minutes) in the lab next week

- There will be a penalty of 20% for each day late. You must inform me in advance of deadline if you have some grave reason for being late.
- If late, **the onus is on you** to email me the code when you have finished and then we will arrange alternate time for demo. The date you email me the code will be taken as date of submission for purposes of the above penalty. It is expected that you will not make further modifications to the demo after this date.

Report (max 1 page), code and youtube video should be submitted with the cumulative submission on 28/02/2016

GOALS

0. **Read Baraff's notes "Lecture 5 Rigid Body Dynamics, PART I Unconstrained Rigid Body Dynamics" in PBM Siggraph 2001 course notes.**
   - Available at: http://www.pixar.com/companyinfo/research/pbm2001/

1. **Rigid Body Geometry**
   - Implement a simple data structure and draw function for a basic Rigid Body

2. **Implement the state data structure for rigid body**
   - TO DO: show that your rigid body updates correctly when you manually change its position $\boldsymbol{x}$ and orientation $\boldsymbol{R}$

3. **State update: Unconstrained Rigid Body Motion**
   - Update position of rigid body and orientation of rigid body based on velocity and angular velocity
   - TO DO: show that your rigid body can update automatically based on a given linear velocity $\boldsymbol{v}$ and angular velocity $\boldsymbol{\omega}$

4. **Force and mass properties**
   - Define centre of mass and Inertial Tensor
   - Add handling of force and torque

1. RIGID BODY GEOMETRY

Goal: Implement a simple geometric representation to associate with your rigid body.

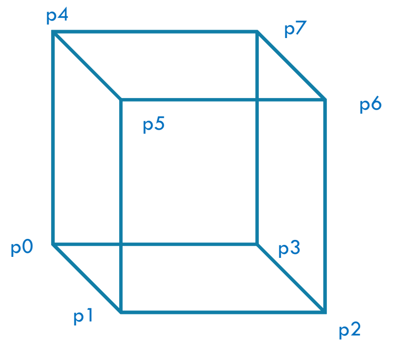
- Detailed geometry representation is slightly outside of the scope of this module so you can keep this simple. Even a simple block (cuboid) will do.
- You'll need a "draw" function to display the geometry
- You may use external functions or 3rd party code (make sure you cite any external sources used). BUT you MUST have access to each individual vertex – before and after it is transformed. **YOU WILL LOSE MARKS IF YOU CAN'T RETURN THE VALUE OF THE TRANSFORMED VERTICES**

Very Simple Example:

```
class VertexList
{
    int num_vertices
    vector3 vlist [];
    void draw();
}
```

```
void VertexList::draw()
{
   for int (i=0; i<num_vertices; i++)
     drawVertex(vlist[i].x,
          vlist[i].y, vlist[i].z);
}
```

p0	-0.5,-0.5,-0.5
p1	+0.5,-0.5,-0.5
p2	+0.5,+0.5,-0.5
p3	-0.5,+0.5,-0.5
p4	-0.5,-0.5,+0.5
p5	+0.5,-0.5,+0.5
p6	+0.5,+0.5,+0.5
p7	-0.5,+0.5,+0.5



2. RIGID BODY STATE DATA STRUCTURE

Goal: Associate the physical state information of your rigid body with the rigid body geometry

You will need vector and matrix classes. Use existing ones or create your own. Make sure you site any third-party sources*

- Following examples in C++ (disclaimer: provided as is; not really bug-tested exhaustively):
  - https://www.scss.tcd.ie/michael.manzke/cs7057/vector.h
  - https://www.scss.tcd.ie/michael.manzke/cs7057/matrix.h
  - https://www.scss.tcd.ie/michael.manzke/cs7057/matrix.cpp

Update your draw function to draw the geometry based on x and R

- All points in the mesh are multiplied by $\boldsymbol{R}$ and $\boldsymbol{x}$ is added to each of their positions. I recommend you do this transform yourself (not using a graphics function e.g. glRotate) For collisions etc, you will later need to know the physical value of this transformed point so make sure you can return it

$$\boldsymbol{p}' = \boldsymbol{R}.\boldsymbol{p} + \boldsymbol{x}$$

where $\boldsymbol{p}$ is a point on the mesh, $\boldsymbol{p}'$ is its transformed world position

TO DO: Change Values of x and R in Rigid Body data structure and redraw to ensure that the rigid body is correctly updated

*: Good detailed example is David Eberly's Wild Magic SDK: http://www.geometrictools.com/

RIGID BODY REPRESENTATION

RigidBody

Mass = m

InertialTensor = $\boldsymbol{I}_{BODY}$

$$\boldsymbol{Y}(t) = \begin{cases} \text{Position} = \boldsymbol{x}(t) \\ \text{Orientation} = \boldsymbol{R}(t) \\ \text{LinearMomentum} = \boldsymbol{P}(t) \\ \text{AngularMomentum} = \boldsymbol{L}(t) \end{cases}$$

STATE VARS

Force = $\boldsymbol{F}(t)$

Torque = $\boldsymbol{t}(t)$

$\downarrow$

Rigid body geometry

Used for display, for collision detection, and for calculating mass properties (Inertial tensor **I**, center of mass)

Simulation involves finding the change of $\boldsymbol{Y}$ over time:

$$\frac{d}{dt}Y(t) = \frac{d}{dt}\begin{bmatrix} \boldsymbol{x}(t) \\ \boldsymbol{R}(t) \\ \boldsymbol{P}(t) \\ \boldsymbol{L}(t) \end{bmatrix} = \begin{bmatrix} \boldsymbol{v}(t) \\ \boldsymbol{\omega}(t)^{*}\boldsymbol{R}(t) \\ \boldsymbol{F}(t) \\ \boldsymbol{\tau}(t) \end{bmatrix}$$

RIGID BODY REPRESENTATION

RigidBody

Mass = m

InertialTensor = $\boldsymbol{I}_{BODY}$

$$\boldsymbol{Y}(t) = \left\{ \begin{array}{l} \text{Position} = \boldsymbol{x}(t) \\ \text{Orientation} = \boldsymbol{R}(t) \\ \text{LinearMomentum} = \boldsymbol{P}(t) \\ \text{AngularMomentum} = \boldsymbol{L}(t) \end{array} \right.$$

STATE VARS

Force = $\boldsymbol{F}(t)$

Torque = $\boldsymbol{t}(t)$

$\downarrow$

Rigid body geometry

Used for display, for collision detection, and for calculating mass properties (Inertial tensor **I**, center of mass)

Simulation involves finding the change of $\boldsymbol{Y}$ over time:

$$\frac{d}{dt}Y(t) = \frac{d}{dt}\begin{bmatrix} \boldsymbol{x}(t) \\ \boldsymbol{R}(t) \\ \boldsymbol{v}(t)m \\ \boldsymbol{\omega}(t)\boldsymbol{I}(t) \end{bmatrix} = \begin{bmatrix} \boldsymbol{v}(t) \\ \boldsymbol{\omega}(t)^{*}\boldsymbol{R}(t) \\ \boldsymbol{F}(t) \\ \boldsymbol{\tau}(t) \end{bmatrix}$$

NB we are using Linear Momentum,

$$\boldsymbol{P}(t) = \boldsymbol{v}(t)m,$$

instead of velocity and Angular momentum,

$$\boldsymbol{L}(t) = \boldsymbol{\omega}(t)\boldsymbol{I}(t),$$

instead of angular velocity

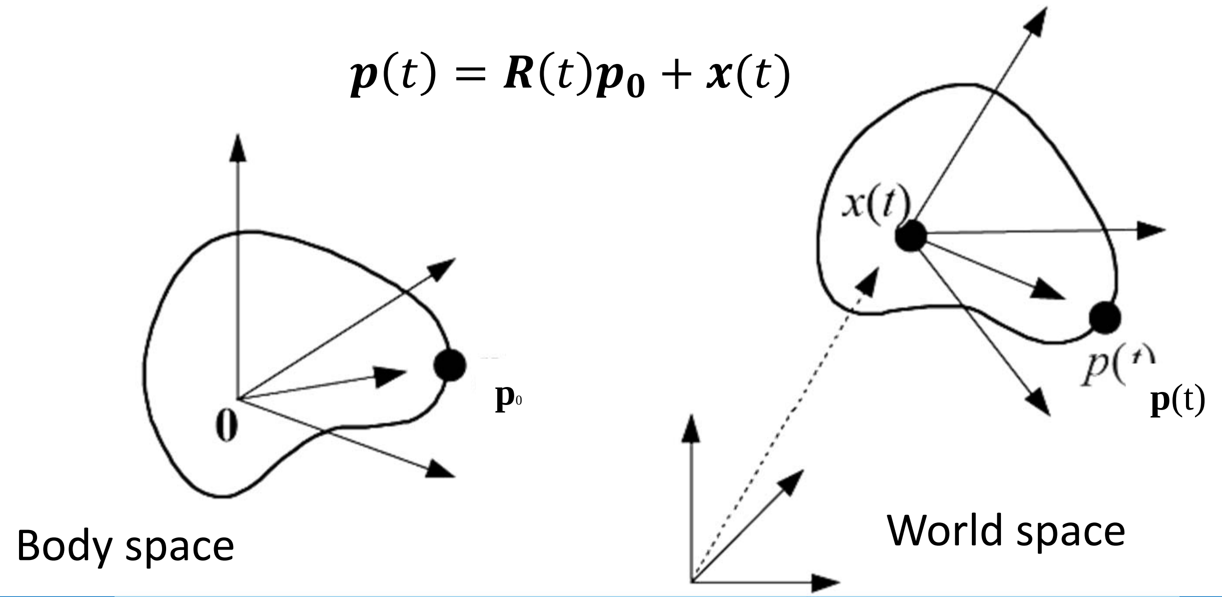
RECALL: ORIENTATION

Can be represented as a 3x3 rotation matrix

$$\boldsymbol{R}(t) = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

Points on object are transformed from body space to world space using:

$$\boldsymbol{p}(t) = \boldsymbol{R}(t)\boldsymbol{p_0} + \boldsymbol{x}(t)$$



3. AUTOMATIC STATE UPDATE

Implement a simple* update function that updates x and R based on the linear and angular velocity (v and ω).

Updating position of centre of gravity is simple

$$\boldsymbol{x}(t+\Delta t) = \boldsymbol{x}(t) + \boldsymbol{v}(t)$$

NOTE: Δt is the timestep (denoted as h in some of the notes)

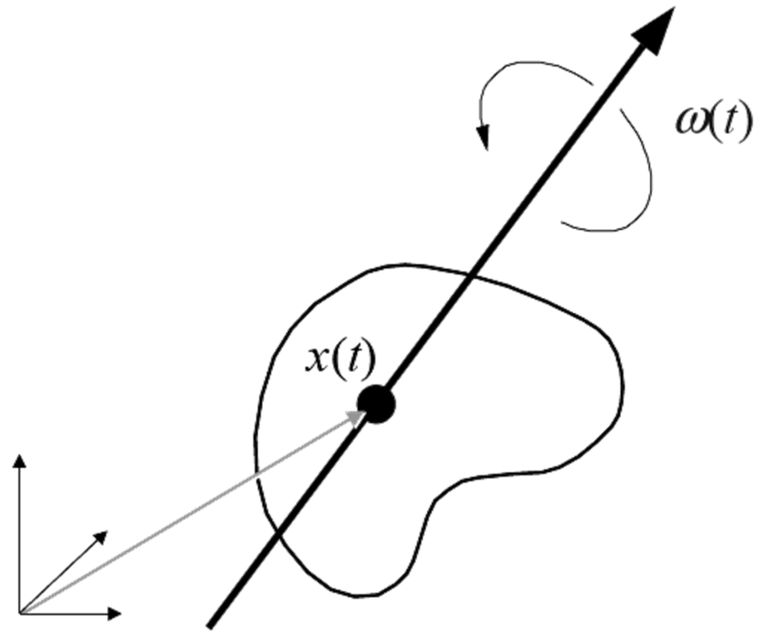
For updating orientation, see following slides

TO DO: Give your rigid body an initial starting velocity v and starting angular velocity ω and show that that your rigid body can update automatically based on v and ω

* Euler integration is sufficient for now

ANGULAR VELOCITY

RECAP FROM LAST LECTURE



Angular velocity $\boldsymbol{\omega}(t)$ = $\begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$

N.B. this does **NOT** imply rotating about X by ω_x, about Y by ω_x, about Z by ω_z

$\boldsymbol{\omega}(t)$ **encodes:**

- the axis of rotation (given by direction of $\boldsymbol{\omega}$)
- speed of the spin (given in radians by the magnitude $\boldsymbol{\omega}$)

RB ROTATIONAL UPDATE

How is R updated based on ω?

- Based on Baraff et al, $\dot{\boldsymbol{R}}(t)$ and $\boldsymbol{\omega}(t)$ are related by:

$$\frac{d}{dt}\boldsymbol{R}(t) = \begin{bmatrix} 0 & -\omega_z(t) & \omega_y(t) \\ \omega_z(t) & 0 & -\omega_x(t) \\ -\omega_y(t) & \omega_x(t) & 0 \end{bmatrix} \boldsymbol{R}(t)$$

$$= \boldsymbol{\omega}(\mathrm{t})^{*}\boldsymbol{R}(t)$$

- Thus we can update $\boldsymbol{R}$ as follows :

$$\boldsymbol{R}(t+\Delta t) = \boldsymbol{R}(t) + \boldsymbol{\omega}^{*}\boldsymbol{R}(t)\Delta t$$

For an explanation see Baraff Witkin Kass, PBM siggraph course notes, 2001, Lecture on Rigid Body Dynamics Part I – Unconstrained Rigid Body Dynamics

DRIFT CORRECTION

Due to numerical drift (this arises from incremental update of the matrix) your object may shear over time, you need to renormalize and re-orthogonalise *R*.

- If C_x, C_y, C_z are the column vectors of ***R*** the following unoptimized Psuedocode should acheive this:

```
$C_x$ = $C_x$/length($C_x$)
$C_y$ = crossproduct ($C_z$, $C_x$)
$C_y$ = $C_y$ /length($C_y$)
$C_z$ = crossproduct ($C_x$, $C_y$)
$C_z$ = $C_z$ /length($C_z$)
```

There are more formal ways of orthogonalizing a matrix e.g. Gram-Schmidt. The following references are slightly more math-friendly for programmers:

- Sec 9.3.3. Orthogonalizinga matrix in “3D Math Primer for Graphics and Game Development” by Fletcher Dunn and Ian Parberry(p135) . Available at: http://www.scss.tcd.ie/michael.manzke/cs7057/Matrix-Orthogonalization.pdf
- Sec 9.6 Matrix Orthogonalization in “Graphcis Gems, Volume 1” by Andrew Glassner(p 464)

ALTERNATIVES FOR UPDATING ORIENTATION

Rotate each column vector of R (in a sense this is the same as what we already suggested, see Baraff Witkin Kass)

- Generate relevant matrix for "rotation about an arbitrary axis" transform $\boldsymbol{R}_{\omega}$ based on $\boldsymbol{\omega}$.

$$\boldsymbol{R}(t + \Delta t) = \Delta t \boldsymbol{R}_{\omega} \boldsymbol{R}(t)$$

Use quaternions,

- Game Physics Engine Development - Ian Millington. Chapter 9
- Baraff et al Rigid Body Dynamics, Part I, Section 4 Quaternions vs. Rotation Matrices

4. FORCE AND TORQUE

Similar to particles, any force acting on an a rigid body causes change in linear momentum

For multiple forces calculate the net force on a rigid body

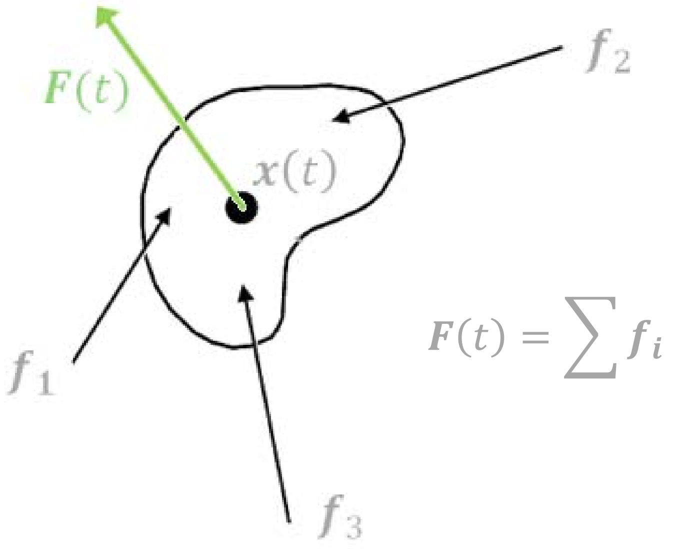
- Accumulate this in the rigid body state

RigidBody

Mass = m
InertialTensor = I_{BODY}

Position = $\mathbf{x}(t)$
Orientation = $\mathbf{R}(t)$
LinearMomentum = $\mathbf{P}(t)$
AngularMomentum = $\mathbf{L}(t)$

Force = $\mathbf{F}(t)$
Torque = $\tau(t)$

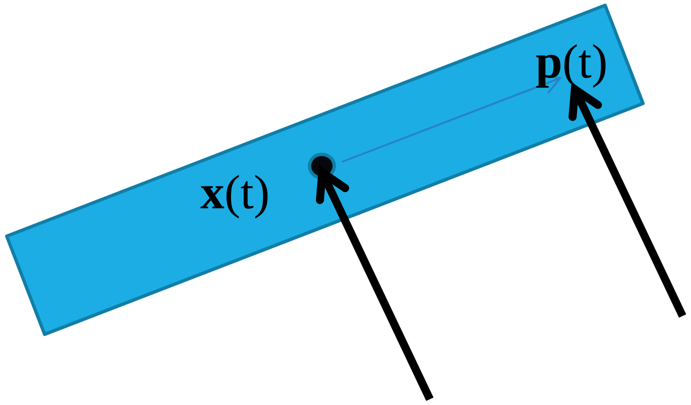


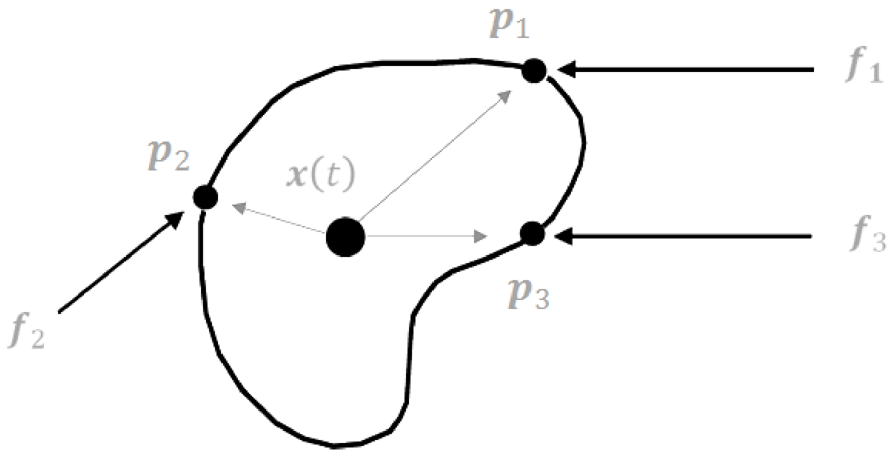
N.B. Unlike particles, $\boldsymbol{F}(t)$ is the net force acting on the object through its c.o.m. **x**

TORQUE

$$\boldsymbol{\tau}_i(t) = (\boldsymbol{p}_i - \boldsymbol{x}(t)) \times \boldsymbol{f}_i(t)$$

For a hinged object, $\boldsymbol{x}$ is the position of the pivot/hinge. Otherwise $\boldsymbol{x}$ is c.o.m.





Similar to force, we can calculate the net Torque as

$$\boldsymbol{\tau}(t) = \sum (\boldsymbol{p}_i - \boldsymbol{x}(t)) \times \boldsymbol{f}_i(t)$$

APPLYING FORCE AND TORQUE

Force causes change in linear momentum

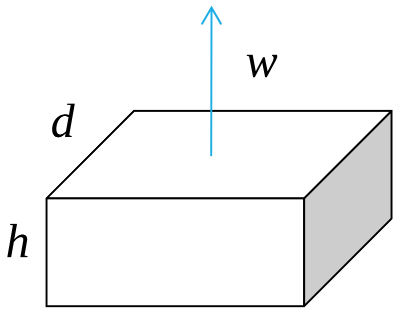
- i.e. $\Delta \boldsymbol{P} = \boldsymbol{F}\Delta \boldsymbol{t}$
- But linear momentum is defined as $\boldsymbol{P}(t) = m\boldsymbol{v}(\boldsymbol{t})$
- So $m\Delta \boldsymbol{v} = \boldsymbol{F}\Delta \boldsymbol{t} \quad \Rightarrow \quad \Delta \boldsymbol{v} = \boldsymbol{F}\Delta t/m$

Similarly Torque causes change in angular momentum

- $\Delta \boldsymbol{L} = \boldsymbol{\tau}\Delta \mathbf{t}$
- Linear momentum defined as $\boldsymbol{L}(t) = \boldsymbol{I}(t)\boldsymbol{\omega}(t)$
- So $\boldsymbol{I}\Delta \boldsymbol{\omega} = \boldsymbol{\tau}\Delta \mathbf{t} \quad \Rightarrow \quad \Delta \boldsymbol{\omega} = \boldsymbol{I}^{-1}\boldsymbol{\tau}\Delta t$

INERTIAL TENSOR: CUBE APPROXIMATION

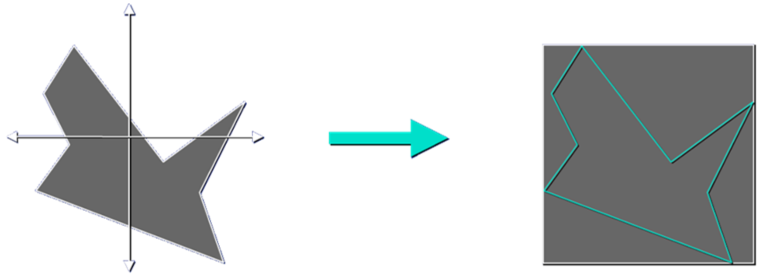
For a cube with axis of rotation about the center of the object:



$$\boldsymbol{I} = \begin{bmatrix} \frac{1}{12}m(h^2+d^2) & 0 & 0 \\ 0 & \frac{1}{12}m(w^2+d^2) & 0 \\ 0 & 0 & \frac{1}{12}m(w^2+h^2) \end{bmatrix}$$

where m the is total mass of the body.

Assuming uniform density we can roughly approximate the Inertial tensor of any object as the inertial tensor of its bounding cube:



This seems drastic but for most convex objects the simulation error is often unnoticeable.
It is common practice to use dynamics proxies such as this and the important thing is we can preserve some sort of asymmetry in the behaviour

INERTIAL TENSORS DURING SIMULATION

Inertial tensor is invariant to translation but changes with a body's orientation

Thankfully we can straightforwardly calculate the inertial tensor $\boldsymbol{I}(t)$ based on $\boldsymbol{I}_{body}$ and $\boldsymbol{R}(t)$

$$\boldsymbol{I}(t) = \boldsymbol{R}(t)\underset{\uparrow}{\boldsymbol{I}_{body}}\boldsymbol{R}(t)^T$$

$\boldsymbol{I}_{body}$ **is a constant associated with the object in body-space.**

REFERENCES

Required Reading

- David Baraff "Lecture 5 Rigid Body Dynamics, PART I Unconstrained Rigid Body Dynamics" in PBM Siggraph 2001 course notes.
  - http://www.pixar.com/companyinfo/research/pbm2001/

Optional References

- Brian Mirtich, "Fast and Accurate Computation of Polyhedral Mass Properties", Journal of Graphics Tools. Vol. 1 (2) , 1996.
  - http://tinyurl.com/mo59tzr
- David Eberly"Polyhedral Mass Properties"
  - http://www.geometrictools.com/Documentation/PolyhedralMassProperties.pdf
  - Source Code: http://www.geometrictools.com/Source/Physics.html

ASIDE: ASSIGNMENT 2 MARKING

Compulsory Requirements:

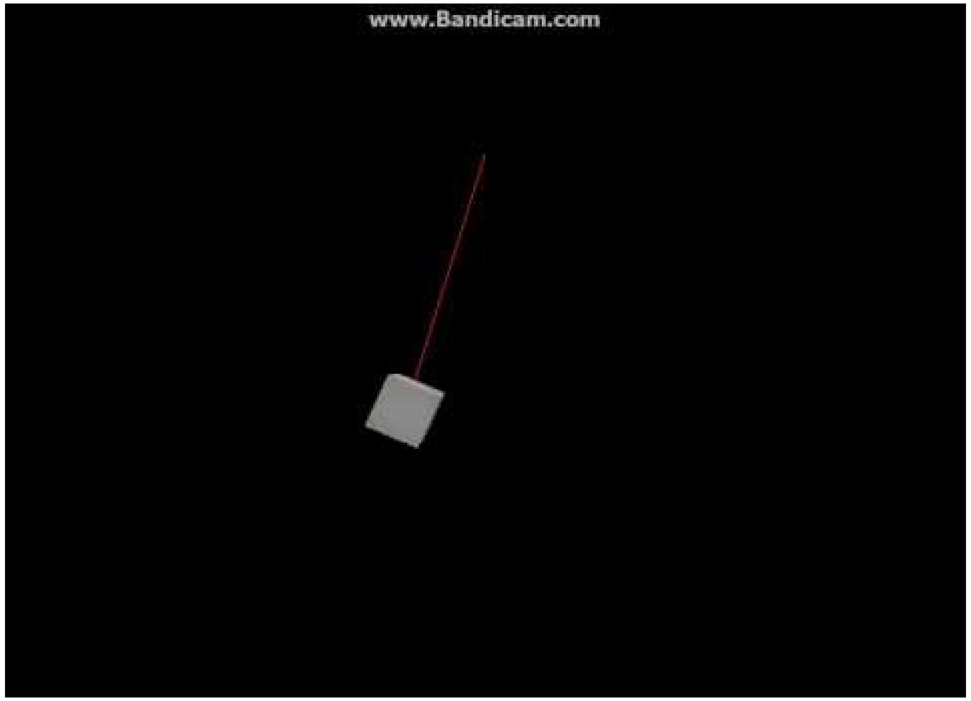
1. Rigid Body Geometry: data structure and draw function
2. Rigid Body State data structure:
   - Minimum Requirement: Show that this updates correctly based on any given position $\boldsymbol{x}(t)$ and orientation $\boldsymbol{R}(t)$
3. Rigid Body State update
   - Minimum Requirement: Show that given a starting velocity and angular velocity the Rigid Body can be updated to move (animate) with it's position and orientation changing correctly
4. Approximated Inertia Tensor, Force and Torque
   - Given a Force and Torque, show that you can simulate the changing state of the Rigid Body over time (Euler Integration is sufficient for now)

60%

Optional (Some of the following)

- Implement a meaningful demo (that does something interesting) e.g. apply a force at a point, calculate the Torque and create the correct simulation
- Generalizable implementation
- Visualisation of variables
- Use of quaternions
- Generalised geometry class implemented (e.g. polygon mesh) with loader
- Add computation of c.o.m. from general mesh
- Add computation/approximation of Inertia tensor beyond just approximating as a cube

40%


www.Bandicam.com

CUMULATIVE SUBMISSION (ASSIGNMENT 1-5)

The full cumulative submission (code, report and youtube video) is due 28/02/2016 at 23:59 and should combine reports, video and code from all labs leading upto reading week.

Goals:

- Implement a rigid body dynamics engine capable of simulating a **convex** object colliding with a plane. Also implement a particle systems simulation.
- Write a report approx 4-5 pages (max 1 page on each point) discussing your implementation:
  - Particle System Simulation
  - Rigid Body Unconstrained Motion
  - Collision detection: broad phase
  - Collision detection: narrow phase
  - Contact Modelling & Collision Response

Submission via http://turnitin.com

Technical report (upto 5 pages) including link to YouTube video

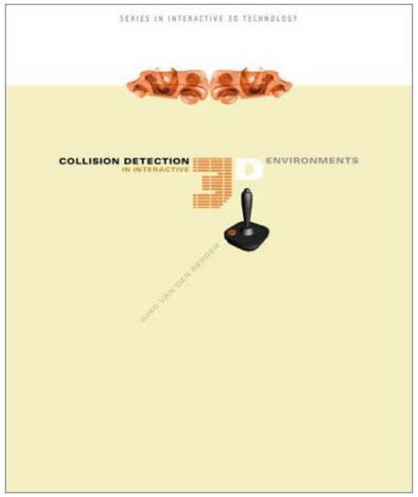
- Source code and executable (in zip file)

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

04: COLLISION DETECTION

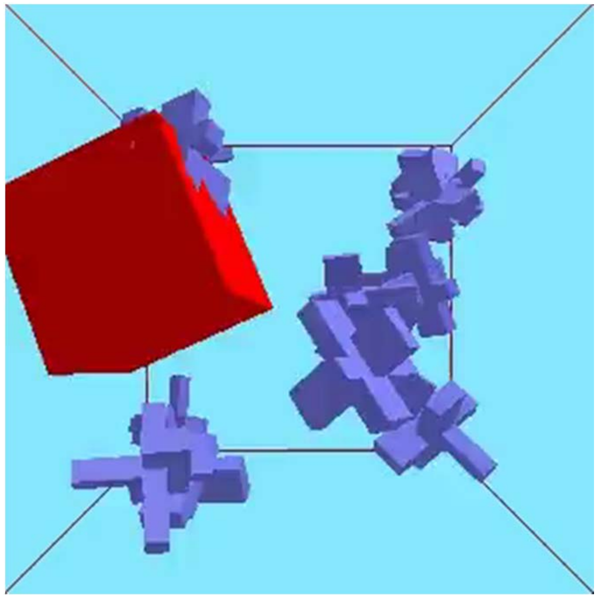
RELEVANT BOOKS (COLLISION DETECTION)

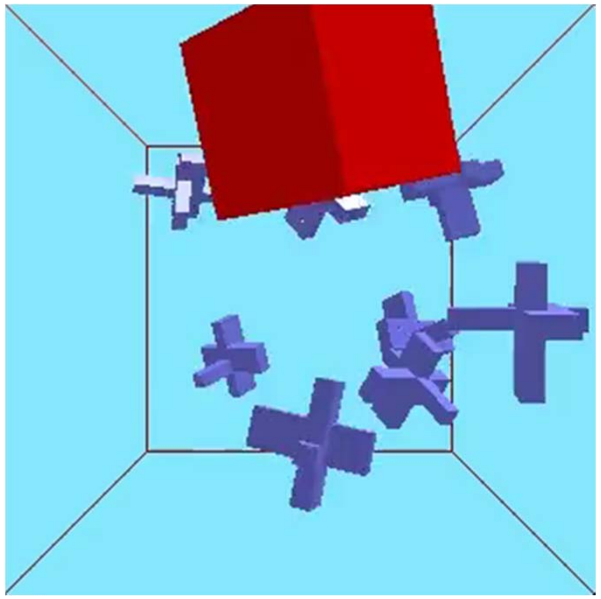
Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3D Technology) - By Christer Ericson


SERIES IN INTERACTIVE 3D TECHNOLOGY
COLLISION DETECTION IN INTERACTIVE 3D ENVIRONMENTS
GINO VAN DEN BERGEN

Collision Detection in Interactive 3D Environments (The Morgan Kaufmann Series in Interactive 3D Technology) - By Gino van den Bergen

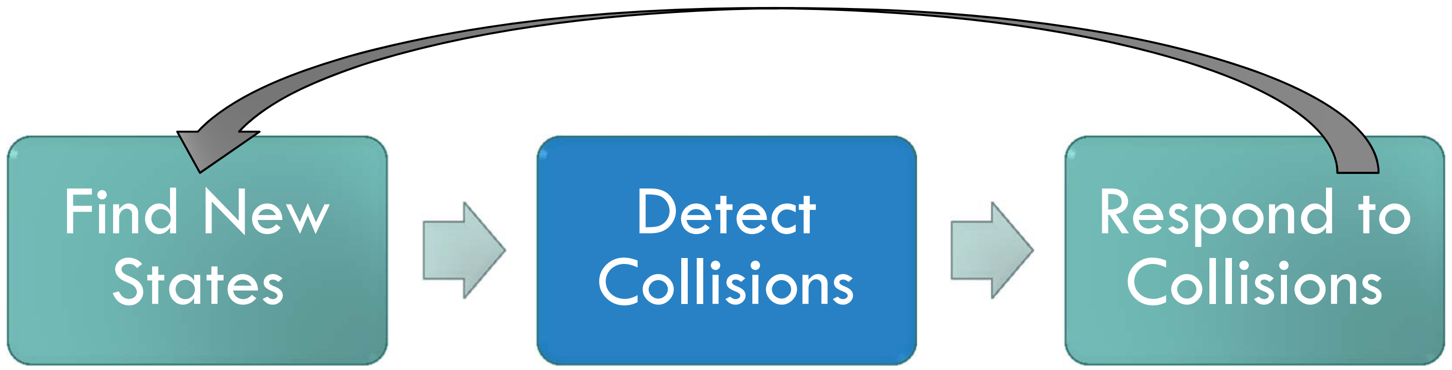
COLLISION DETECTION





Collisions are an important part of interactive animation
(Video courtesy of Carol O'Sullivan)

BASIC SIMULATION LOOP



[Erleben et al] Physics-based Animation

COLLISION DETECTION

Related Terms:

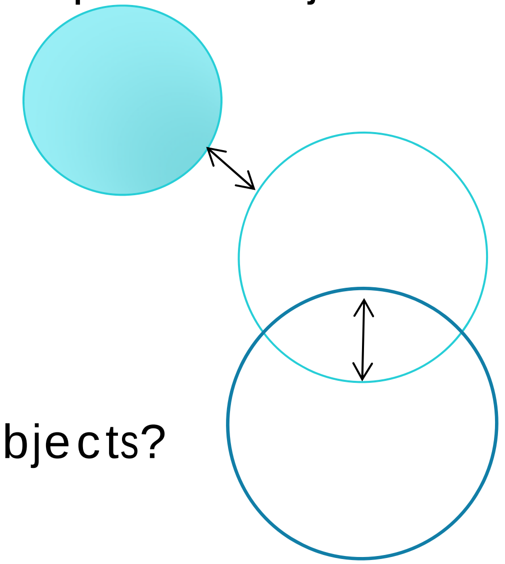
- Proximity Queries, Intersection Testing, Penetration depth testing, Contact modelling, Collision Avoidance

Also used in:

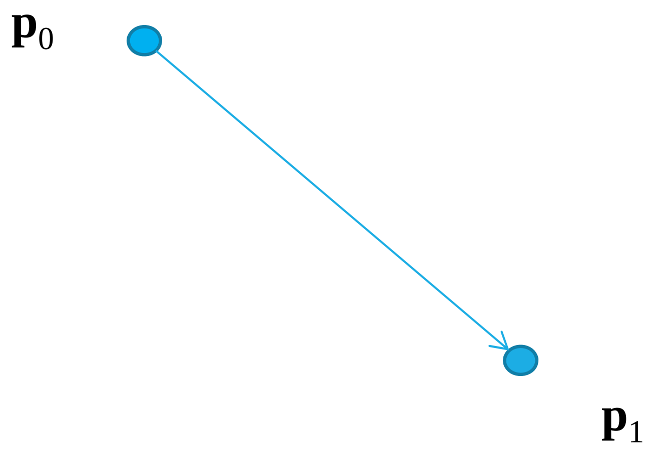
- Rendering : raytracing, Shadow rendering, Non-planar Projection
- Visibility determination, Occlusion culling
- Behaviour simulation, navigation, robotics

Do objects intersect while moving?

- NO: How far apart are the objects?
- YES: How much is the overlap between the objects?



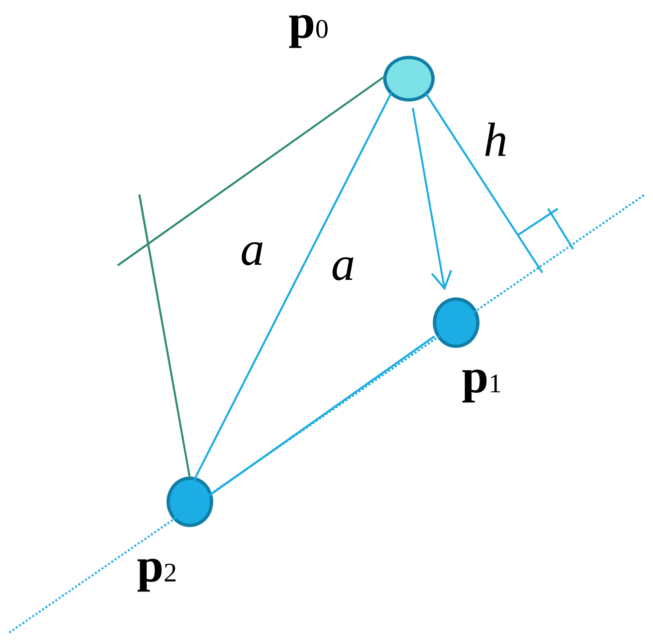
DISTANCE: POINT TO POINT



$$d = \left|\mathbf{p}_0 - \mathbf{p}_1\right| = \sqrt{(\mathbf{p}_0.x - \mathbf{p}_1.x)^2 + (\mathbf{p}_0.z - \mathbf{p}_1.y)^2 + (\mathbf{p}_0.z - \mathbf{p}_1.z)^2}$$

DISTANCE: POINT TO EDGE

$$d = \min\left\{ \frac{\left|(\mathbf{p}_2 - \mathbf{p}_1)\times(\mathbf{p}_0 - \mathbf{p}_1)\right|}{\left|\mathbf{p}_2 - \mathbf{p}_1\right|},\quad \left|\mathbf{p}_0 - \mathbf{p}_1\right|,\quad \left|\mathbf{p}_0 - \mathbf{p}_2\right| \right\}$$



Area of the parallelogram *A:*

$$A = \left|(\mathbf{p}_2 - \mathbf{p}_1)\times(\mathbf{p}_0 - \mathbf{p}_1)\right|$$

The area of the triangle *a*

$$a = \frac{\left|(\mathbf{p}_2 - \mathbf{p}_1)\right|\ h}{2} \quad \textit{where} \quad a = \frac{A}{2}$$

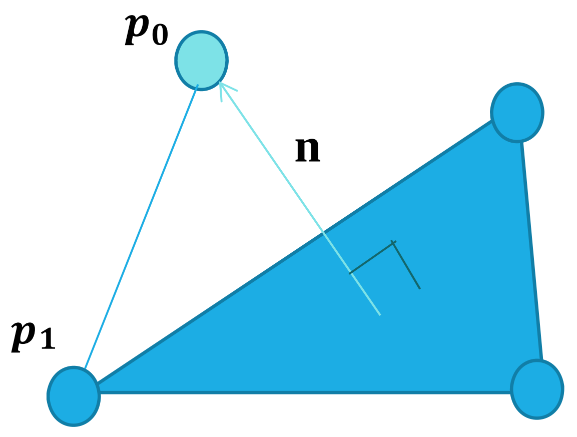
Thus

$$h = \frac{A}{\left|(\mathbf{p}_2 - \mathbf{p}_1)\right|} = \frac{\left|(\mathbf{p}_2 - \mathbf{p}_1)\times(\mathbf{p}_0 - \mathbf{p}_1)\right|}{\left|(\mathbf{p}_2 - \mathbf{p}_1)\right|}$$

And

$$d = \min\left\{ \frac{\left|(\mathbf{p}_2 - \mathbf{p}_1)\times(\mathbf{p}_0 - \mathbf{p}_1)\right|}{\left|\mathbf{p}_2 - \mathbf{p}_1\right|},\quad \left|\mathbf{p}_0 - \mathbf{p}_1\right|,\quad \left|\mathbf{p}_0 - \mathbf{p}_2\right| \right\}$$

DISTANCE: POINT TO FACE

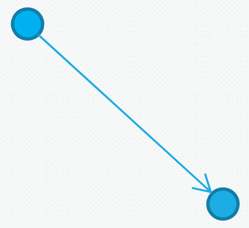


$$d = \min\left\{(\mathbf{p_o} - \mathbf{p_1}).\widehat{\mathbf{n}}, \quad \mathbf{p_o}\ to\, e1, \mathbf{p_o}\ to\ e2, \ldots.\, \mathbf{p_o}\ to\ eN\right\}$$

DISTANCE TESTS

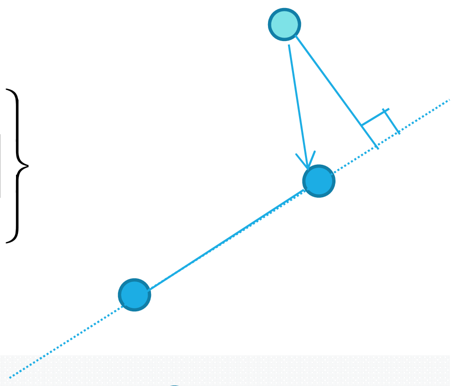
Point to point

$$d = \left|\mathbf{p}_0 - \mathbf{p}_1\right|$$



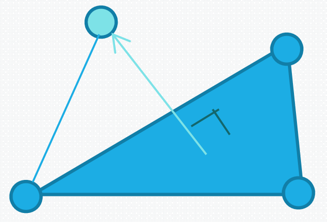
Point-Edge

$$d = \min\left\{ \frac{\left|(\mathbf{p}_2 - \mathbf{p}_1)\times(\mathbf{p}_0 - \mathbf{p}_1)\right|}{\left|\mathbf{p}_2 - \mathbf{p}_1\right|},\quad \left|\mathbf{p}_0 - \mathbf{p}_1\right|,\quad \left|\mathbf{p}_0 - \mathbf{p}_2\right| \right\}$$



Point –face

$$d = \min\left\{ (\mathbf{p}_o - \mathbf{p}_1).\widehat{\mathbf{n}},\quad \mathbf{p}_o \ to\, e1, \mathbf{p}_o \ to\ e2, \ldots.\ \mathbf{p}_o \ to\ eN \right\}$$

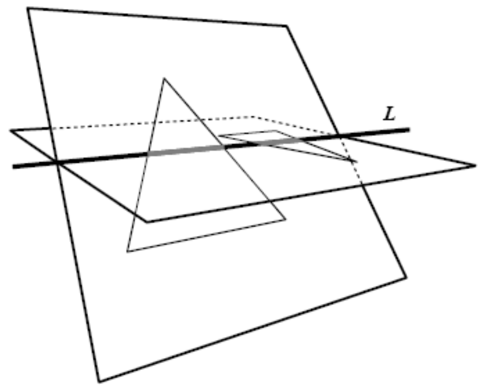


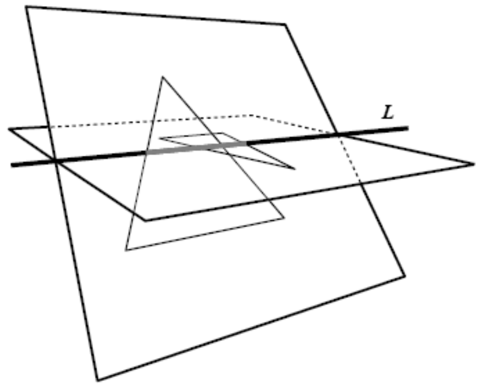
… Edge-edge, edge-face, face-face, object-object

The above is just to illustrate complexity of the operations and is not written for efficiency.

TRIANGLE INTERSECTION

1. Compute plane equation of triangle 2.
2. Reject as trivial if all points of triangle 1 are on same side.
3. Compute plane equation of triangle 1.
4. Reject as trivial if all points of triangle 2 are on same side.
5. Compute intersection line and project onto largest axis.
6. Compute the intervals for each triangle.
7. Intersect the intervals.





From – "A Fast Triangle Triangle Intersection Test" - Tomas Moeller. Available at: http://knight.cis.temple.edu/~lakaemper/courses/cis350_2004/etc/moeller_triangle.pdf
Code: http://fileadmin.cs.lth.se/cs/Personal/Tomas_Akenine-Moller/code/opttritri.txt

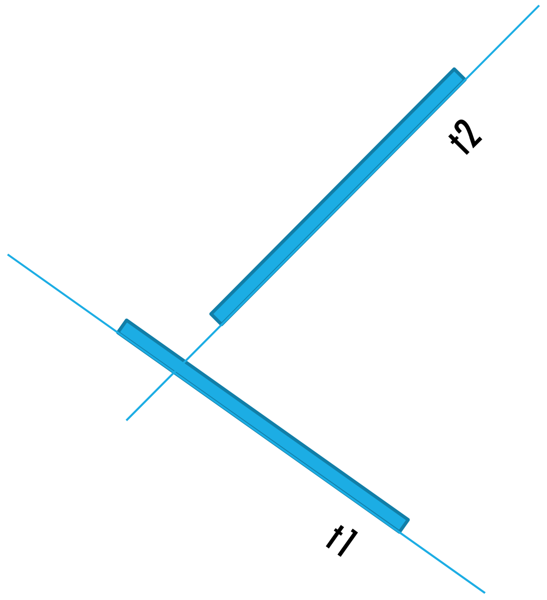
WHY DO WE NEED TO CHECK BOTH PLANES?

If a triangle is eliminated by step 2, then we're done.

But if NOT eliminated here it can still be eliminated by step 4.

e.g. In example on the right: all t1 verts are on both side of t2's plane

However they can be eliminated at step 4 as t2 verts are all on one side of t1.



TYPICAL SCENE

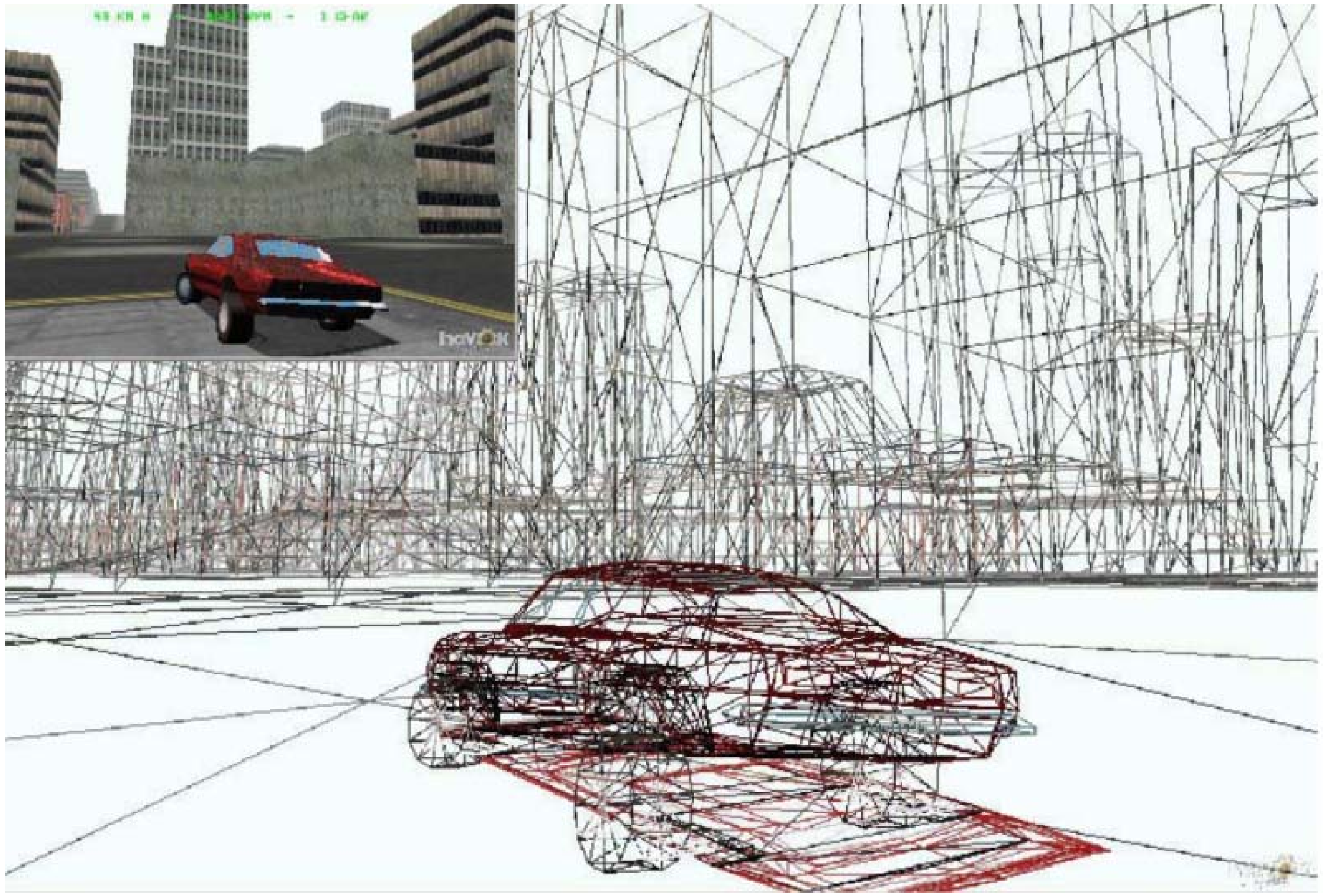
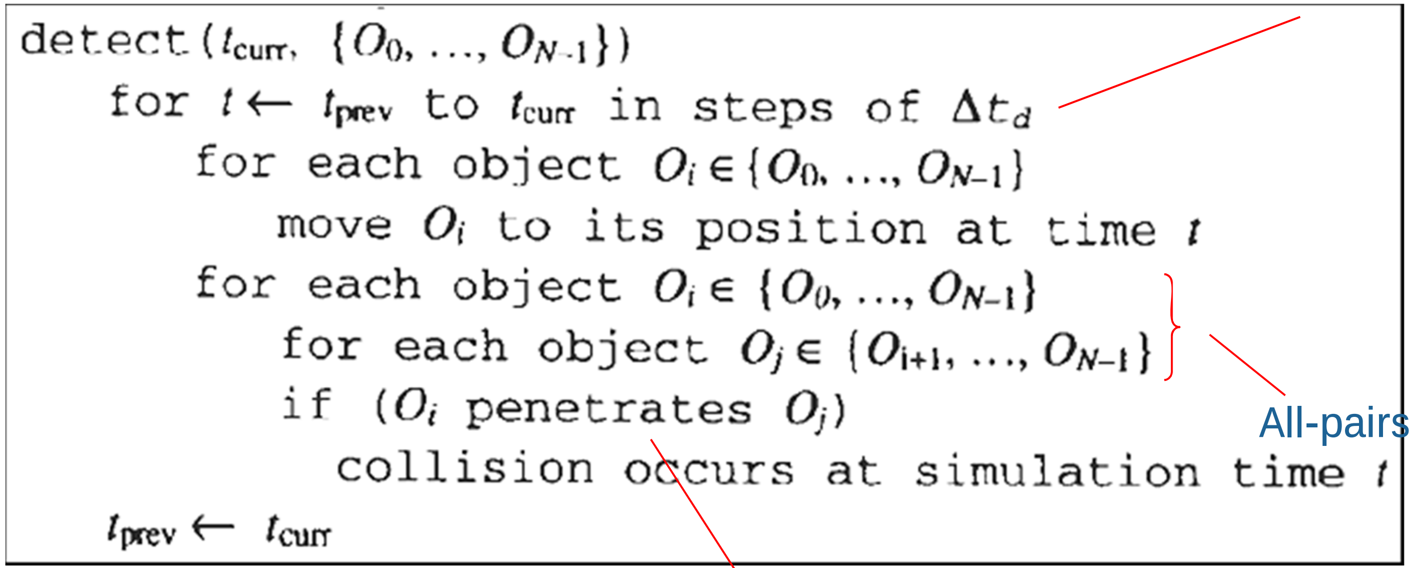


Image © Havok 2000

COLLISIONS

© Havok ~2005

NAIVE COLLISION DETECTION



```
detect($t_{curr}$, $\{O_0, \ldots, O_{N-1}\}$)
    for $t \leftarrow t_{prev}$ to $t_{curr}$ in steps of $\Delta t_d$
        for each object $O_i \in \{O_0, \ldots, O_{N-1}\}$
            move $O_i$ to its position at time $t$
        for each object $O_i \in \{O_0, \ldots, O_{N-1}\}$
            for each object $O_j \in \{O_{i+1}, \ldots, O_{N-1}\}$
            if ($O_i$ penetrates $O_j$)
                collision occurs at simulation time $t$
    $t_{prev} \leftarrow t_{curr}$
```

Fixed-timestep weakness

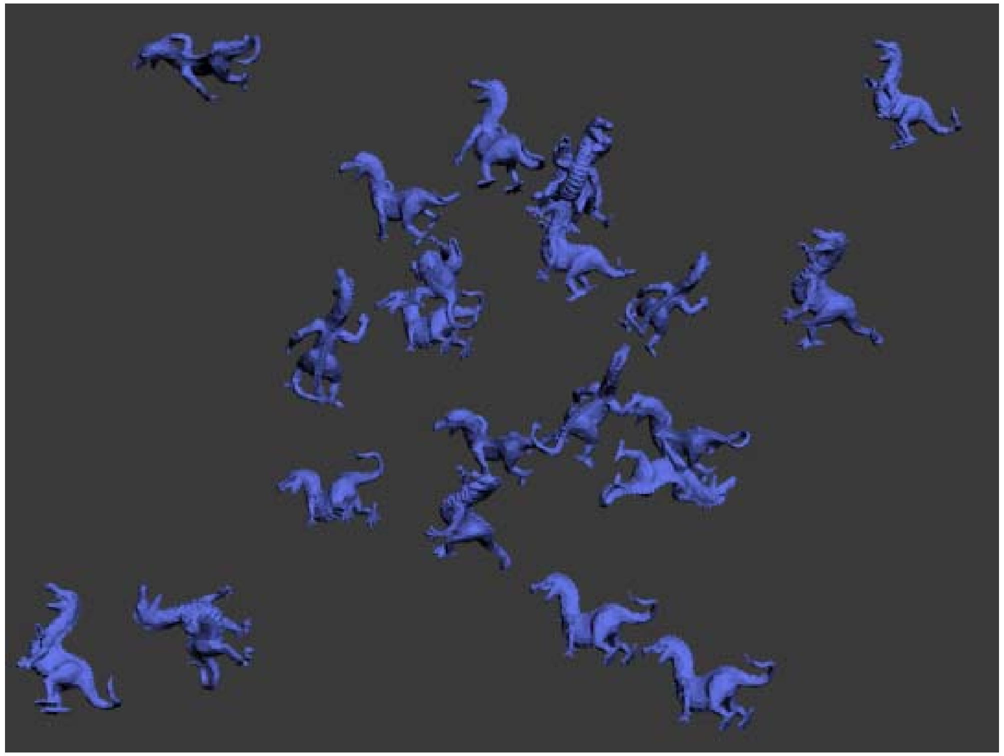
All-pairs weakness

Pair-processing weakness

[Hubbard 1993]

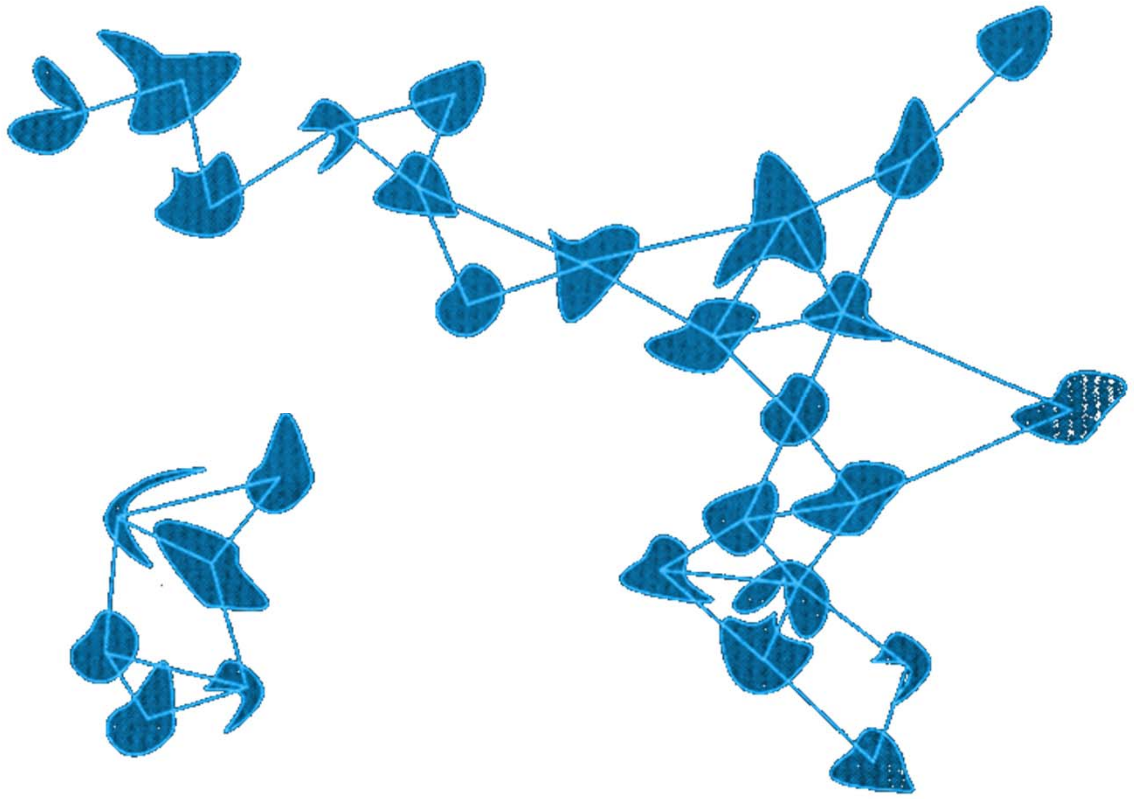
ALL PAIRS WEAKNESS

Pairwise $\frac{n(n-1)}{2} \approx O(n^2)$ **comparisons**



20 – objects = 190 pair comparisons

PAIR LOCALITY

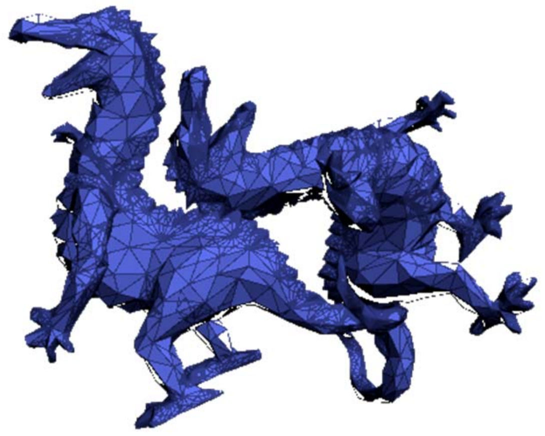


28 Objects = 378 pairs ~ 40 by locality

PAIR-PROCESSING WEAKNESS

Even for n=2, there are significant problems

2 objects (100 polygons) ~ 10000 polygon pairs to check

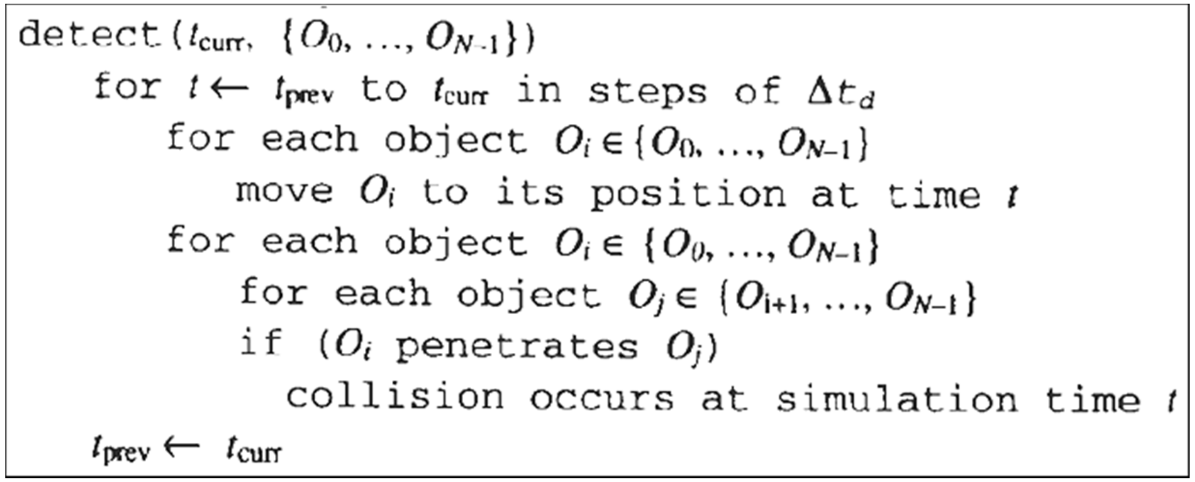


If we can exploit locality/spatial coherence, we shouldn't need to check every single primitive for a pair of objects

- A number of techniques exist to cull computations. E.g. separating axis, closest features
- More on these in upcoming lectures

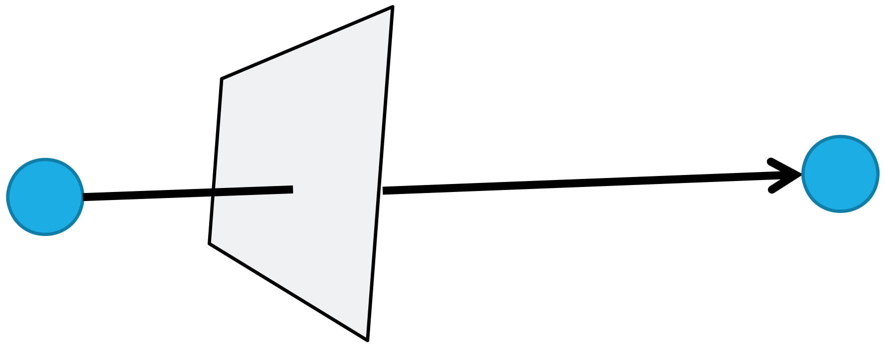
FIXED TIME-STEP WEAKNESS

Naive collision detection algorithm



```
detect($t_{curr}$, $\{O_0, \ldots, O_{N-1}\}$)
    for $t \leftarrow t_{prev}$ to $t_{curr}$ in steps of $\Delta t_d$
        for each object $O_i \in \{O_0, \ldots, O_{N-1}\}$
            move $O_i$ to its position at time $t$
        for each object $O_i \in \{O_0, \ldots, O_{N-1}\}$
            for each object $O_j \in \{O_{i+1}, \ldots, O_{N-1}\}$
            if ($O_i$ penetrates $O_j$)
               collision occurs at simulation time $t$
    $t_{prev} \leftarrow t_{curr}$
```

But if Δt is large you get tunnelling!



[Hubbard 1993]

FIXED TIME-STEP WEAKNESS

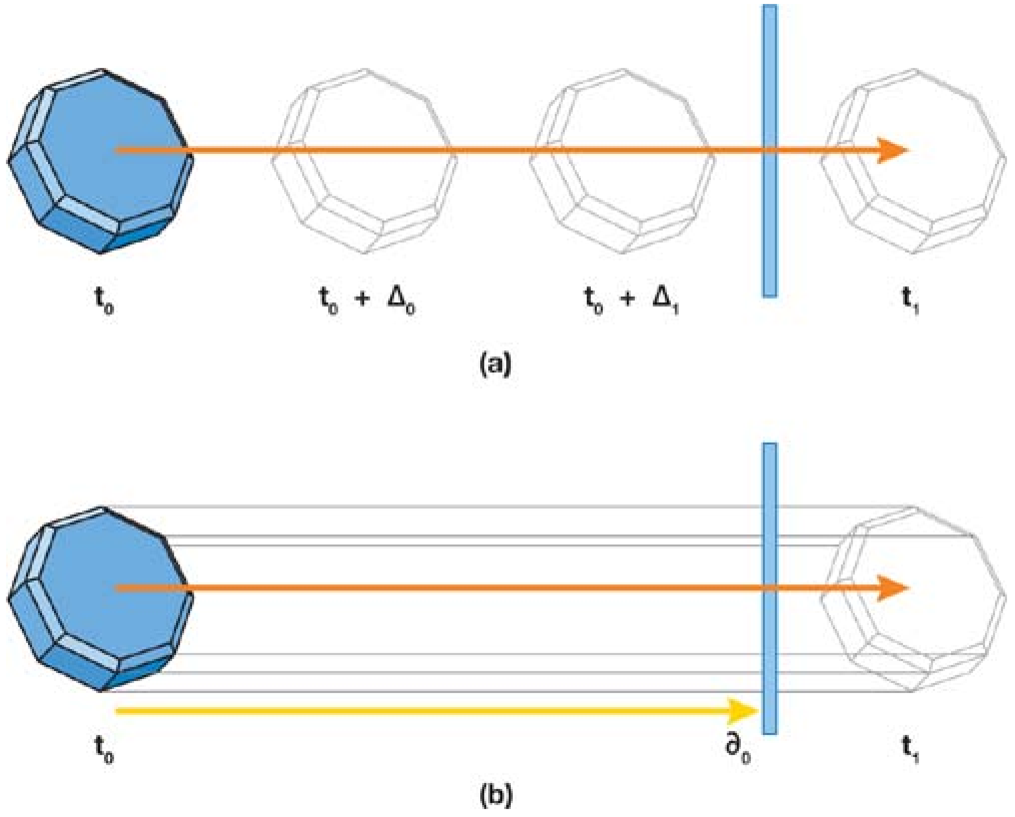
Position and orientation of body is updated at each time-step

- Interactions are only handled at these discrete time-steps

Simulation time is incremented by fixed amount Δt

- Large Δt **reduces accuracy** of collision detection
- Smaller Δt **reduces efficiency** of simulation

CONTINUOUS COLLISION DETECTION

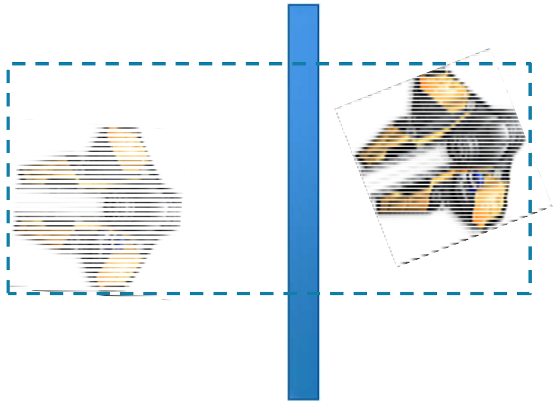


http://http.developer.nvidia.com/GPUGems3/gpugems3_ch33.html

TWO-PHASE COLLISION DETECTION

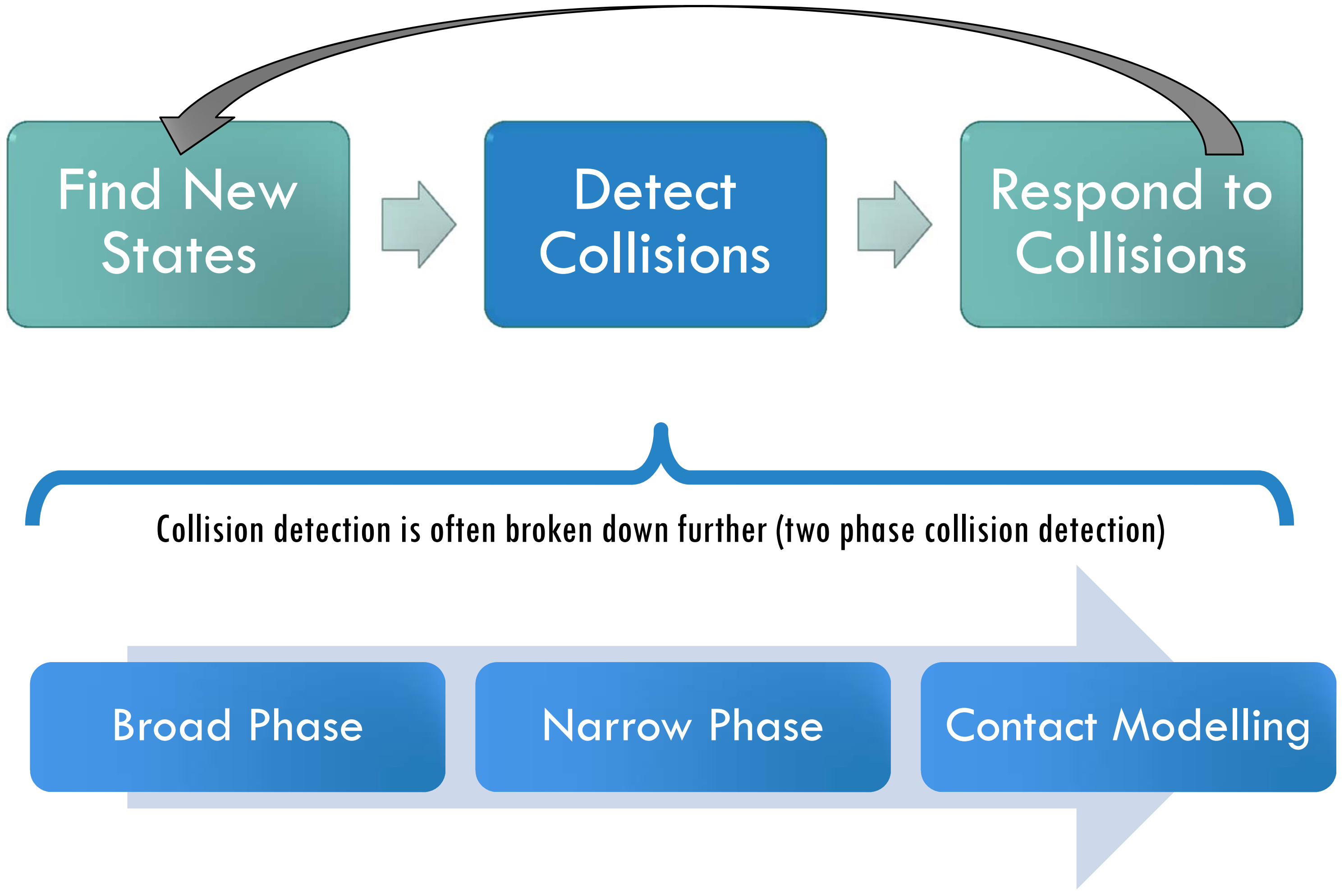
Addressing the *weaknesses*

- **Broad phase** : efficiently cull pairs whose bounding boxes don't overlap. *"Sweep and Prune" exploits inter-frame* coherence to achieve near O(n) performance [Cohen et al.1995]
- **Narrow phase** : utilise spatial localisation techniques to narrow in on the areas of objects which are in contact and use the underlying polygon model to determine the actual points of contact



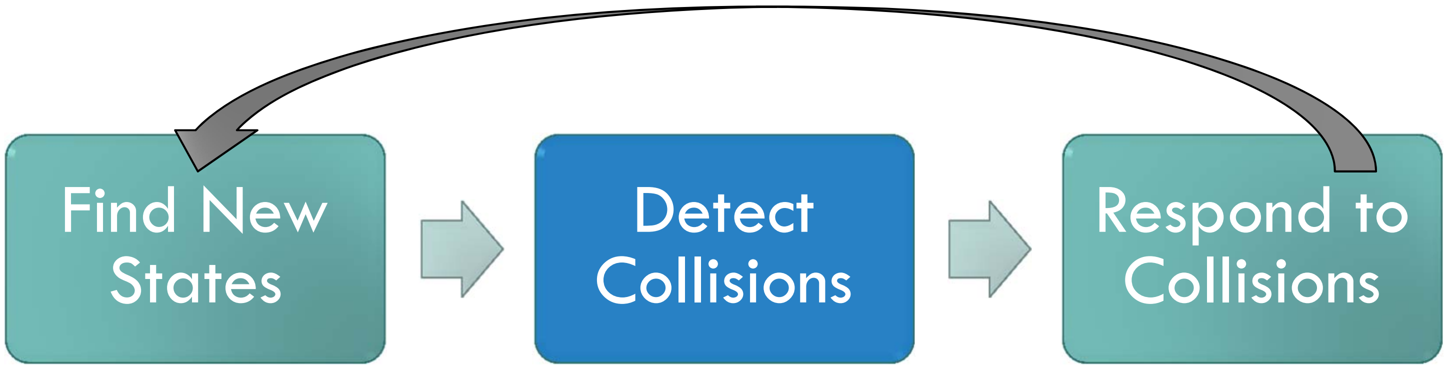
[Cohen et al. 1995] Cohen, Jonathan D.; Lin, Ming C.; Manocha; Ponamgi, Madhav K. (April 9–12, 1995), I–COLLIDE: an Interactive and Exact Collision Detection System for Large Scale Environments), Proceedings of the 1995 Symposium on Interactive 3D Graphics

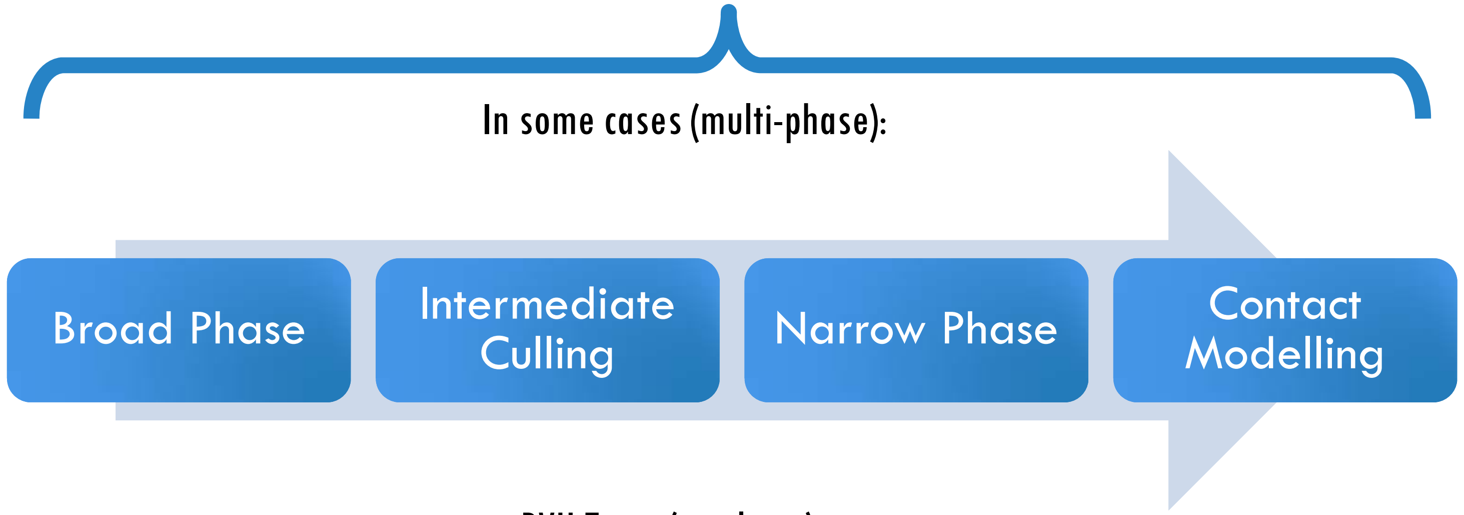
BASIC SIMULATION LOOP



Collision detection is often broken down further (two phase collision detection)

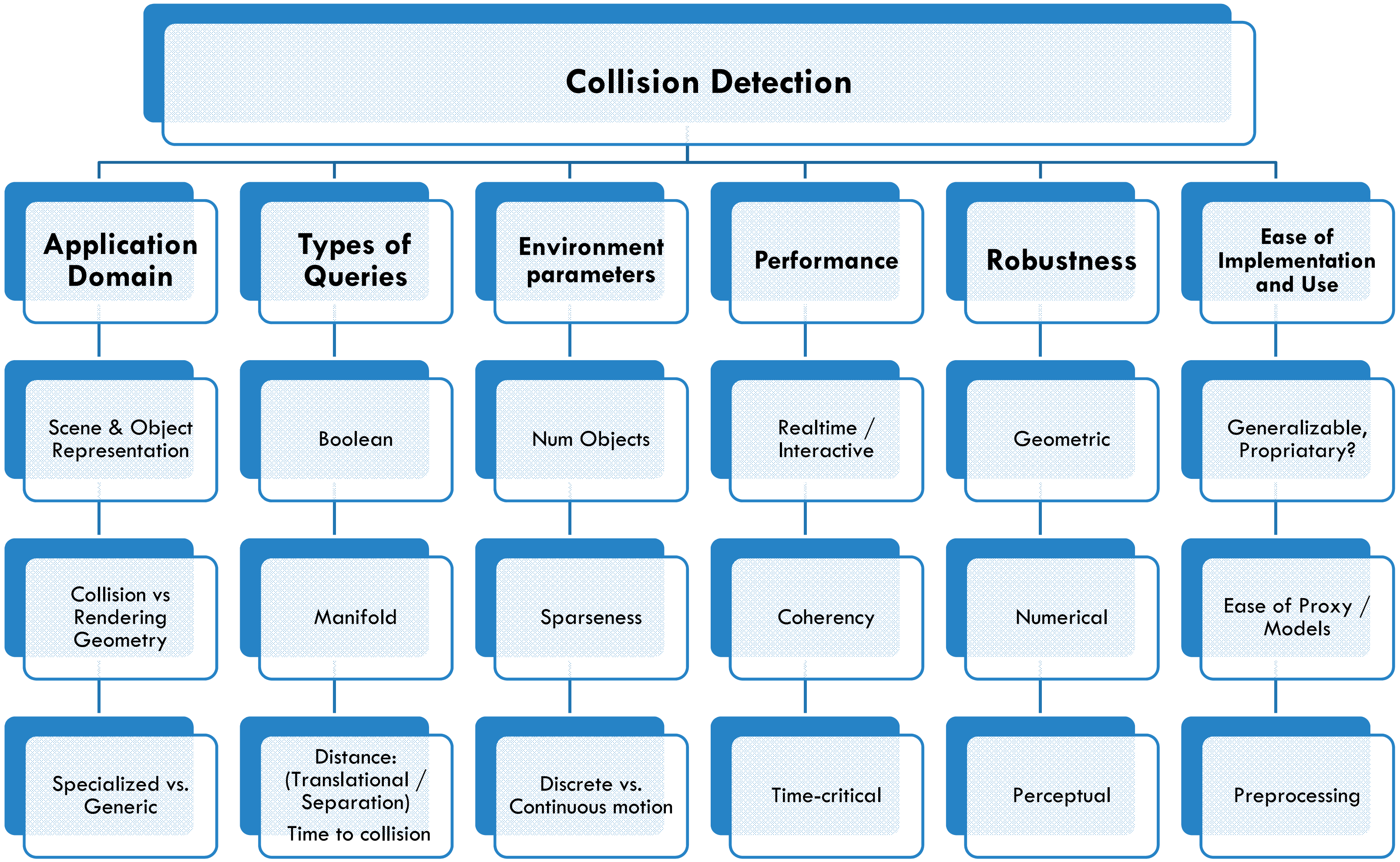
BASIC SIMULATION LOOP





e.g. BVH Trees (see later)

ALGORITHM DESIGN ISSUES



CHALLENGES

Physics data complexity

- Multivariate, multi-dimensional
- Time-variant / transient
- Special cases / singularities

But require: Efficiency, Plausibility, Fidelity, Robustness … + relative ease of implementation

Tips for Debugging a Collision Detection System [Erricson]

- Keep a buffer of the last n collision queries: output data for further analysis
- Provide a means to visualise collision geometry
- Implement a simple reference algorithm: e.g. Brute force variant to isolate cause
- Maintain a test suite of queries run code against different tests

REFERENCES IN THIS LECTURE

[Erricson05] Real-Time Collision Detection - By Christer Ericson. Elsevier, 2005.

[Erleben05] Physics Based Animation (Graphics Series) – By K. Erleben,J. Sporring, K. Henriksen, H. Dohlmann. Charles River Media, 2005.

[Hubbard93] Interactive Collision Detection – By Philip Hubbard, in Proceedings of IEEE Symposium on Research Frontiers in Virtual Reality, 1993