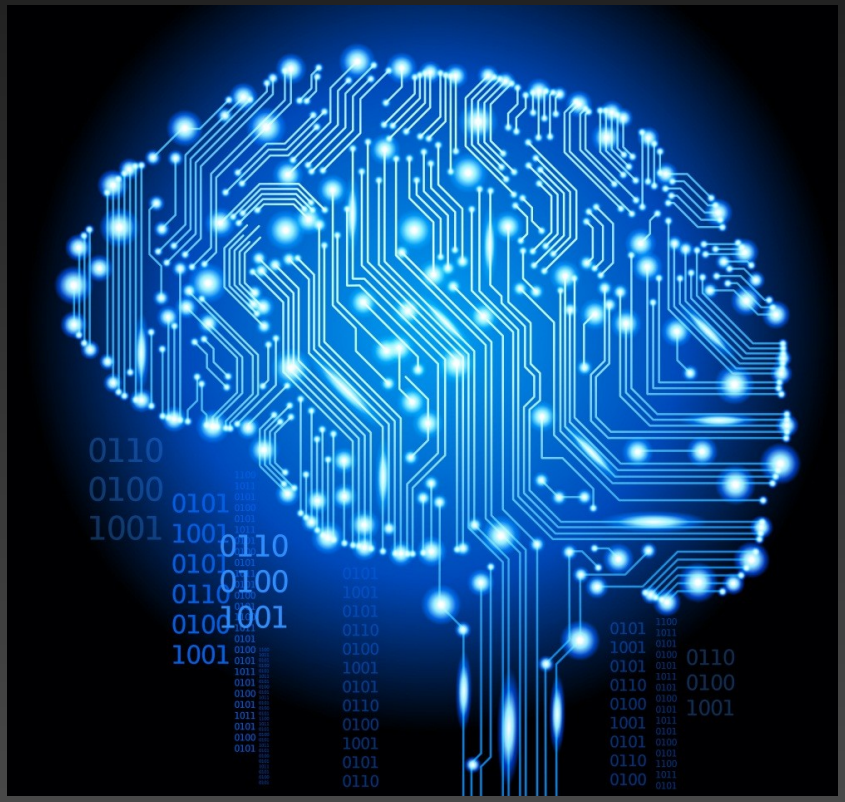
AI



Searching in space

Collision detection

Searching in space

Applications

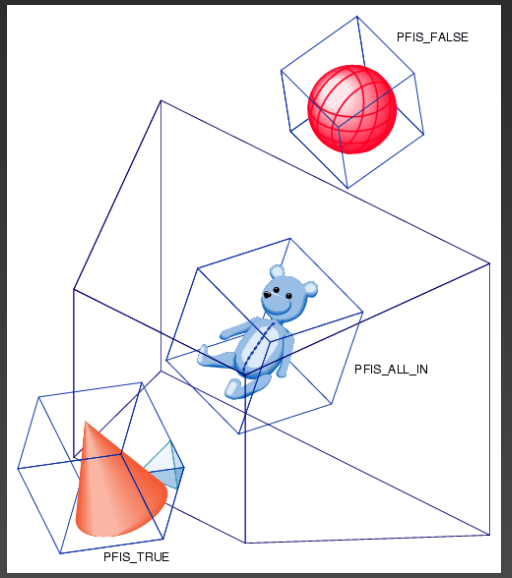
- Computer graphics
  - speed-up of 2D and 3D algorithms (ray-tracing, intersection search…)
- Image analysis and processing, recognition
- Geographic information systems (GIS)
  - point, line and planar objects
  - database can handle millions of objects
- Industry, CAD …
  - part layout, collision detection
- Robot trajectory planning
  - shortest/fastest path search
- Databases
  - multidimensional data search

Elementary tasks

- Localisation of point in 2-3D space
  - find object, that contains analyzed point
- Find closest n points from center
  - global variant: find two points closest to each other
- Object processing by distance from center
- Intersection search in set of lines (curves)
- Find closest intersection with 3D scene

Elementary tasks

- Interval search in 2-3D space
- Windowing problem
  - clipping by predefined polygon window
- Set operations on map objects
  - areas, paths, points
- Collision testing and minimal object distance



Example

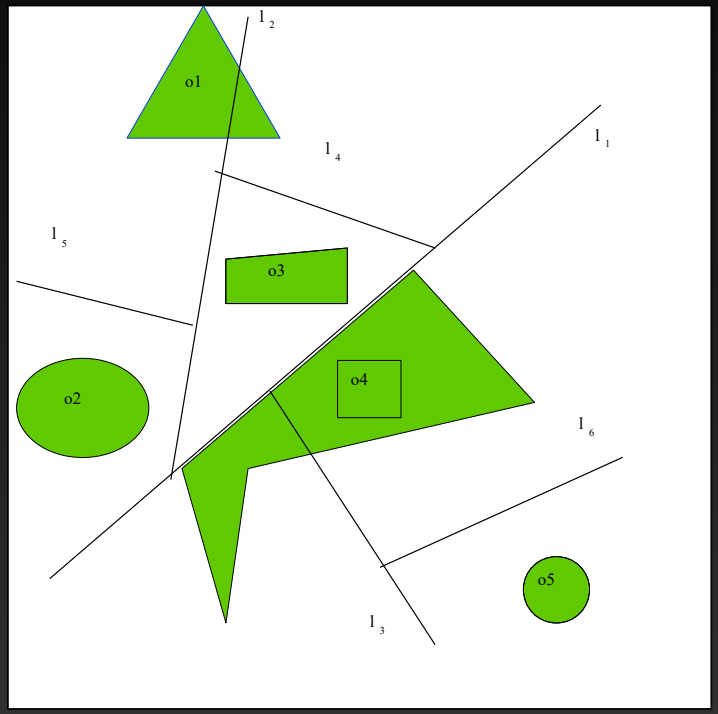
https://media.giphy.com/media/xUPGcgiYkD2EQ8jc5O/source.gif

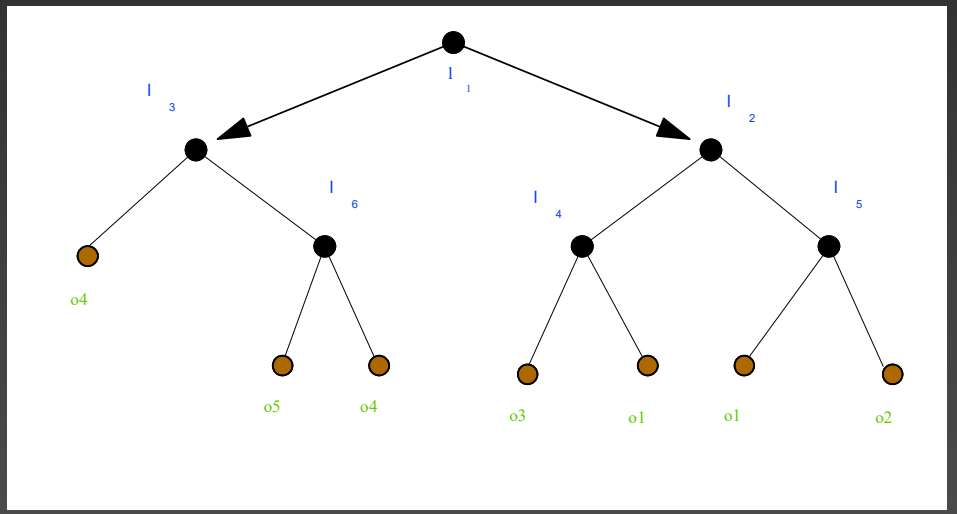
Binary Space Partition (BSP) trees

- BSP is binary tree used to general sorting of scene objects in n-dimensional space
- Whole scene is recursively divided by hyperplane
- Leaves of BSP tree corresponds to convex space partitioning. Subspace defined by leave contains maximally one object (or its part).

BSP tree

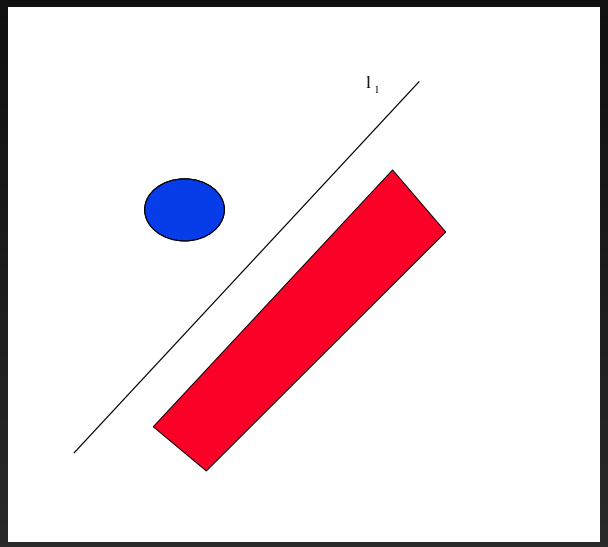
- partitioning of space to convex parts
- recursive dividing by arbitrary hyperplane
- objects (parts of obj.) stored in leaves

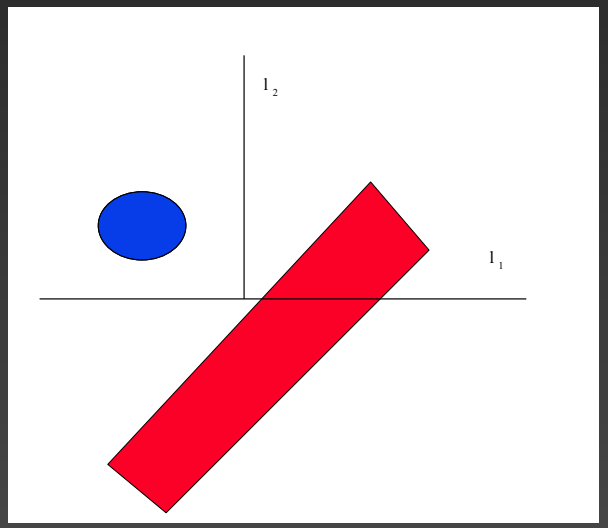




Creating of BSP tree

- Hyperplanes oriented
  - in any direction
  - by axes



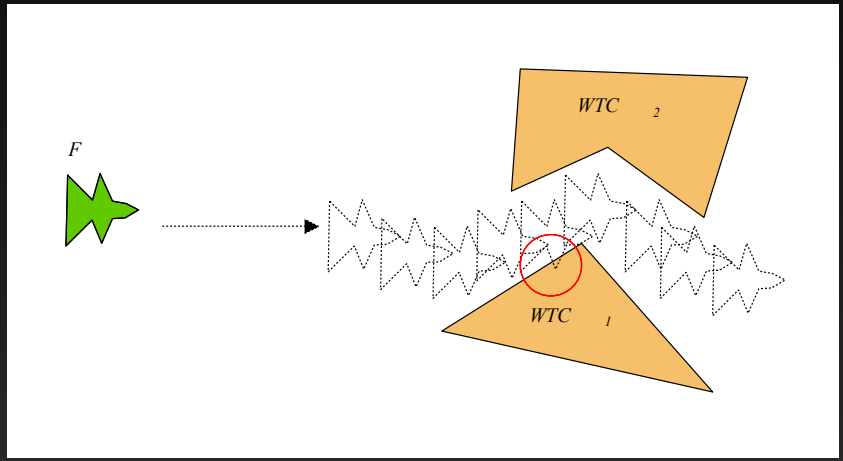


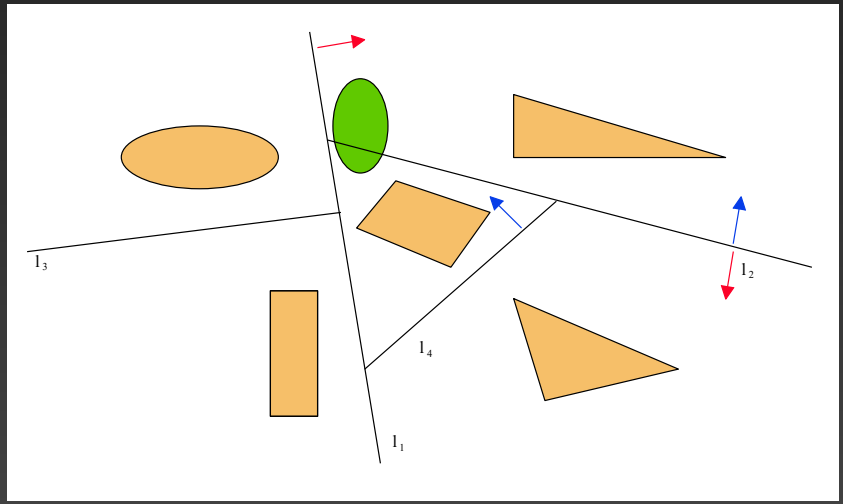
Possible applications of BSP trees

- Painter's algorithm
  - draw from back to front
    - walk though tree, each leave one object
    - compare object distance
- Raytracing speed-up
  - find, which plane is intersected
  - walk through tree to object, and solve reflection
- Collision detection
  - find leave(s), that contains current moving object
  - solve collision of single object from scene and moving object

Collision detection using BSP tree

```
fce Detect_collision(BSP);
Begin
    if BSP.list=True
    then
        Test object F and object stored in leave
    else begin
        if object F interfere in left subtree region
        then
            Detect_collision(BSP.left_subtree);
        if object F interfere in right subtree region
        then
            Detect_collision(BSP.right_subtree);
    end;
end.
```



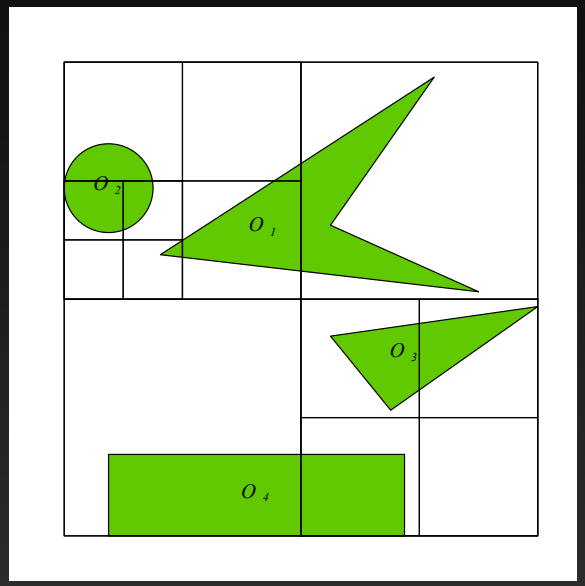


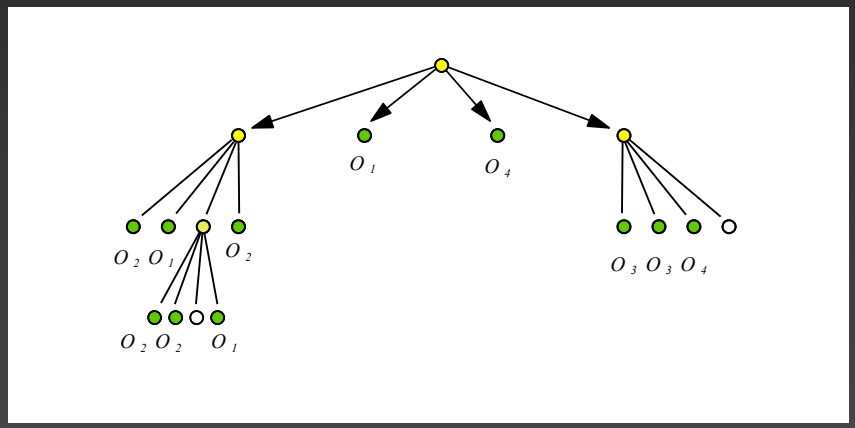
Quad tree, K-D tree, Oct tree

- Special type of BSP tree
- Scene partitioning in orthogonal directions
- Easier construction nad evaluation
- 2D
  - Quad
    - each step divides in x, y → 4 branches (quad)
  - K-D
    - binary, alternating division by x, y, x, y...
- 3D
  - Oct – divides by x, y, z → 8 branches

Quad tree

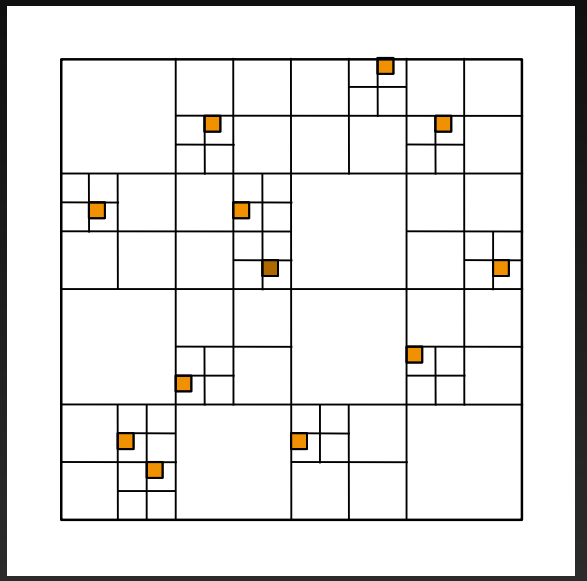
- Divide exactly in half
- Object information
  - in leaves
  - in any depth
  - max 1 obj./leave

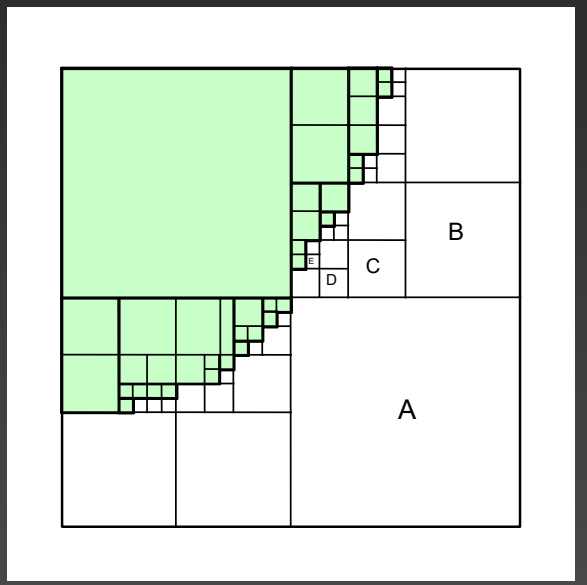




Variants of quad tree

- MX quad tree
  - (matrix q. t.)
  - all objects in same depth
- Region quad tree
  - for areas
  - divide in half
  - information only in leaves

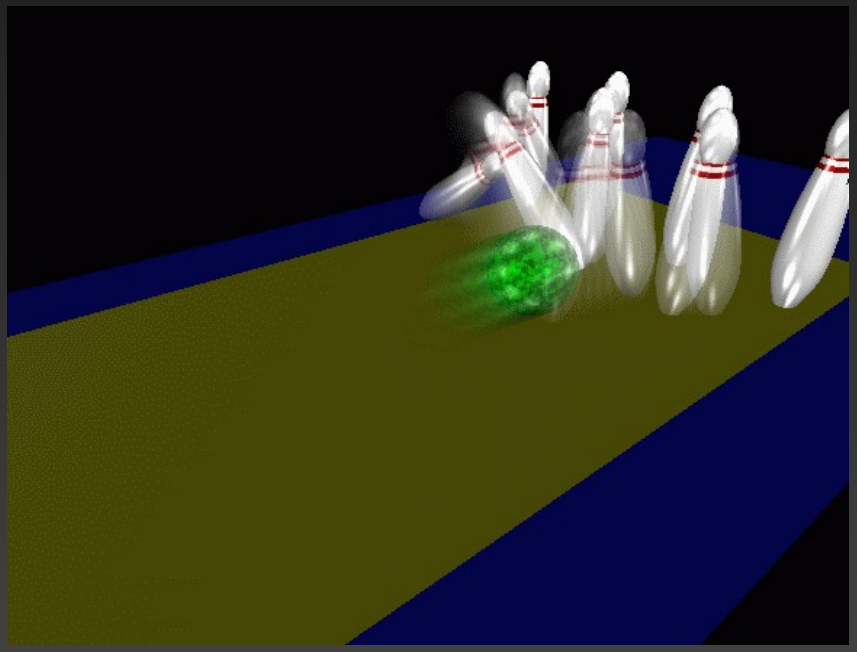




Other options

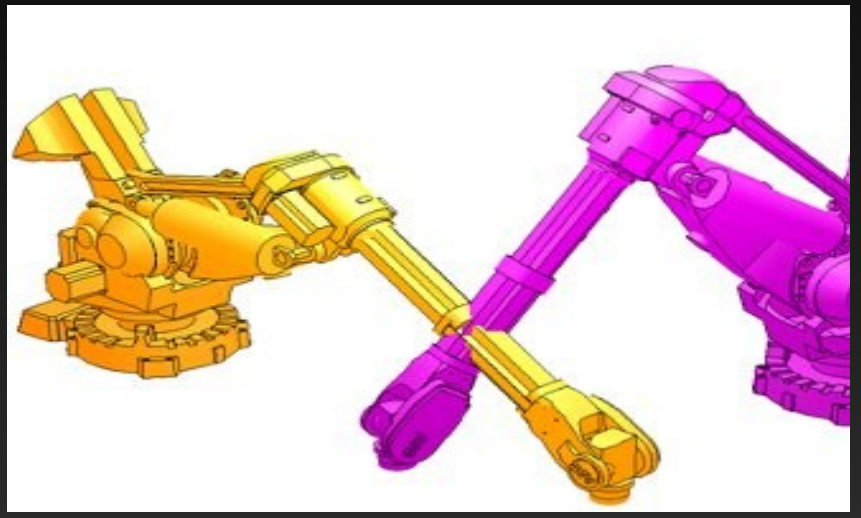
- Point region quad tree
  - for point objects
- Bucket point region quad tree
  - more than one point in region
- Interval search
  - using K-D tree, or 1D interval tree
  - interval in more dimensions – hierarchy of 1D trees
- Partition tree
  - divide points into triangle classes
- Interval trees
  - for fast search of line segments intersected by straight line
- Segment trees
- …

Collision detection



Application

- Virtual reality
- Simulators
- Games
- Industry
- Construction
  - Is it possible to assembly ?
- Scientific applications
  - protein bonding, …

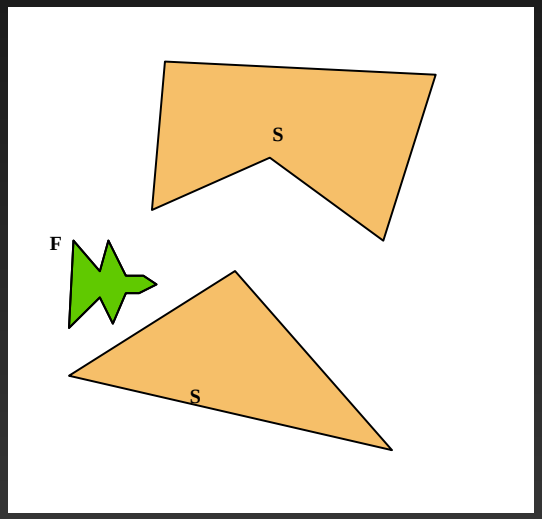


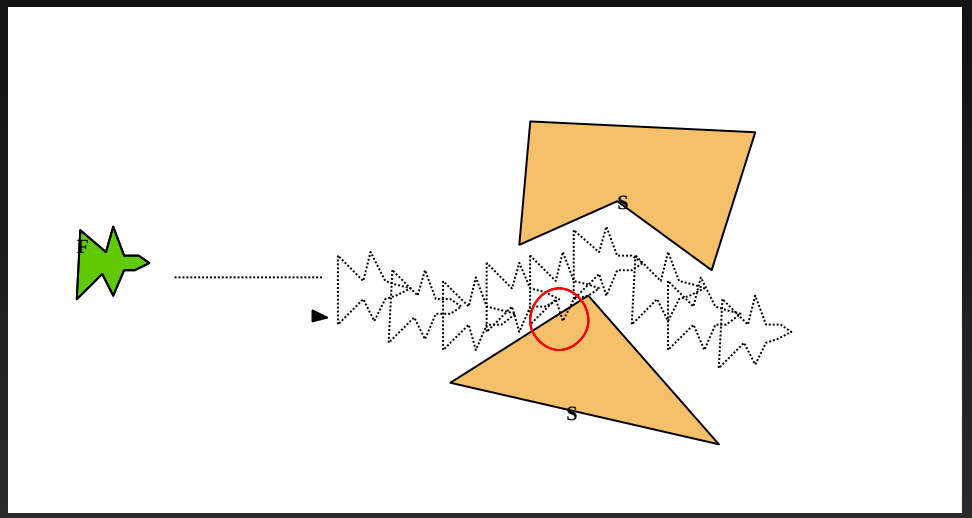
Classification

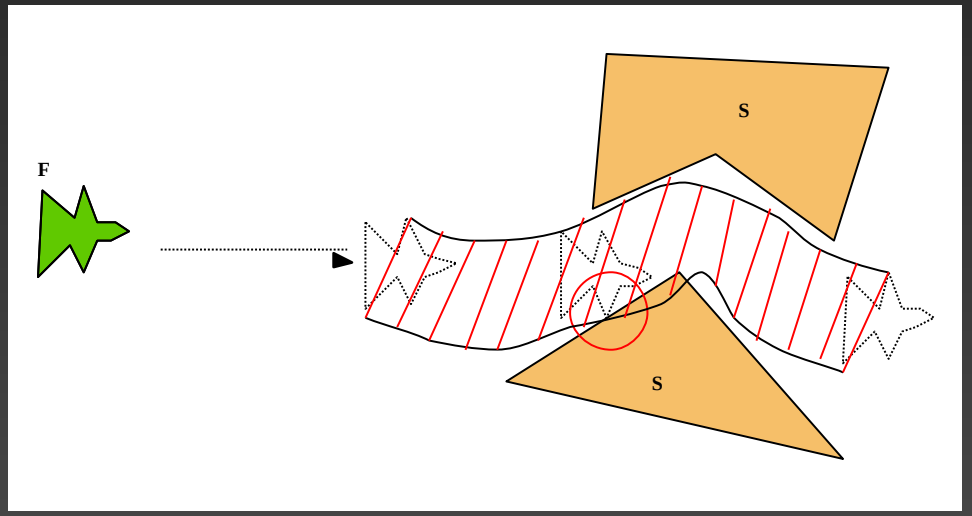
- Scene S, moving object F
  - did F intersected S in any moment?

- Static collision detection
  - Perform one test in exactly one time point **t**, i.e. in one space point and object orientation.
- Pseudo-dynamic collision detection
  - Perform tests on discrete set of positions of F corresponding to trajectory.
- Dynamic collision detection
  - Test volume created by dragging F through scene intersects S.

Examples







Collision detection methods

- Bounding volumes hierarchy
  - Approximate object by simple bounding volumes (sphere, AABB, OBB, FDH)
  - Create hierarchy
  - Refine approximation by walking through hierarchy
- Space partitioning
  - Create space partitioning (eg. BSP)
  - Recursively find all leaves, that intersects object F
  - Find if F really intersects object in leaves

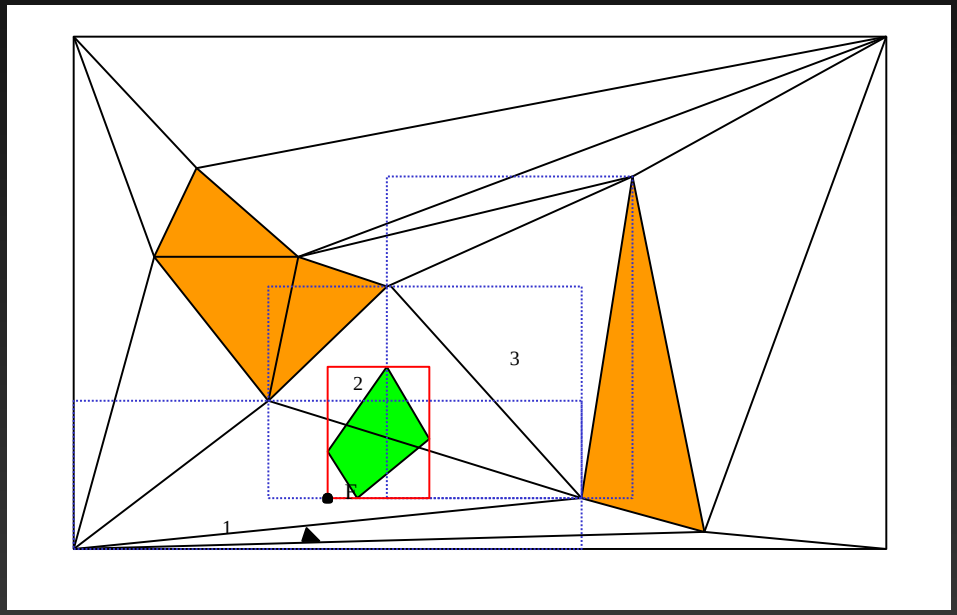
Space partitioning

1. Space is broken down to parts with similar structure
2. Collision detection is performed on these parts and refined locally

Object does not collide if any of its triangles does not intersect with scene (with exception of F completely inside S or vice versa).

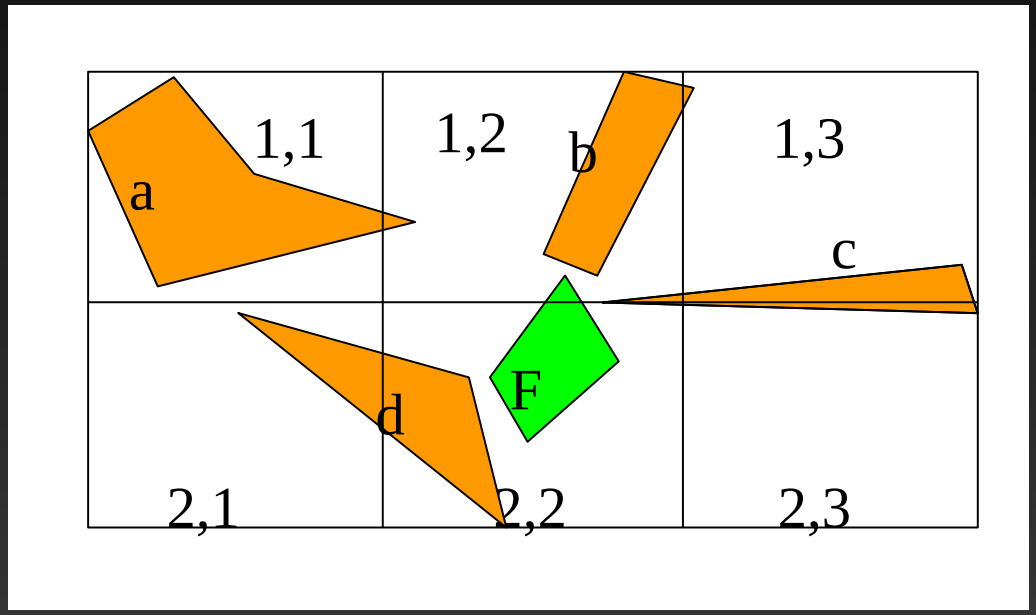
Tetrahedrons

- Triangles in 2D
- Method
  - find intersected volumes
  - for each volume test all triangles of F with all triangles of S



Voxel grid

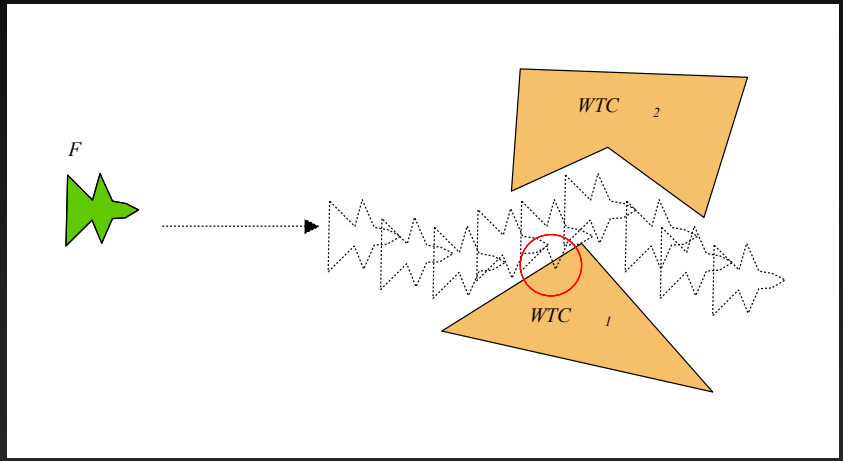
- 1,1 -> a
- 1,2 -> a, b, c
- 1,3 -> c
- 2,1 -> d
- 2,2 -> c, d
- 2,3 -> c

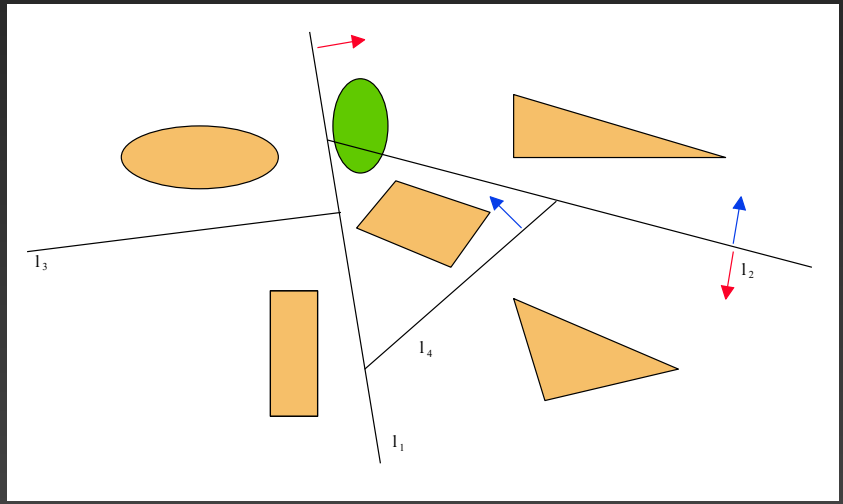


- Method
  - Find volume for F
  - Test collision for all triangles in volume

Collision detection using BSP tree

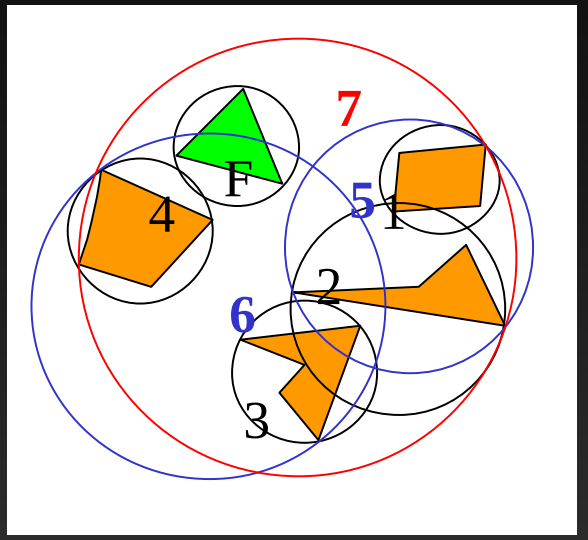
```
fce Detect_collision(BSP);
Begin
    if BSP.list=True
    then
        Test object F and object stored in leave
    else begin
        if object F interfere in left subtree region
        then
            Detect_collision(BSP.left_subtree);
        if object F interfere in right subtree region
        then
            Detect_collision(BSP.right_subtree);
    end;
end.
```

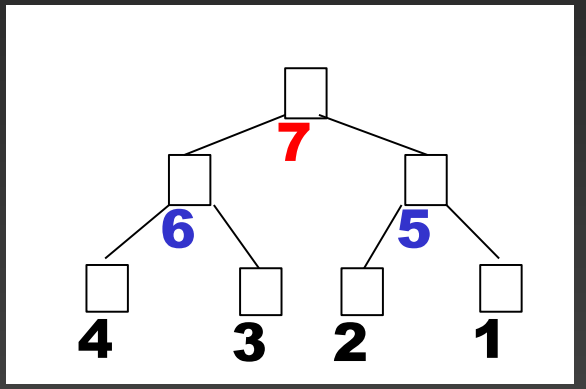




Bounding volumes hierarchy

- Tree of decreasing volumes
- Basic problems
  - Choice of bounding volume
    - should envelope object as tightly as possible
    - testing should be fast
  - Tree
    - construction should be as fast as possible
    - search should be as fast as possible



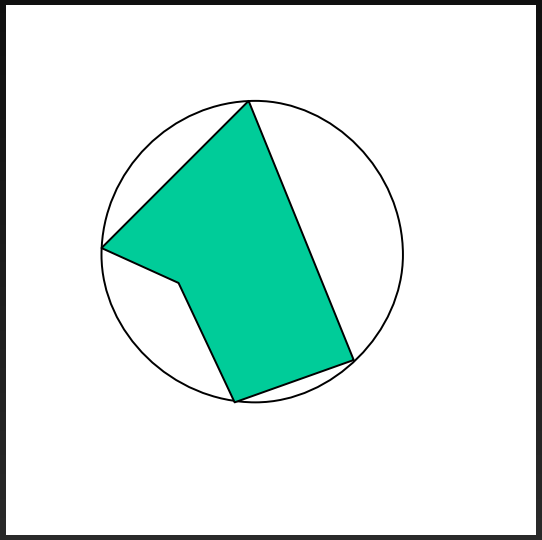


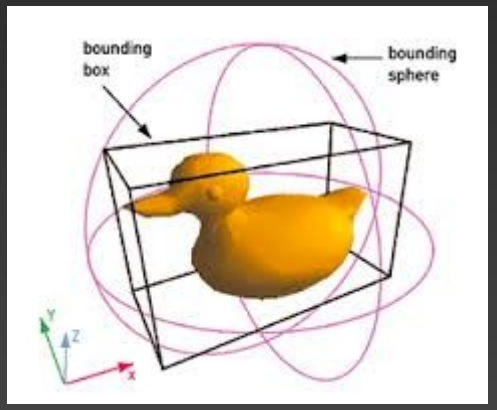
Bounding volumes

- Sphere
- Axis Aligned Bounding Box (AABB)
- Oriented Bounding Box (OBB)
- k-DOP (FDH) k-Discrete Orientation Polytopes (Fixed Dimensional Hull)
- …

Sphere (cylinder)

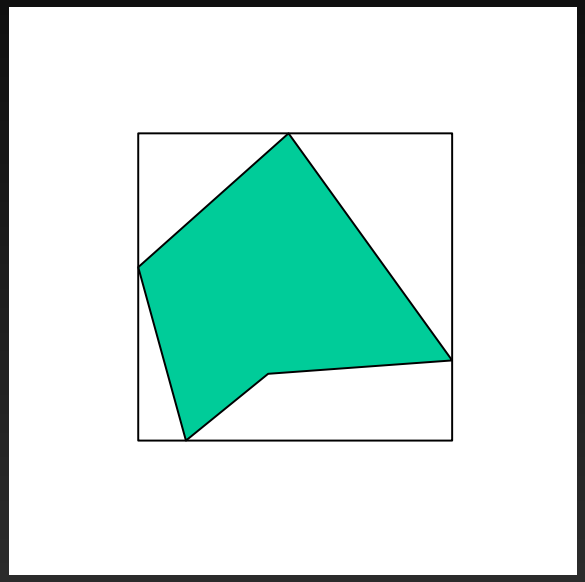
- Most simple
- Fastest test
  - $r_1 + r_2 <$ distance
- Insensitive to orientation
- Most wasted space
  - intersected → you must compare triangles!
- Complicated creation
  - slow → precompute

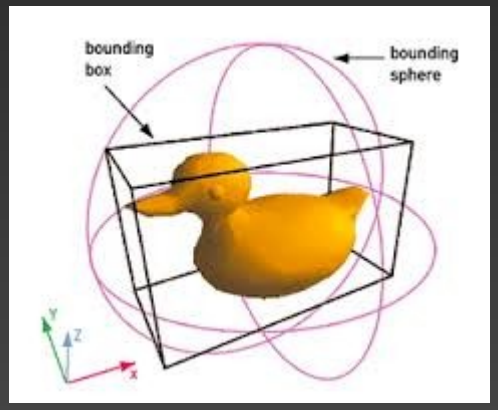




AABB

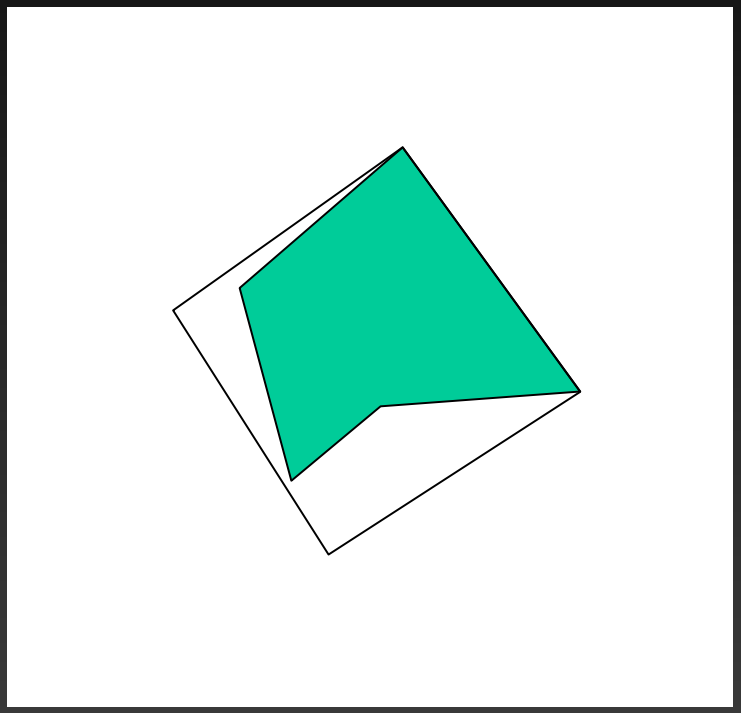
- Very simple
  - Test only sides of box
  - 6 comparisons
- Not very sensitive to shape or orientation
- Fast creation
  - min & max of x, y, z of all vertices





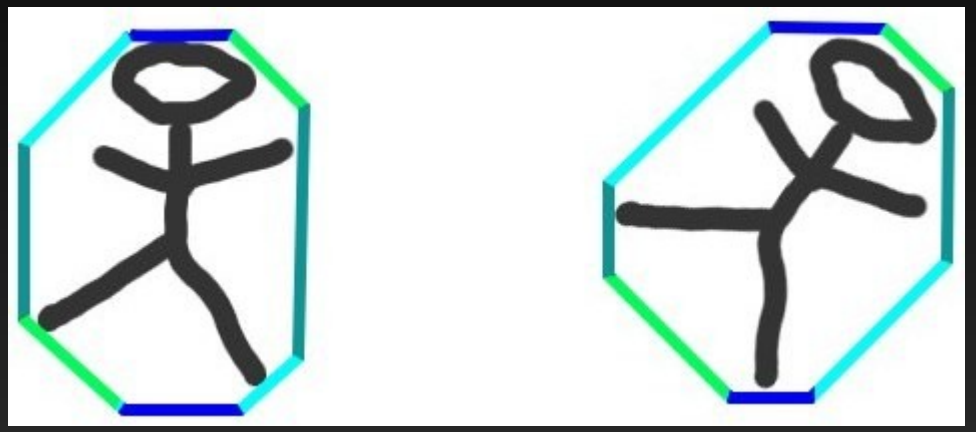
OBB

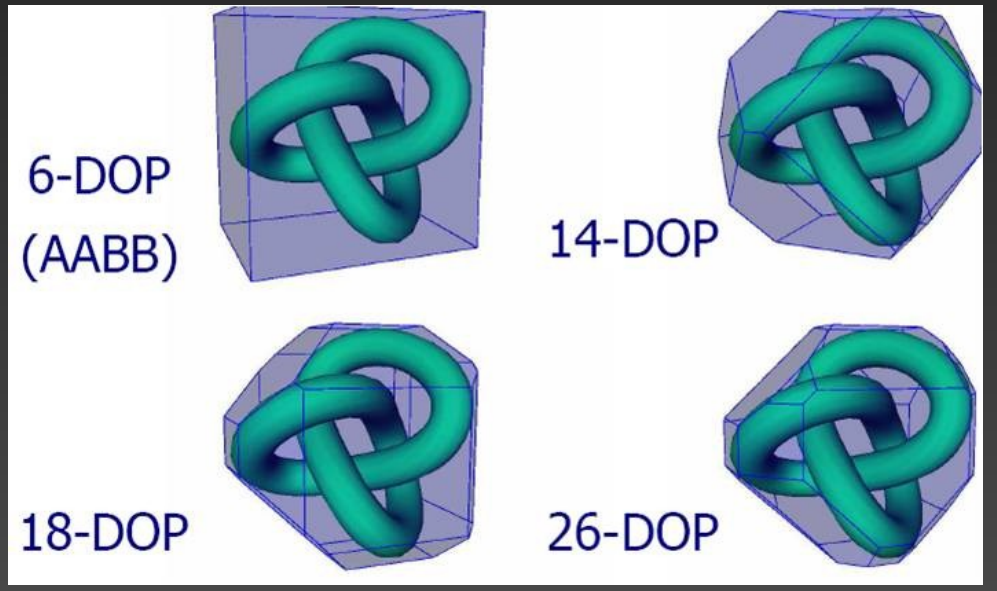
- Middle complexity
- Test is fast
  - 15 operations
    - projection to x, y, z axes and compare intervals
- Tighter envelope than BS or AABB
- Complicated creation



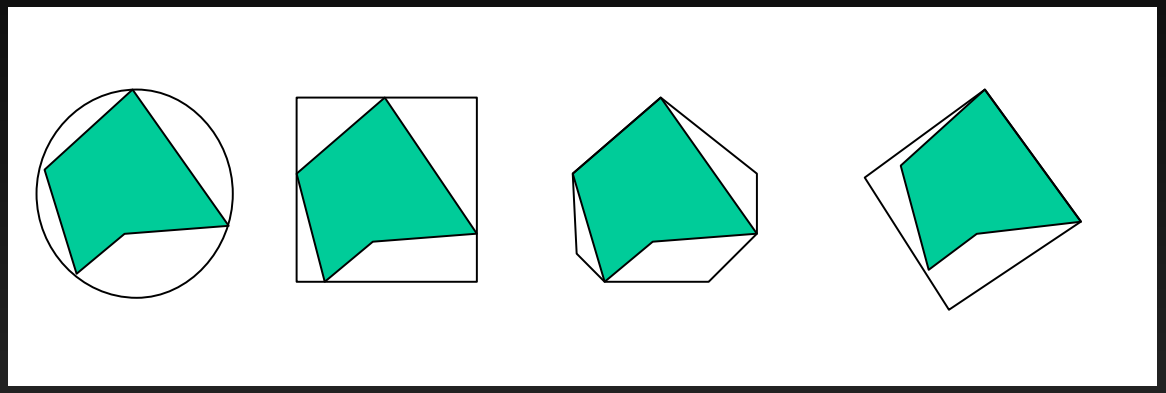
k-DOP

- Discrete Oriented Polytope
- Fast enough test
  - testing only parallel planes, distance of general planes
- Tighter than AABB
- Fast enough construction
  - min & max of projection



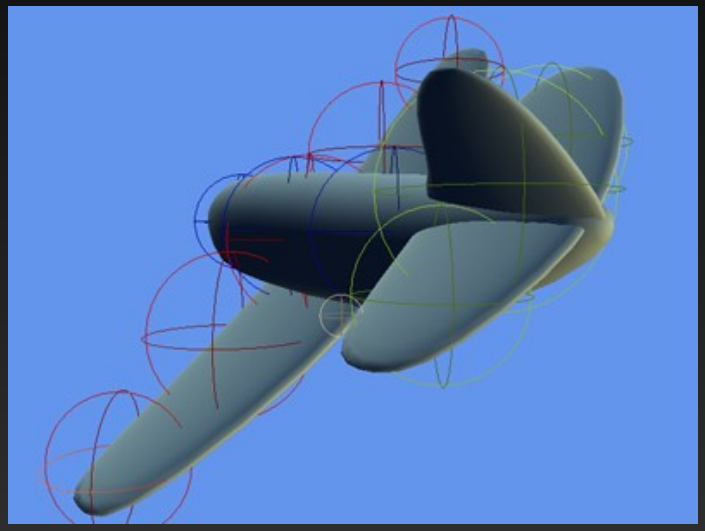


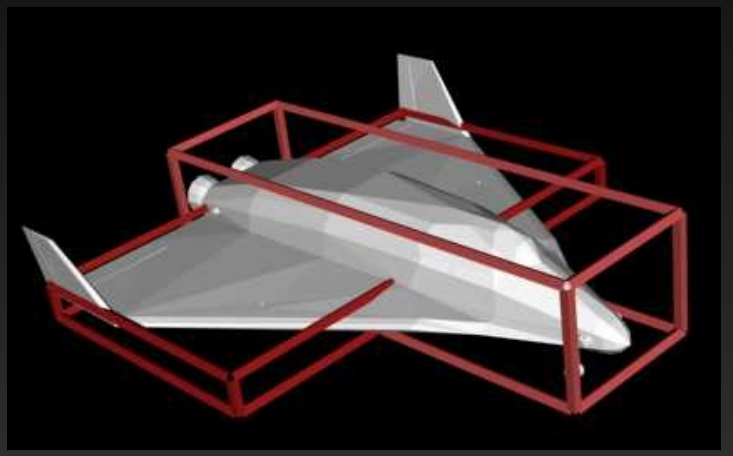
Comparison

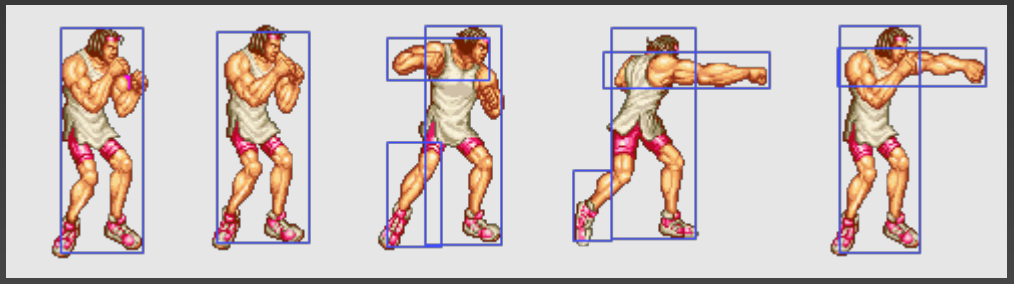


- Sphere, AABB = fast test + worse object approx.
  - easy to program and understand
- k-DOP + OBB = slower test + better approx.
  - better results

Examples







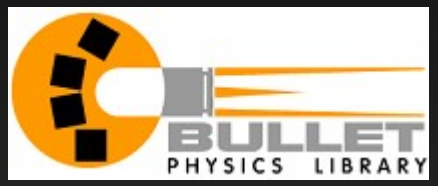
Terrain collision – height search

- Much more simple
  - [x,y] coords of F are known
  - terrain is regular grid with z displacement

    → find z
  - find related terrain polygon (quad, triangle) by x,y
  - interpolate z inside polygon from [x,y,z] of vertices

Application & Usage

- Mostly part of libraries

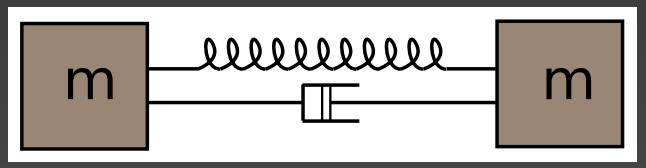


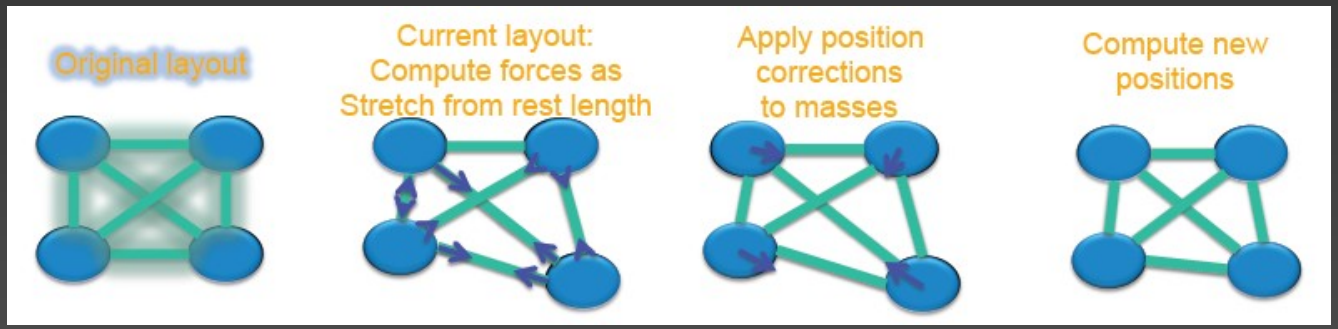
- Simulations and collisions
  - rigid body (various bounding volumes, destruction)
    - easier, theoretical (everything is softbody in reality)
  - soft body (cloth, ropes, deformation)
    - allows deformation, compression etc.
  - fluids
  - artificial inteligence (path search, optimization)

Principle of Soft Body Simulation
One of possible models

- Particle-Spring-Mass model
  - stores position **x**, velocity **v**, force **F**, mass **m**
- Euler integration
  - Every frame for every point {

```
F = 0;
F += gravity (-9,8 * m), spring forces (next slide), …
v += F * delta_t / m
x += v* delta_t
}
```





Soft Body Simulation

- Spring forces
  - Spring: start + end point (A, B), stiffness k_s, rest length L_0, damping factor k_d

Hook law

$$F_s = k_s \cdot (|B - A| - L_0)$$

Damping = dot product of position vector and relative speed; (result force is in opposite direction to movement)

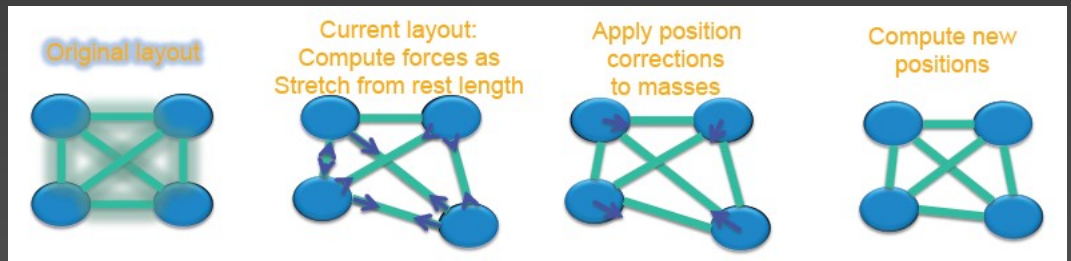
$$F_d = \left(\frac{B-A}{|B-A|}\right) \cdot (\boldsymbol{v_b} - \boldsymbol{v_a}) \cdot k_d$$

Total spring force $F_t = F_s + F_d$

$$F_A = F_t \cdot \left(\frac{B-A}{|B-A|}\right)$$

$$F_B = F_t \cdot \left(\frac{A-B}{|A-B|}\right)$$

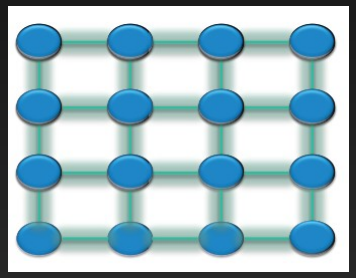
positive force = towards



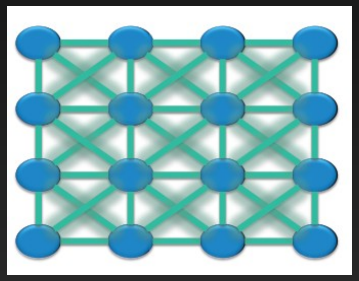
Soft Body Simulation

- Connected (usually) by 3 types of springs

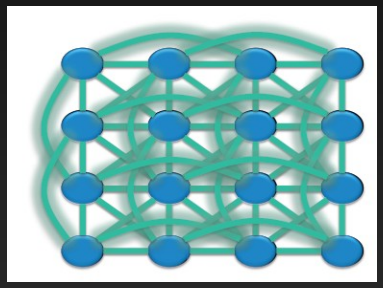
structure



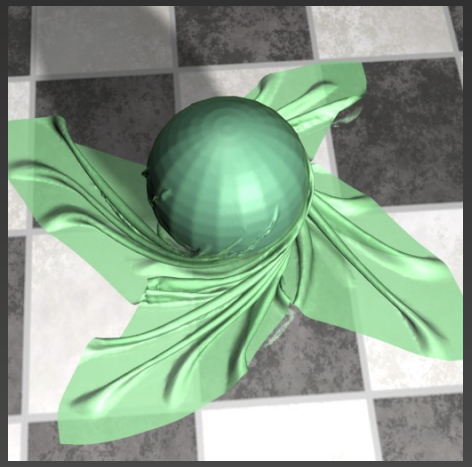
shear

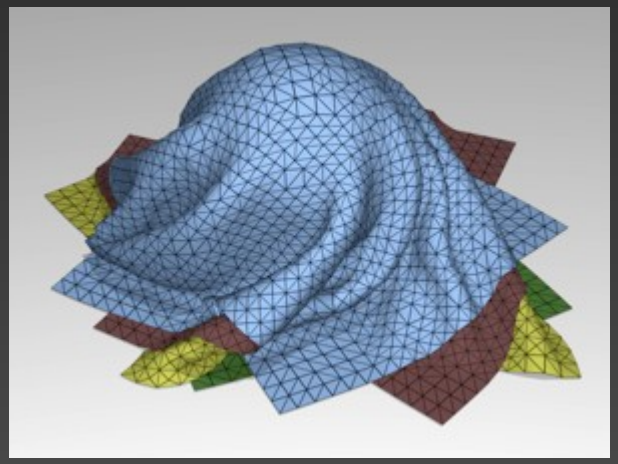


bend (3D)



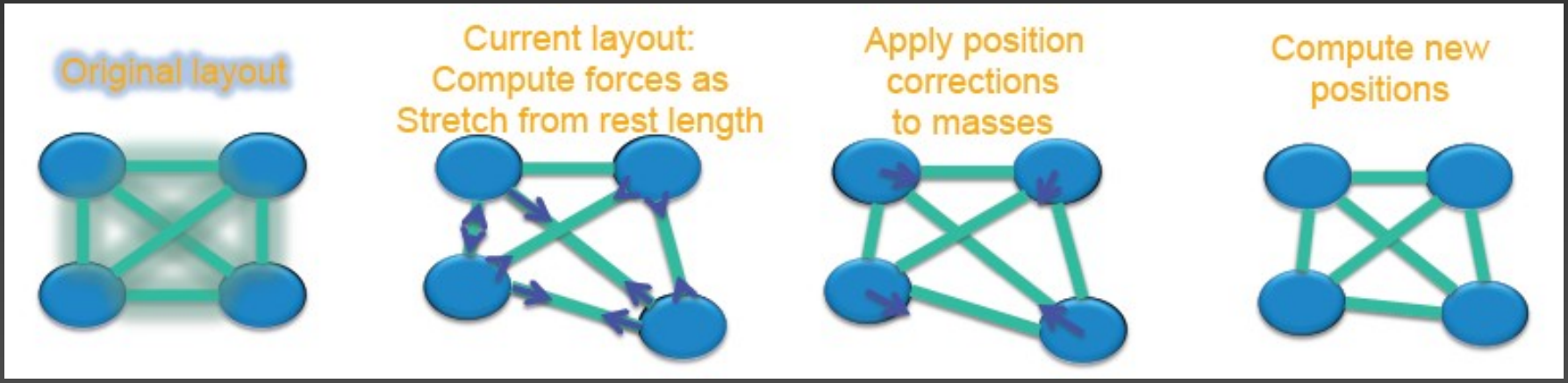
DEMO!





Soft Body Simulation

- Parallel processing
  - Lots of points and connections
  - Linear equations
- Possible to improve by adding self collisions (point + radius)
- For hollow objects
  - Pressure-Spring-Mass



Fluid simulation principle

- Reality
  - Navier-Stokes
  - Vortex
  - Combustion, diffuse, …
- Simplification – discretization

  → **particles** (or grid with average attribute)
  - sum of discrete particles in cube volume
  - only neighbouring cubes interact
  - replace integral → sum
  - paralell procesing

no sources or sinks

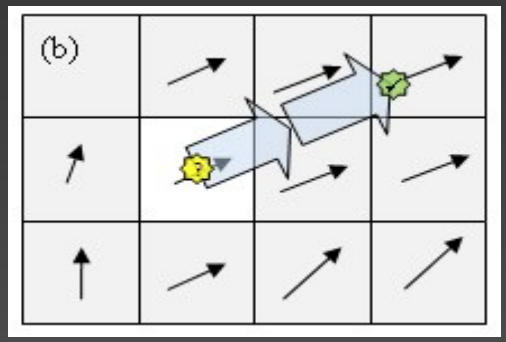
$$\nabla \boldsymbol{v} = 0$$

acceleration = pressure gradient + viscosity + external forces

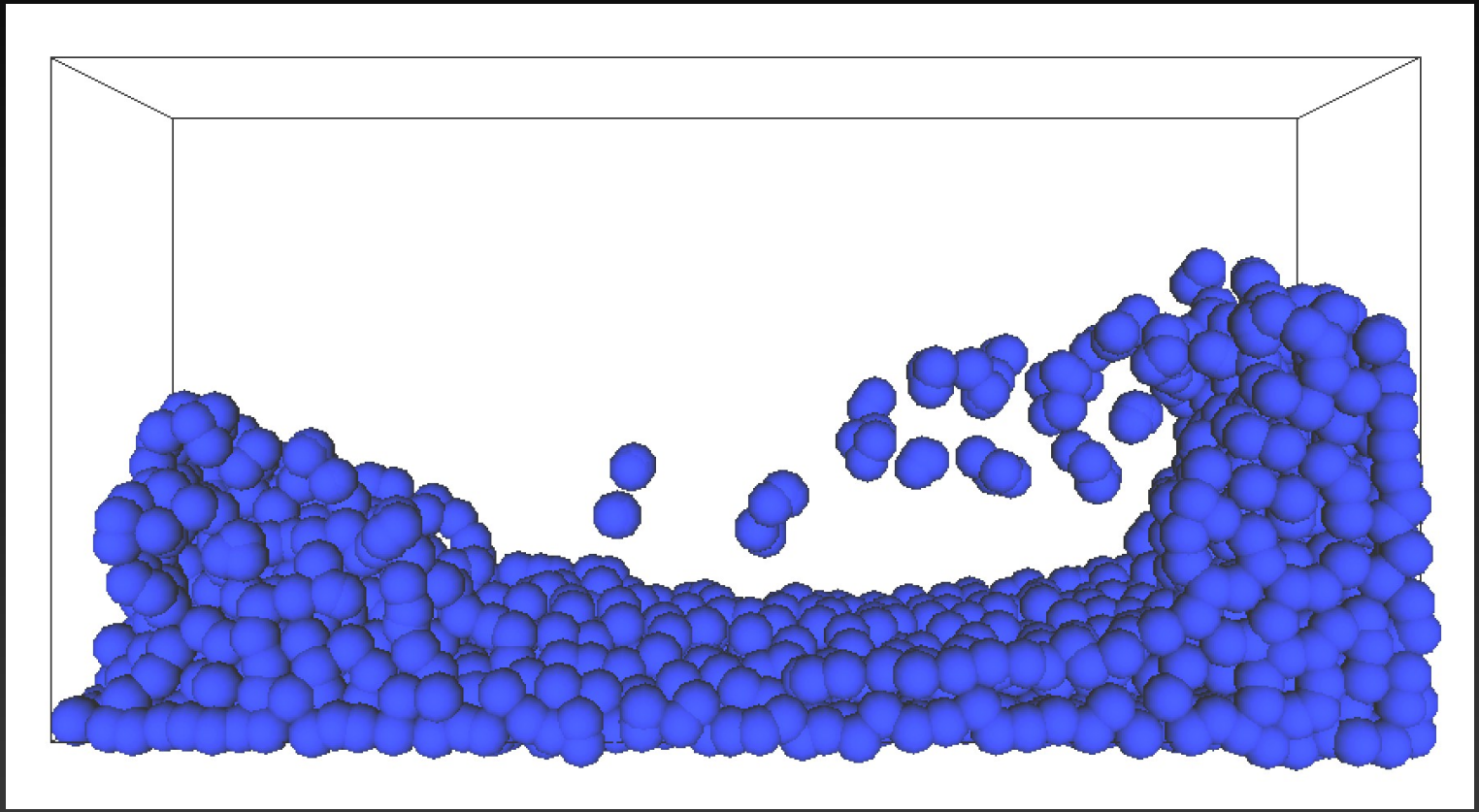
$$\rho \frac{\partial \boldsymbol{v}}{\partial t} = -\nabla p + \mu \nabla^2 \boldsymbol{v} + \rho f_{ext}$$

$$\frac{\partial \vec{\boldsymbol{v}}}{\partial t} + (\vec{\boldsymbol{v}} \cdot \vec{\nabla}) \vec{\boldsymbol{v}} = \frac{-\vec{\nabla} p}{\rho} + \frac{\mu}{\rho} \nabla^2 \vec{\boldsymbol{v}} + \frac{\vec{f}_{ext}}{\rho}$$

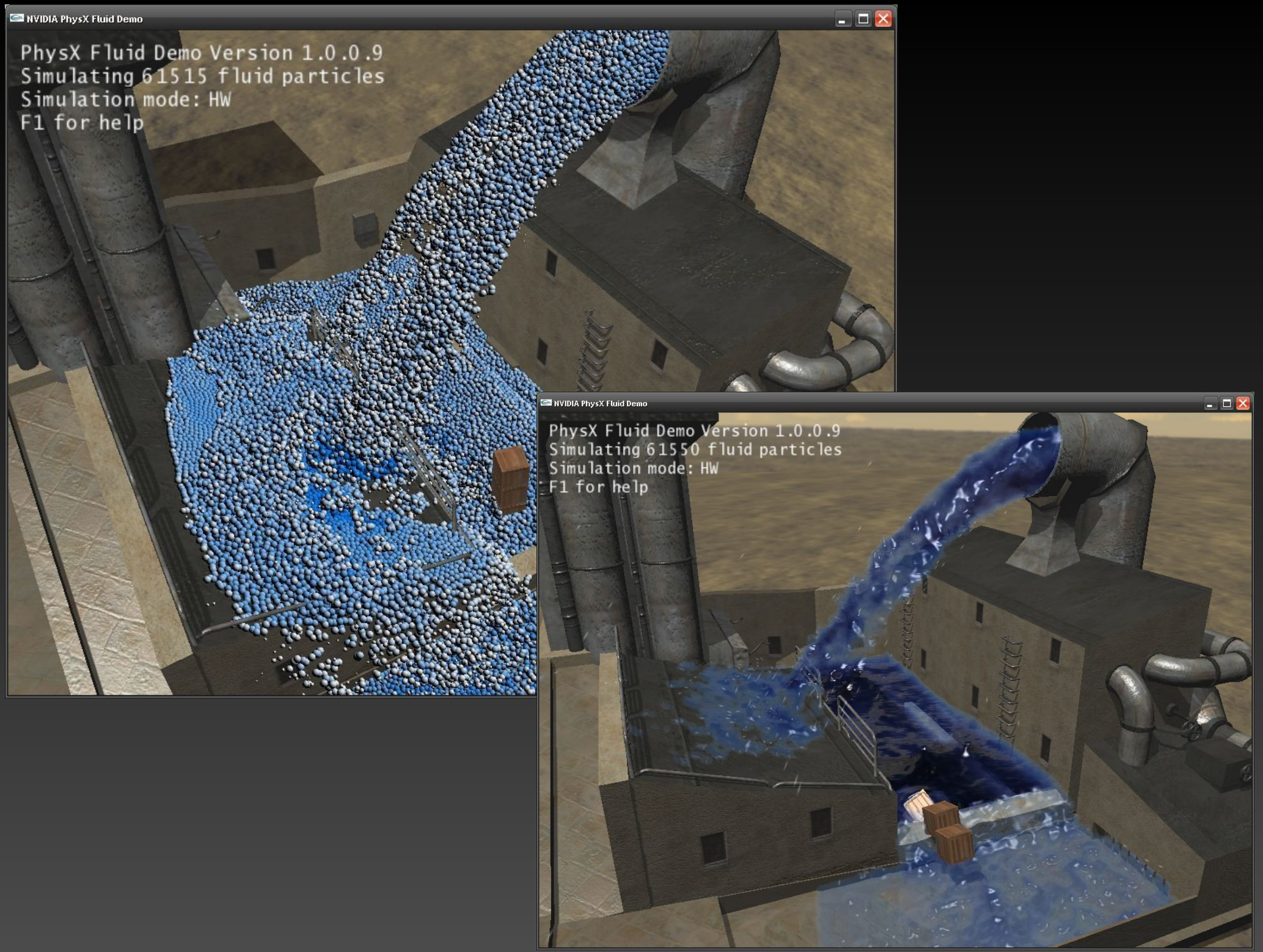
$$\frac{D\vec{\omega}}{Dt} = (\vec{\omega} \cdot \vec{\nabla}) \vec{\boldsymbol{v}} + \nu \nabla^2 \vec{\omega} + \frac{\vec{\nabla} \rho \times \vec{\nabla} p}{\rho^2} + \vec{\tau}$$

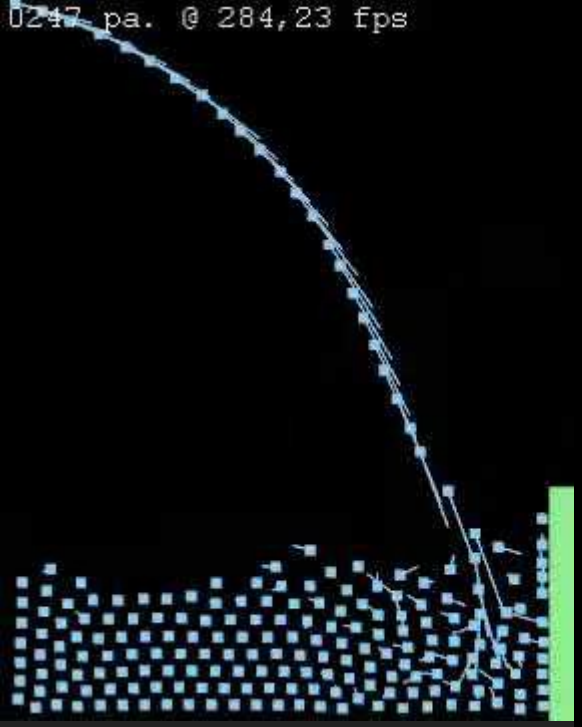


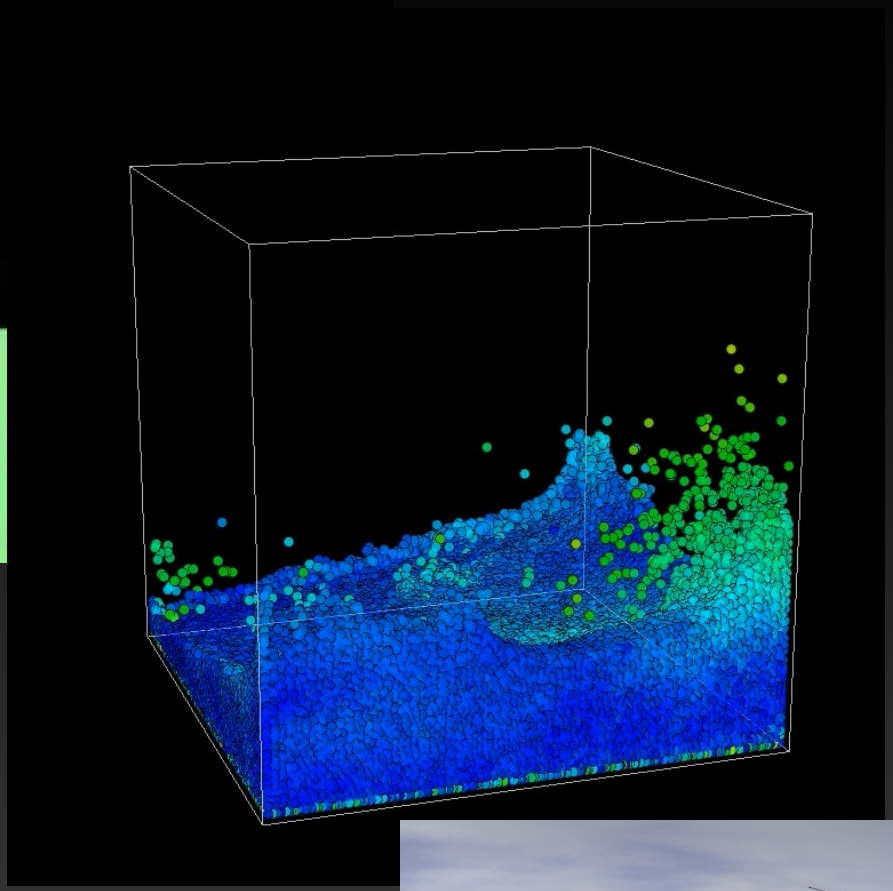
Fluid simulation



http://madebyevan.com/webgl-water/


NVIDIA PhysX Fluid Demo
PhysX Fluid Demo Version 1.0.0.9
Simulating 61515 fluid particles
Simulation mode: HW
F1 for help
NVIDIA PhysX Fluid Demo
PhysX Fluid Demo Version 1.0.0.9
Simulating 61550 fluid particles
Simulation mode: HW
F1 for help





https://vimeo.com/120475526

- 12 days
- 30 minutes / frame
- 27 × 6 × 12 m
- particle size 0,4 cm