

물리 엔진 사용을 위한
게임 물리 프로그래밍 가이드

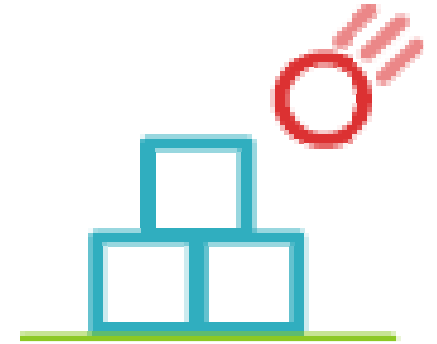
김성익(noerror@hitel.net)
2008/06/22

Overview

- **물리 엔진 사용을 위한 백그라운드 정보**
 - 개념들 소개/정리
 - 구현을 다루지 않음
 - Box2D 소개
- **PhysX 소개**

Box2D

- Erin Catto씨 개인 프로젝트
- 2D 오픈 소스 물리 엔진
- 사용한 프로젝트 : [Crayon Physics](#)
- <http://www.box2d.org>
- 최적의 물리 처리 레퍼런스



Kinematics

- **Newtonian Physics**
 - **$F = ma$**
 - **Spring ($F = -kx$)**
 - **Drag Force ($F = -mpv$)**
 - **Gravity**
 - **$F = dp/dt, P = mv$**
 - **$a = dV/dt$**
 - **$V = dX/dt$**

Calculus

- **Position**
- **Velocity**
- **Acceleration**

$$\mathbf{F} = \sum_k \mathbf{F}_k$$

$$\mathbf{a}_i = \frac{\mathbf{F}}{m}$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + h\mathbf{v}_i$$

$$\mathbf{v}_{i+1} = \mathbf{v}_i + h\mathbf{a}_i$$

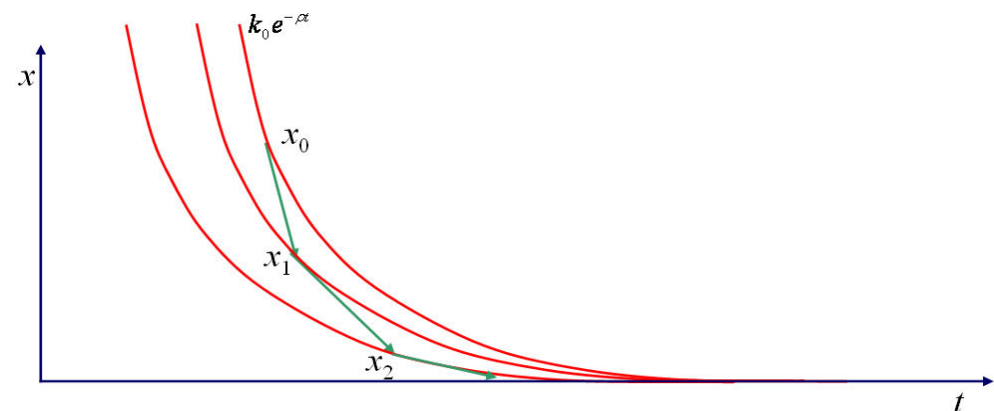
Solution

- **Analytic Solution**
- **Numeric Solution**
 - **Euler Method**
 - **Midpoint Method**
 - **Runge Kutta**
 - **Implicit Method**
 - **Verlet**

$$\dot{f} = \lim_{h \rightarrow 0} \frac{f(t+h) - f(t)}{h}$$

$$\dot{f} \approx \frac{f(t+h) - f(t)}{h}$$

$$f(t+h) \approx f(t) + h\dot{f}$$



Solution

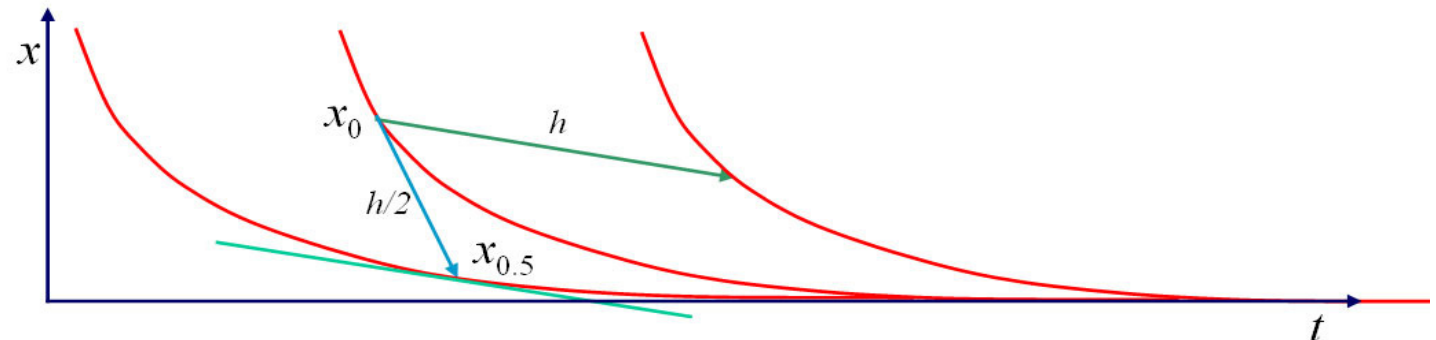
- **Analytic Solution**
- **Numeric Solution**
 - Euler Method
 - **Midpoint Method**
 - Runge Kutta
 - Implicit Method
 - Verlet

$$\mathbf{x}_{i+1/2} = \mathbf{x}_i + \frac{h}{2} \mathbf{v}_i$$

$$\mathbf{v}_{i+1/2} = \mathbf{v}_i + \frac{h}{2} \mathbf{F}(\mathbf{x}_i, \mathbf{v}_i) / m$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + h \mathbf{v}_{i+1/2}$$

$$\mathbf{v}_{i+1} = \mathbf{v}_i + h \mathbf{F}(\mathbf{x}_{i+1/2}, \mathbf{v}_{i+1/2}) / m$$



Solution

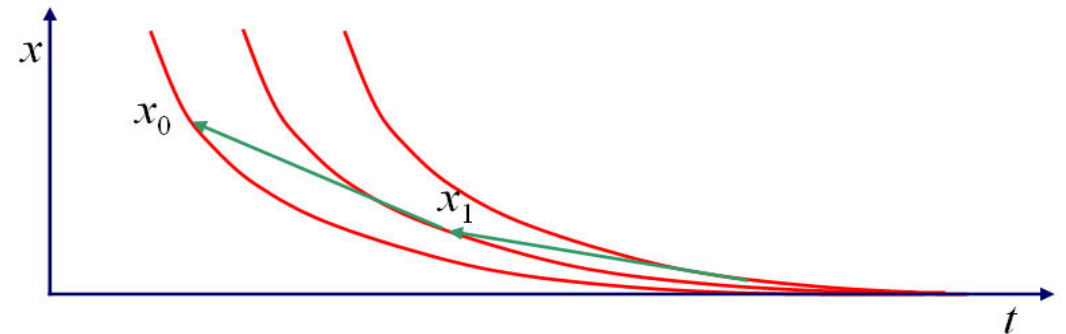
- **Analytic Solution**
- **Numeric Solution**
 - Euler Method
 - Midpoint Method
 - **Runge Kutta**
 - Implicit Method
 - Verlet

$$\begin{aligned}k_1 &= h f(x_n, y_n) \\k_2 &= h f\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right) \\y_{n+1} &= y_n + k_2 + O(h^3),\end{aligned}$$

$$\begin{aligned}k_1 &= h f(x_n, y_n) \\k_2 &= h f\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right) \\k_3 &= h f\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2\right) \\k_4 &= h f(x_n + h, y_n + k_3) \\y_{n+1} &= y_n + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 + O(h^5)\end{aligned}$$

Solution

- **Analytic Solution**
- **Numeric Solution**
 - Euler Method
 - Midpoint Method
 - Runge Kutta
 - **Implicit Method**
 - Verlet



Solution

- **Analytic Solution**
- **Numeric Solution**
 - Euler Method
 - Midpoint Method
 - Runge Kutta
 - Implicit Method
 - **Verlet**

$$\mathbf{x}_{i+1} = 2\mathbf{x}_i - \mathbf{x}_{i-1} + h^2 \cdot \mathbf{a}_i$$

$$x(t + \Delta t) = (2 - f)x(t) - (1 - f)x(t - \Delta t) + a(t)(\Delta t)^2.$$

Rotational Dynamics

- **Center of Mass**
- **Linear vs Angular**
 - **Force vs Torque**
 - **Mass vs Moment of inertia**
 - **Momentum vs Angular Momentum**
 - **Velocity vs Angular Velocity**
 - **Position vs Orientation**

Box2d CodeReview(1)

- **Simulation iteration step**
 - **broadphase**
 - 충돌검사 (충돌정보 Arbiter 개체로 저장)
 - **force 적용**
 - force→velocity
 - torque→angular velocity
 - **prestep**
 - 충돌 포인트에 대해서 massnormal, masstangent, bias 계산

Box2d CodeReview(2)

- 충돌에 의한 impulse
 - 충돌체에 impulse
 - velocity, angular velocity 적용
 - Joint 에 의한 impulse
 - 양 joint에 충돌체에 impulse
- 위치 및 회전에 적용
 - velocity => position
 - Angular velocity => rotation

PhysX(1)

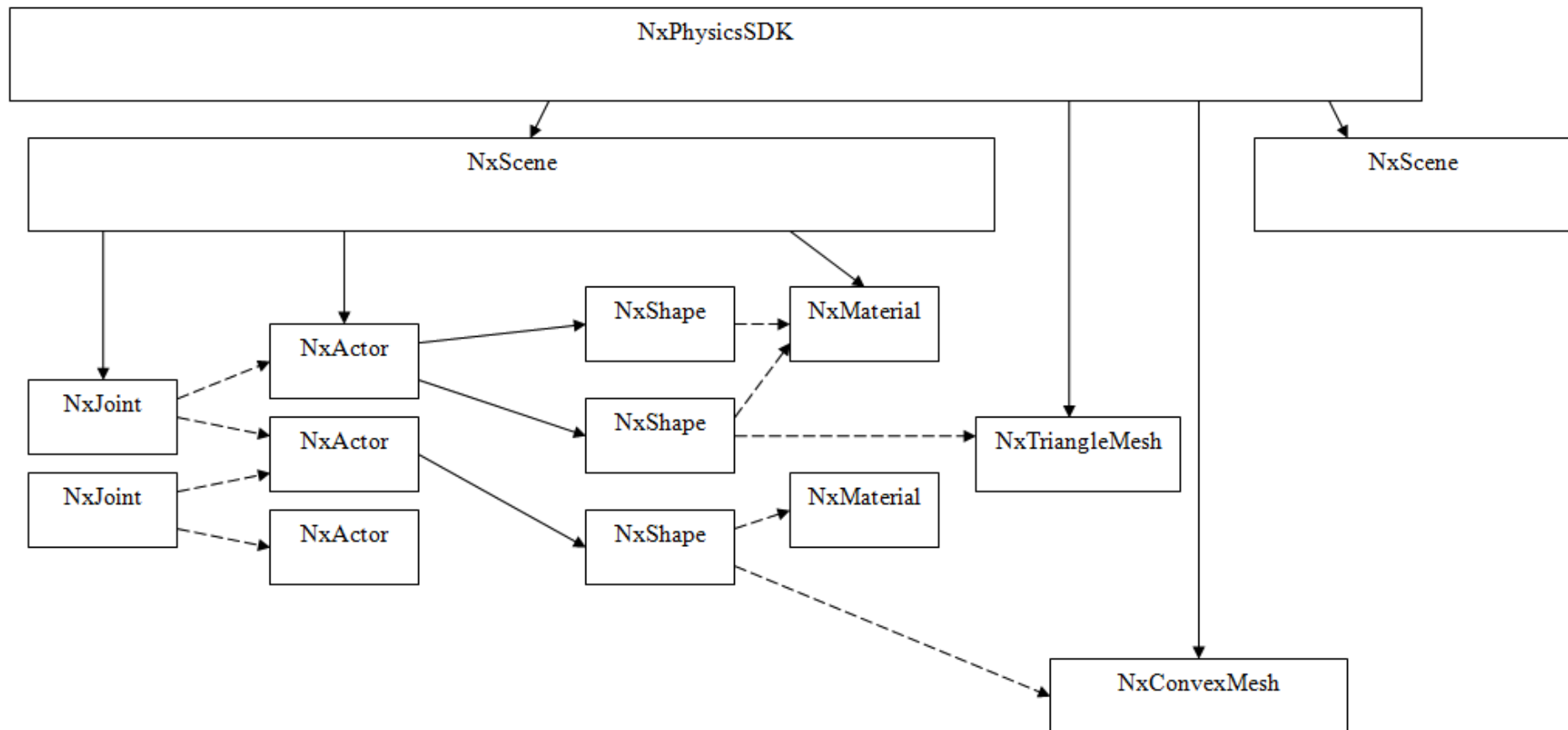
- NVidia (<http://www.ageia.com>)
- 무료 (일부 플랫폼 및 소스코드 라이선스는 유료)
- 충실한 튜터리얼 샘플 제공
- 메인스트림 물리 엔진
- 사용 프로젝트 : 언리얼3, 외 다수

PhysX(2)

• 기능

- 복잡한 강체 물리 시스템
- 캐릭터 컨트롤
- 탈 것 다이내믹스
- 멀티쓰레드/멀티 플랫폼/PPU지원
- 유체 시뮬레이션
- 옷, 깃발등의 천 시뮬레이션
- 소프트 바디
- 역장(forcefield) 시뮬레이션

PhysX-architecture diagram



Shape

- **Shape**

- **Sphere**

- **Capsule**

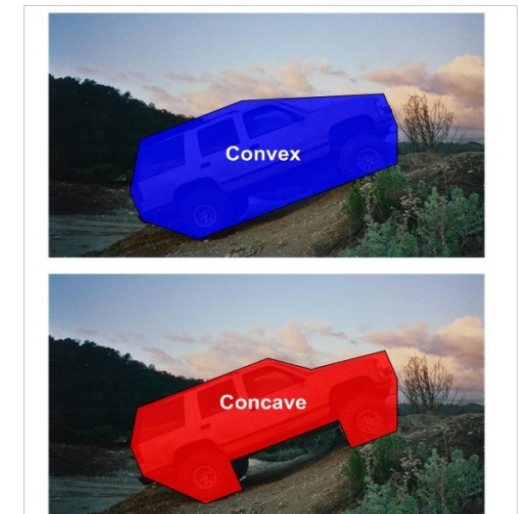
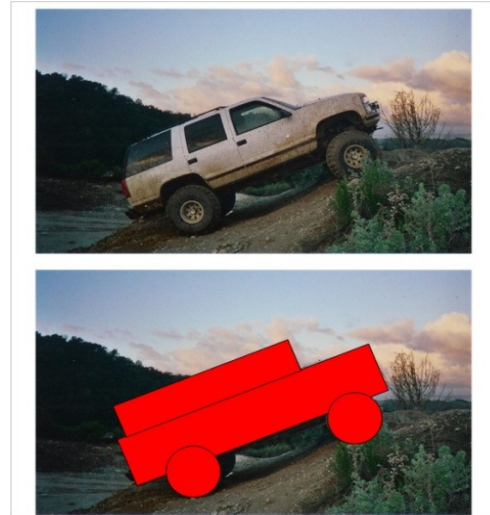
- **BoundBox**

- **AABB** (Axis Aligned Bound Box)

- **OBB** (Object Oriented Box)

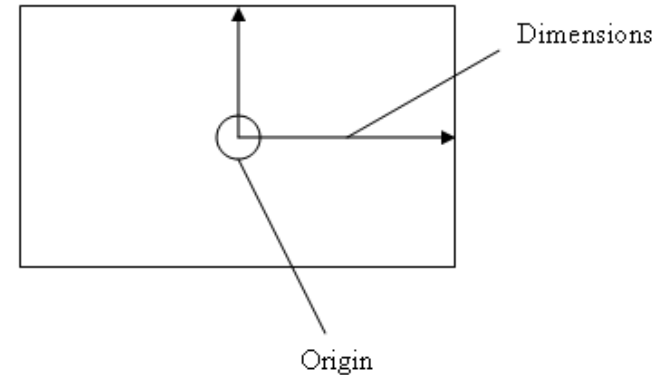
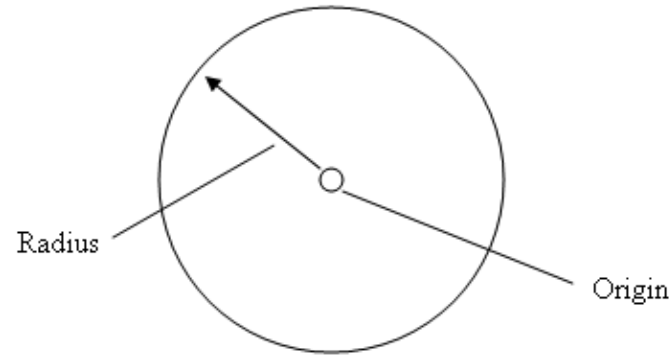
- **Convex**

- **(height field)**

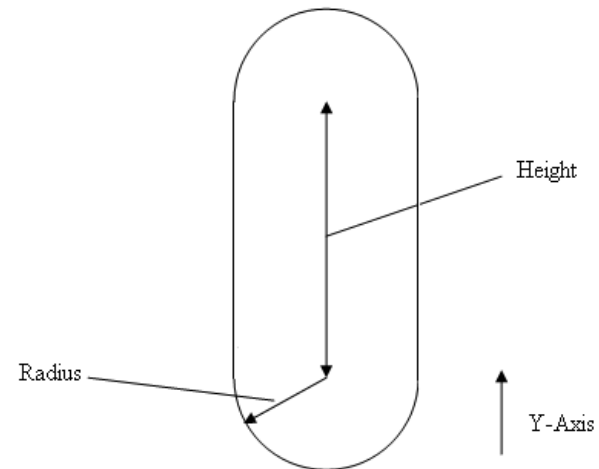


PhysX-Shape

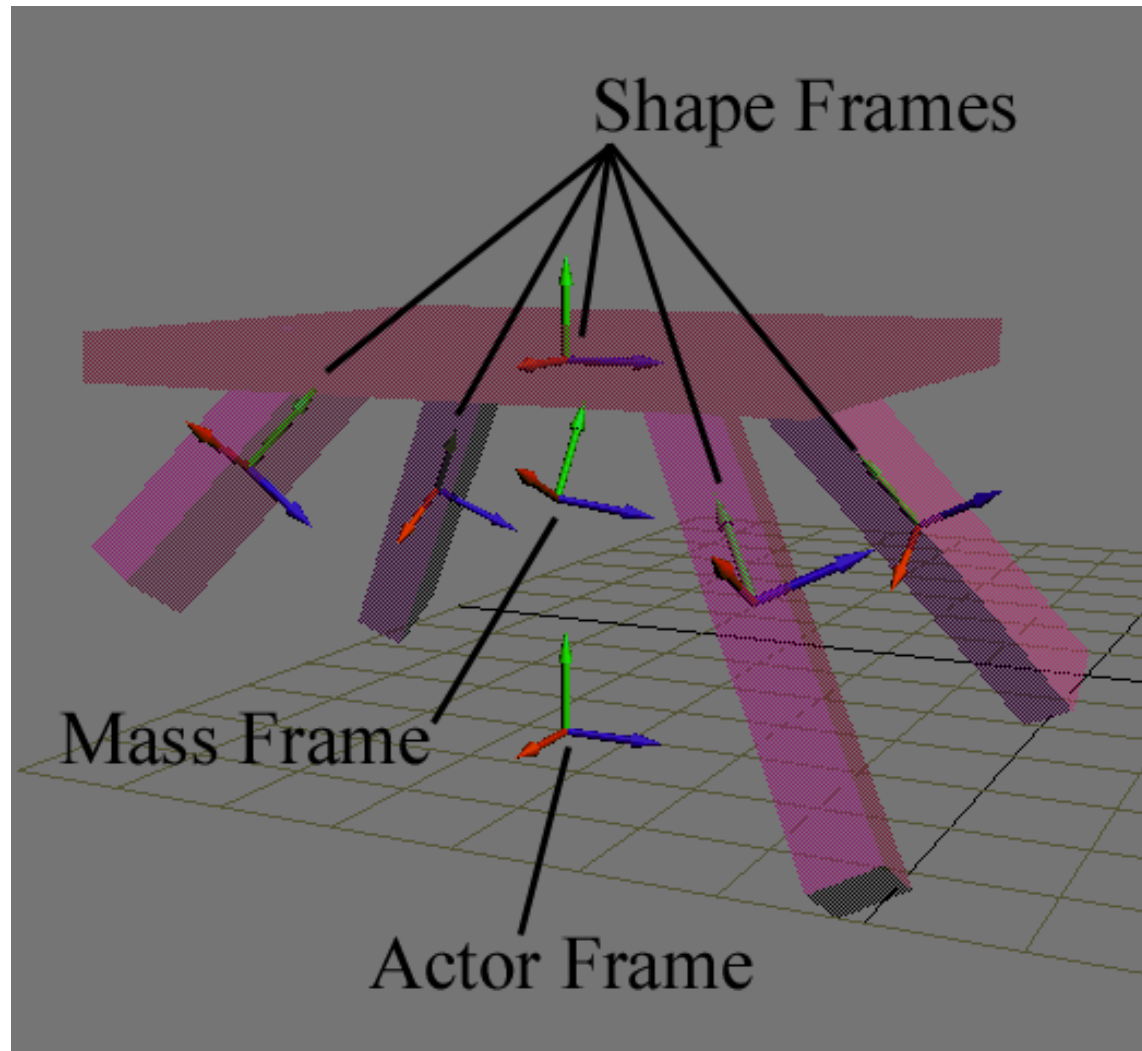
- **Sphere**
- **Box**
- **Capsule**



- **Plane**
- **Height field**
- **Convex meshes**
- **Triangle meshes**



PhysX-Actor

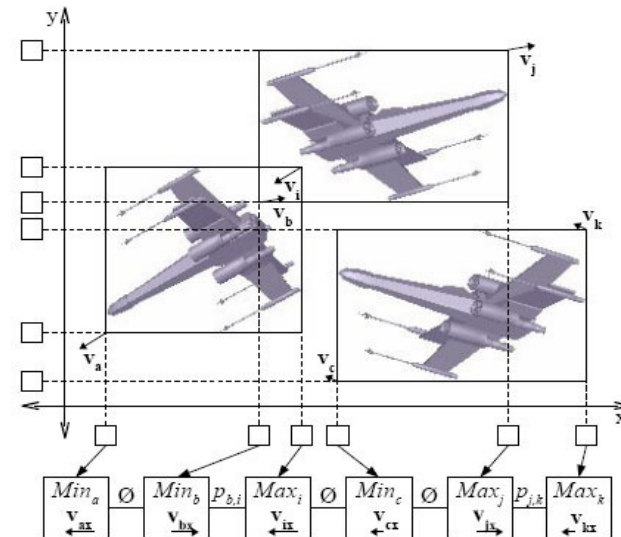


Collision Detection(1)

- **Distance Test**
 - Point-point distance
 - Line-point distance
 - Line-line distance
 - Segment-segment distance
- **Rough Bounding**
 - Sweep and prune

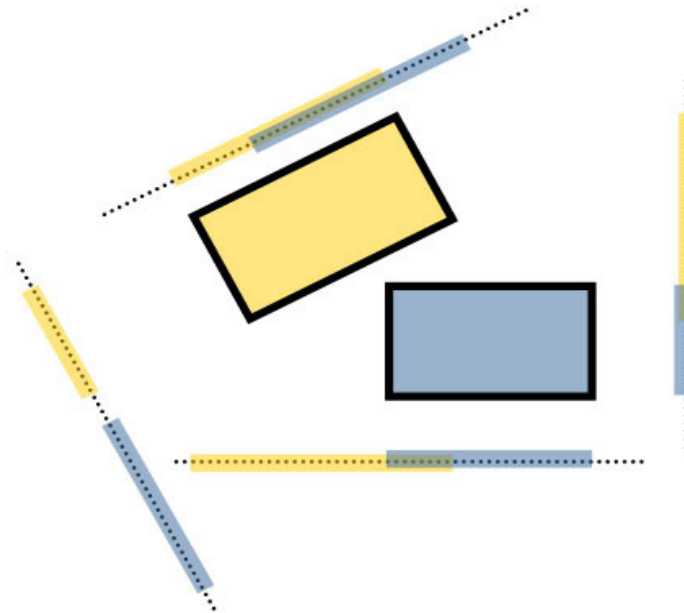
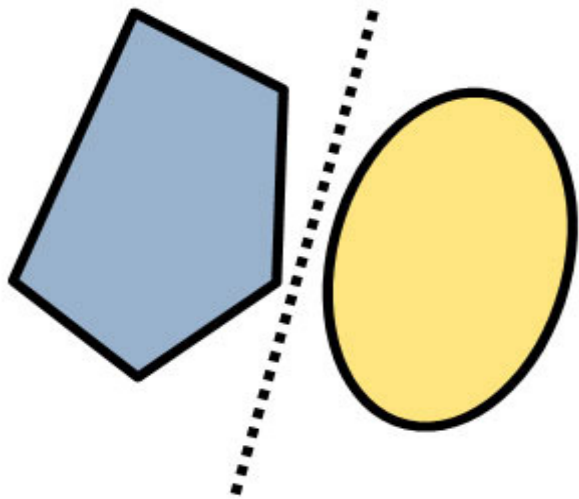
Collision Detection(1)

- Distance Test
 - Point-point distance
 - Line-point distance
 - Line-line distance
 - Segment-segment distance
- Rough Bounding
 - Sweep and prune



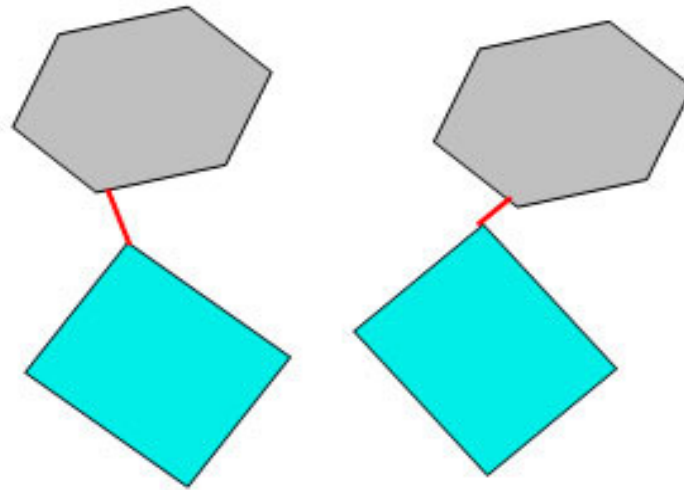
Collision Detection(2)

- **SAT(separation axis test)**
- Lin Canny
- Minkowski Difference



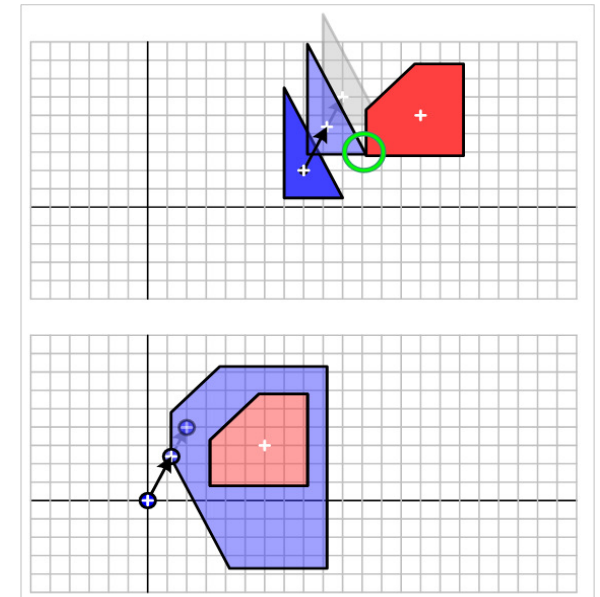
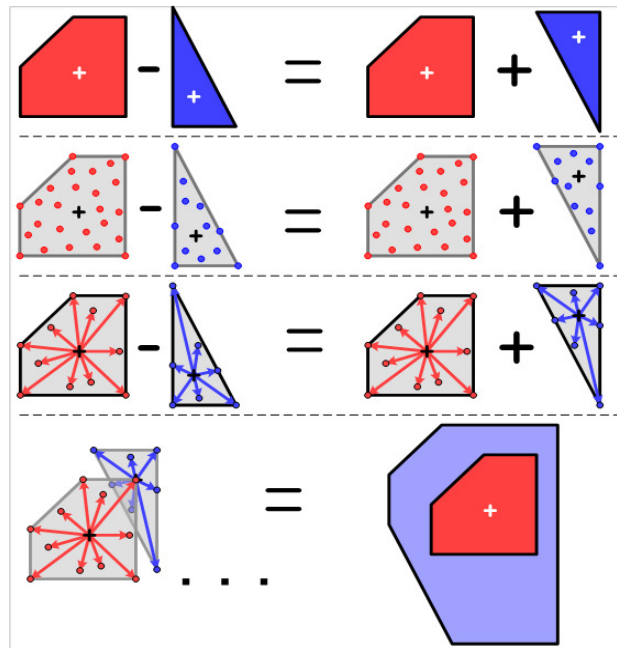
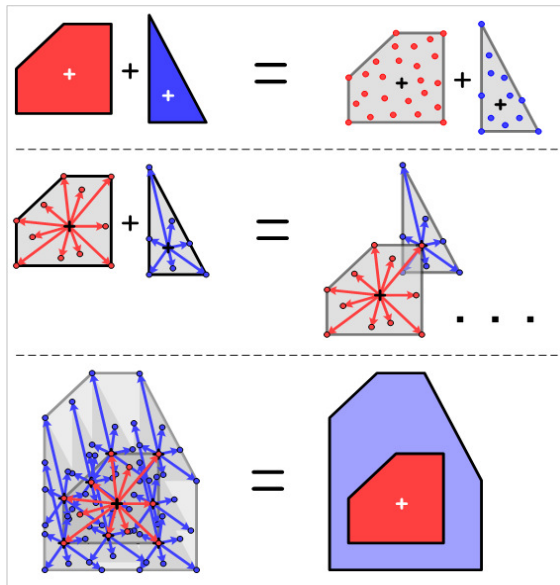
Collision Detection(2)

- SAT(separation axis test)
- **Lin Canny**
- Minkowski Difference



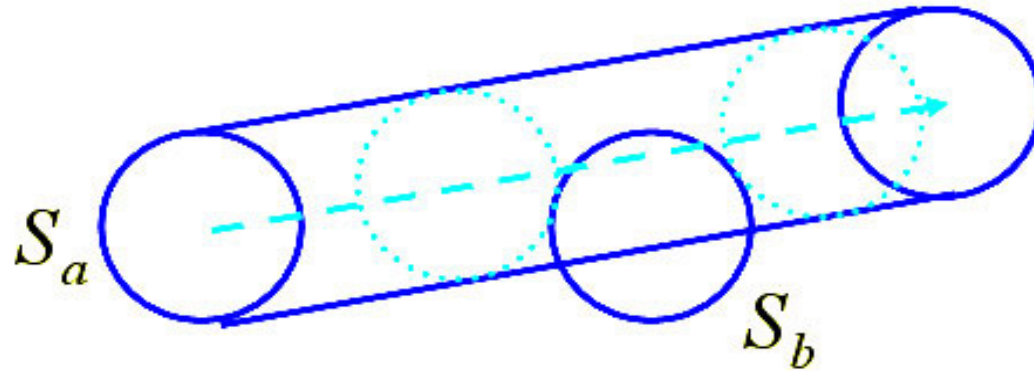
Collision Detection(2)

- SAT(separation axis test)
- Lin Canny
- **Minkowski Sum/Difference**



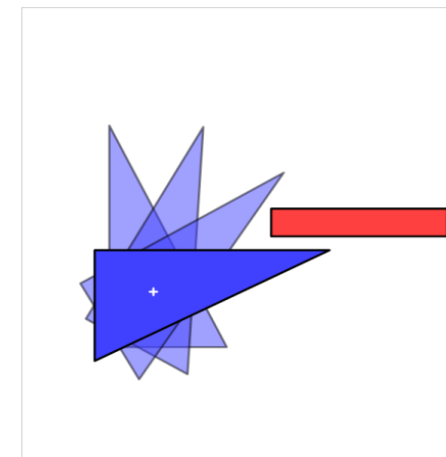
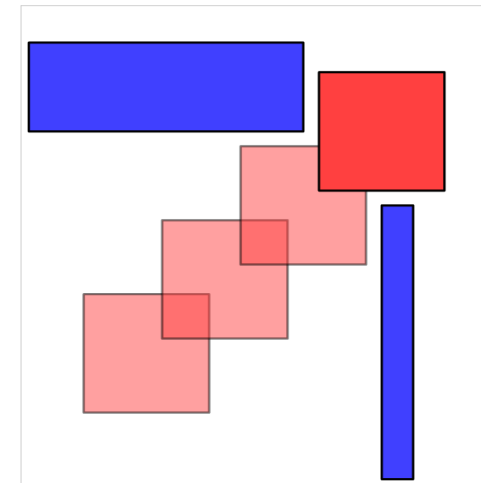
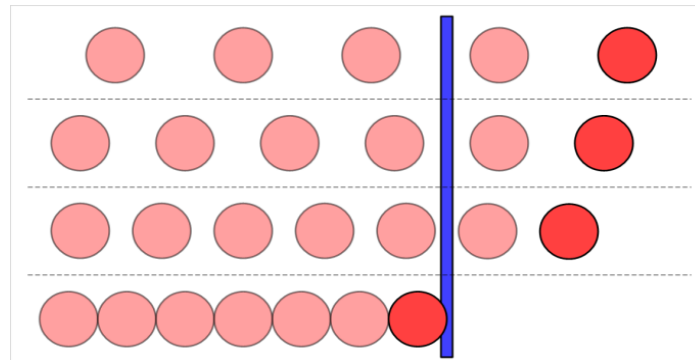
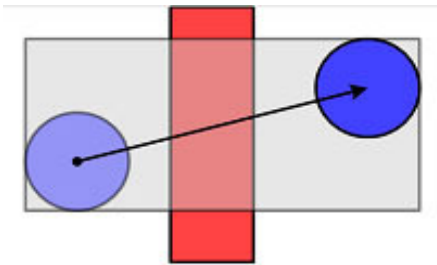
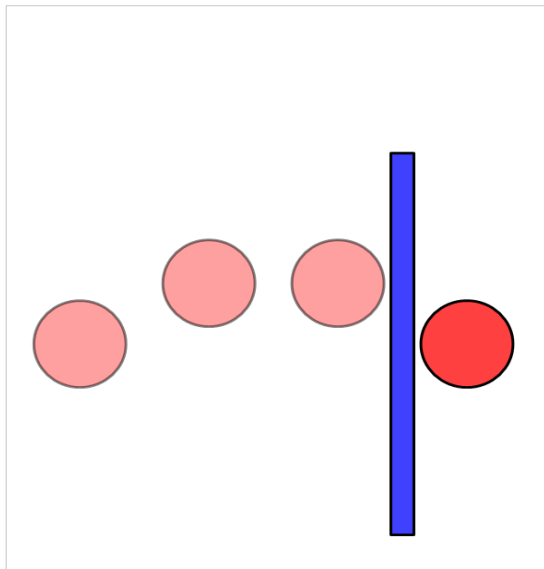
Collision Detection(3)

- **Determine Collision Time**
 - Time of impact
- **Tunnelling (CCD)**



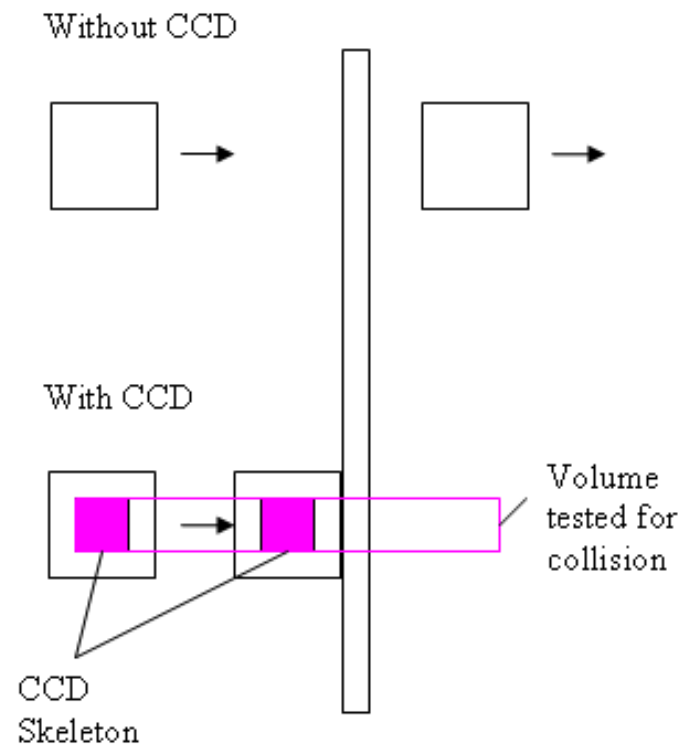
Collision Detection(3)

- Determine Collision Time
 - Time of impact
- Tunnelling



PhysX - CCD

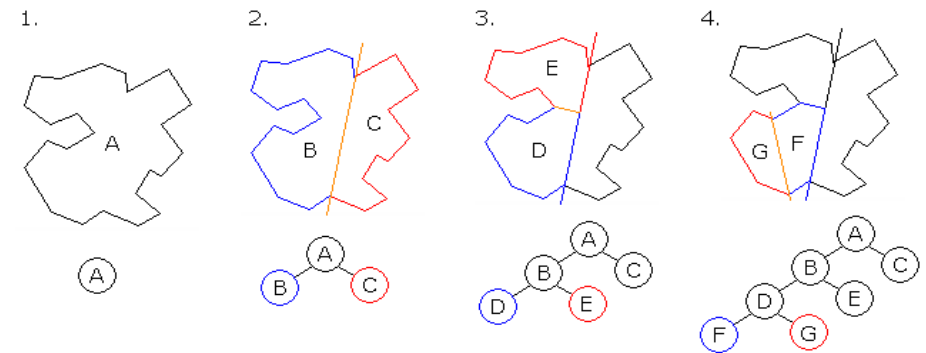
- **Continuous Collision Detection**



Collision Detection(4)

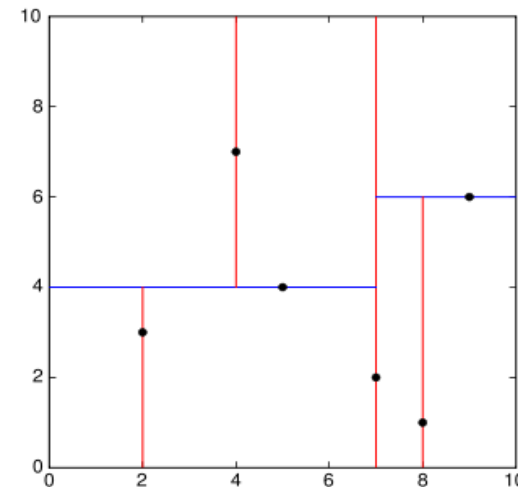
- **Spatial Subdivision**

- **Bsp, k-d tree**
- **Portal**
- **Quad tree / octree**



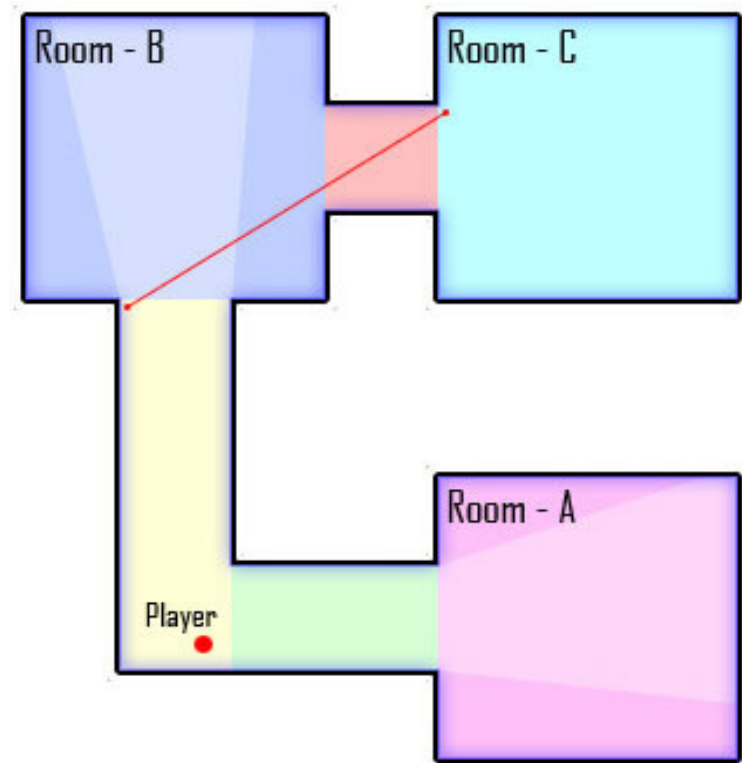
- **Grouping**

- **Sleeping**



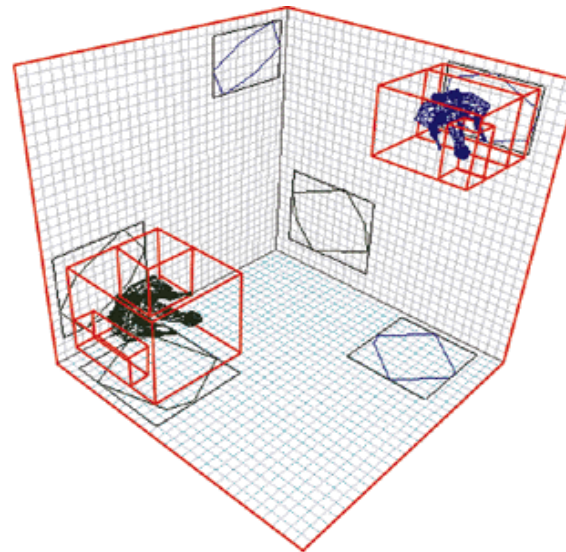
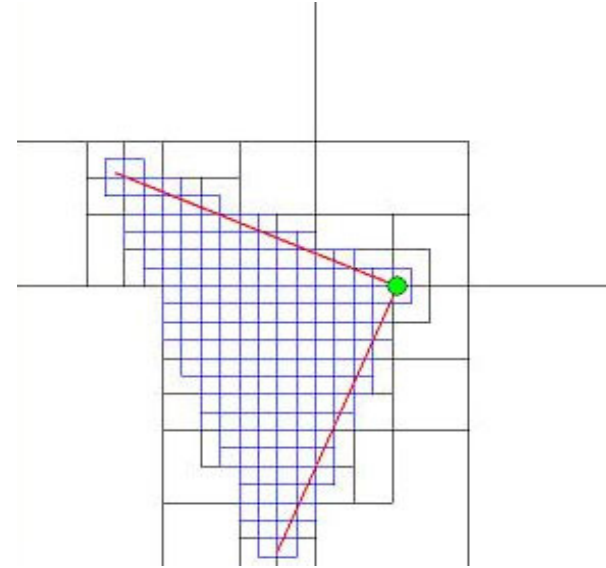
Collision Detection(4)

- **Spatial Subdivision**
 - Bsp, k-d tree
 - **Portal**
 - Quad tree / octree
- **Grouping**
 - **Sleeping**



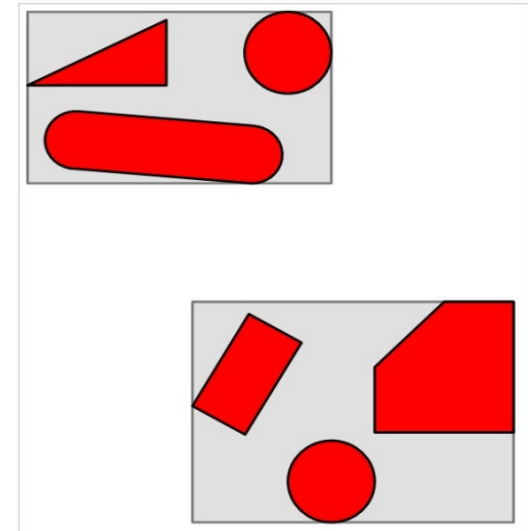
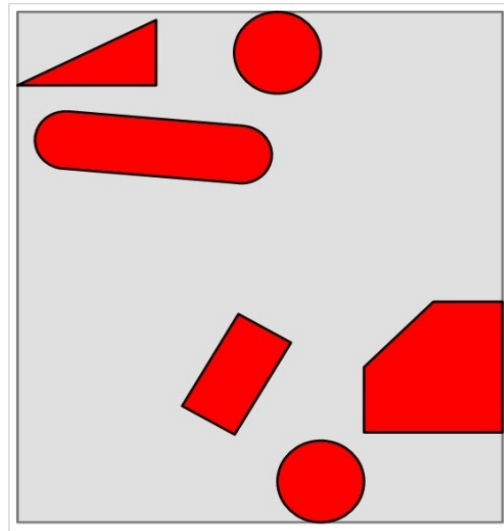
Collision Detection(4)

- **Spatial Subdivision**
 - Bsp, k-d tree
 - Portal
 - Quad tree / octree
- **Grouping**
 - Sleeping



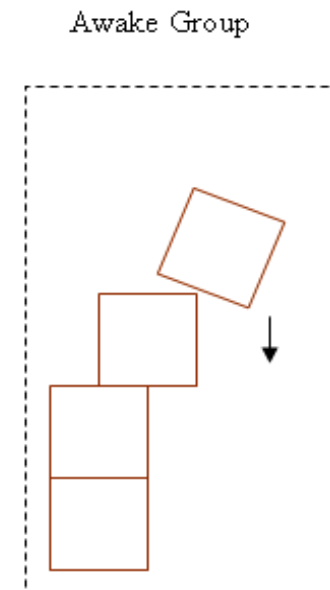
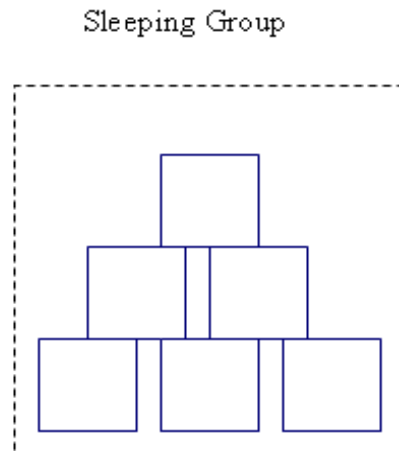
Collision Detection(4)

- **Spatial Subdivision**
 - Bsp, k-d tree
 - Portal
 - Quad tree / octree
- **Grouping**
 - Sleeping



Collision Detection(4)

- **Spatial Subdivision**
 - Bsp, k-d tree
 - Portal
 - Quad tree / octree
- **Grouping**
 - **Sleeping**



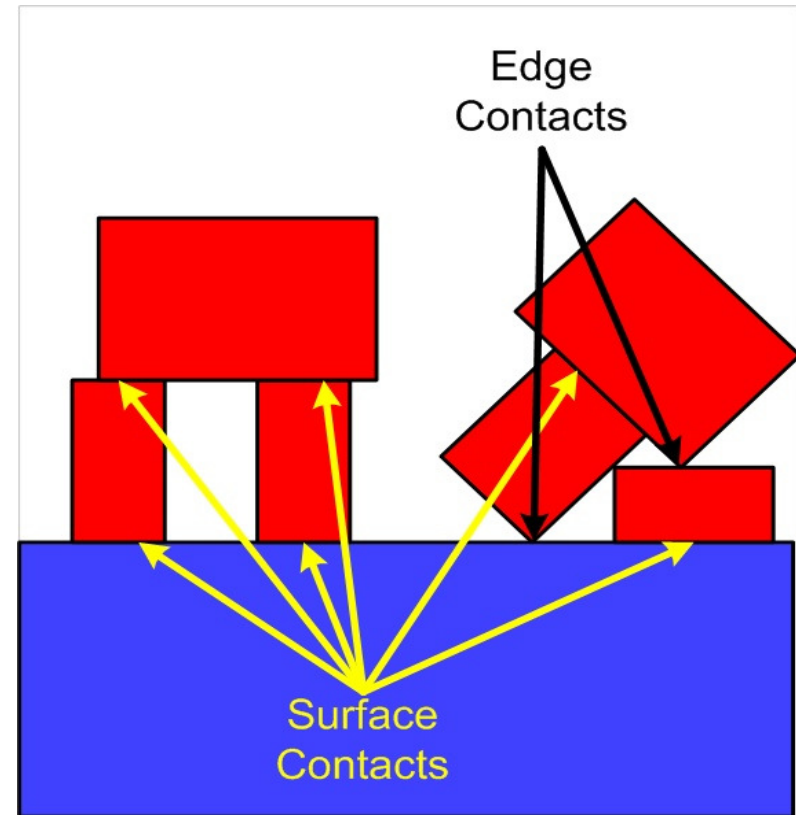
PhysX-Collision Detection

	NxSphereShape	NxBoxShape	NxCapsuleShape	NxPlaneShape	NxConvexShape	NxHeightfieldShape	NxWheelShape	NxTriangleMeshShape	NxTriangleMeshShape (Heightfield)	NxTriangleMeshShape (pmap)
NxSphereShape	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
NxBoxShape	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
NxCapsuleShape	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
NxPlaneShape	✓	✓	✓	✗	✓	✗	✓	✓	✗	✗
NxConvexShape	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
NxHeightfieldShape	✓	✓	✓	✗	✓	✗	✓	✗	✗	✗
NxWheelShape	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓
NxTriangleMeshShape	✓	✓	✓	✓	✓	✗	✓	✗	✗	✗
NxTriangleMeshShape (heightfield)	✓	✓	✓	✗	✓	✗	✓	✗	✗	✗
NxTriangleMeshShape (pmap)	✓	✓	✓	✗	✓	✗	✓	✗	✗	✓

- **Contact Filtering**
- **Triggers**
- **CCD**
- **Dominance Group**
- **Overlap Testing**

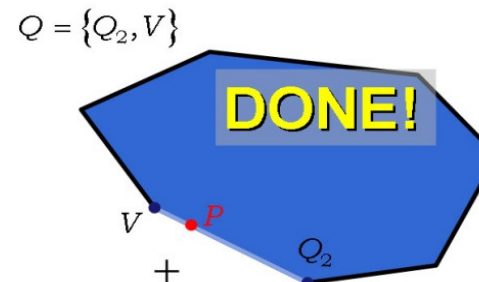
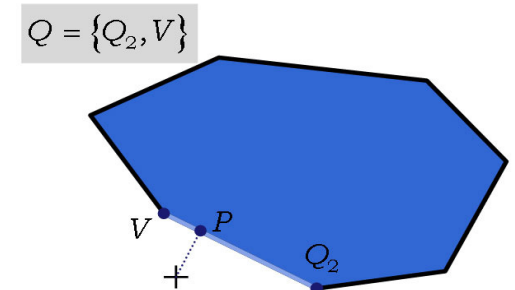
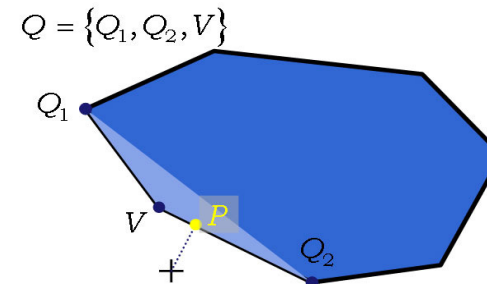
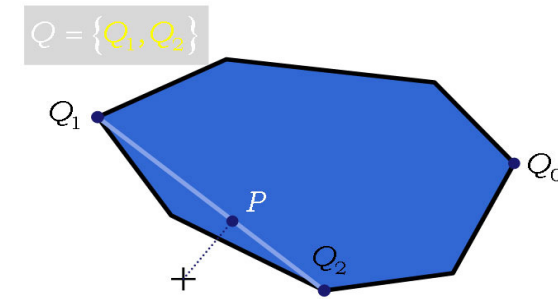
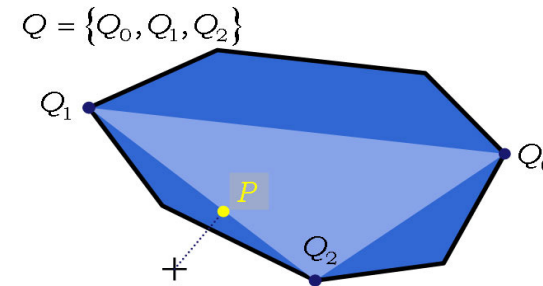
Collision Response(1)

- **Contact Point**
 - EE (Edge-Edge)
 - FV (Face-Vertex)
 - GJK
- **Normal**
- **Penetration**



Collision Response(1)

- Contact Point
 - EE (Edge-Edge)
 - FV (Face-Vertex)
 - GJK
- Normal
- Penetration

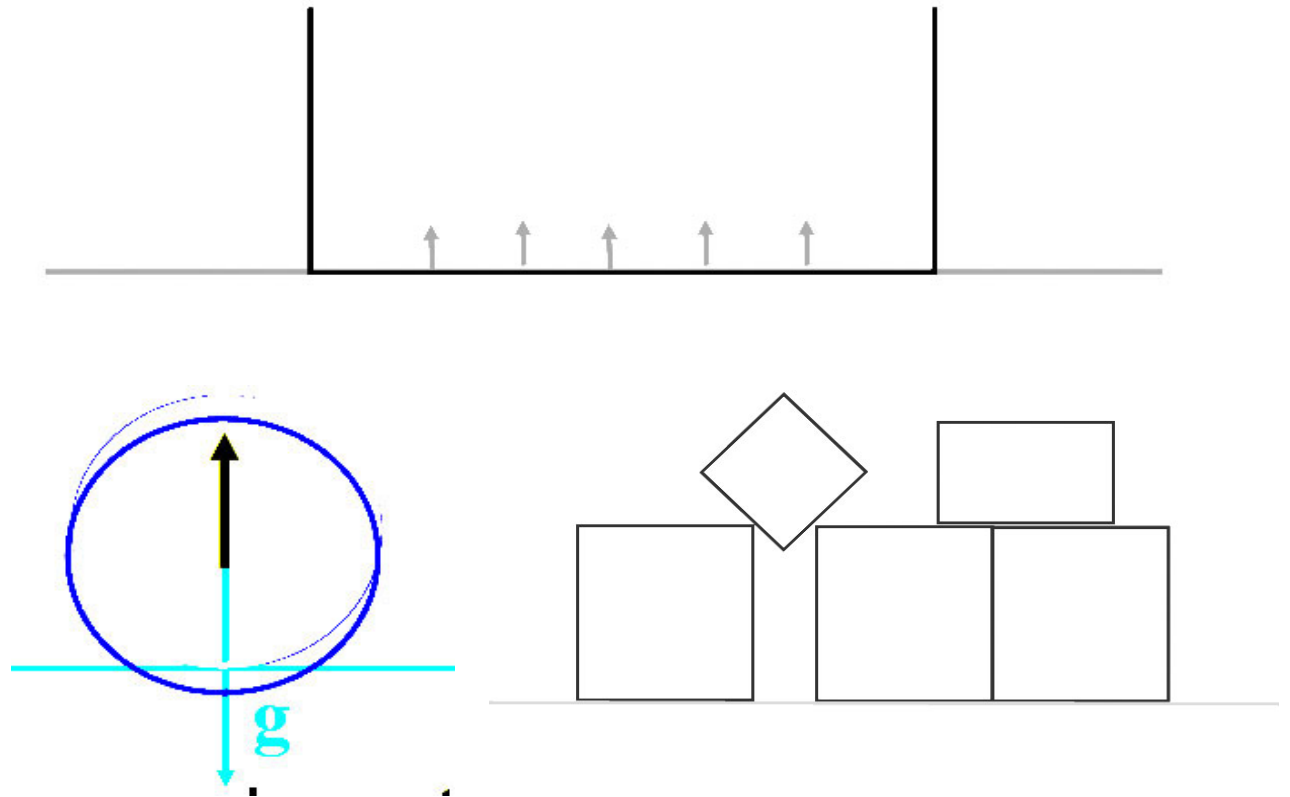


Collision Response(2)

- **Colliding**
 - **Bouncing**
- **Resting**
 - **Rolling**
 - **Sliding**

Collision Response(2)

- Colliding
 - Bouncing
- Resting
 - Rolling
 - Sliding

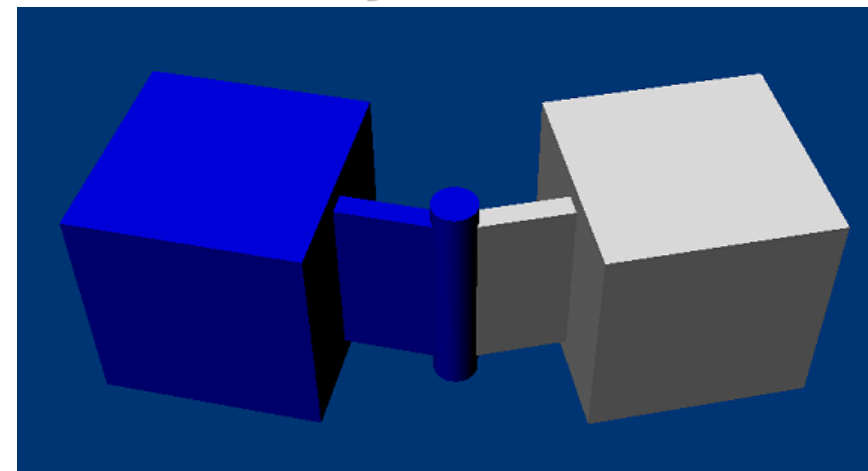
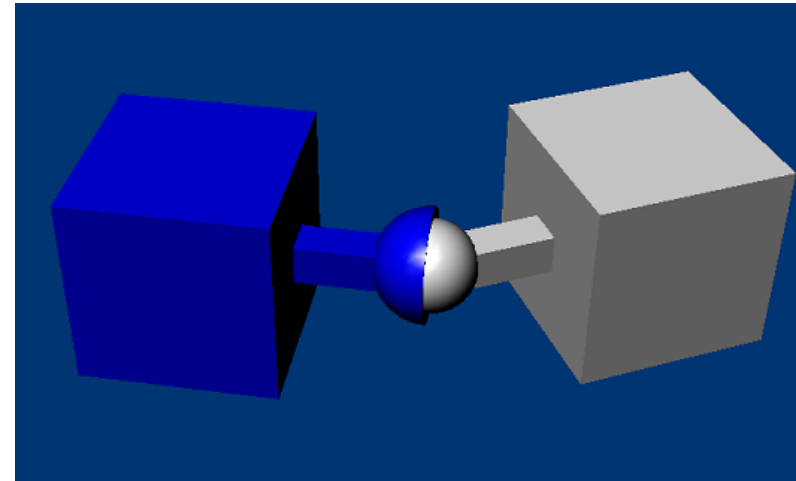


Constraint

- **Joint**
 - **Pin Joints**
 - **Hinge**
- **Angular Constraints**
- **Distance Constraints, ...**

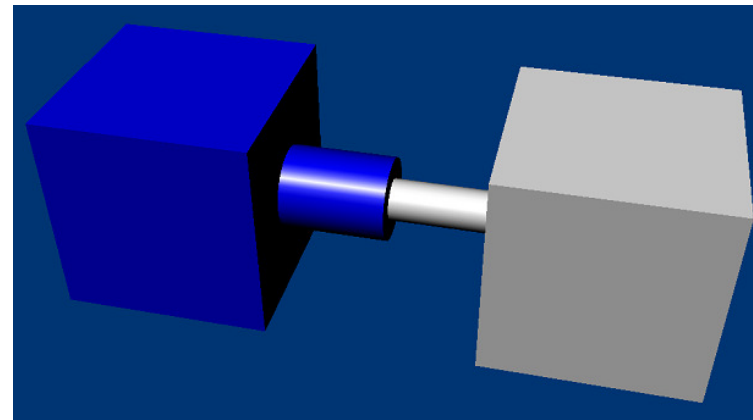
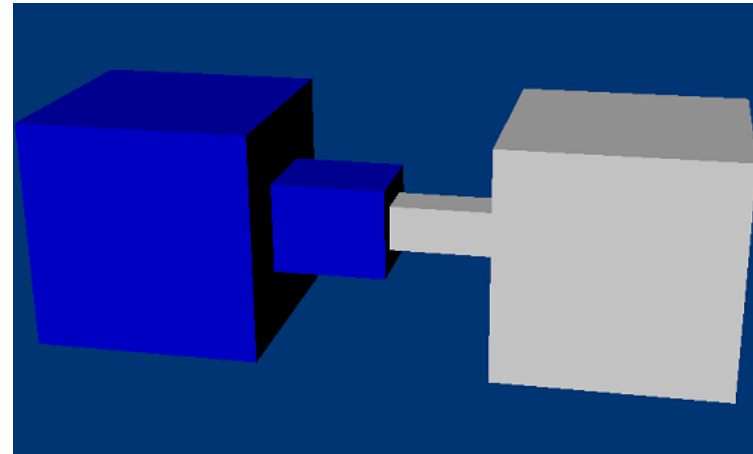
PhysX-Joint

- **spherical joint**
- **revolute joint**
- prismatic joint
- cylindrical joint
- fixed joint
- distance joint
- point in plane joint / point on line joint
- pulley joint
- d6



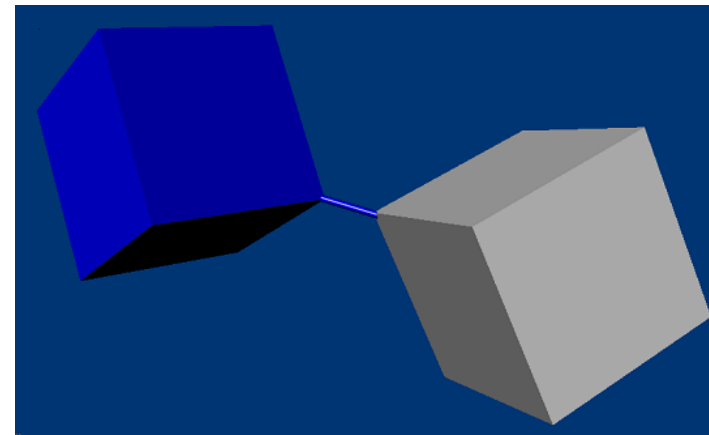
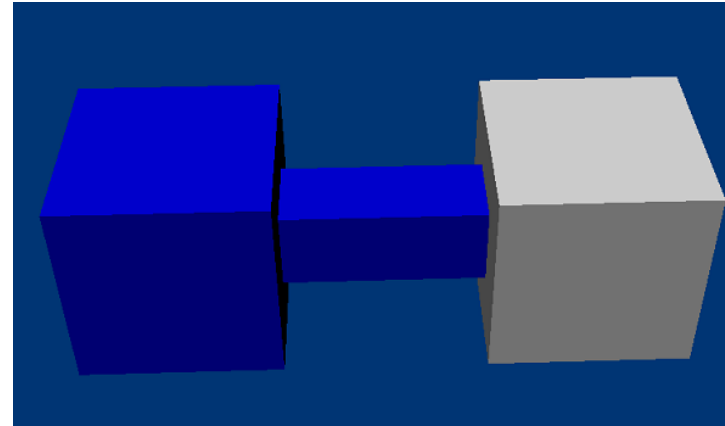
PhysX-Joint

- spherical joint
- revolute joint
- **prismatic joint**
- **cylindrical joint**
- fixed joint
- distance joint
- point in plane joint / point on line joint
- pulley joint
- d6



PhysX-Joint

- spherical joint
- revolute joint
- prismatic joint
- cylindrical joint
- **fixed joint**
- **distance joint**
- point in plane joint / point on line joint
- pulley joint
- d6



PhysX-Example Review

- **예제 : SampleBoxes**
- **InitNx**
 - Initialize PhysicsScene => gPhysicsSDK
 - CreateScene => gScene
 - Create Ground => gScene->createActor
- **RenderCallback**
 - Simulate
 - GetResult/Render All actors => actor->getGlobalPos
 - FetchSimulation => flushScream, fetchResult
- **CreateCube**
 - shapes.pushBack
 - gScene->createActor

More

- **Breaking**
- **Force Field**

- **Character Controller**
- **Raycast/SweepTest**

- **Multithreading Model**

Reference

- <http://www.essentialmath.com>
- 게임물리바이블, David H Everly, 사이텍미디어