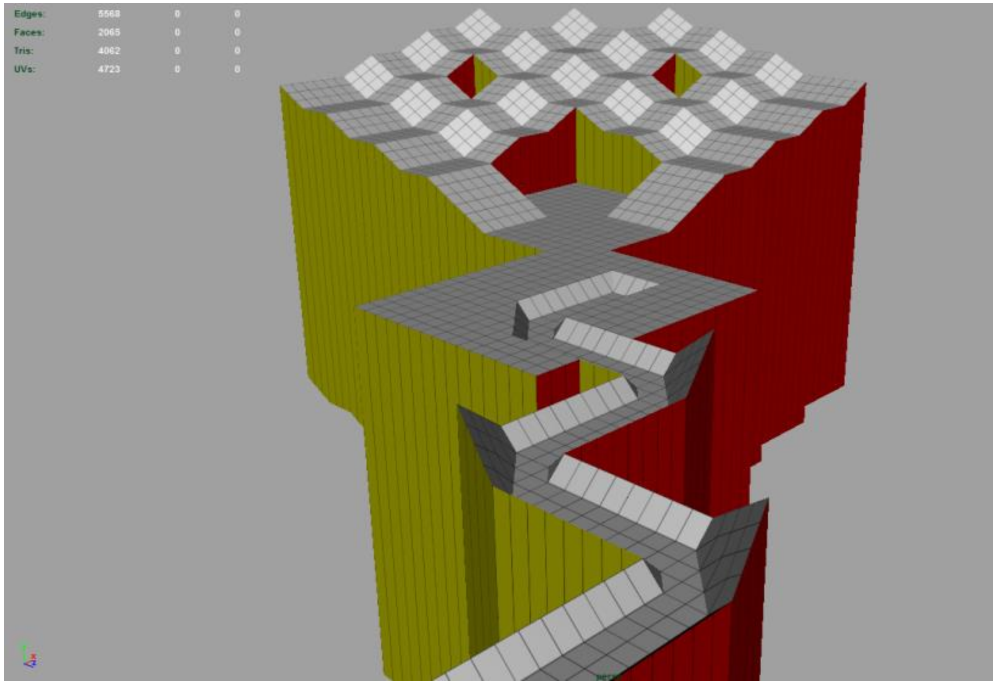
# Game Technology

Lecture 9 – 19.12.2015
Physics 2



Dr.-Ing. Florian Mehm
Dipl-Inf. Robert Konrad

Prof. Dr.-Ing. Ralf Steinmetz
KOM - Multimedia Communications Lab

# Organization

| Date | Lecture | Topic |
|---|---|---|
| 24.10.2015 | 1 | Input and Output |
| | 2 | The Game Loop |
| | 3 | Software Rendering |
| | 4 | Advanced Software Rendering |
| 28.11.2015 | 5 | Basic Hardware Rendering |
| | 6 | Bumps and Animations |
| | 7 | Physically Based Rendering |
| | 8 | Physics 1 |
| **19.12.2015** | **9** | **Physics 2** |
| | **10** | **Procedural Content Generation** |
| | **11** | **Artificial Intelligence** |
| | **12** | **Multiplayer** |
| 23.1.2016 | 13 | Audio |
| | 14 | Compression and Streaming |
| | 15 | Scripting |

# Organization

## Winter break

- No exercise work scheduled for winter break
- → 2 exercises (8, 9)
- Exercise 10 might be released during winter break, but will be due after the last block

## Last block (January 23)

- 3 lectures, maybe a guest speaker
- Document with example questions

## Lecture recordings for this block

- Audio: Today
- Video: December 23
- I‘ll be unavailable until December 23 (forum, mail)

## Exercise results uploaded to „points“ branch

# Background

## „Marbellous“

- Clone of „Marble Madness“ (1984)
- Roll a marble through a maze

## Ball Physics

- Apply force based on key inputs
- Bounce off off the level geometry
- (Fall from too high)

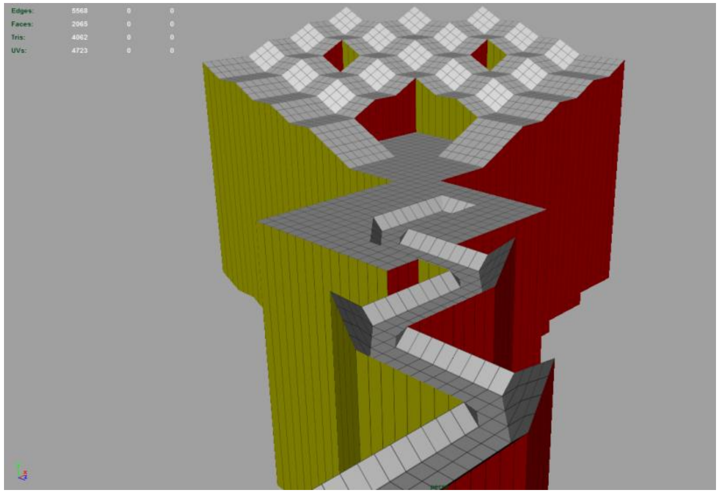
## Level

- Provided as a mesh
- „2D in 3D“

# Adding Physics to "Marbellous"

## Collision with the level

- Level supplied by artist as 3D mesh
- How to handle the collisions with the mesh?



## Friction

- Handle rotations
- Add friction

## Controls

- Apply forces when keys are pressed

## (Camera control)

- Keep the ball in view
- Don‘t follow every single movement

# Hand-placed colliders

## Sometimes good placeholders for objects or level geometry

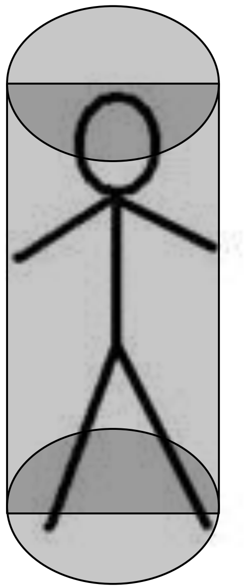
**Planes**

- Ground plane
- Simple intersection

**Boxes**

**Spheres**

**Capsules**

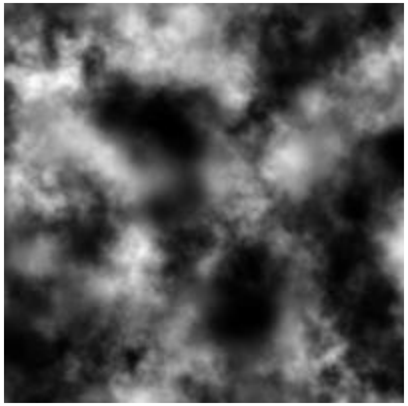


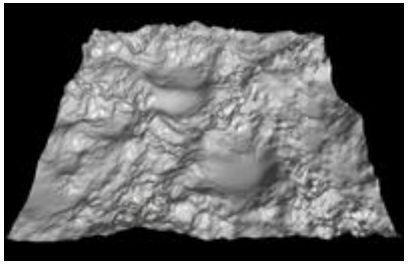
# Height map

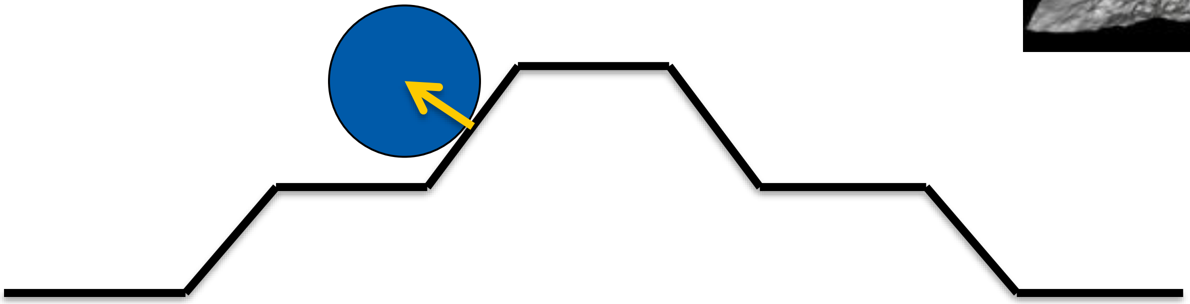
- Supplied as a texture or generated

- Gives height values at grid points

- By interpolating, we can find the height of the mesh under the sphere and the normal





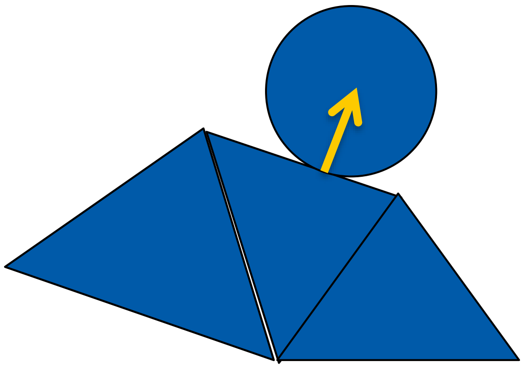


# Using the mesh itself

**Intersection with triangles**

**Check all triangles**

**If sphere intersects a triangle, handle the collision**

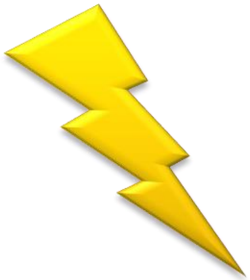


# Using the mesh itself

**Intersection with triangles**

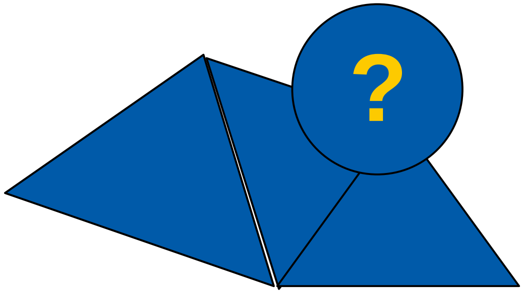
**Check all triangles**

**If sphere intersects a triangle, handle the collision**



**If there are multiple collisions**

- Handle only one (most prominent)
- Handle all



# Separating Axis Test

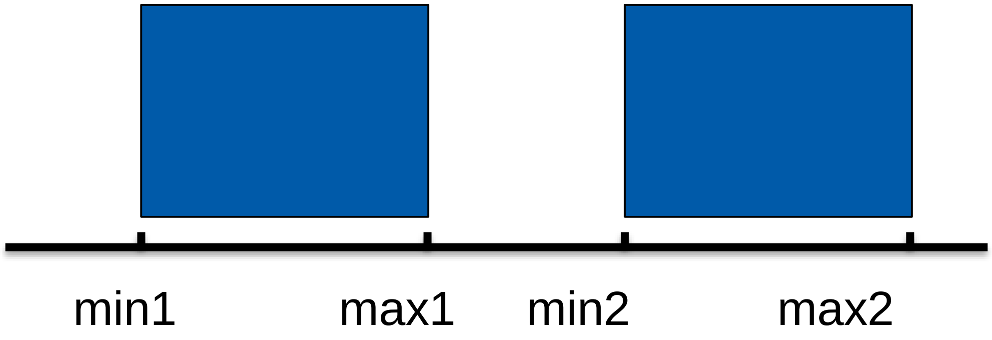
**If two objects are separated, there must be an axis which separates the two objects**

- („Separating Axis Theorem“ → Not a theorem – follows from Hyperplane separation theorem by Hermann Minkowski)
- First mentioned in computer graphics in 1995

# Separating Axis Test

## More exact

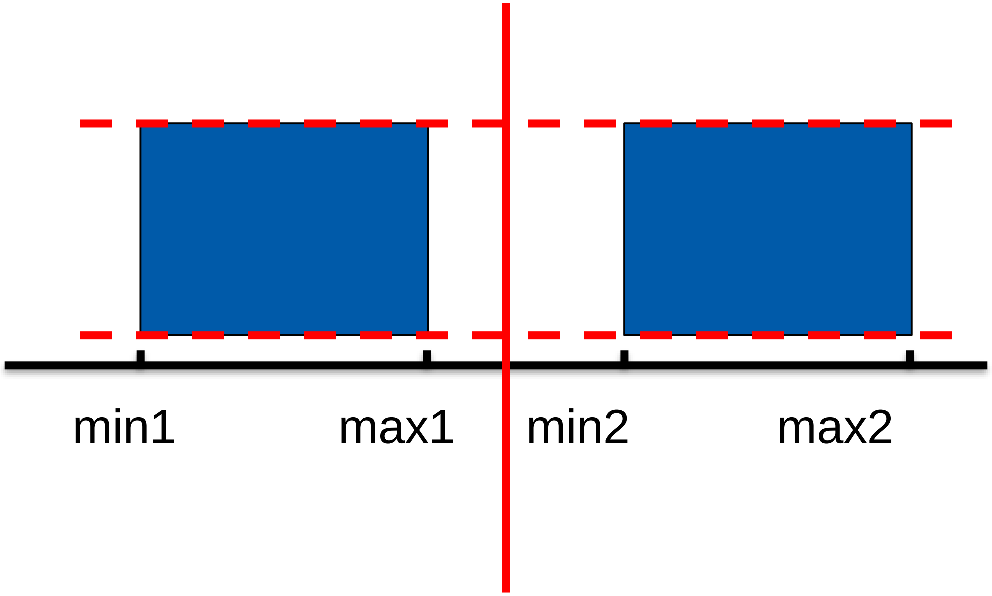
- There must be points P1 and P2 of objects 1 and 2 such that the normal resulting from P2 – P1 is a separating axis
- Separating Axis
  - Project all points of the objects onto the separating axis
  - We get the minimal and maximal points min1, min2 and max1, max2
  - The objects are separated iff max1 < min2 or max2 < min1



# Separating Axis Test

## What the separating axis is NOT

- The separating axis is not a line between the objects
- If the projections overlap, it is not a separating axis
- → This can be referred to as separating plane



# Separating Axis Test

**Infinite set of possible points to test for**

**It can be proven that an upper boundary exists**

- Only the relevant axes have to be tested for
- If separation exists on any axis, the test is done → early out for positive test result
- If no separation exists, we still have to test all combinations of features → no early out for negative tests
  - Can be more efficient to reject the test based on other information, e.g. bounding boxes

**For polygonal objects, the features are**

- Faces
- Edges
- Vertices

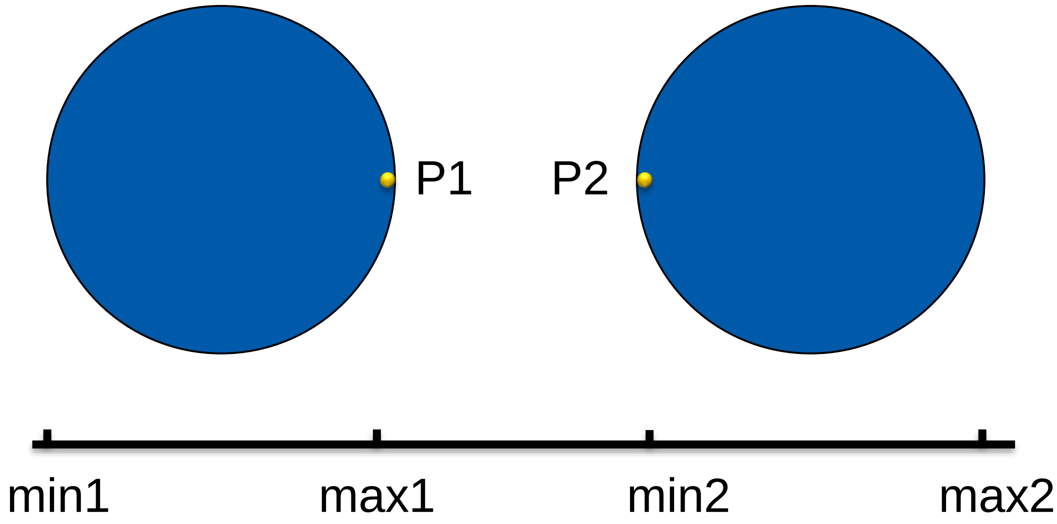
# Separating axes for spheres

TECHNISCHE UNIVERSITÄT DARMSTADT

## Spheres have no clear feature points

## We have already used the separating axis test, though

- The relevant features for two spheres are the two closest points of the spheres
- We find them by finding the axis from one sphere‘s center to the other‘s center
- The intersection test in the previous lecture used this axis for testing intersections



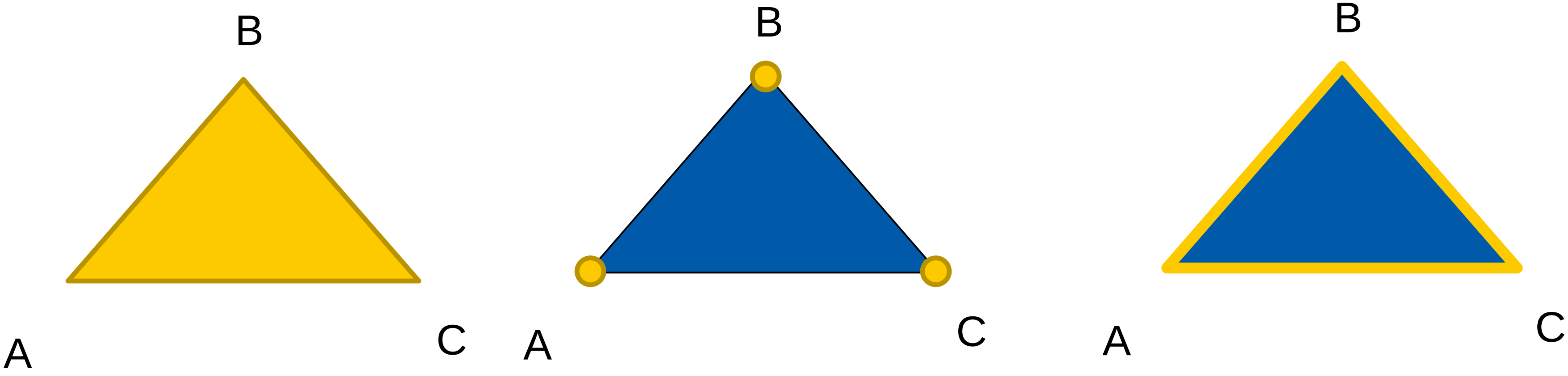
# Triangle-Sphere-Test

## Relevant Features of the Triangle

- Face (x1)
- Vertices (x3)
- Edges (x3)

## Relevant feature of the sphere

- The point on the surface closest to the feature of the triangle



# SAT: Testing the plane of the triangle

**(We have done this test already – need to define the plane)**

**Normal: Use the cross product (very useful for finding normal vectors)**

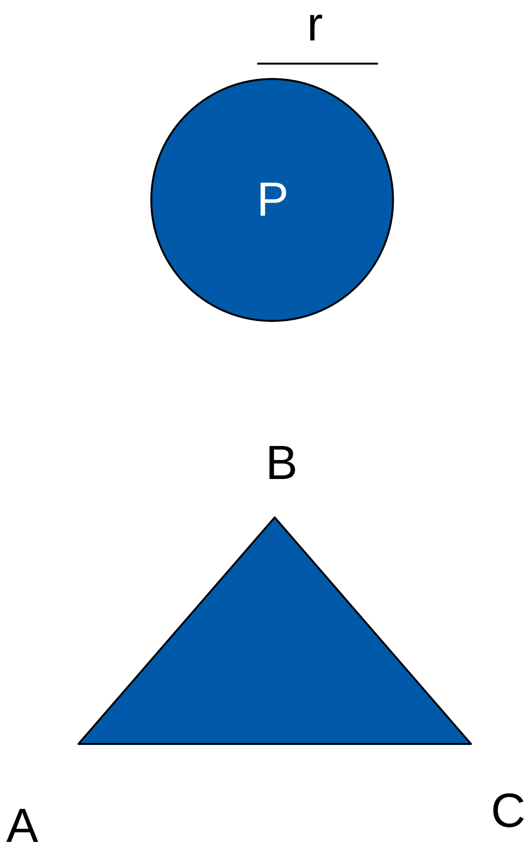
- n = normalize((B – A) x (C - A))

**Distance**

- Insert one of the points into the equation for distance
- n*A – d = 0 (since A lies on the plane of the triangle)
- →n*A = d

**Test for separation**

- Separation = distance(Plane, P) > r



# SAT: Vertices

**Here shown for A (similar for B and C)**

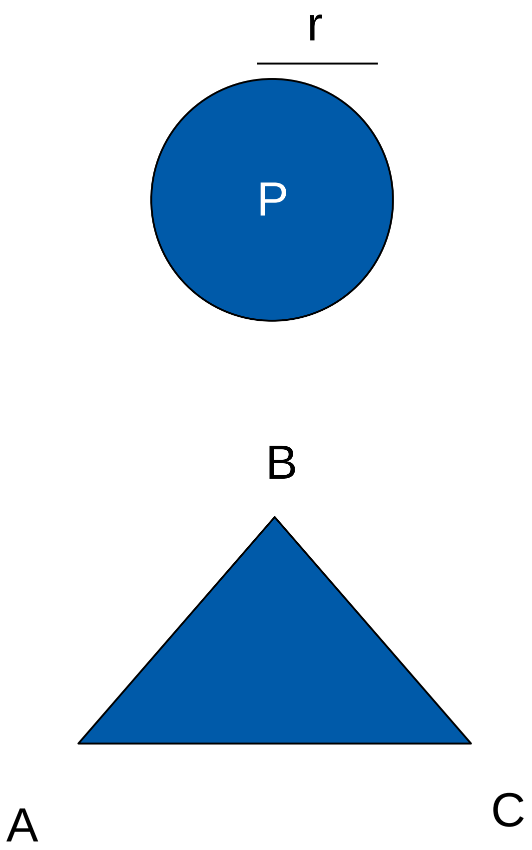
**Finding the sphere‘s feature**

- Along the line from A to P

**Compute distance from A to P**

**Separation (along this axis) iff**

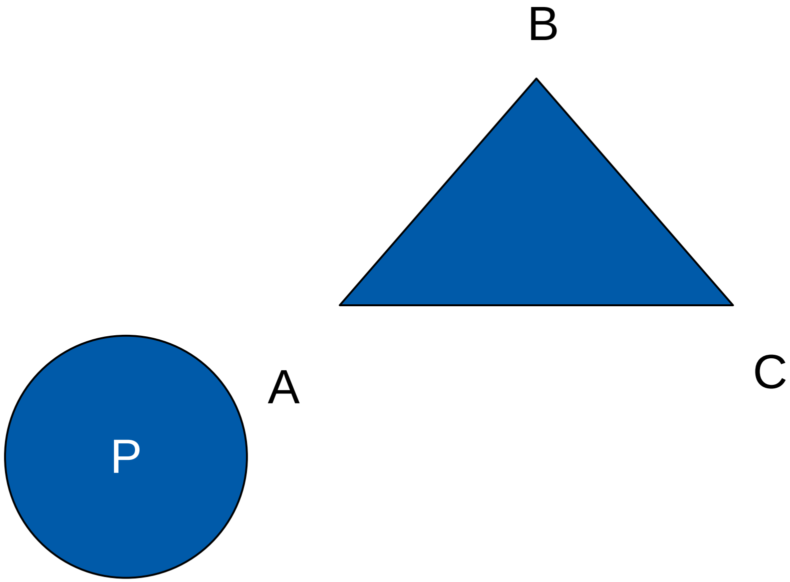
- Distance d > r
- And B and C lie on the opposite side



# SAT: Vertices

**Separation (along this axis) iff**

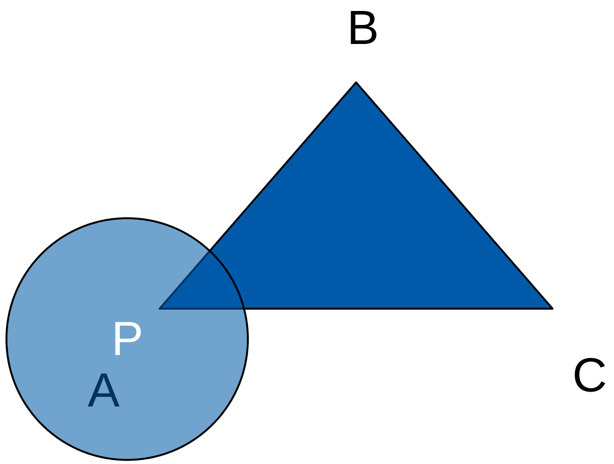
- Distance d > r
- And B and C lie on the opposite side



# SAT: Vertices

**Separation (along this axis) iff**

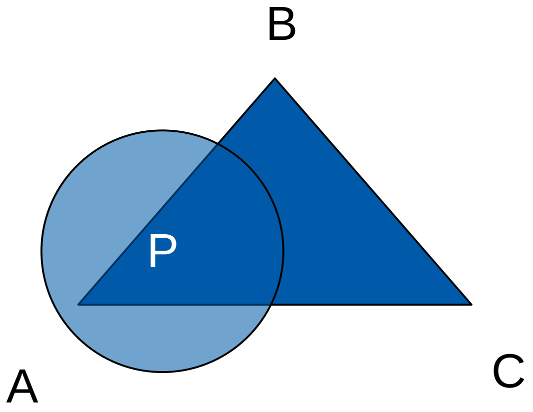
- Distance d > r
- And B and C lie on the opposite side



# SAT: Vertices

**Separation (along this axis) iff**

- Distance d > r
- And B and C lie on the opposite side

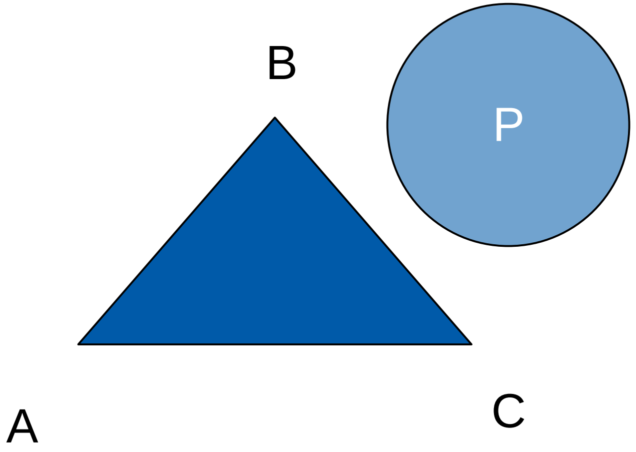


# SAT: Vertices

TECHNISCHE UNIVERSITÄT DARMSTADT

**Separation (along this axis) iff**

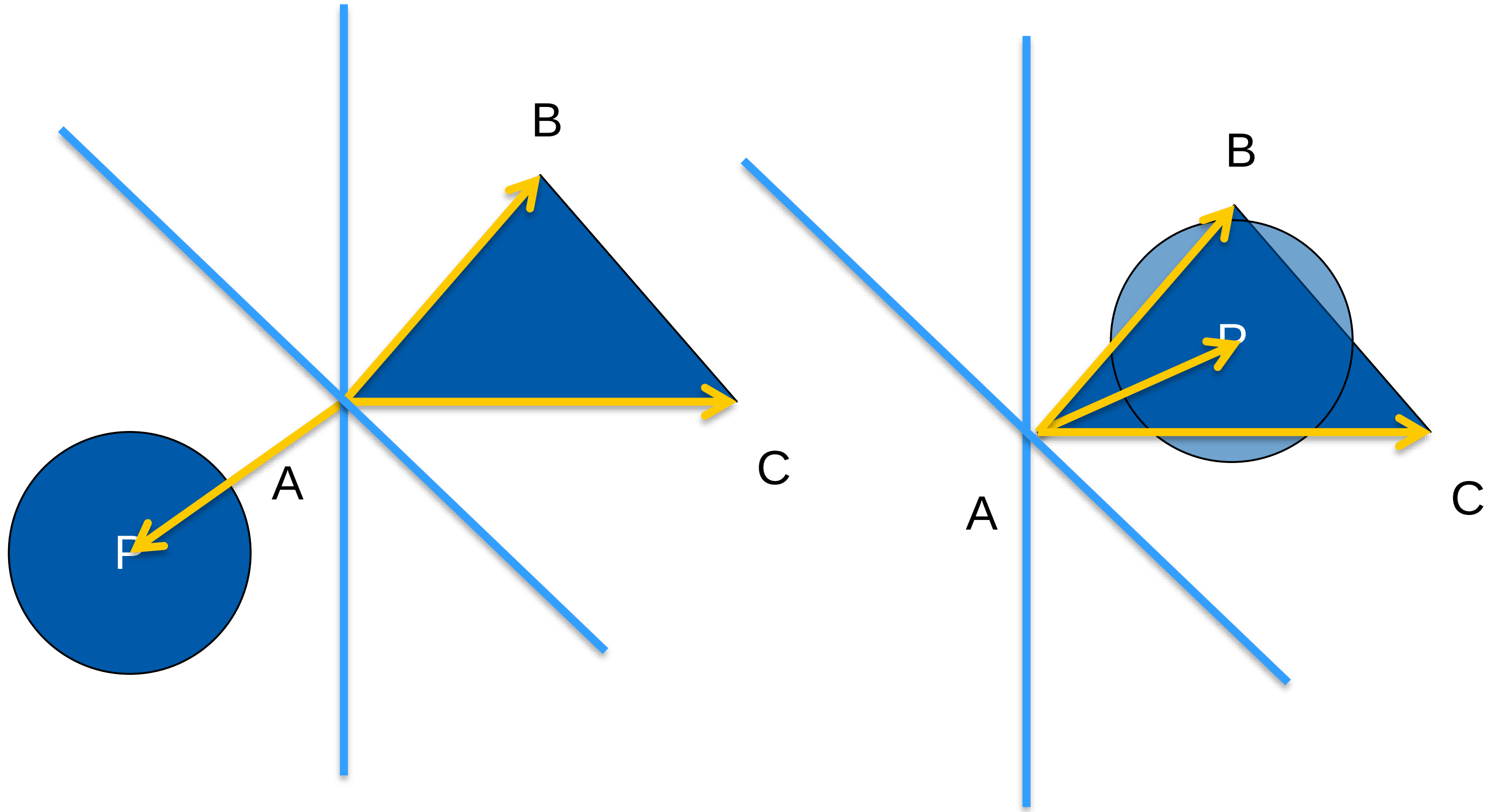
- Distance d > r
- And B and C lie on the opposite side



# Separating Axes Test

**Demonstration of “on the opposite side”**

**Calculate using the dot product of AC and AP, AB and AP**



# SAT: Vertices

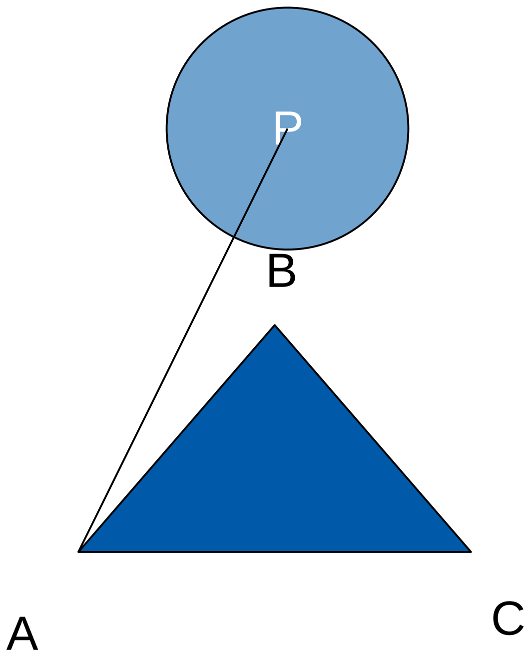
**Separation (along this axis) iff**

- Distance d > r
- And B and C lie on the opposite side

**→ We assume that A-P is the separating axis**

**→ No check if A is the closest point**

**→ B and C might be separating axes!**



# SAT: Edges

**Here shown: AB**

**Find a point for Q for which Q-P is a normal vector orthogonal to AB**

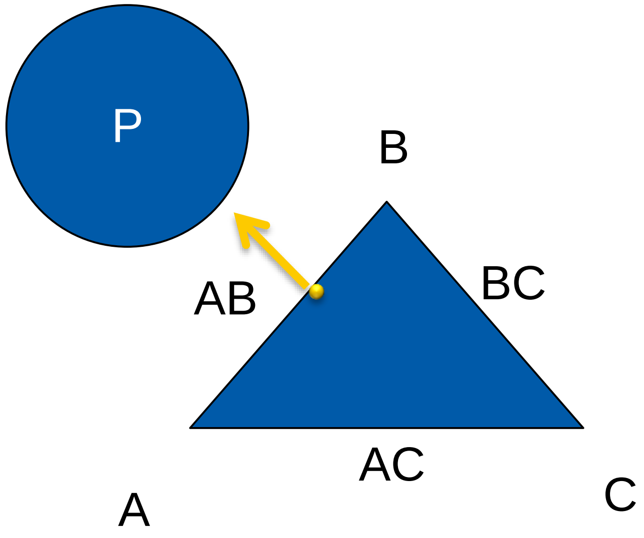
**→ Projection of P onto AB**

→ Use the dot product (ideal for projecting vectors onto each other)

**Determine the distance d of Q to P**

**AB defines a separating axis iff**

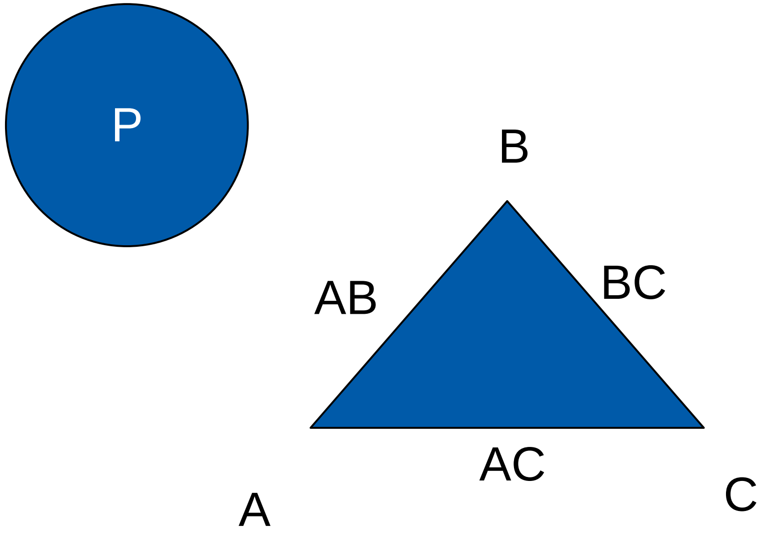
- Distance d > r
- C lies on the other side of the plane through AB with normal PQ



# SAT: Edges

**AB defines a separating axis iff**

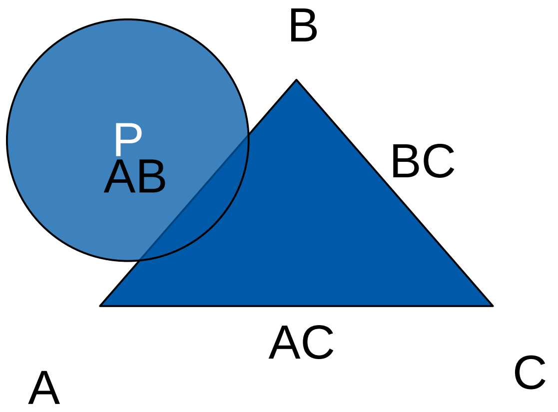
- Distance d > r
- C lies on the other side of the plane through AB with normal PQ



# SAT: Edges

**AB defines a separating axis iff**

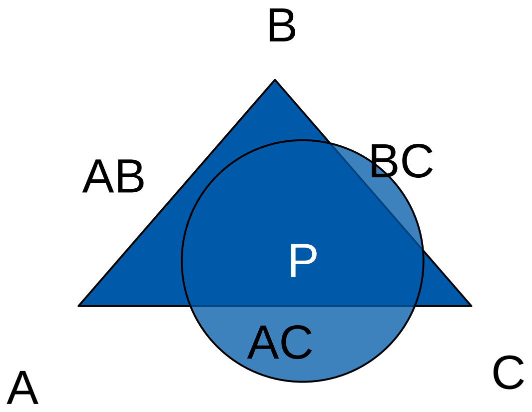
- Distance d > r
- C lies on the other side of the plane through AB with normal PQ



# SAT: Edges

**AB defines a separating axis iff**

- Distance d > r
- C lies on the other side of the plane through AB with normal PQ



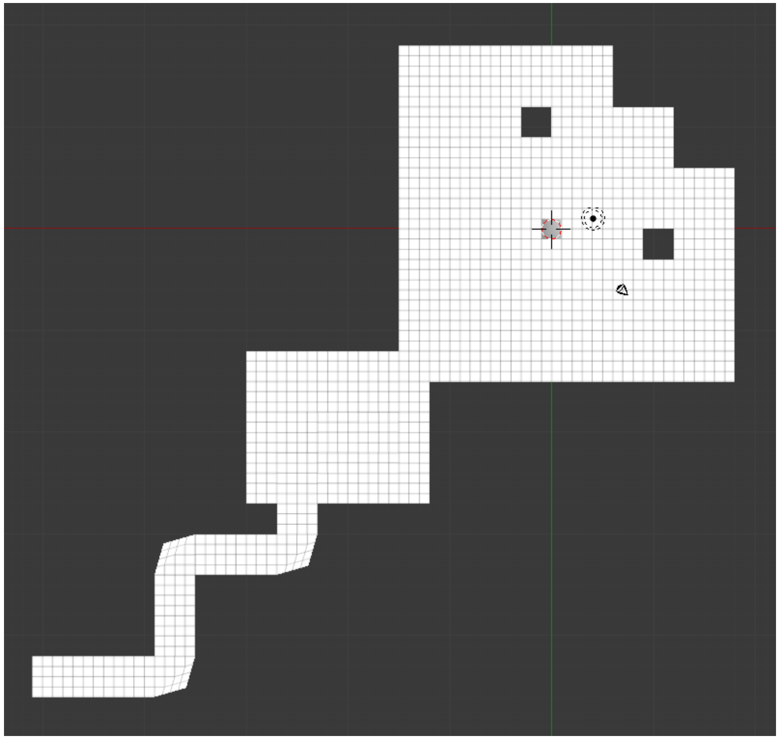
# Speeding the calculation up

## Note: In our case, the level is essentially 2D

- Most of our collisions will be from the top of the level

## Use a space partition

- Regular Grid
- Quadtree
- KD-Tree
- BSP



# Regular grid

## Subdivide space regularly

### E.g. specify

- Cell size in units
- Start point

### For each cell

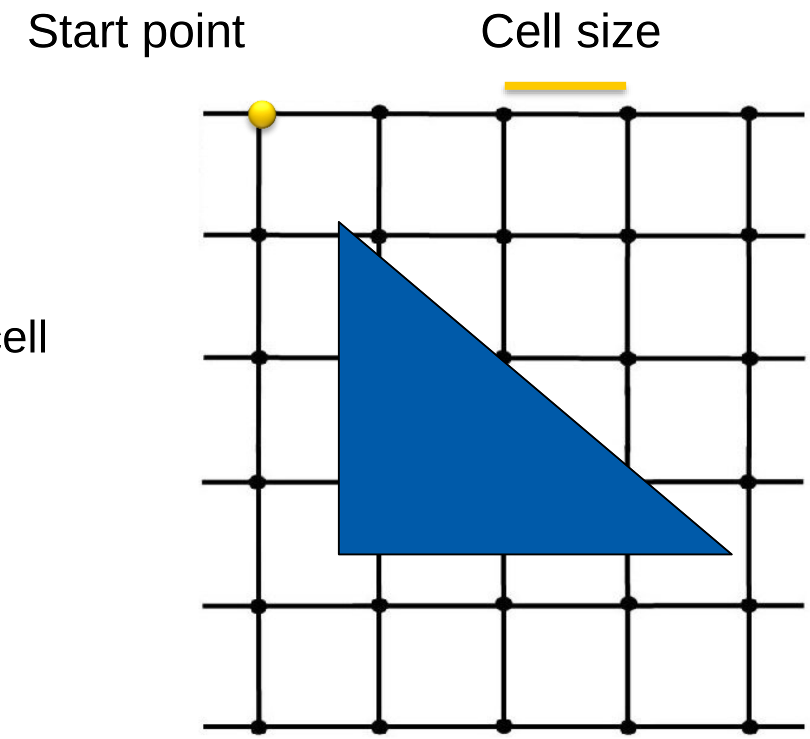
- Test if an object intersects (partly) with the cell
- If so, save a reference to this object
- (Objects can be in several cells)

### Advantages

- Easy to compute
- Lookup of cells is trivial

### Disadvantages

- Sparsity kills the performance
- Clusters

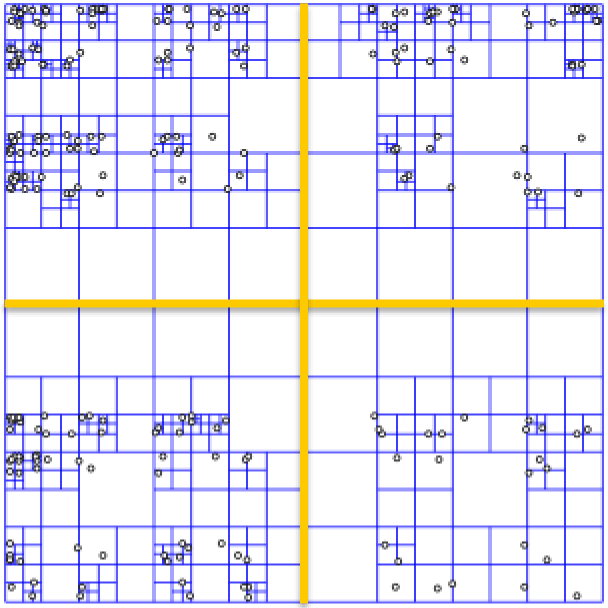


# Quadtree(2D), Octree(3D)

**Start with a rectangular shape**

**Subdivide the space into 4 or 8 subdivisions of equal size if the number of contained objects is too large**

**Until the required minimal number of objects per subdivision is found**



**Advantages**

- Still simple lookup where an object is placed
- Can handle clusters better

**Disadvantages**

- Can cope less with changing number and position of objects

# KD-Tree

**Similar idea to Quad/Octree**

**Subdivide starting from a rectangular shape**

**Choose the subdividing line**

- E.g. median point of the contained objects (cutting them in half)

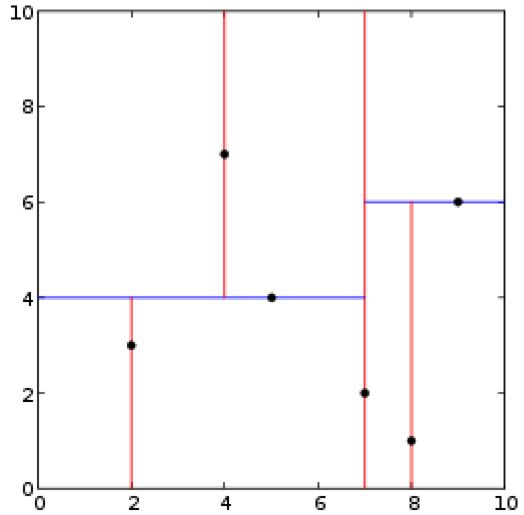
**Alternate axes for subdivision**

**Advantages:**

- Well suited for clusters

**Disadvantages**

- Lookup harder than octree

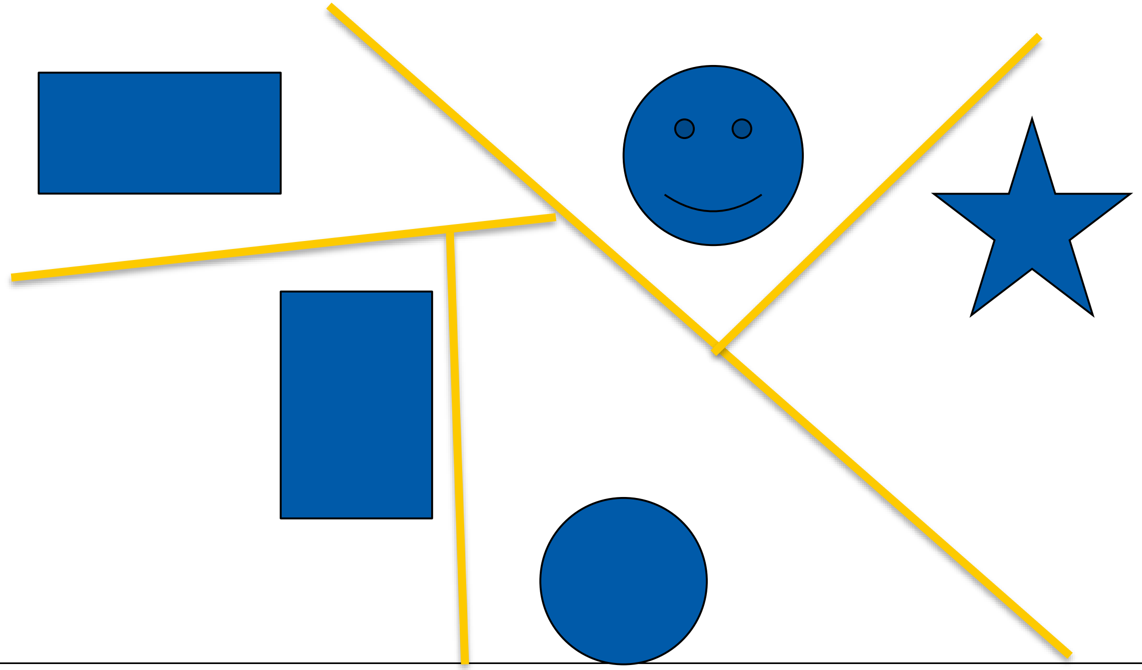


# Binary Space Partition

Generalization of KD-Tree

Subdivide the space into half-spaces with arbitrary planes

Used previously to speed up rendering (Quake Engine)



# Reducing the dimensionality

TECHNISCHE UNIVERSITÄT DARMSTADT

**Many problems in 3D games are essentially 2D**

- Heightmaps
- Top-down shooters
- Real-Time Strategy games
- ...

**In Marbellous, we can expect that**

- No overhangs are present in the level
- The sphere will stay close to the mesh at all times

**If we look at the level from above, we can see that if we put a grid over the game world, only the triangles in the same 2D cell can be possibly colliding**

**→ During initial setup and the lookup, project everything into 2D**

# Lookup

## Saving the triangles

- We should save only the triangles that are contained in the grid cell
- → We need to check intersection between a rectangle and a triangle

## Minimizing storage

- Re-use the vertex and index buffer
- Save only the index of the triangle
  - (Ideally, we will not suffer from too many cache misses, since the goal in the first place is to reject most collision tests early)

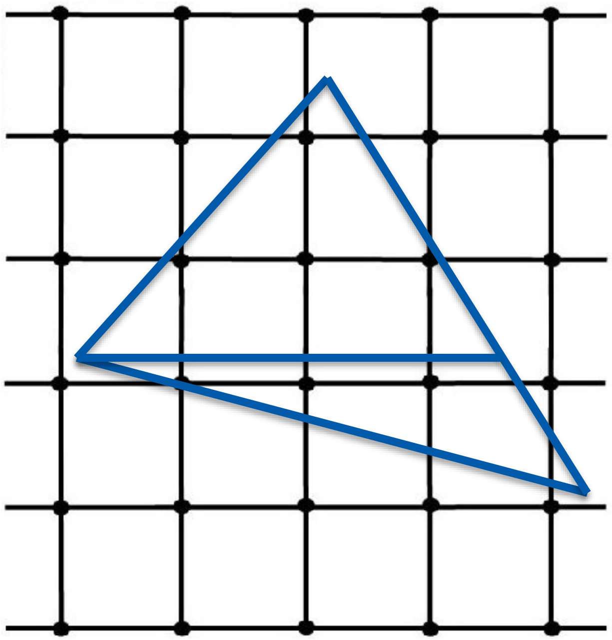
# Intersection between the triangles and the grid

## Re-use the scanline rasterization algorithm

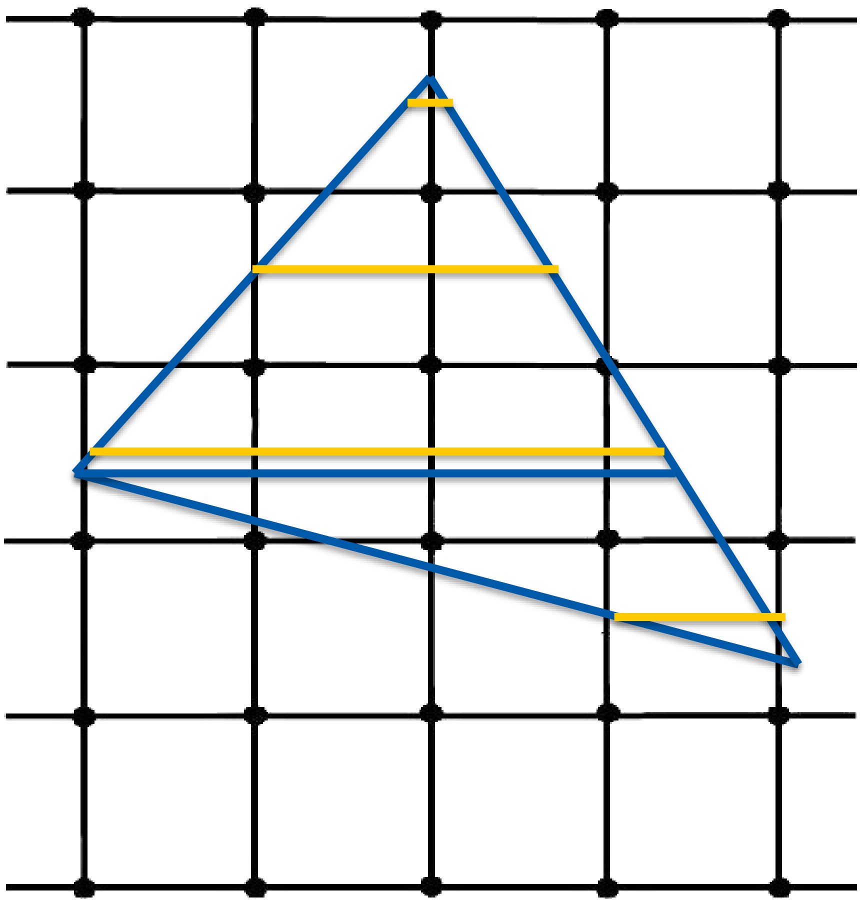
- Very similar task
- But have to watch out due to larger cell size

## Original algorithm

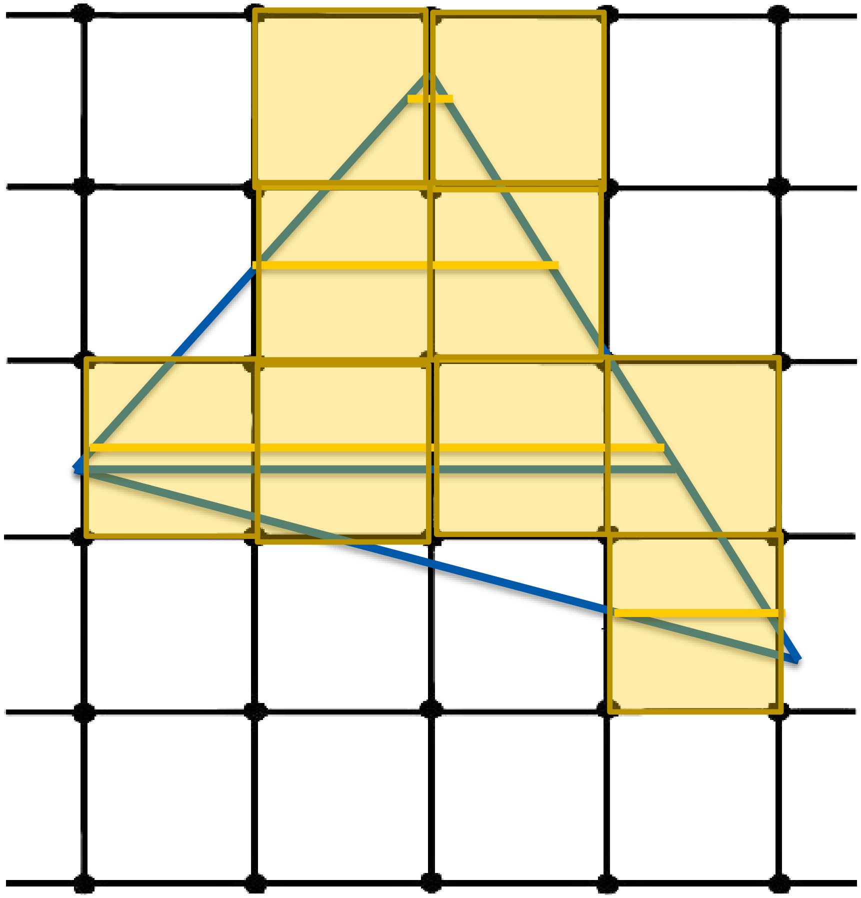
- Find edge longest with biggest ydif
- Fill lines between long edge and other edge 1
- Fill lines between long edge and other edge 2



# Triangle Rasterisation



# Triangle Rasterisation



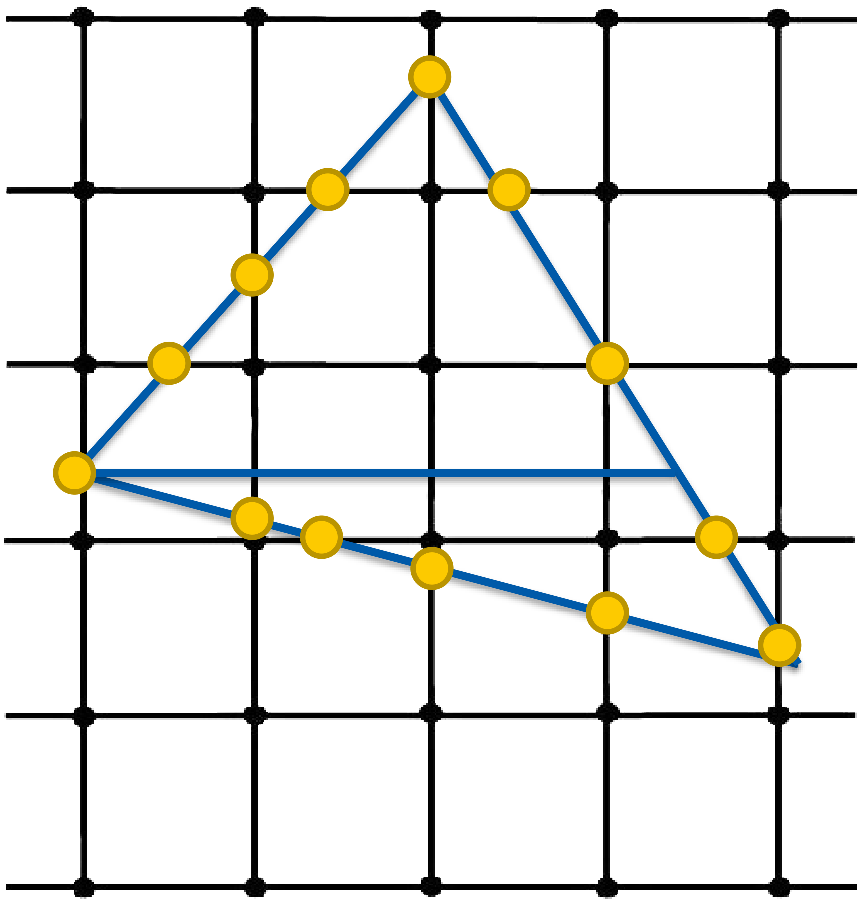
# New algorithm

**Calculate intersection with all grid lines**

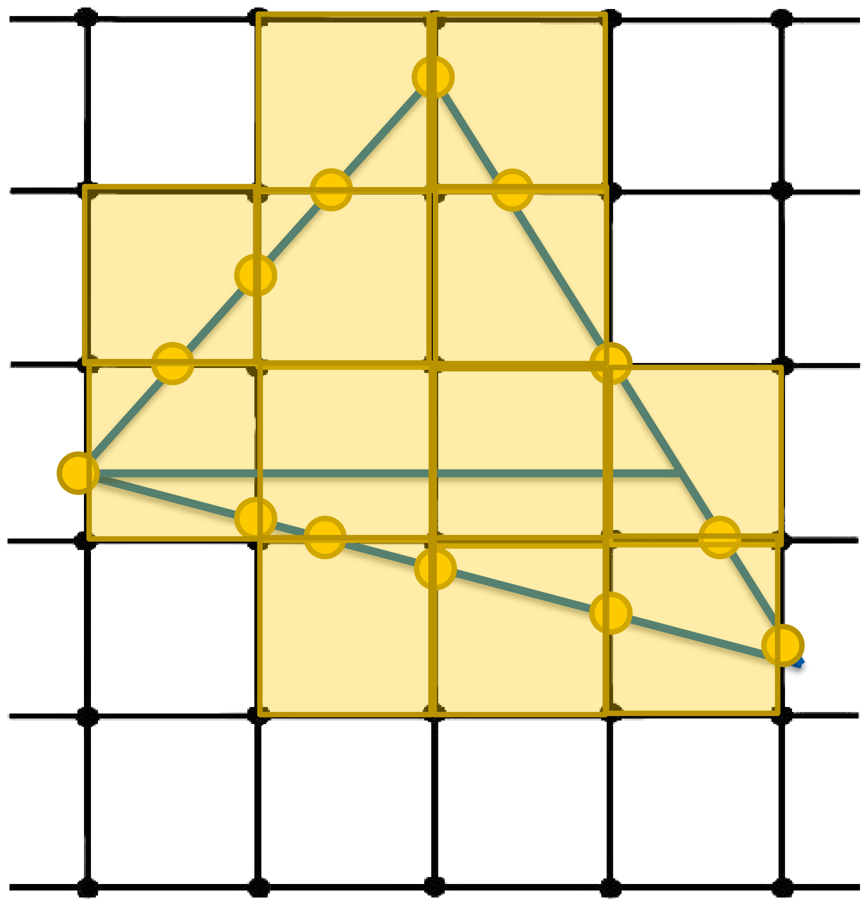
**For each row**

- Left extent is the minimal intersection point
- Right extent is the maximal intersection point

# Triangle Rasterisation



# Triangle Rasterisation

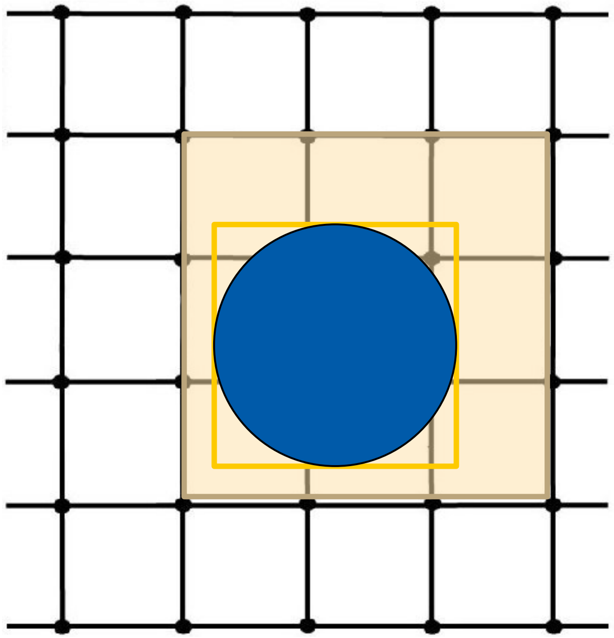


# Intersection between the sphere and the grid

Use the bounding box of the sphere

Defined by the extents in the x-z-Plane

(Or implement rectangle-sphere intersection)



# Is it worth it?

No (at least not for our exercise)

On a Core2 Duo @2.7 GHz, the intersection with the mesh takes about 0.908 ms in Release mode

But, for production code, larger meshes and more objects, it could become relevant

(Triangle-Sphere Intersection implemented with optimized code by Christer Ericson, http://realtimecollisiondetection.net/blog/?p=103)

# Broad Phase vs. Narrow Phase

**Broad Phase**

**Rule out as many possible collisions**

**Only call narrow phase if separation can not be proven here**

**Reduce the problem**

- Use spatial data structures (grid, octree, etc.)
- Use bounding volumes (and bounding volume hierarchies)

**Narrow Phase**

**Check for exact collisions**

**Use exact tests**

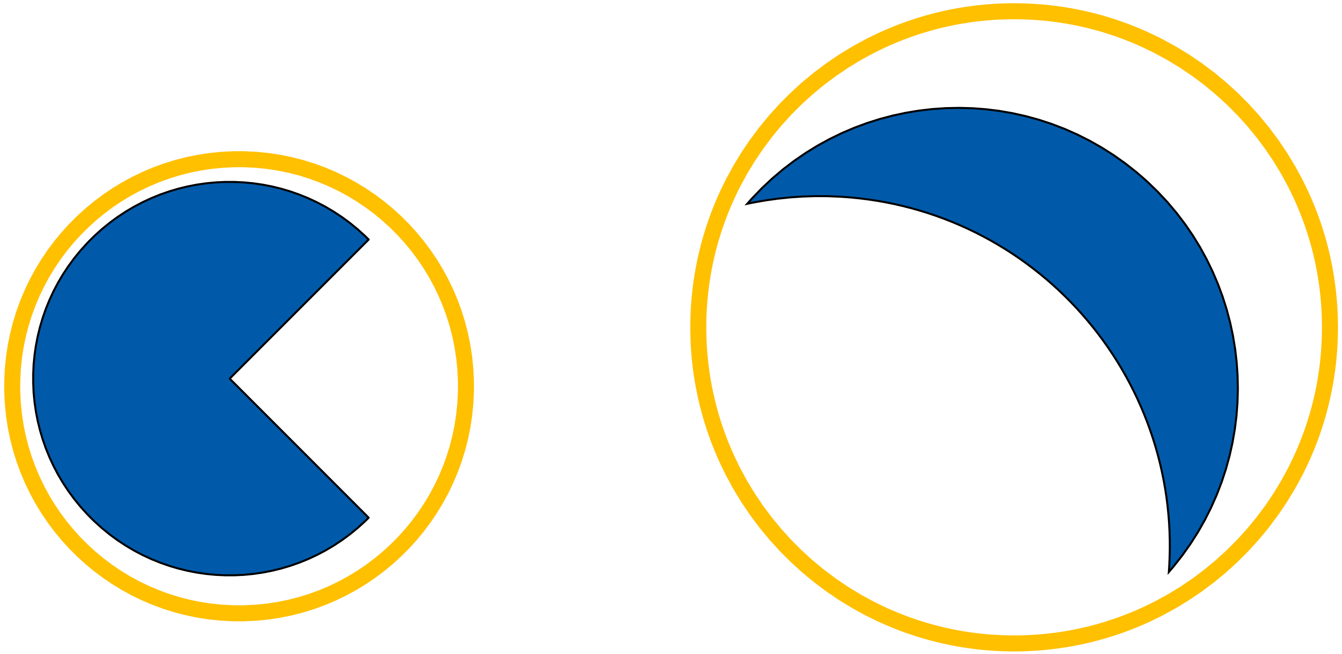
- E.g. based on SAT

**Should be much slower than broad phase and therefore seldomly called**

**Provide collision data to resolver**

# Broad phase

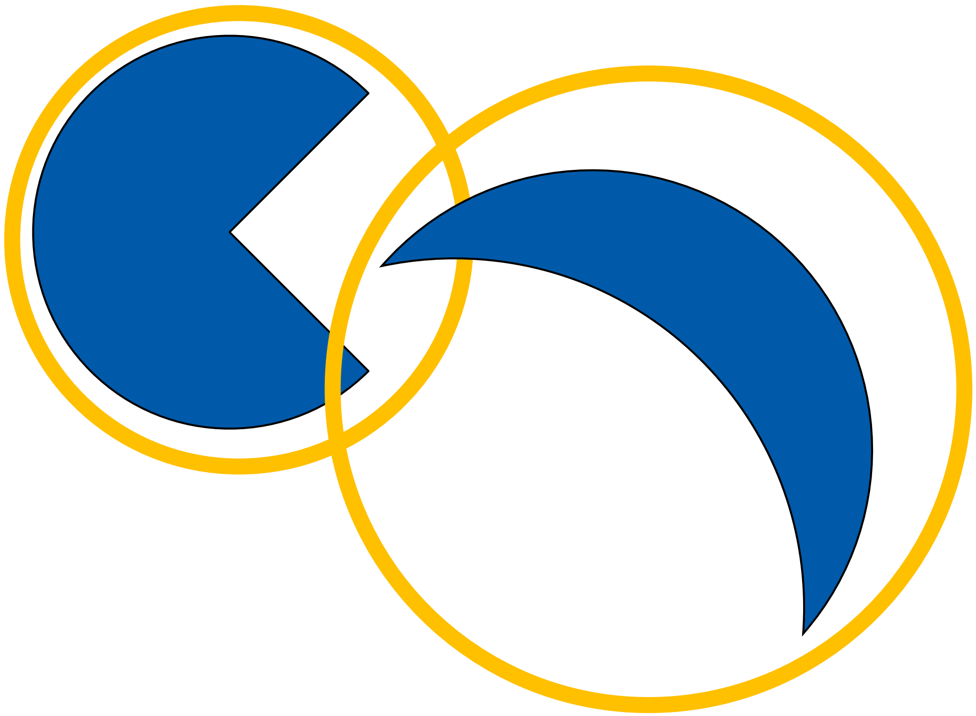
## No collision possible (surrounding bodies are not overlapping)



# Broad phase

**False positive, need to do more detailed collision test**

**→ Go into narrow phase**



# Time Handling

## Fixed Time Step

- Explicit Time Step → Our method
- (Semi-)Implicit Time Step Method
  - Try to predict the times of collisions and handle them at the beginning

## Adaptive Time Step

- Retroactive Detection
  - If there is interpenetration at t +deltaT, use deltaT *= 0.5 and retry
- Conservative Advancement
  - Predict the next time of collision
  - Advance to this time

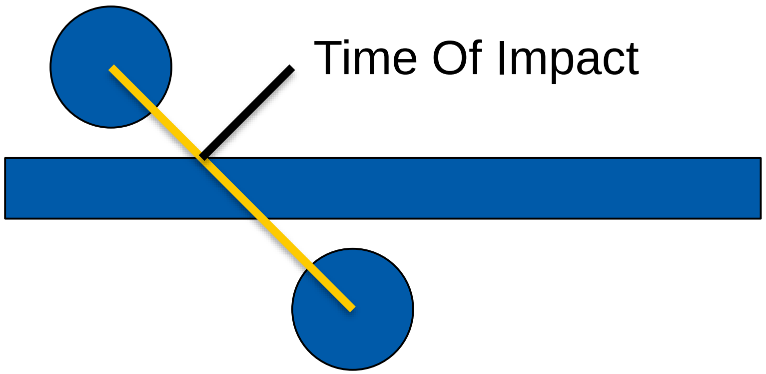
# Continuous Collision Detection

**Check if an object moved through another in the frame**

- On one side before, on one side after
- Swept shape algorithms

**Time of impact ordering**

**Go to time of impact, resolve there**

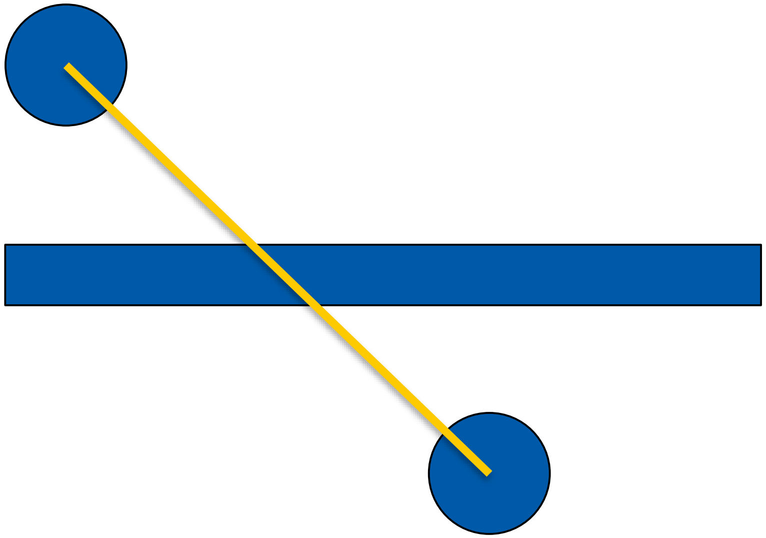


# Speculative Contact

TECHNISCHE UNIVERSITÄT DARMSTADT

**Calculate the distance to the collider**

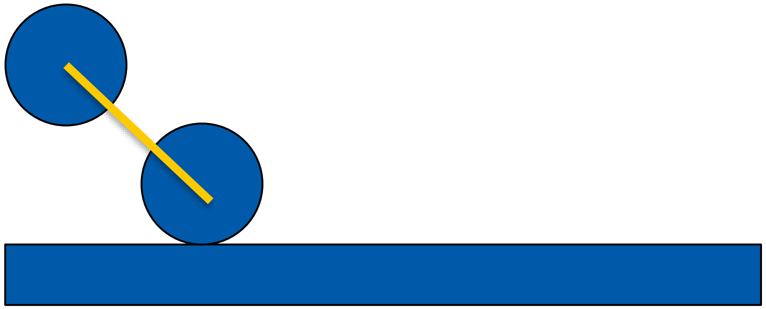
**Remove just enough velocity so they touch in the next frame**



# Speculative Contact

**Calculate the distance to the collider**

**Remove just enough velocity so they touch in the next frame**



# Constraints

## Stiff constraints

- Keep objects at an exact length compared to each other
- E.g. when attached to a steel cable

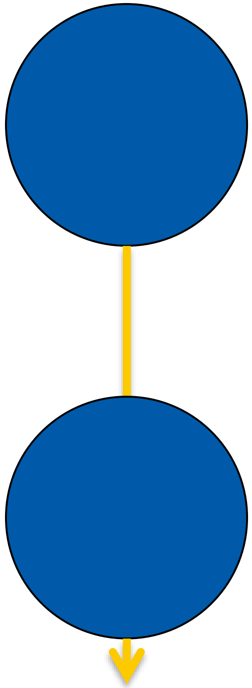

## Springs

- Variable length between objects
- E.g. when attached to a bungee rope

# Stiff Constraints - Rods

**Distance between two objects is determined to stay constant**

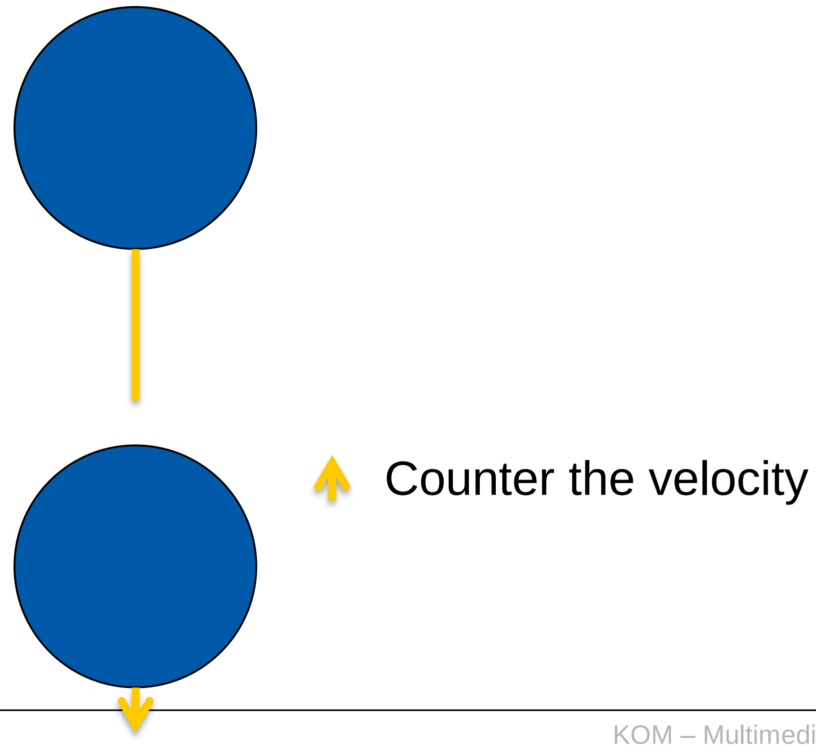
**→ Separating Velocity between the two objects along the vector from one to the other should be 0 at all times**



# Stiff Constraints - Rods

**Distance between two objects is determined to stay constant**

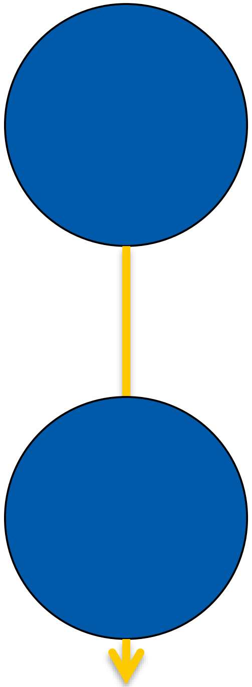
**→ Separating Velocity between the two objects along the vector from one to the other should be 0 at all times**



# Stiff Constraints - Rods

**Distance between two objects is determined to stay constant**

→ **Separating Velocity between the two objects along the vector from one to the other should be 0 at all times**

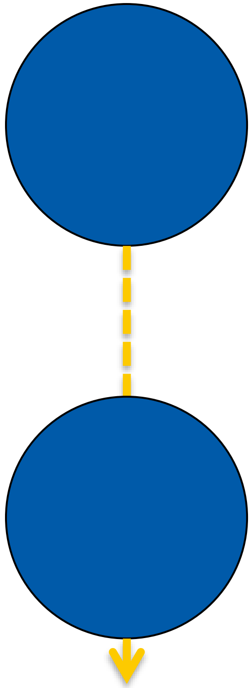


# Spring Constraints

## Model a spring between two objects (one might be stationary)

### Spring force

- Rest length (no force)
- Stiffness
- (Breaking point)



# Hooke‘s Law

**$F = -k * (l - l0)$**

**F: Spring force**

**k: Spring constant (stiffness)**

**l: Current length of the spring**

**l0: Rest length of the spring**

**Apply the resulting force to the objects that are attached**

**(One might be immovable)**

# Stiffness

**Also a property of numerical systems**

**The stiffer, the more problems we face → exploding systems**

**J. D. Lambert : “If a numerical method with a finite region of absolute stability, applied to a system with any initial conditions, is forced to use in a certain interval of integration a steplength which is excessively small in relation to the smoothness of the exact solution in that interval, then the system is said to be stiff in that interval.”**

# Particle networks

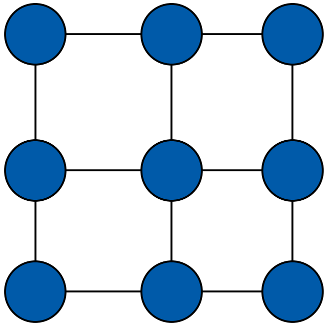
**Connect multiple particles with springs**

**Approximation for deformable objects**

**Often used for cloth**

**Problems/Challenges**

- Stiff constraints
- Self-intersections
- Stability



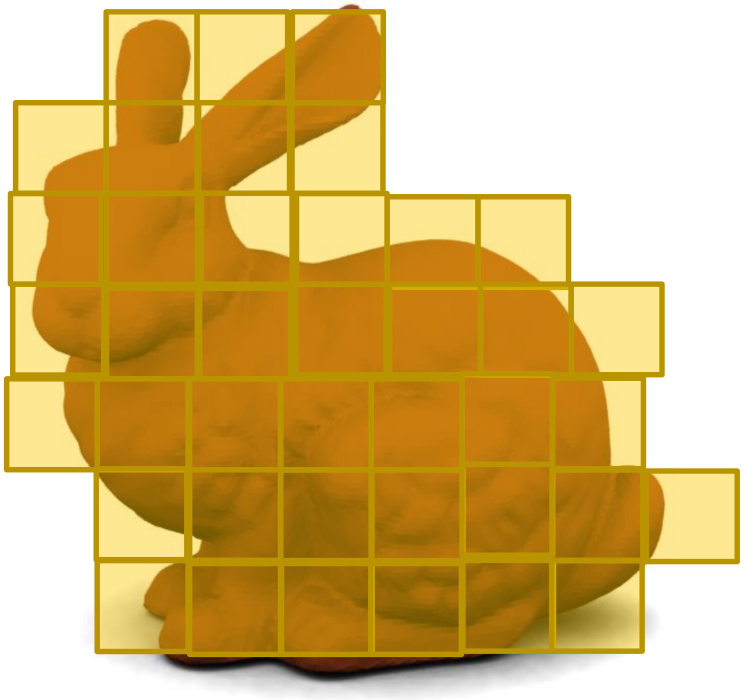
# Deformable objects

**Generalization of particle networks**

**Finite Element Method from Mechanics**

**Model forces inside the object**

- Stress
- Strain



**Gasses, Liquids**

- Discretize into a vector field
- Calculate flow by solving the Navier-Stokes-Equations

# Collision handling schemes

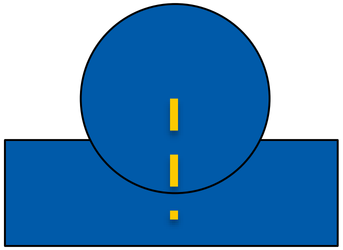
TECHNISCHE UNIVERSITÄT DARMSTADT

## Impulse-based Micro-Collisions

- What we are using

## Spring-Based

- Insert a spring at the point where the collision is detected
- Forces the objects out again



## Constraint-Based

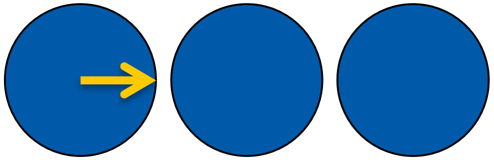
- Formulate the collisions as violations of constraints

# Sequential Impulses

## Aka. Propagating Impulses

## Stability

- Add iterations
- Solve impulses in order of importance



## Adaptive schemes

- Few, „large“ contacts → need fewer iterations
- Many contacts → need more iterations

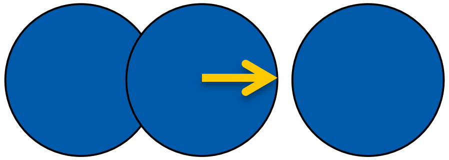
# Sequential Impulses

TECHNISCHE UNIVERSITÄT DARMSTADT

## Aka. Propagating Impulses

## Stability

- Add iterations
- Solve impulses in order of importance



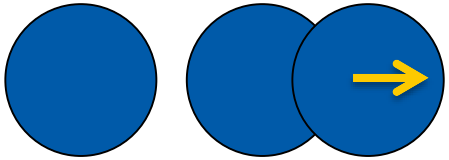
## Adaptive schemes

- Few, „large“ contacts → need fewer iterations
- Many contacts → need more iterations

# Sequential Impulses

TECHNISCHE UNIVERSITÄT DARMSTADT

## Aka. Propagating Impulses



## Stability

- Add iterations
- Solve impulses in order of importance

## Adaptive schemes

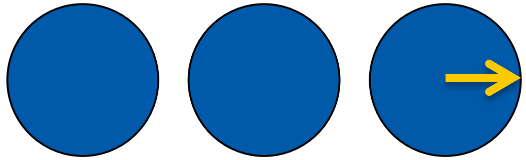
- Few, „large“ contacts → need fewer iterations
- Many contacts → need more iterations

# Sequential Impulses

## Aka. Propagating Impulses

## Stability

- Add iterations
- Solve impulses in order of importance



## Adaptive schemes

- Few, „large“ contacts → need fewer iterations
- Many contacts → need more iterations

# Constraint-based

## Constraint Vector

- For each collision or constraint
- Equality constraint
  - Objects should stay at a fixed relative position
- Inequality constraints
  - E.g. for separating objects after collisions

## For each collision, add a constraint to the constraint vector

## Results in a large system of equations

## Solve via Linear Complementarity Problem (LCP)

# Other integrators



## Runge Kutta 4th order (RK 4)

- Approximate from 4 values

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$t_{n+1} = t_n + h$$

$$k_1 = f(t_n, y_n)$$

$$k_2 = f\left(t_n + \frac{h}{2}, y_n + \frac{1}{2}k_1 h\right)$$

$$k_3 = f\left(t_n + \frac{h}{2}, y_n + \frac{1}{2}k_2 h\right)$$

$$k_4 = f(t_n + h, y_n + k_3 h)$$

## Velocity-less Verlet integration

- Uses no explicitly saved velocity
- Instead, uses position difference between this and previous calculation

- x(t + deltaT) = 2 * x(t) – x(t – deltaT) + deltaT^2 * a

# Rotation

## Angular Velocity, Acceleration

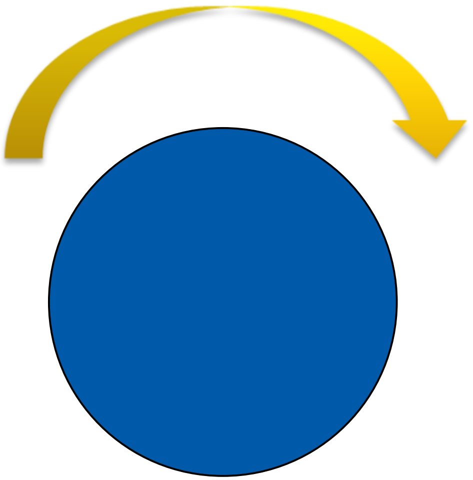
- Save as additional properties
- Velocity: 3-Vector, Rotations around x, y, z axis
- Acceleration: Change in angular velocity

## Mass Moment of Inertia

- Property that resists the change in angular velocity

## Torque

- Force acting off-center



# Torque

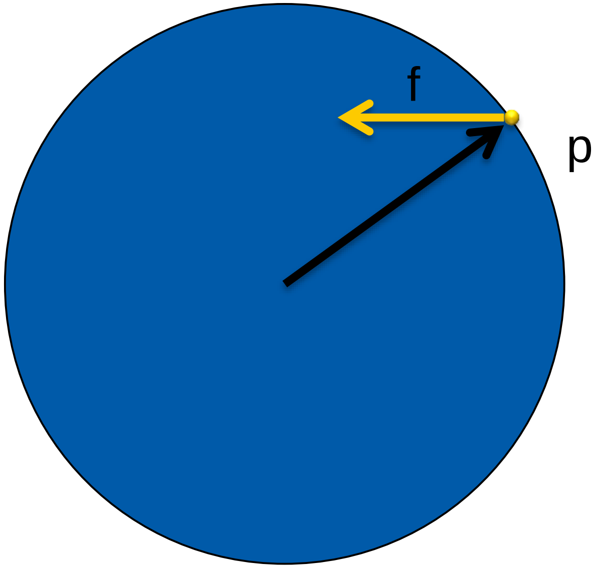
**torque = p x f**

**p is the point of application**

**f is the force applied**

**Note: If p and f are in the same direction**

**→ No torque**



# Mass Moment of Inertia

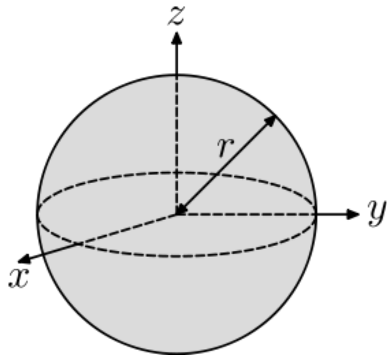
Inertia Tensor – Generalized version of a matrix

For purposes of games, most often 3x3

Diagonal Matrix for moments of inertia about x, y, z-axis

Off-center entries encode product of inertia

See http://en.wikipedia.org/wiki/List_of_moments_of_inertia




$$I = \begin{bmatrix} \frac{2}{5}mr^2 & 0 & 0 \\ 0 & \frac{2}{5}mr^2 & 0 \\ 0 & 0 & \frac{2}{5}mr^2 \end{bmatrix}$$

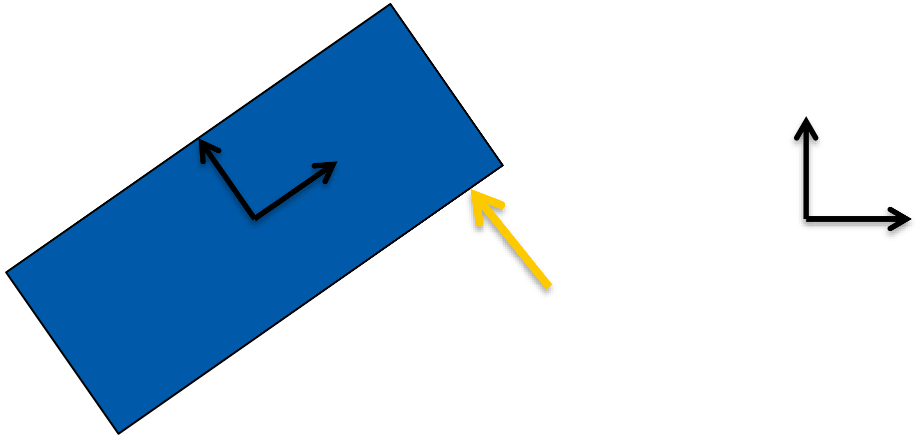
# Inverse Inertia Tensor

**Remember the calculation of forces**

**F = m * a → a = F / m**

**We need the inverse of the inertia tensor for the equivalent formula**

**Additionally, need to transform to the world coordinate system**
**→ Torques given in world coordinates**



# Integration

**Add an accumulator for Torque**

**(D'Alambert's Principle also works here)**

**Add all forces to linear accumulator**

**Calculate torque for each force**

**Add torques to torque accumulator**

**Integration**

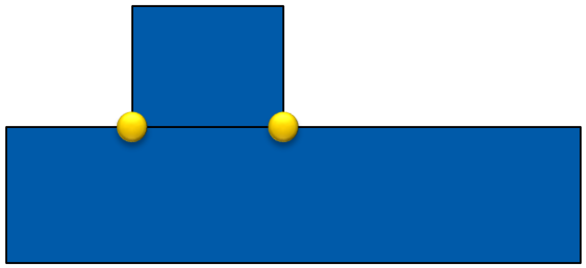
- Multiply inverse mass moment of inertia with sum of torques

# Handling non-spherical rigid bodies

**E.g. a box**

**Can collide with any feature**

- Face
- Vertex
- Edge



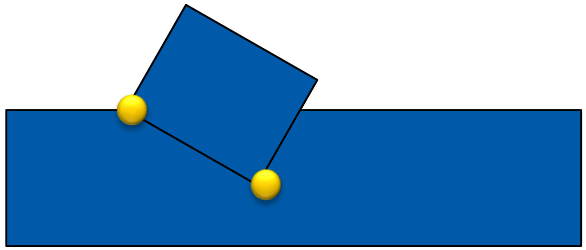
**If we handle only one feature, the others would sink**

**Sequential impulses**

- One part starts sinking into the floor
- Push up → Rotation
- Continue
- Needs iterations to get stable

# Handling non-spherical rigid bodies

## E.g. a box



## Can collide with any feature

- Face
- Vertex
- Edge

## If we handle only one feature, the others would sink

## Sequential impulses

- One part starts sinking into the floor
- Push up → Rotation
- Continue
- Needs iterations to get stable

# Friction

TECHNISCHE UNIVERSITÄT DARMSTADT

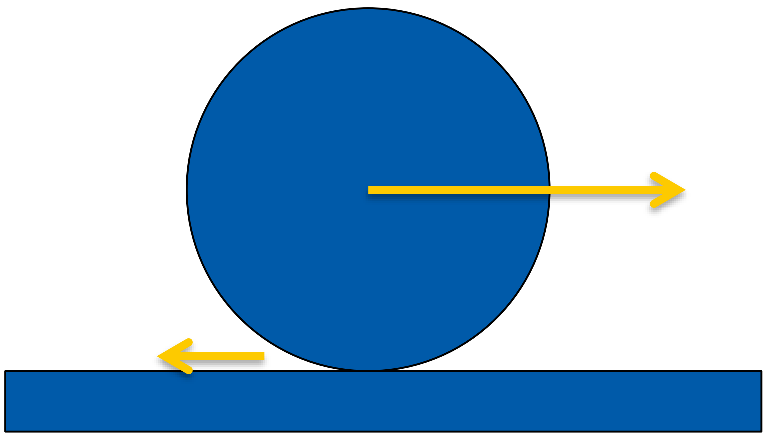
**In the previous exercise, our spheres slided over the floor**

→No rotation

→They came to rest because of dampening and not friction

**Friction resists the spheres at the point of contact with the floor**

- Rolling along the floor
- Different coefficients
  - Ice
  - Smooth floor
  - Sand
  - …



# Coulomb‘s Law



**Depends on**

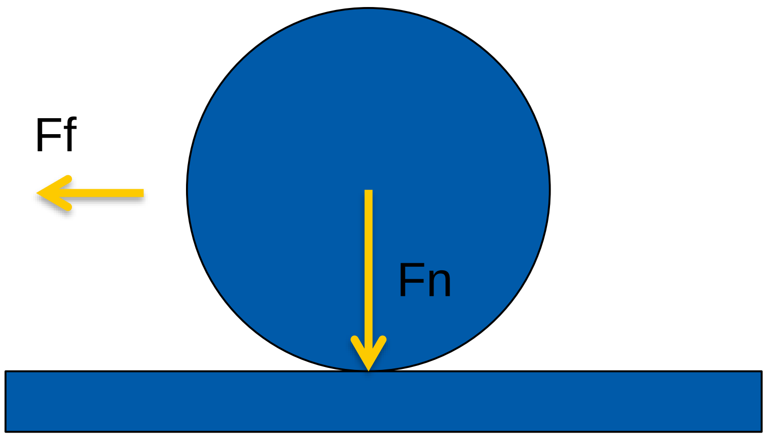
- normal force that presses the surfaces together
- coefficient of friction
  - Most dry materials have a coefficient of friction of 0.1 to 0.6

**Ff: Friction force**

**$\mu$: Coefficient of friction**

**Fn: Normal force**

$$F_f \leq \mu F_n$$




**In 3D**

- Tangential plane
- Force lies in this plane

# Friction

## Static friction

- Keeps objects in place
- Start moving when the limit is overcome
- f_static <= k_static * |r|
- k_static: Constant for friction between the involved materials
- r: Reaction force of the ground at the point of contact

## Dynamic friction

- Force between the objects while they are sliding across
- f_dynamic = -v_planar * k_dynamic * |r|
- v_planar: The velocity of the object across the surface
- k_dynamic: Constant for dynamic friction

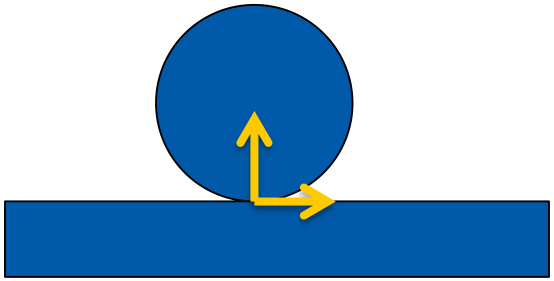
# Contact basis

**Calculating friction requires us to calculate the velocity along the contact**

**Handle collision with a collision basis**

**3 orthonormal vectors**

- x: collision normal
- y, z: Perpendicular to x, define the plane of the contact



# Calculating the contact basis

x: Contact normal

y: Choose a vector perpendicular to x

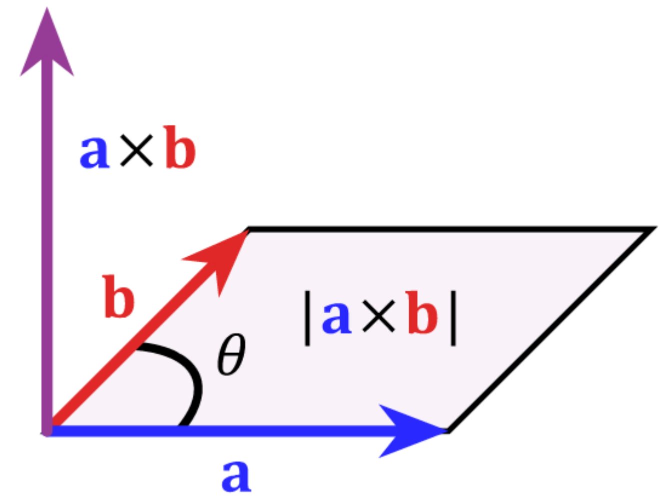
Cross product: A x B is perpendicular to A and B (unless they are parallel)

Use an axis, e.g. global z

y = x x (0, 0, 1)

Choose third vector to be perpendicular to x and y

z = x x y



# Velocity resolution

Find contact basis

Calculate the change in velocity of the contact paint per unit impulse

Invert this to get a way to counter velocities

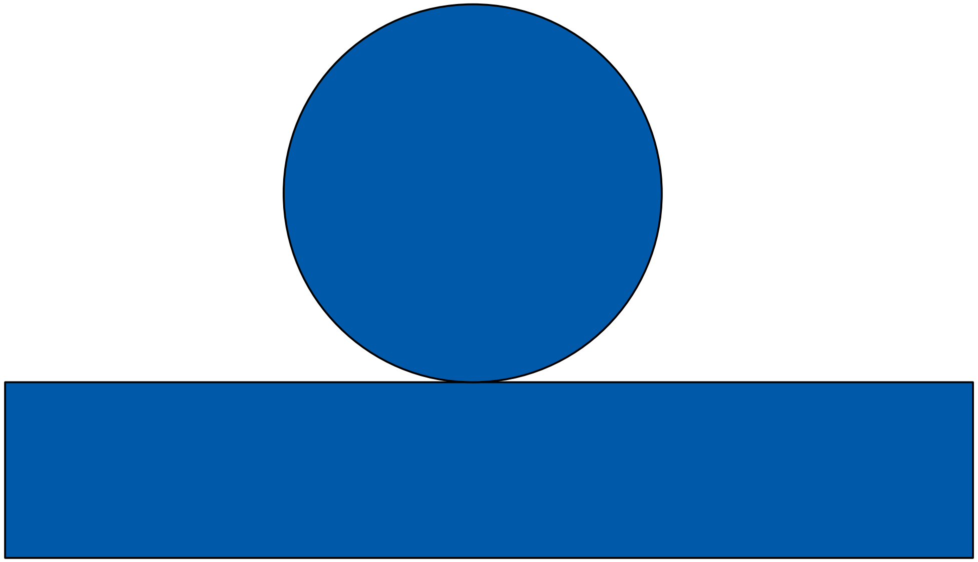
Calculate the x-term of the impulse (along the collision normal – our old calculation)

Calculate the y and z-terms of the impulse (for friction)

Apply the impulse

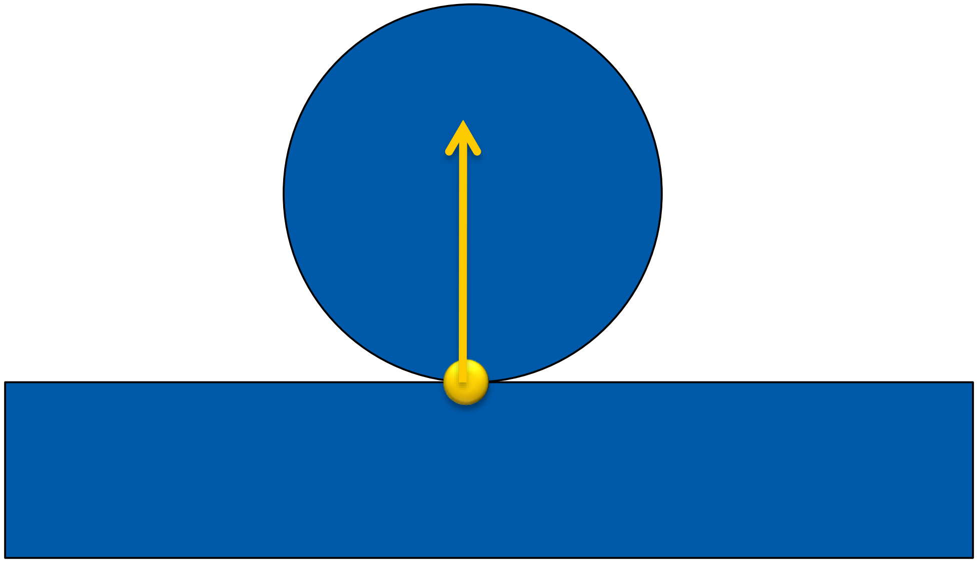
# Velocity resolution

## Find the collision collision normal and point of collision



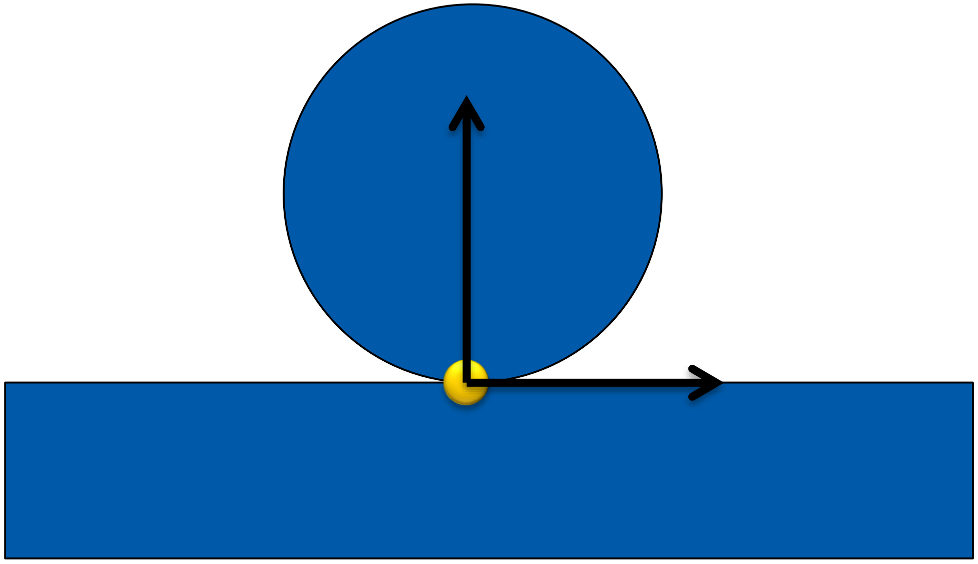
# Velocity resolution

## Find the collision normal and point of collision

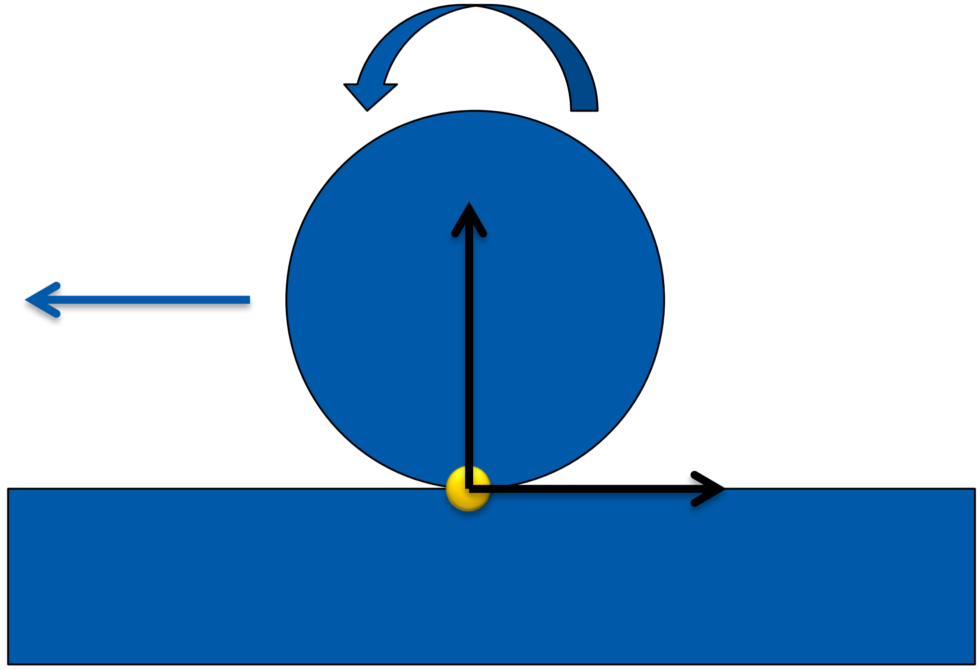


# Velocity resolution

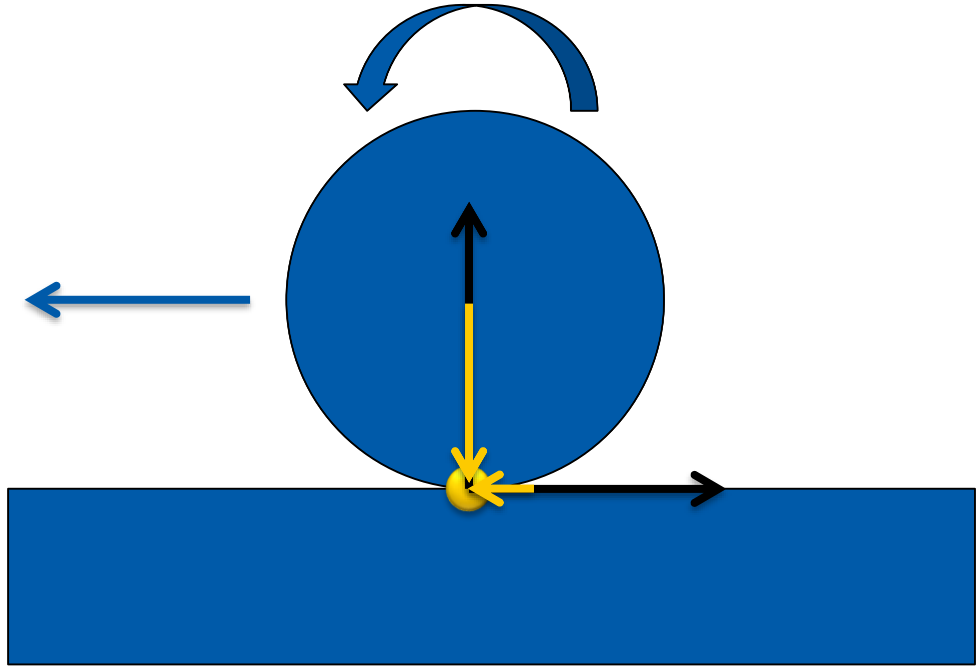
## Find the collision basis



# Identify the velocity of the collision point

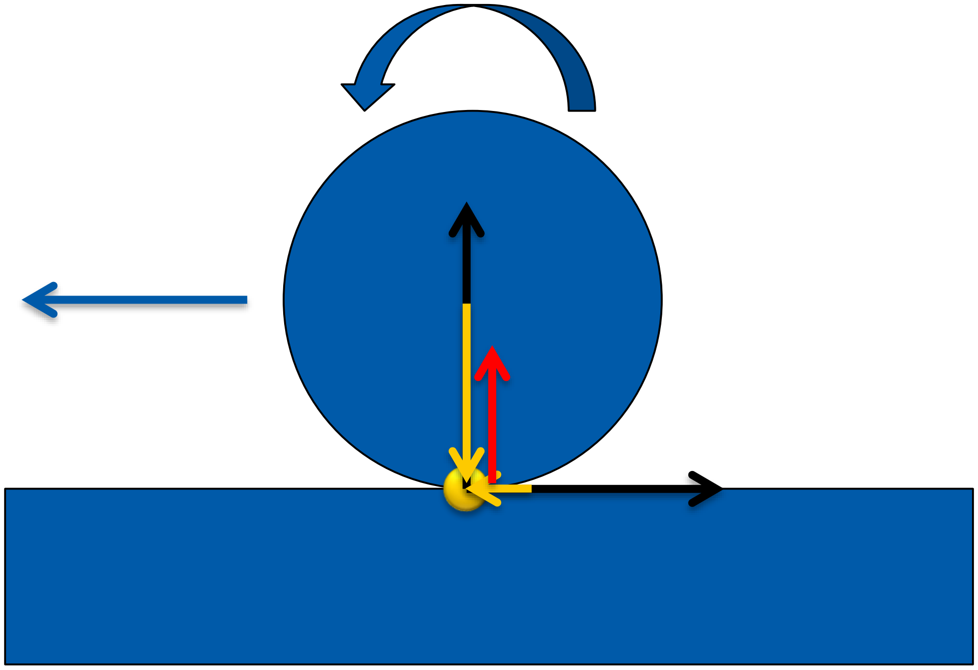


# Map velocity into the collision basis



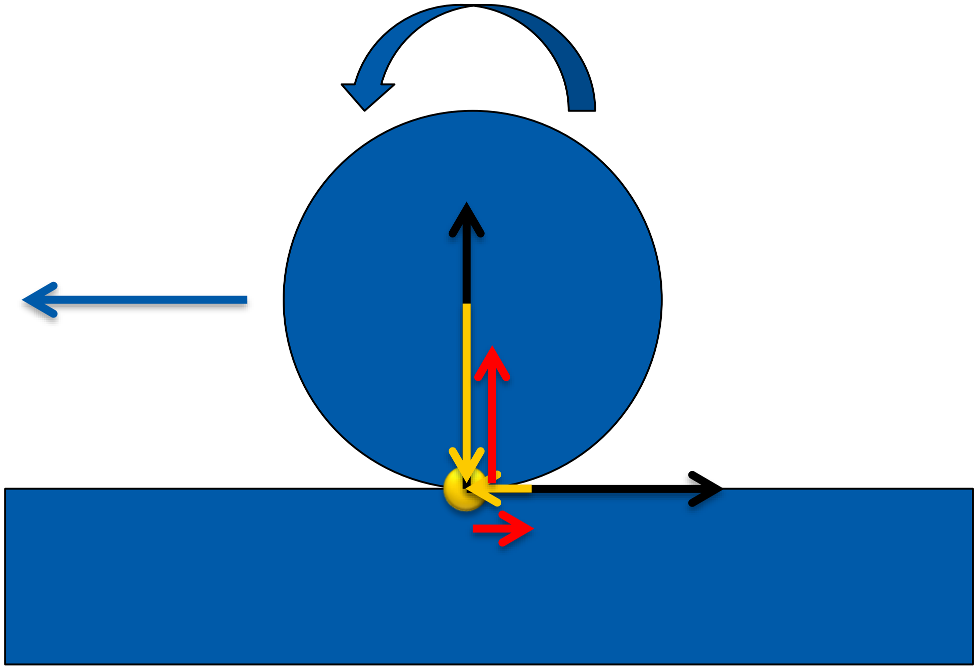
# X-Axis (= collision Normal)

## Separate the objects (what we did last lecture)



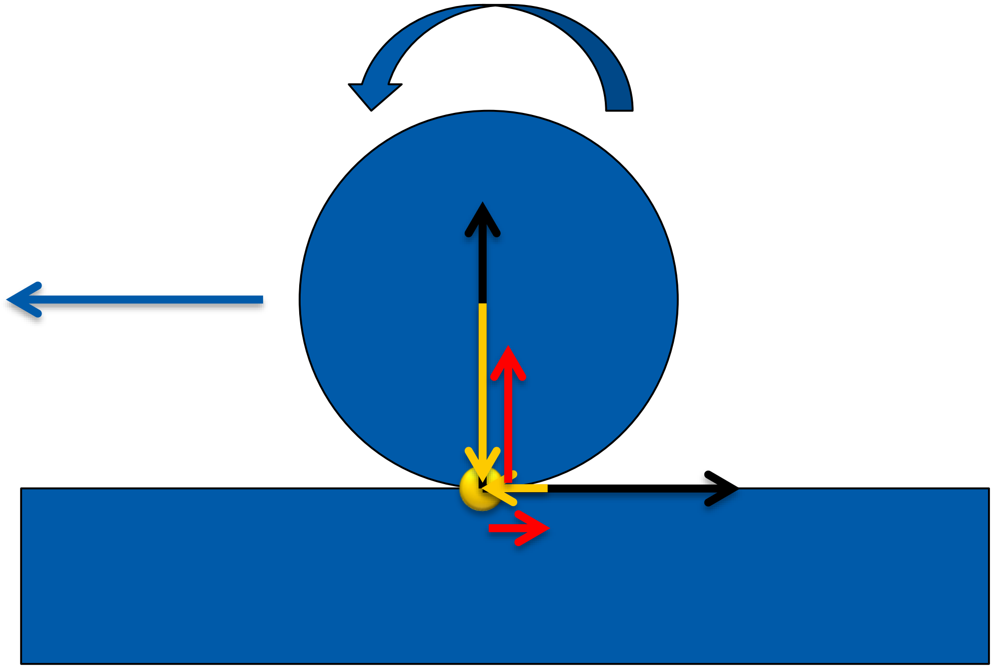
# Y and Z-Axis

## Handle Friction



# Apply changes as impulses

## Changes to velocity and rotation



# Summary

## Collision Detection

- Narrow vs. Broad phase
- Geometrical data structures
- Separating axis test

## Physics Implementations

- Different integrators
- Different schemes

## Rotation

- Torque
- Resolving velocities with friction