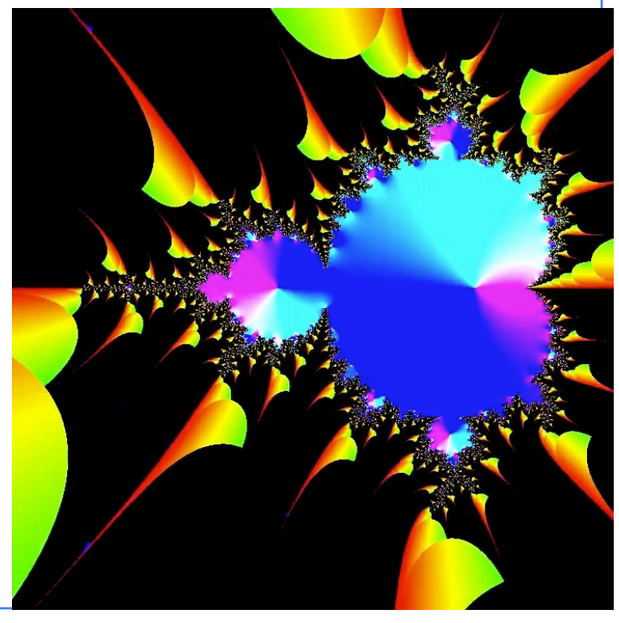
TSBK07/TSBK11

Computer Graphics

Ingemar Ragnemalm, ISY





Lecture 11

Animation

Particle systems

Collision detection

Next lab

Dugga 5 - last before the retake

Many have already reached "pass" level.

Note the lower demands for TSBK11.

Lecture questions

1. What depth cues give 3D effects in 2D animations?

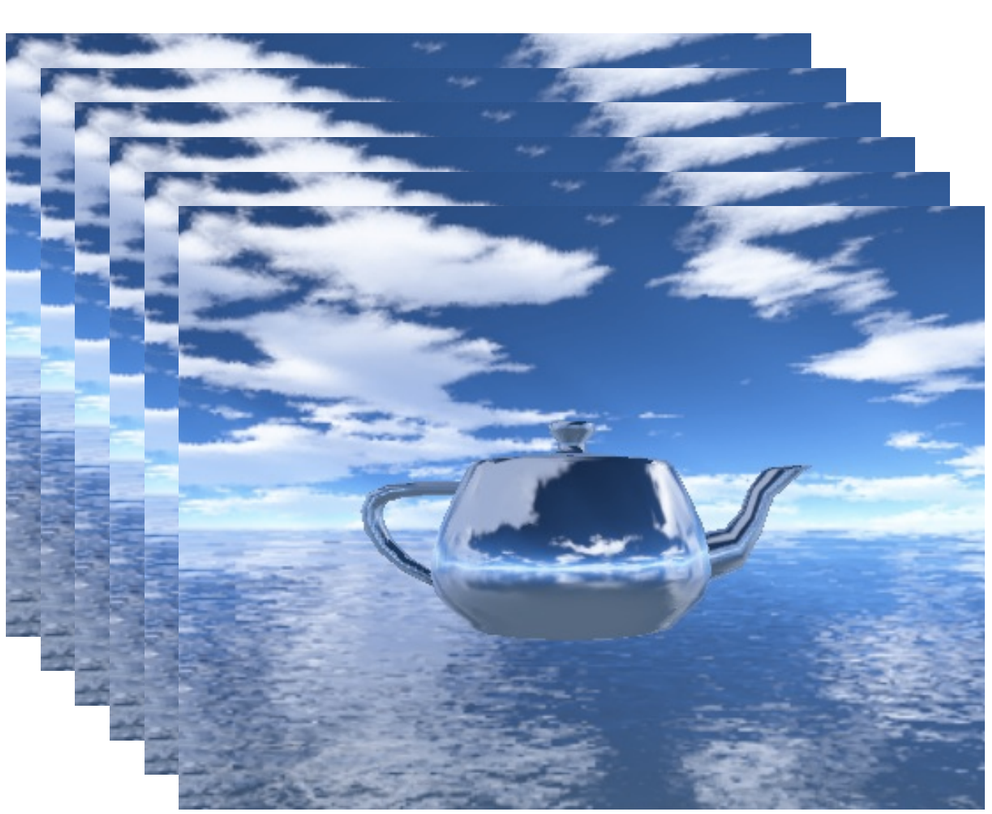
2. How can you provide data to instanced animations?

3. What bounding shapes are suitable for the broad phase?

4. What does SAT mean, as in the SAT theorem, and how can you apply it to collision detection?

Animation

Essentially a question of flipping between many still images, fast enough



Animation as a topic

- Page flipping, double-buffering
- Sprite animation
- Movement and posing
- Collision detection and handling
- Deformations

Double buffering

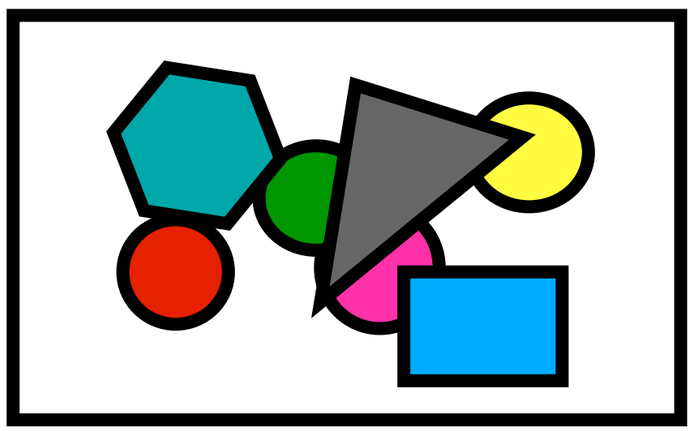
Flicker-free animation

The double buffering problem

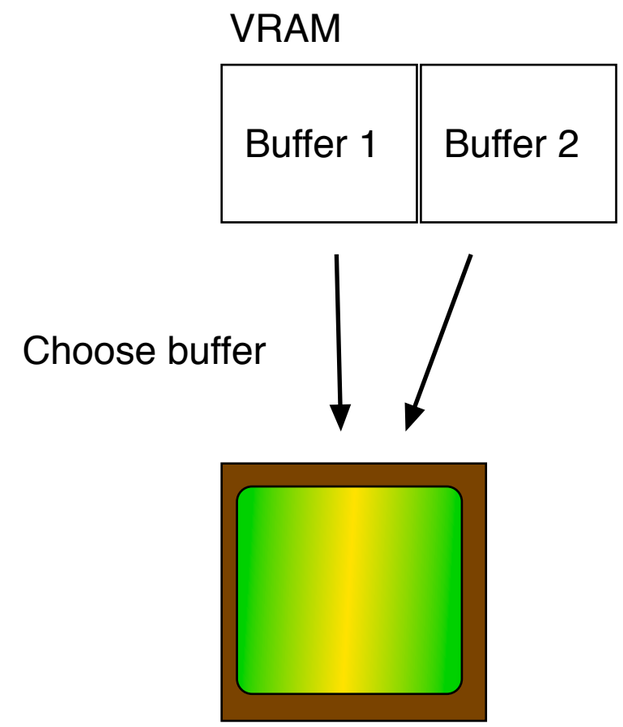
When animating a scene with many objects in real time, it is not just a question of showing images:

- Erase the entire scene
- Draw each visible object in new positions

This procedure may be visible if done on-screen!



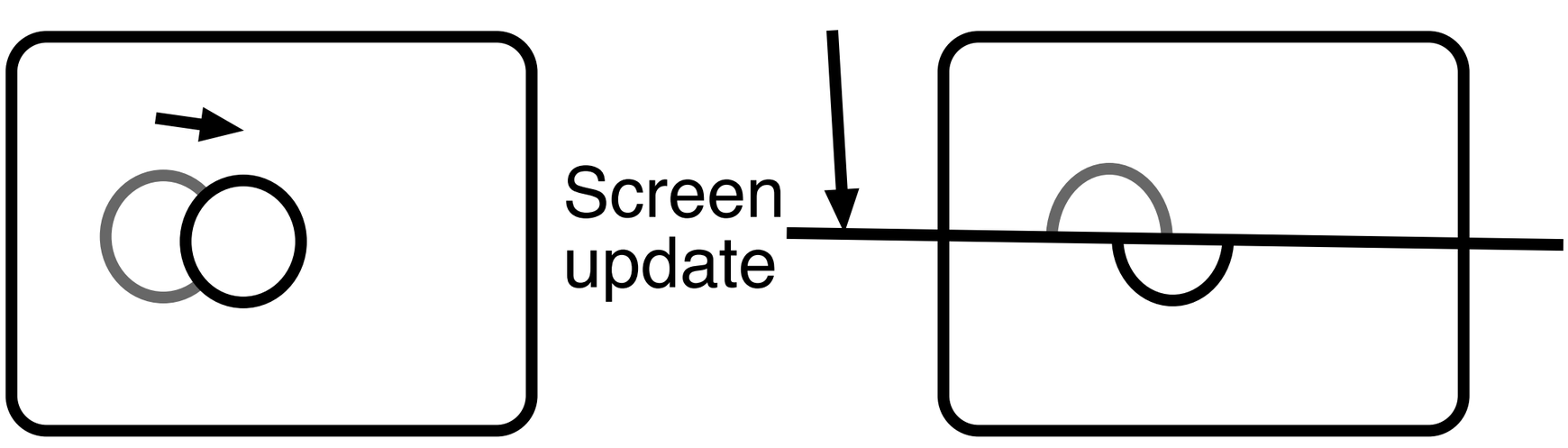
Double buffering



Switch between the buffers by moving a pointer!

Double buffered animation

Tearing



Occurs if buffers switch while the screen is being redrawn.

Synch with vsync to avoid.

Built-in VBL sync (vsync)

Modern systems have VBL sync built-in - even mandatory double buffering. You may need to turn ”vsync” off to test maximum frame rate (but don't keep it off!).

Double buffering in OpenGL

Double buffer

Not actually part of OpenGL

- Pass GLUT_DOUBLE to glutInitDisplayMode
- glutSwapBuffers();

Different in all systems! Inside MG:

Mac: Built in, always used.
Windows: SwapBuffers(hDC); (part of Windows window system)
Linux: glXSwapBuffers(dpy, win); (part of Xwindows)

Sprite animation

2D animation based on 2D images.

Extremely common in games! (Often indie games and/or mobile games.)

Sprites in OpenGL

Use textured polygons with transparency! (Like billboards but without 3D.)

Special “blitter” calls existed in GL2, but they were not guaranteed to be fast!

Take advantage of OpenGL in 2D!

High performance - use high-performance effects

Particle systems

Light. E.g. bump mapping to give some 3D feeling

Pseudo-3D effects

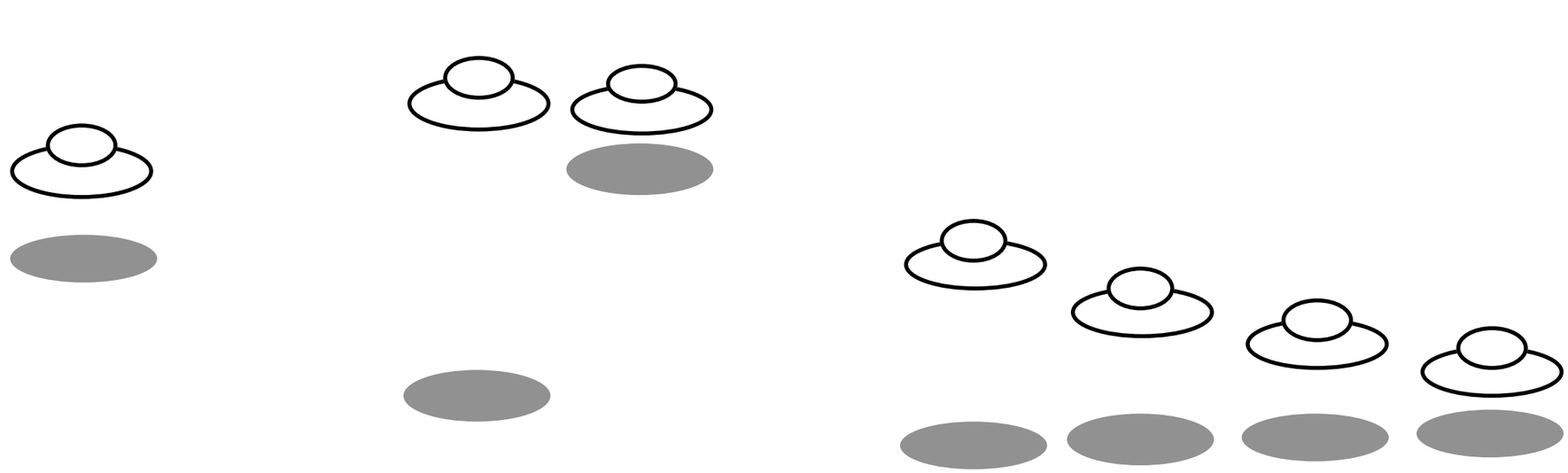
Scaled sprites on background with perspective:

Depth cue by size

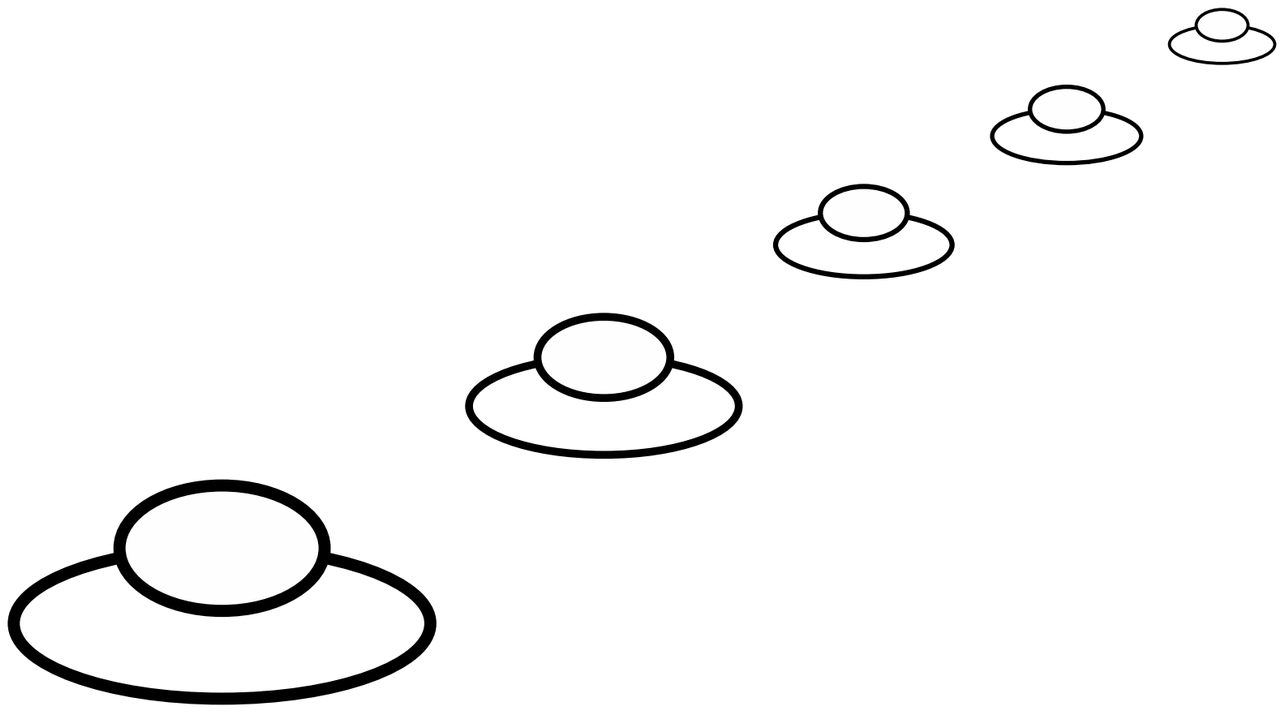
Side-scrolling with parallax scroll: Depth cue by movement

Depth due by shadows: Distance between object and shadow gives important information

Depth from shadows



Depth from size



Classic pseudo-3D game with depth from shadows and depth from size

Zaxxon

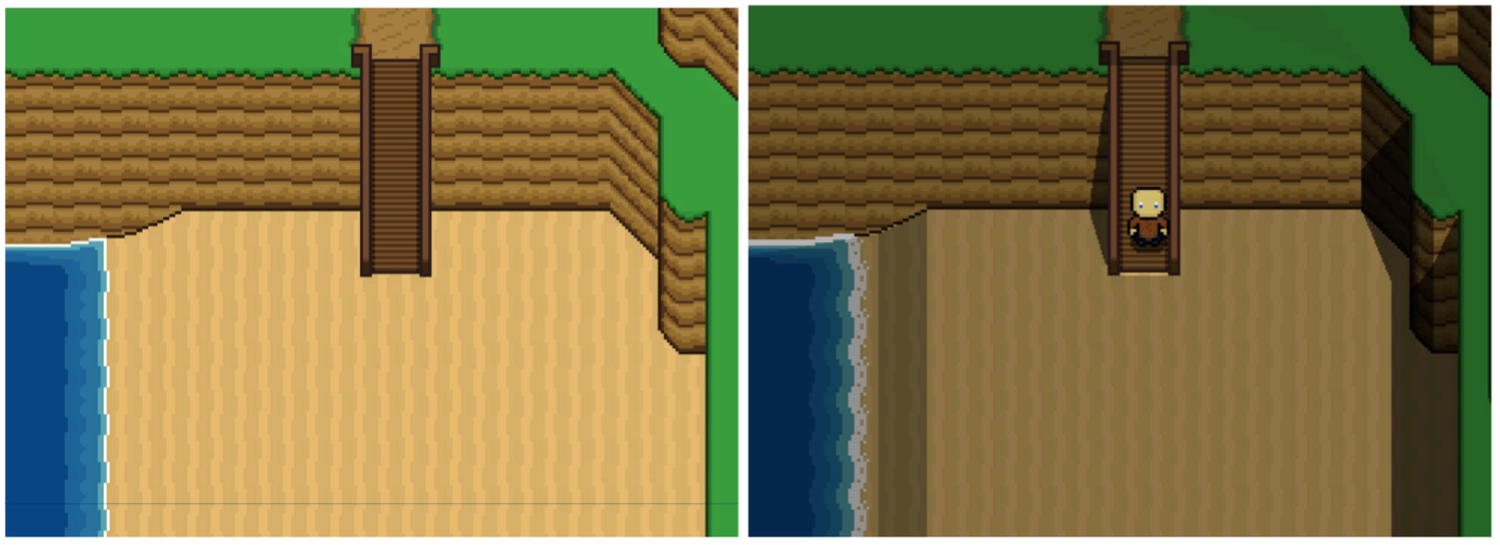
Depth from movement

Parallax scroll

Classic example:
Moon Patrol

Not only history

Modern 2D games incorporate 3D effects



Project by Blomqvist & Detterfelt 2019

Pseudo-3D effects vs 3D

- Depth from size = perspective projection

- Parallax scroll: Comes for free to some extent, but can be emphasized with cameras observing the viewer

- Depth from shadows: That is why shadows are important in 3D! It is needed for ”full 3D” experience.

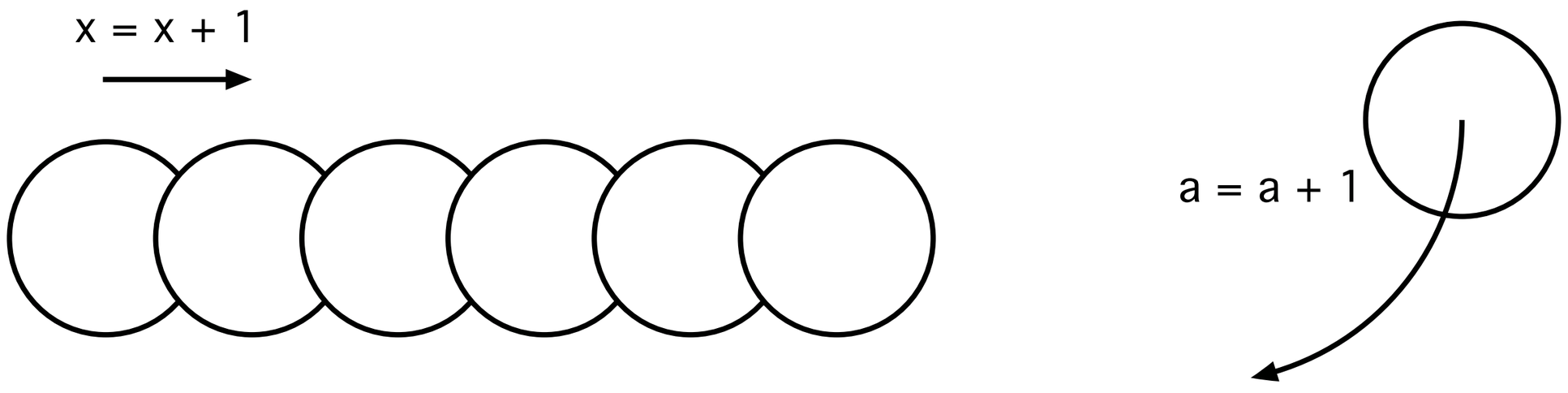
Animation techniques for moving objects

- Procedural animation
- Physics-based animation
- Pre-programmed animation paths

Procedural animation

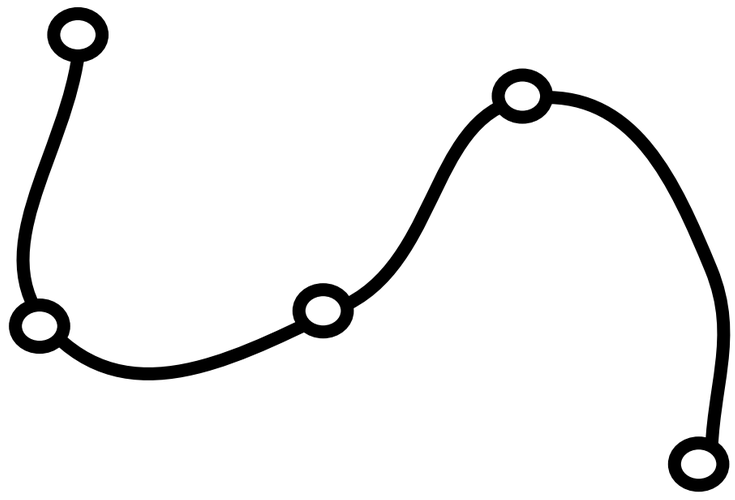
Simple frame by frame movement following some procedural rule.

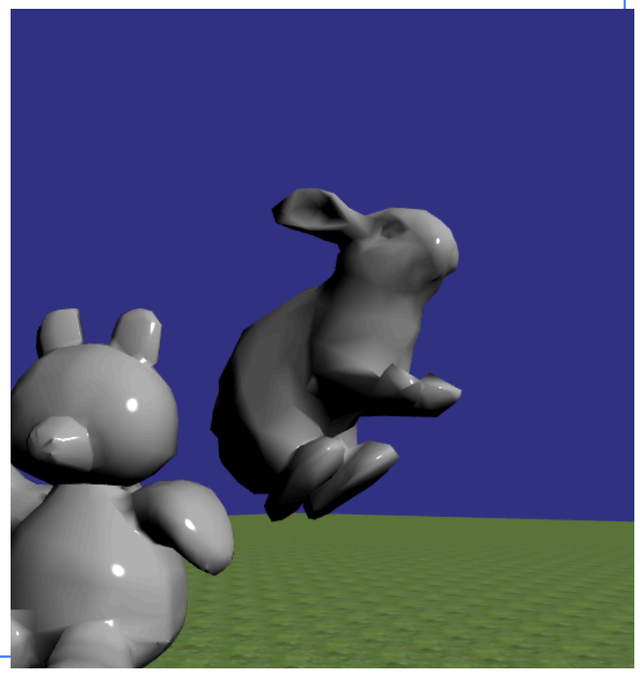
What we are already doing in the labs!



Animation paths

Use Catmull-Rom splines! Predictable, smooth, continuous!





Character animation

- Pre-defined poses
- Key-frame animation
- Forward kinematics
- Inverse kinematics
- Physics based animation
- Motion capture

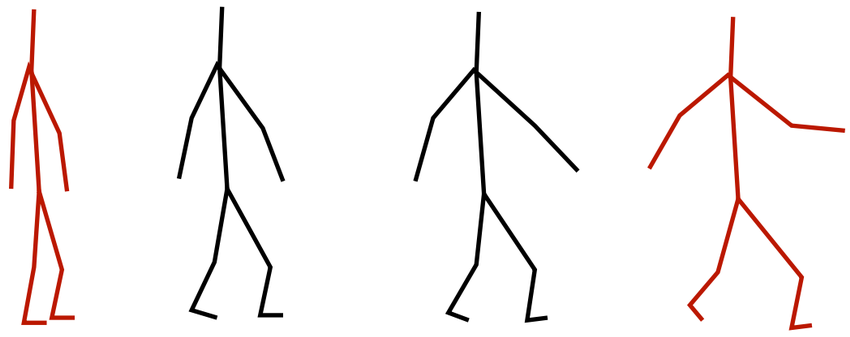
Key-frame animation

Fewer pre-defined poses

Key-frames are designed at suitable intervals

Frames between keyframes are interpolated (morphed)

Very common method for real-time animation



Kinematics

How to find these poses!

Kinematics = movement without forces

Forward kinematics:

Specify poses by specifying rotation of joints. Easy to implement, but specifying poses is much trial-and-error.

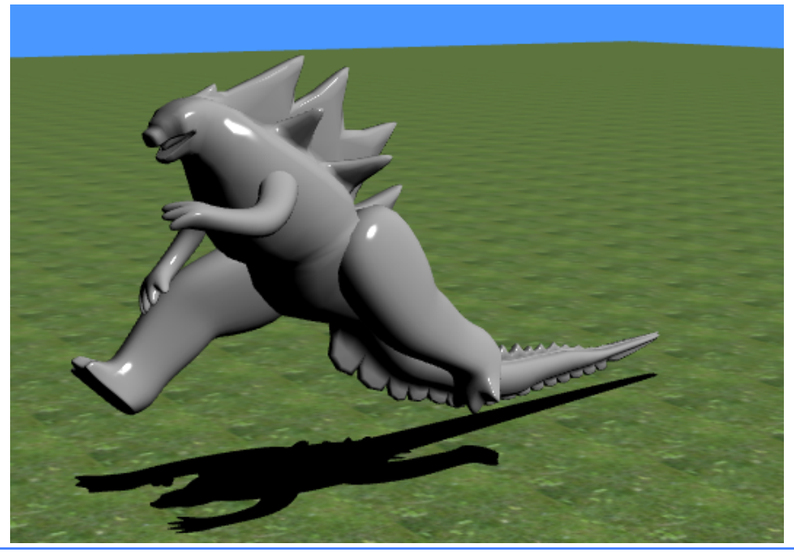
Inverse kinematics:

Goal-driven posing. Specify where some part should go (i.e. a hand) and calculate necessary rotations

Character animation in this course
(using the tools central to the course)

Animation of rigid parts (robot style/windmill style)

Model in lab material: Godzilla model!



Motion capture

Extremely common in movies!

- Record by natural visuals only
- Tracking markers
- Active sensors on the body

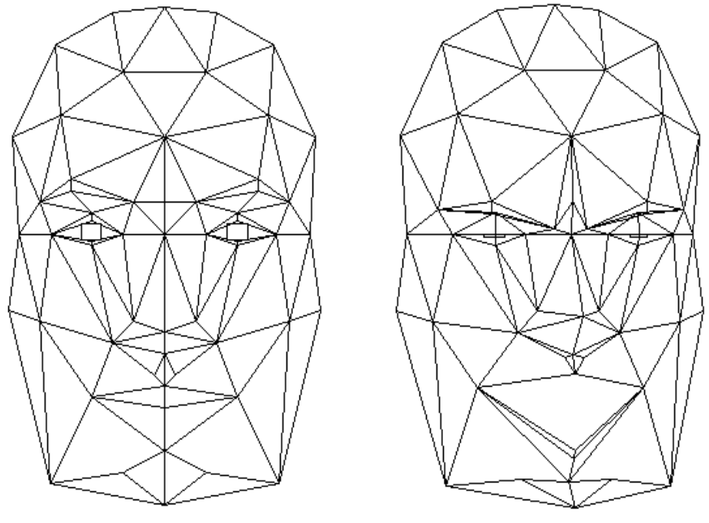
Perfect for pre-generated animations.

Face animation

Hard problem - we are very sensitive to errors!

Animate by action units (muscle based) or face animation parameters (extreme detail)

FAPs part of the MPEG-4 standard.



The Candide model

Some advanced animation topics

- Bones and skinning systems
- Deformations
- Physics-based animation
- Quaternions, SLERP

Mainly subjects for later courses (TSBK03)

Particle systems

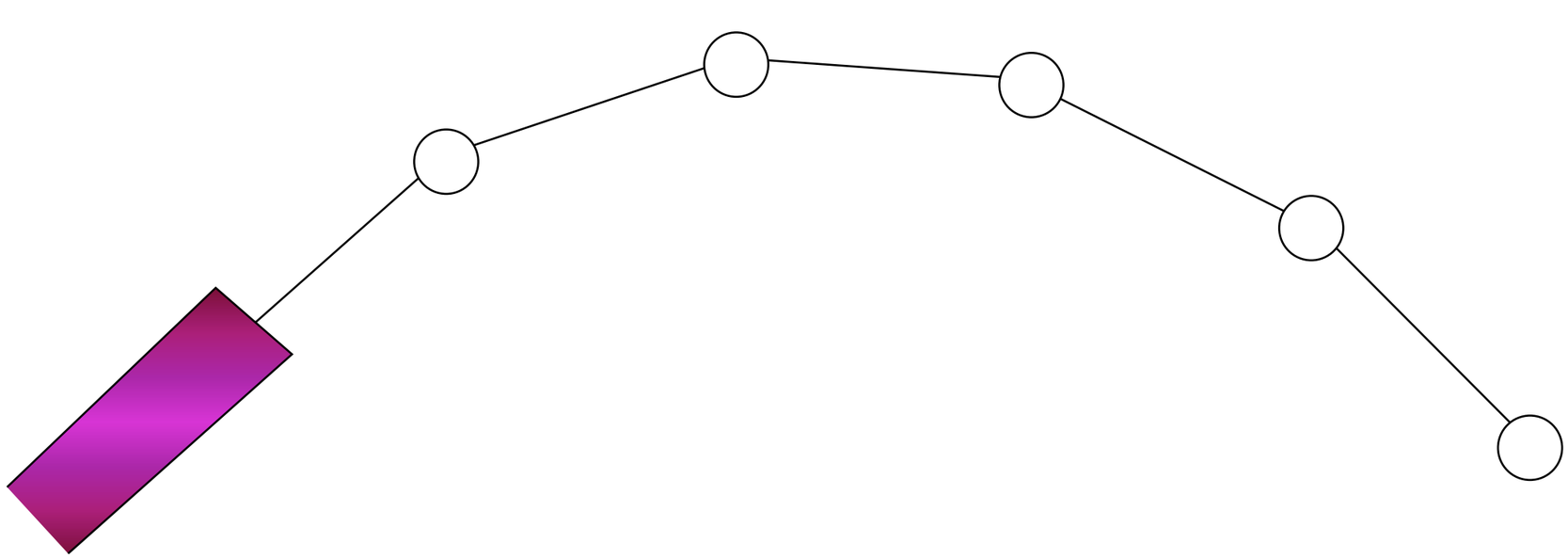
Spectacular effects with little effort!

Many small moving objects.

- Explosions
- Water
- Fire
- Snow
- Rain

Particle system

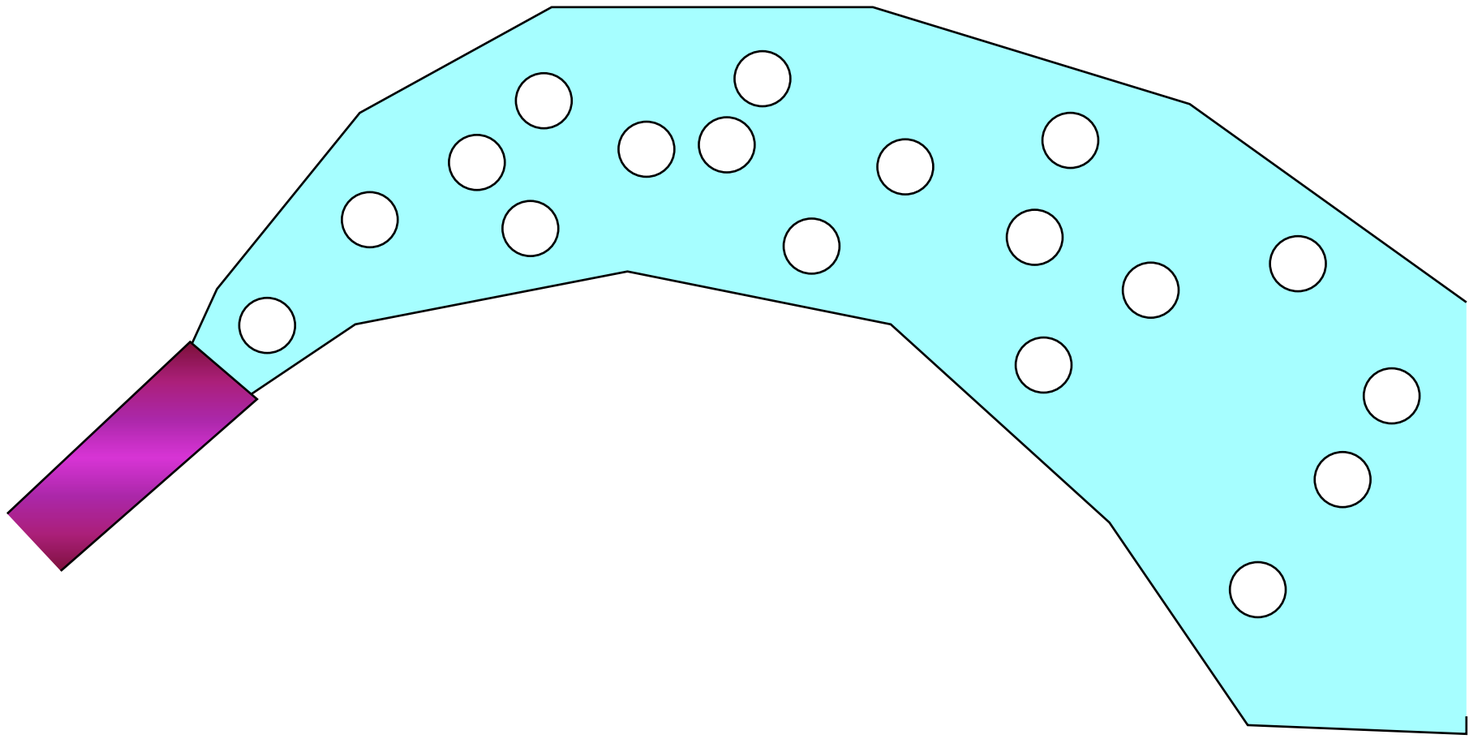
Example: Water



No randomness - bad

Particle system

Example: Water



Life of a particle

- Initial position
- Initial speed (usually with some randomness))
- Movement (usually independent, physically realistic)
- Termination rule (e.g. hits ground, fades away after some time...)

Particle physics

Point-based physics usually fine

Movement according to fundamental physics:

acceleration = gravity + forces/mass
speed = speed + acceleration
position = position + speed

“Euler integration”

Particle system on CPU

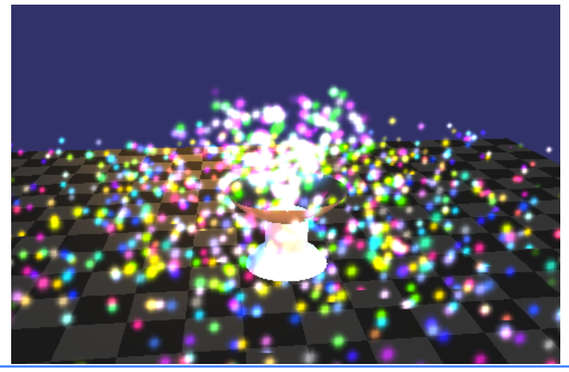
CPU-driven particle systems OK up to a certain size

Make arrays of positions and velocity.

Data transfer (new positions) of all particles can be a bottleneck. Multiple calls for each?

How much can we compute on the GPU?

This was not a problem:



Drawing particle systems

Large number of very simple models (billboards)!

Modest demands on GPU, but very large number of function calls!

Solution: Instancing

Instancing, revisited

Draw a large number of the same model!

Each instance has an index, the instance number.

glDrawArraysInstanced(GL_TRIANGLES, 0, 3, 10);

draws a triangle 10 times!

gl_InstanceID tells the shader which instance we have.
Use for affecting position.

Feeding data to instancing

Much data must be transferred with few function calls!

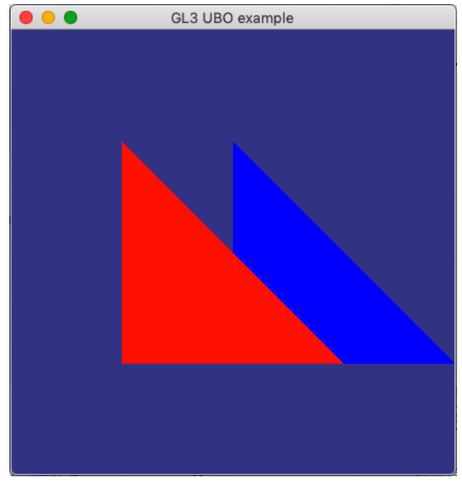
Possible methods:

Textures
UBO (Uniform Buffer Object)
Instanced arrays

UBO demo

One call for drawing 2 triangles as separate instances.

A buffer with two colors is indexed with the instance id.

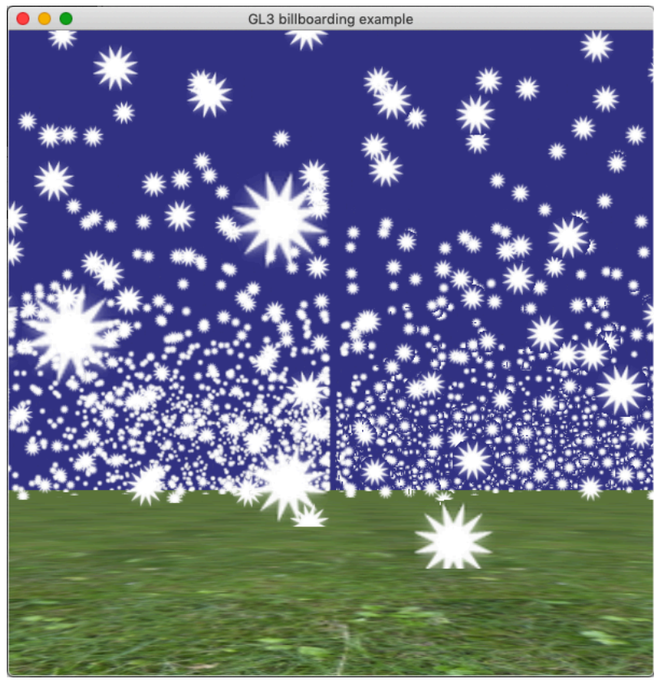


Demo: Data in texture

A texture containg random numbers is uploaded.

Positions are calculated from time and one texel per instance.

Make sure to hit the *center* of each texel!

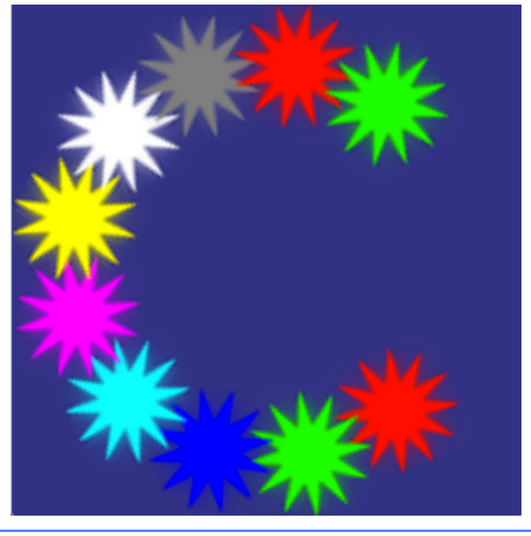


Instanced arrays

An array passed one item at a time, like vertex attributes

Uses glVertexAttribDivisor to split the array in instances

Conclusion: Several ways to get that info in!



TSBK03

Particle system on GPU

CPU-driven particle systems OK up to a certain size

Data transfer (new positions) of all particles can be a bottleneck

Can the whole particle system be computed on the GPU?

TSBK03

Texture based particle systems

Use textures to store x, y, z, dx, dy, dz

Store as color components (r, g, b)

Needs advanced texturing features (render to texture, floating-point buffers)

Particles as billboards. Each polygon must identify its particle data.

TSBK03/
TDDD56

Separate compute kernels for particle systems

CUDA, OpenCL, Compute shaders

Free choice of data formats

Less integration with the OpenGL pipeline

TSBK03/
TMN084

Randomness in GPU

The GPU has no random number generator!

Rain example: Random numbers in texture generated on CPU.

Mathematical calculation in shader is possible. (Other course but ask me as needed.)

Collision detection and handling

A major problem in many real-time animations.

Even modern high-budget games have problems!

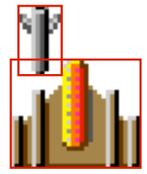
2D collision detection

Even in 2D, collisions is a non-trivial problem.

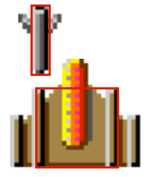
- Rectanges or circles - easy
- Enclosing convex polygon(s)
- Pixel-level testing

2D collision detection with rectangles

Testing with the bounding box: often unsatisfactory.
Too rough approximation!



Smaller rectangles make decent compromises.



Special cases

When using pseudo-3D, sprites in perspective, the shape outline is not usable.

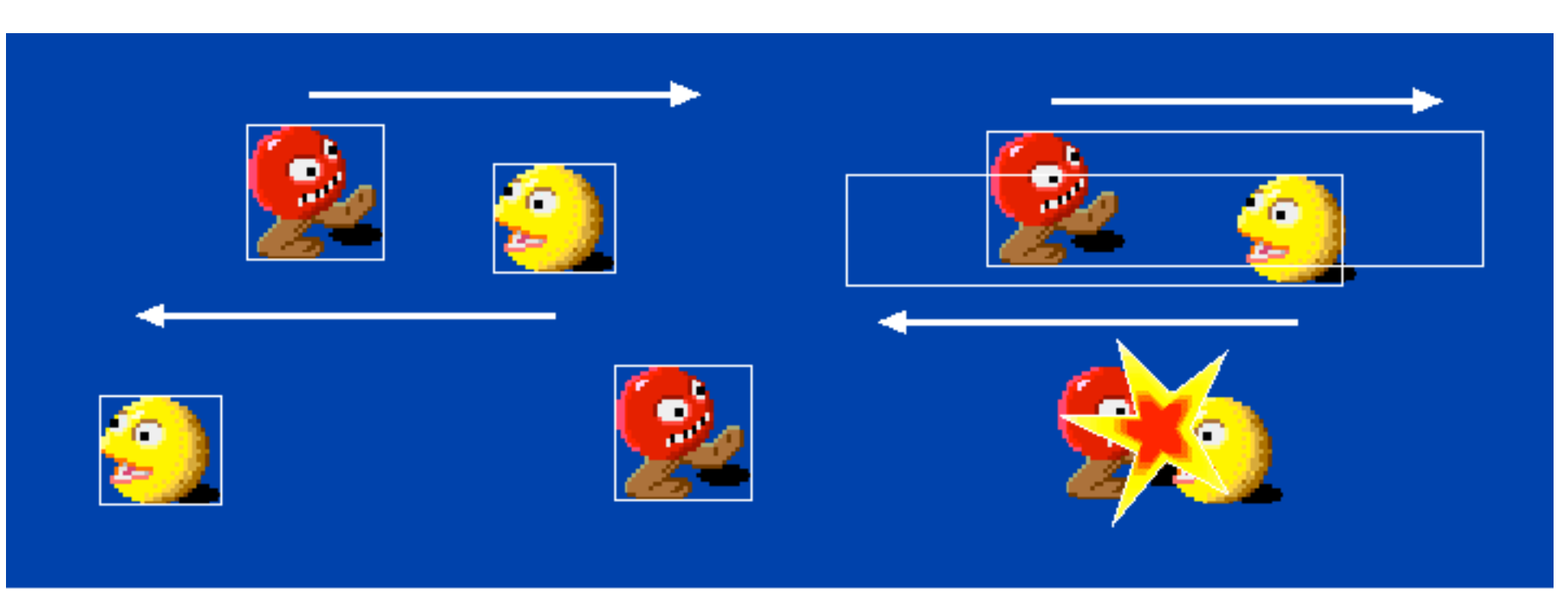
Simple approach: Modified box:

Wrong

Right

Moving objects

Fast moving objects may pass right through each other.



Wrong Right

An example of *temporal aliasing*

Fast-moving objects

- Test with elongated shapes (sweeping)
- Test several times along the trajectory (multisampling)

Polyhedra-polyhedra collisions

Naive method: Check all polygons in all objects against all polygons in all objects!

How would you do that?

Practical methods

1) Separate into broad phase and narrow phase

2) Object hierarchies

Broad phase - narrow phase

Really three phases:

Global phase

Bounding shapes phase

} Broad phase

Narrow phase

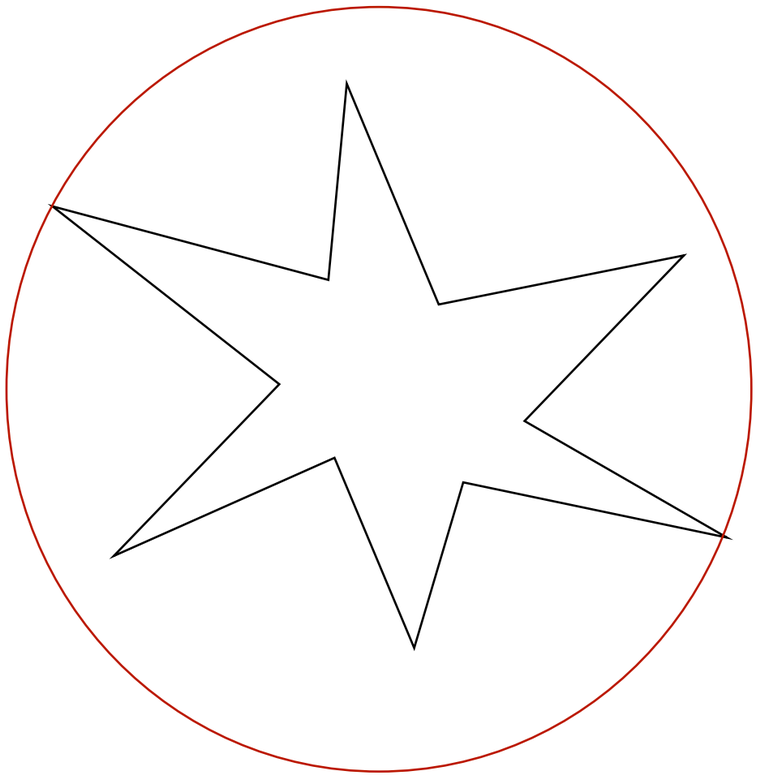
Bounding shapes phase

Make simple checks to see if objects are so close that we should test in detail!

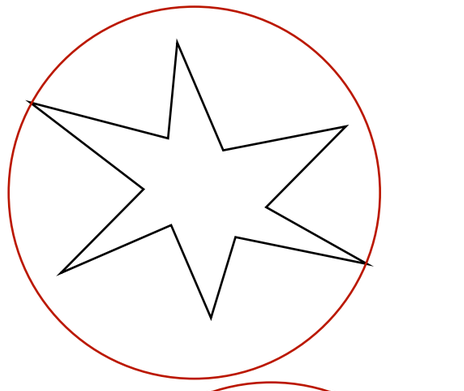
Use simple bounding shapes!

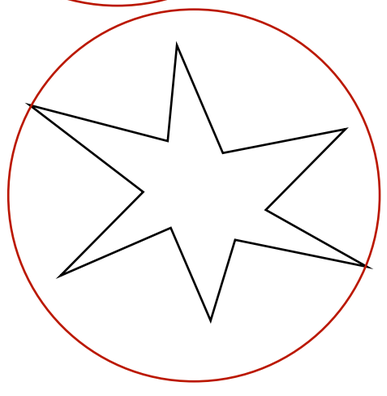
Typical shapes:
- Sphere
- AABB (axis aligned bounding box)
- OBB (oriented bounding box)

Sphere



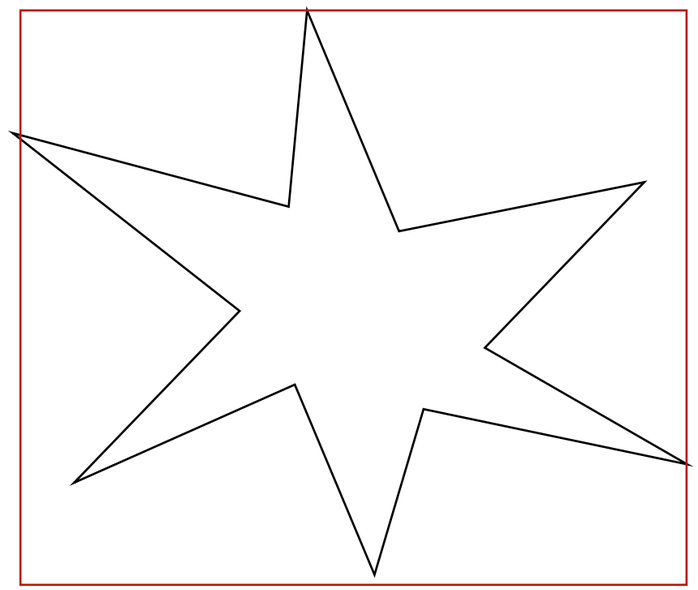
Sphere



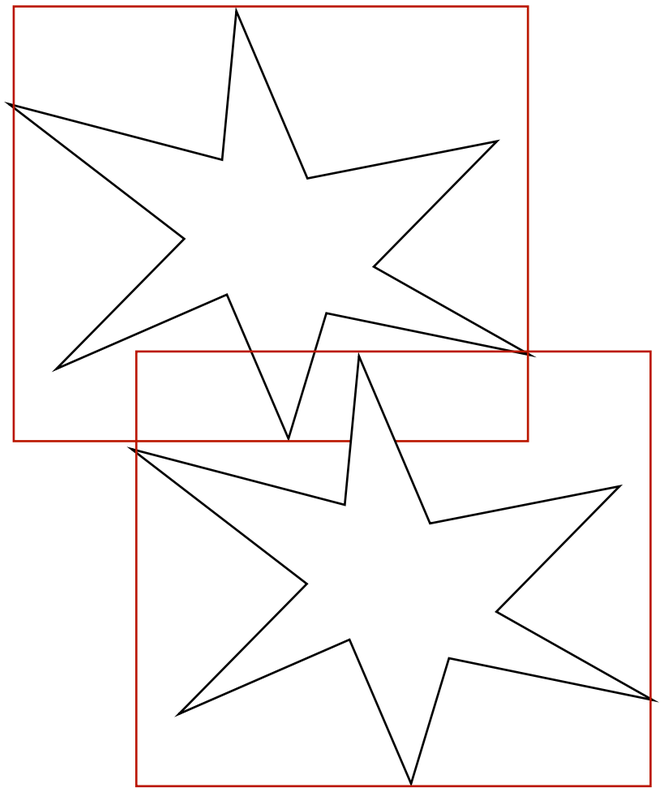


- Very simple tests. $d^2 < (r_1 + r_2)^2$
- Calculate radius once and for all. Never changes for rigid objects. Loop through all vertices, pick biggest distance to center.
- Rotation allowed!
- Good fit for compact objects.
- No flat surfaces, object can not rest on each other.

AABB



Axis aligned bounding box



AABB

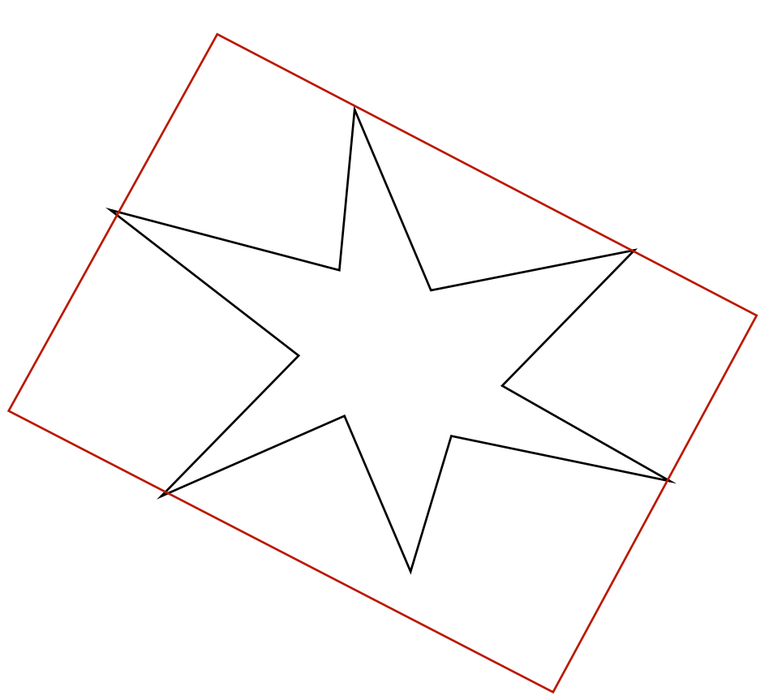
- Very simple tests. 6 tests, like the 2D case.

- Calculate by finding max and min along each axis.

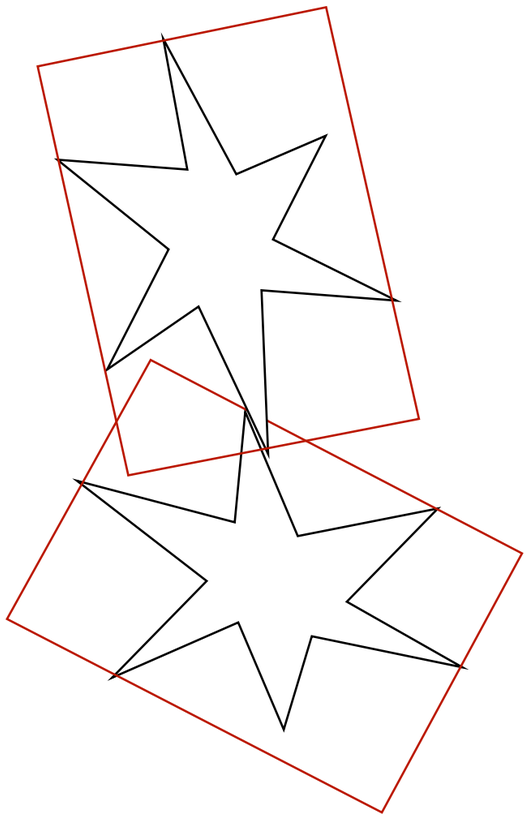
- Can’t be rotated freely!
1) Recalculate on rotation.
2) Make new AABB from rotated vertices of AABB.

- Not very good fit on many objects.

OBB
Oriented bounding box

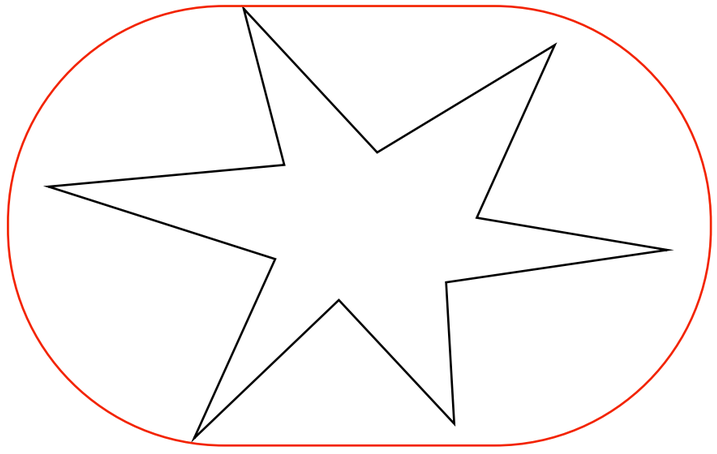


OBB
Oriented bounding box



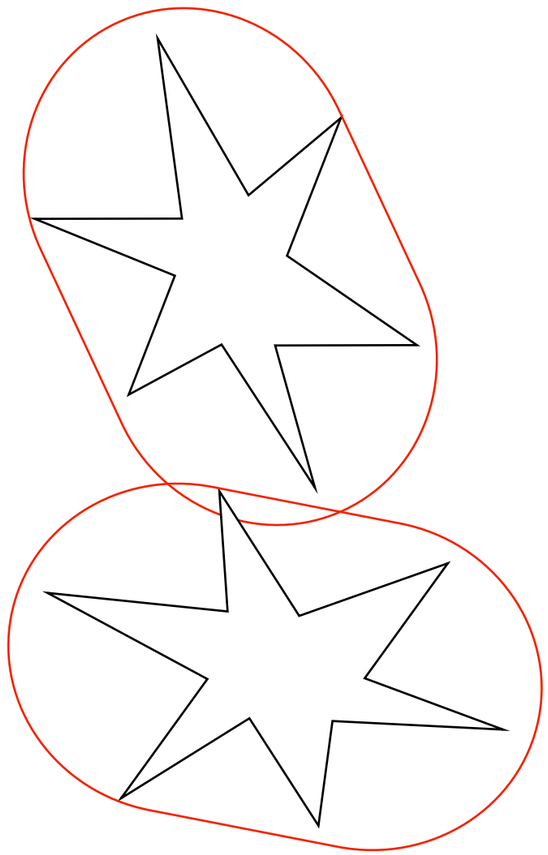
- Good fit, you can find smallest possible box. Few false hits.
- Complicated tests - but can be simplified. (SAT, more later)
- Rotation allowed.

Capsule



Capsule

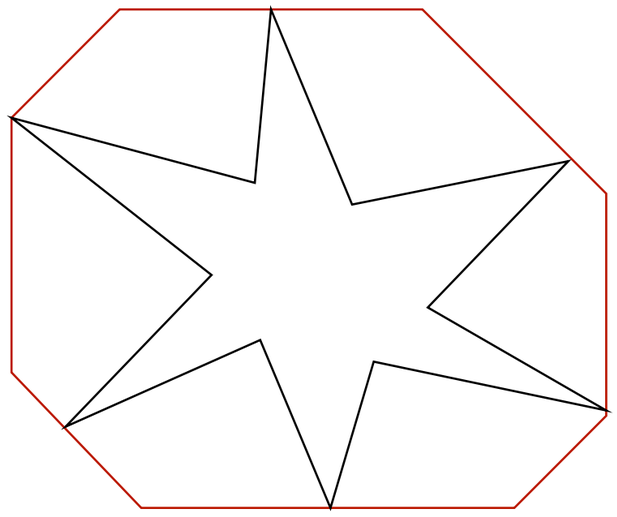
Cylinder + 2 half spheres



- Good fit, you can find smallest possible box. Few false hits.
- Fairly complicated tests (sphere + OBB)
- Rotation allowed.

k-DOP

Discrete oriented polytope with k sides



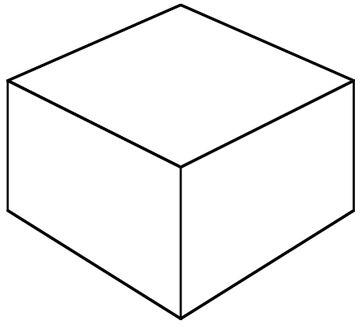
Like AABB but you can cut off corners and sides
Fairly simple tests
Fewer false hits

k-DOP

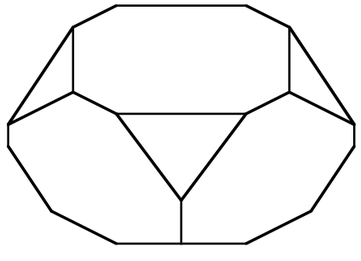
Discrete oriented polytope with k sides

Four kinds:

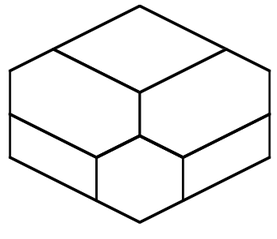
6-DOP (= AABB)
14-DOP (corners)
18-DOP (sides)
26-DOP (sides and corners)



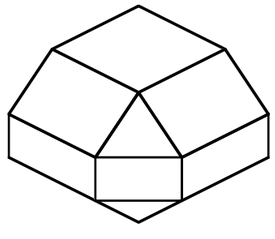
6-DOP



14-DOP



18-DOP



26-DOP

Balancing the broad and narrow phases

$$T = N_V C_V + N_p C_p$$

Simple broad phase - many false hits - high N_p.

Elaborate broad phase - few false hits - lower N_p but high C_V.

Narrow phase

Separating Axis Theorem (SAT)

Containment/intersection test

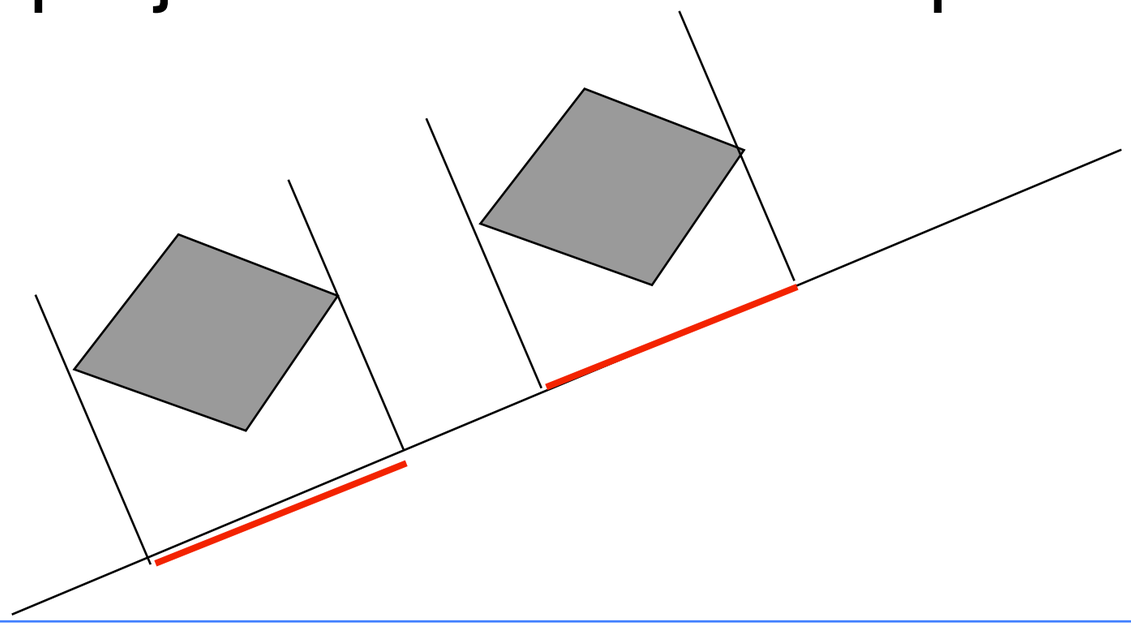
Advanced methods: GJK

Separating axis theorem (SAT)

and its application on collision detection

Separating axis theorem

Two convex objects do not overlap if there exists a line (axis) onto which the two object's projection do not overlap



Really ”Separating plane”

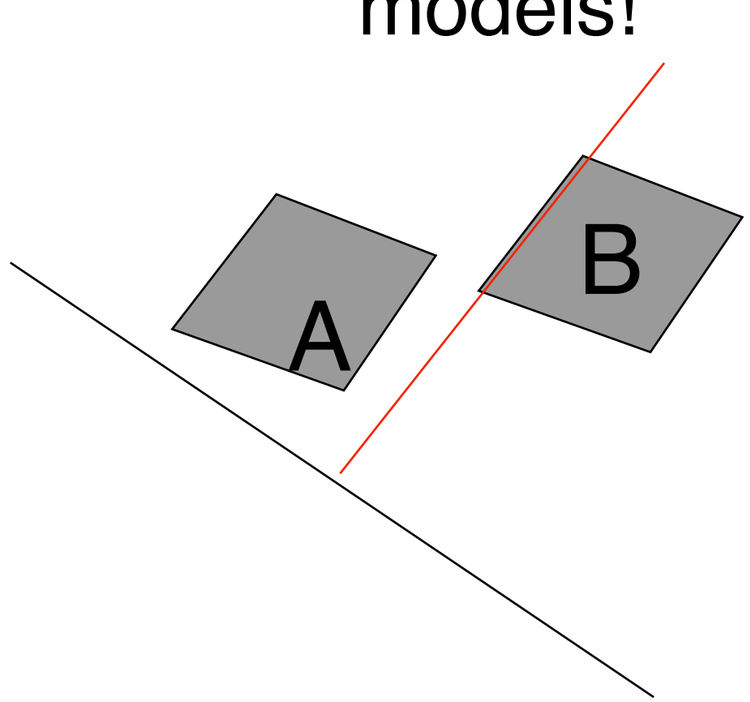
Axis = normal vector of separating plane

Can you find a plane between them?

Problem: The number of possible planes/axes to test are infinite!

SAT in practice

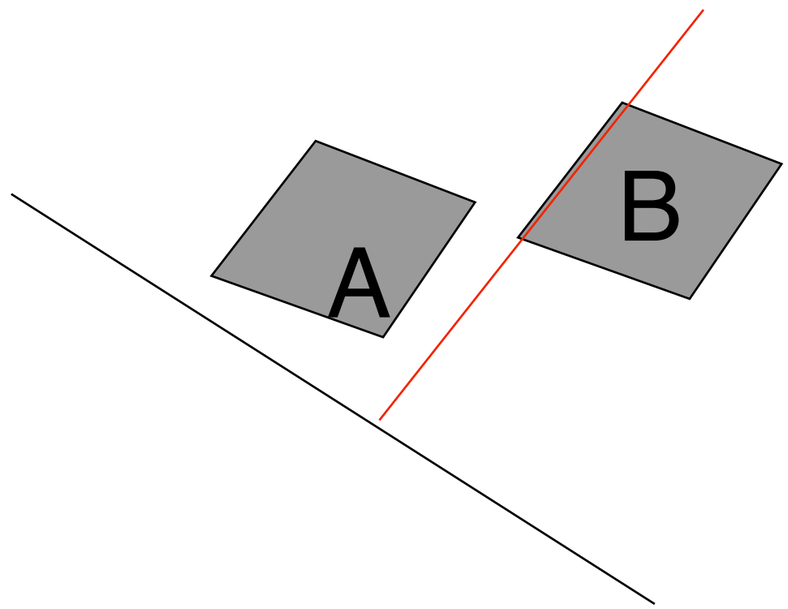
For polyhedra models, use the faces of the models!



Fairly simple in the 2D case

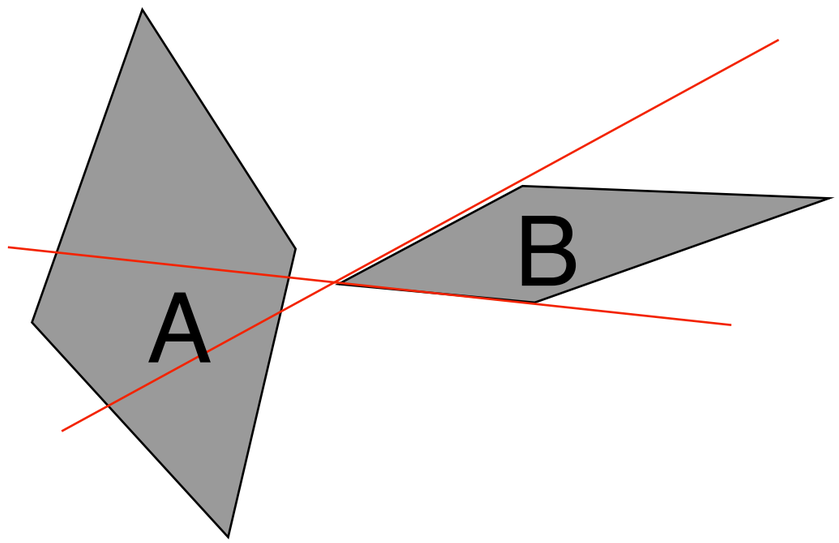
For every face in B, find plane equation

Test all vertices in A with dot product!



But you must go both ways!

There are configurations where one model has no single face that will exclude the other!



2D SAT algorithm

for all faces in A
let a be a vertex in A
hit = false
for all vertices b in B
diff = n • a - n • b
if diff > 0 then
hit = true
if not hit then
return false

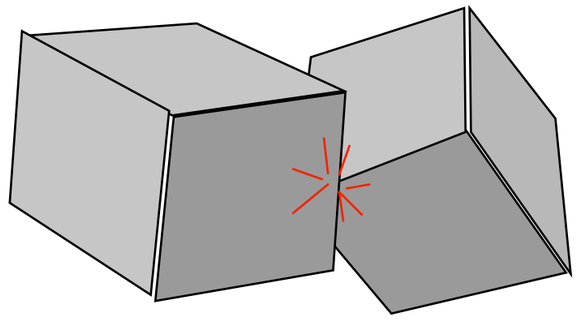
for all faces in B
(same algorithm with A and B reversed)

return true

SAT in 3D

Not as simple!

Two object meeting at an edge may fail the test!



We must add tests with more planes.

SAT in 3D

We must add tests with more planes. Which ones?

Cross product of an edge in a with an edge in b = additional potential separating plane

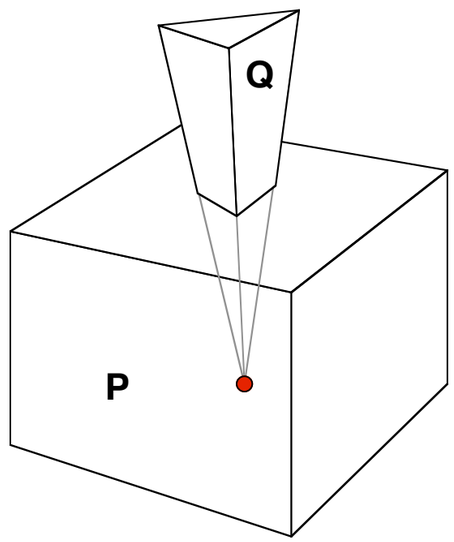
All edges in a * all edges in b!

=> Complete SAT on complex 3D objects is expensive!

So how can SAT be useful?

- Simplify. Skip some tests, accept some false hits.
- Use SAT on simplified bounding shapes. (OBB)
- Remember separating axis from previous test!

Containment/intersection



Containment/Intersection

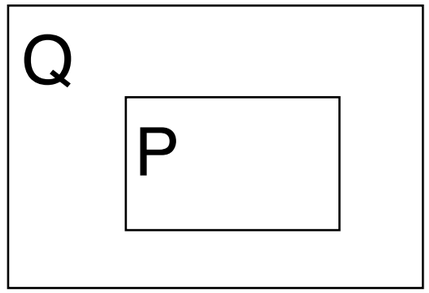
Exact test between two polyhedra, Q and P.

Two tests are needed:

1) Is any vertex in Q contained in P or vice versa?
2) Does any edge of Q penetrate a face of P?

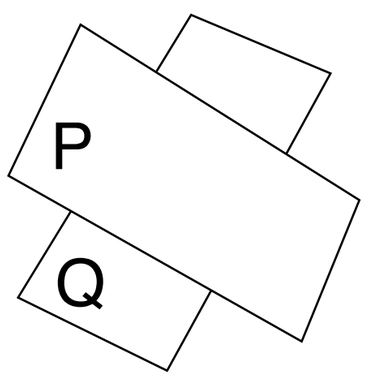
The need for all tests

1) in both directions:



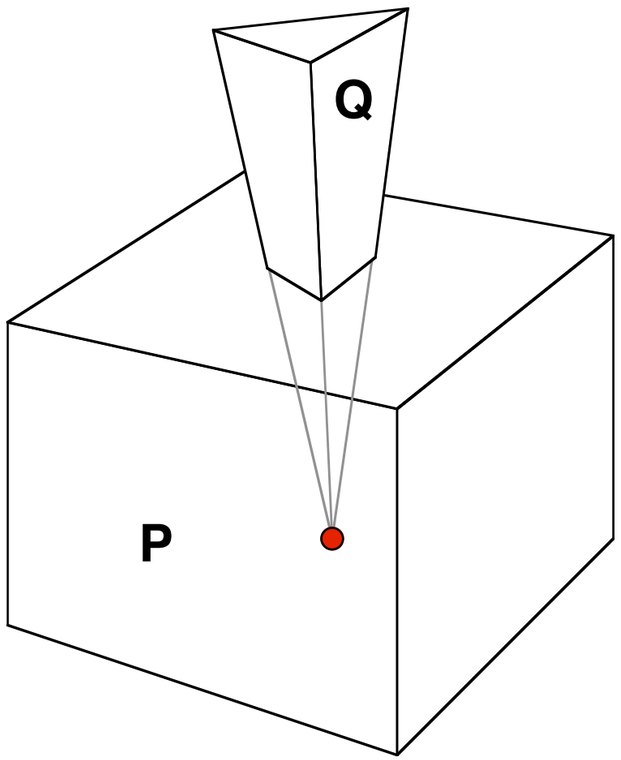
Q contains points in P,
P contains no points in Q

2) when 1) fails:



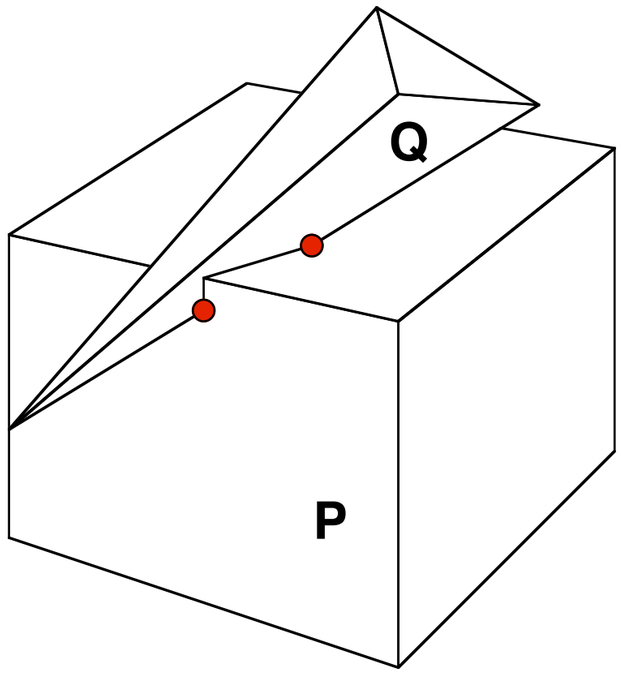
Neither P nor Q contain points of each other, but still overlap!

Test 1:



A point in Q is inside P

Test 2:



An edge of Q intersects sides of P

Comparison

Full SAT

$V_p*P_q+P_p*V_q + Ep*Eq*Vp + Ep*Eq*Vq$ $O(N^3)$

Containment/intersection:

$V_p*P_q+P_p*V_q + E_p*P_q+P_p*E_q$ $O(N^2)$

Acceleration method: Optimize order of tests

Similarity over time

Remember last time, test that first, the ”right test”.

Search in the right direction

Use the center of each shape, start with most likely tests

Level-of detail

Simplified models are extremely useful in collision detection!

We can go much lower than what we can do when rendering!

Reduce narrow phase complexity (C_p)!

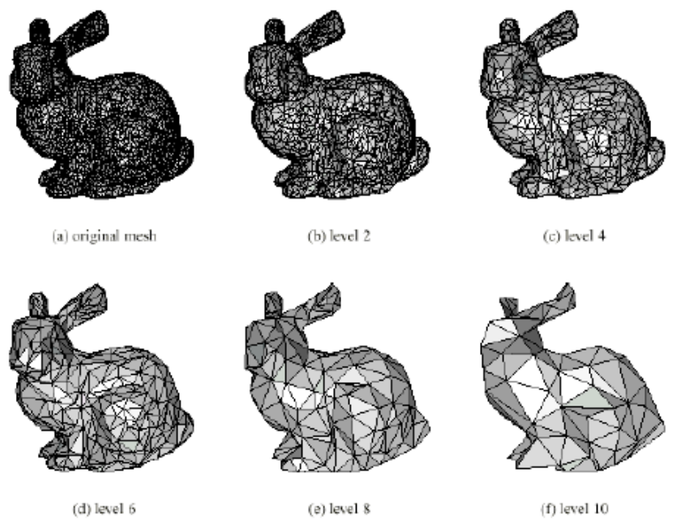
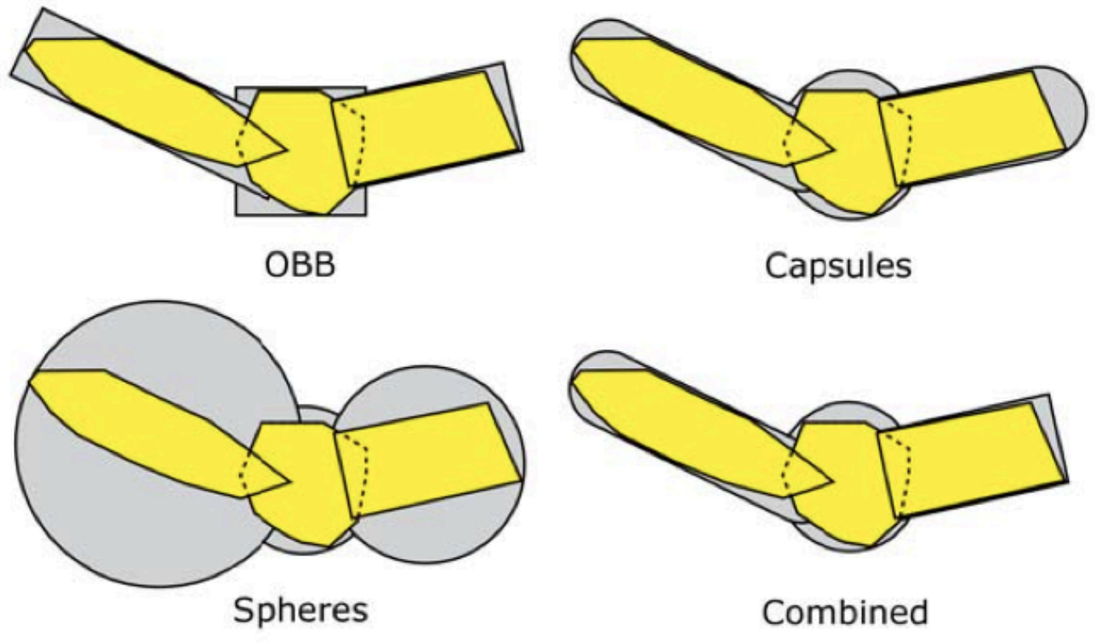


Figure 8. Stanford Bunny. Simplified meshes with 10000, 4577, 2106, 988, 463, 245 triangles

Split to parts

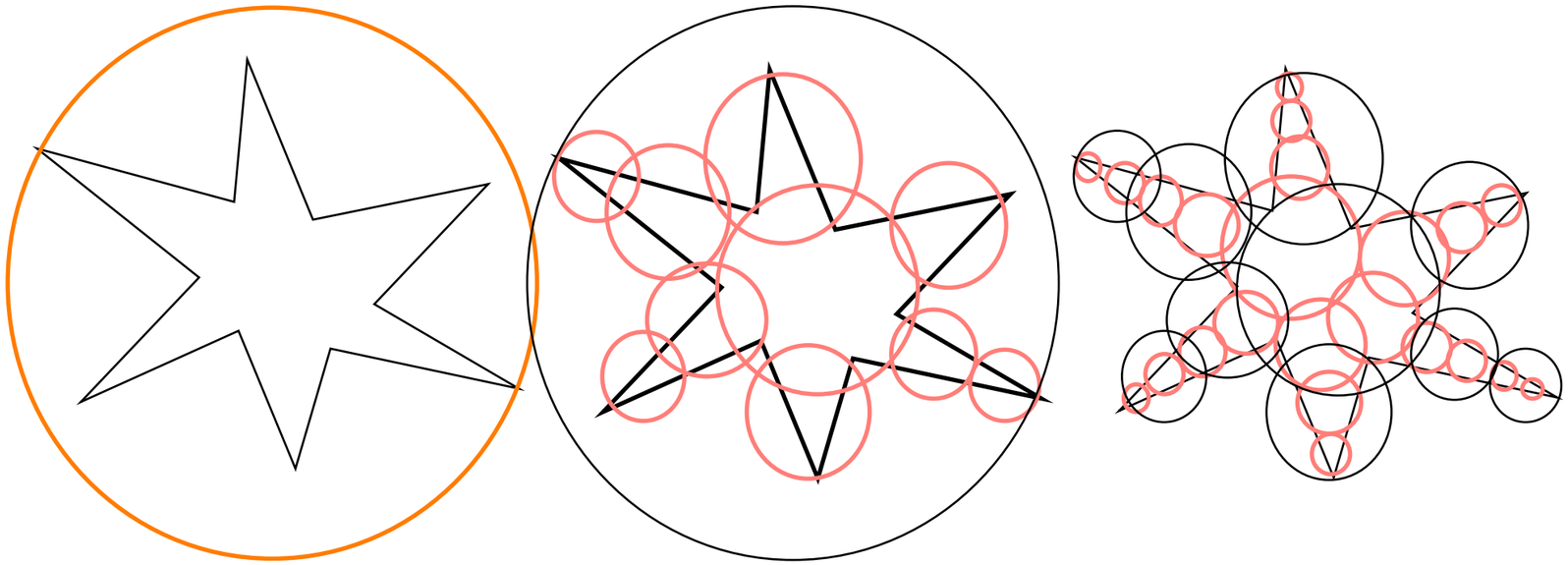
Convex only!

Split to convex parts.



From:
Henrik Bäcklund,
Niklas Neijman,
"AUTOMATIC MESH DECOMPOSITION FOR REAL-TIME COLLISION DETECTION",
2013

Single phase algorithms
Object hierarchies

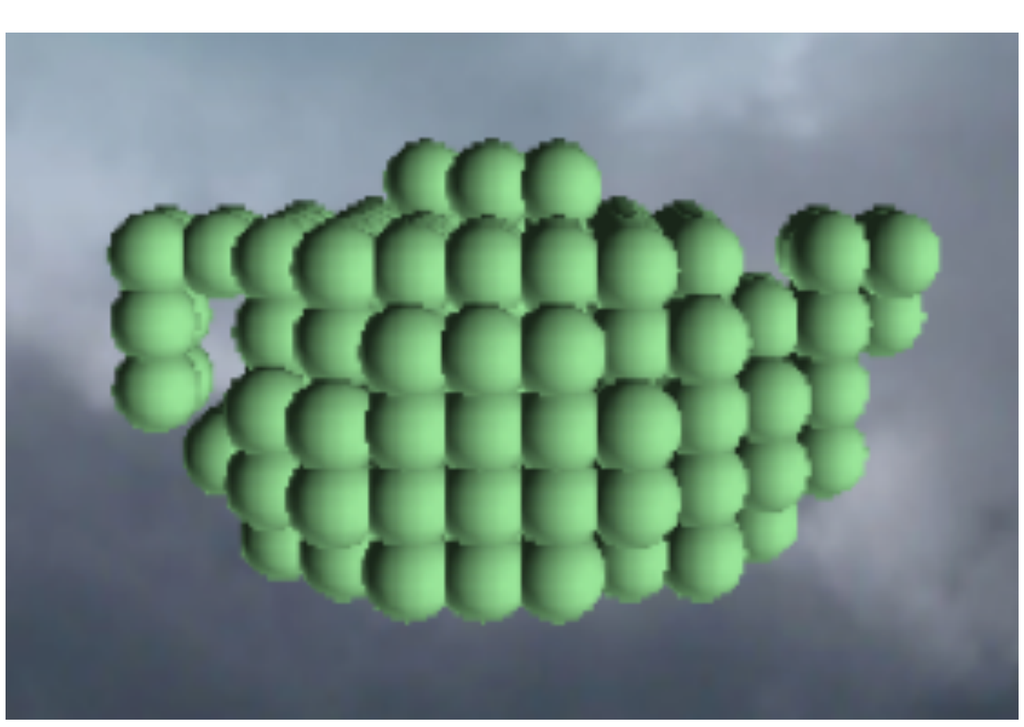


"Sphere trees"

Collision detection is done to the level that available time permits

Voxelization

Break down model to voxels, test with one sphere per one voxel



Teapot voxelized to 171 spheres

From Edhammer, "Rigid Body Physics for Synthetic Data Generation", 2016

Next time: Collision handing

What do you do when a collision is detected?

Also: Global phase and fractals.