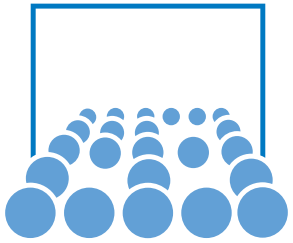
PSE Game Physics

Session (2)
Collisions: Sphere-Sphere, Sphere-Plane Interpenetrations

Atanas Atanasov, Philipp Neumann, Martin Schreiber

13.05.2011



Outline

Some very very basic math

- Left-handed system / Right-handed system
- Cross product
- Dot product

Linear transformations

- Object properties
- Object movement

Collisions

- Overview
- Sphere-Sphere
- Collision Normal
- Sphere-Plane
- Collision points

Outline

Right-handed system

- Our engine is written in a Right-handed system (OpenGL):

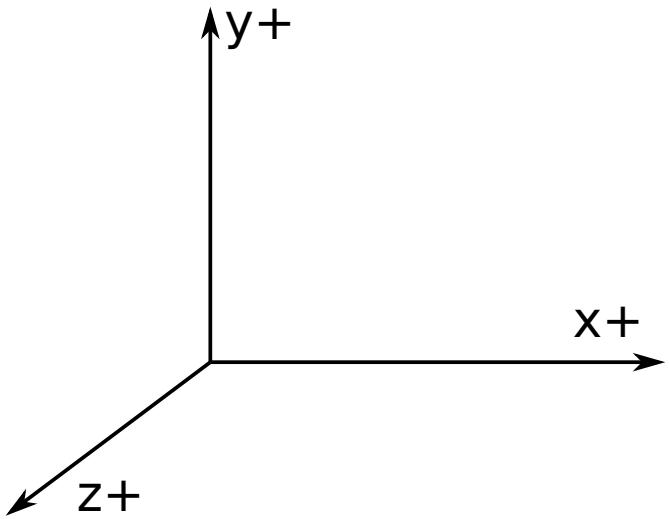
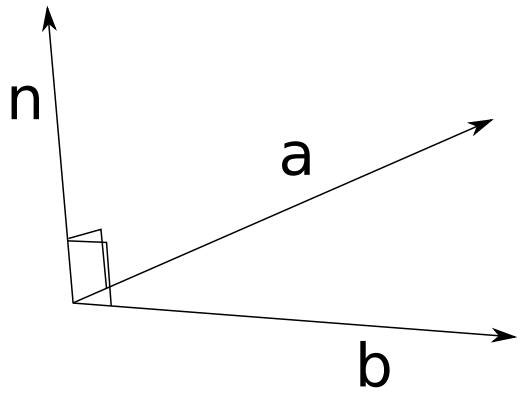


Figure: right-handed coordinate system

- Thus moving an object further away means **decreasing** the z-coordinate!
- All formulae are given in left handed basis!

Cross product



$$\vec{n} = \vec{a} \times \vec{b} = \begin{bmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{bmatrix}$$

- We mainly use it to compute a vector $\vec{n}$ which is perpendicular to two given vectors $\vec{a}$ and $\vec{b}$
- E. g. if a plane is given by a single point and 2 vectors, the **plane normal** can be computed
- Remember: We work with a right-handed coordinate system!

Dot product

$$\vec{a} \cdot \vec{b} = \sum_{i=1}^{3} a_i b_i$$

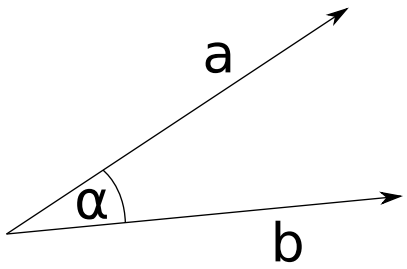


Figure: dot product

- Can be used to compute the cosine of the angle created by $\vec{a}$ and $\vec{b}$:

$$\vec{a} \cdot \vec{b} = |\vec{a}||\vec{b}| \cos(\alpha)$$

Outline

Object properties

- Position
- Rotation
- Velocity
- Mass
- ...

How can we express e. g. Position and Rotation?

How can we express e. g. Position and Rotation?

Modifications of Position and Rotation which we need for our physics engine:

- Translation: Move object from position $p(t)$ to $p(t + \Delta t)$
- Rotation: Rotate an object

Other Modifications not necessary for our physics engine:

- Shearing (Meaningless for rigid body objects)
- Scaling (Meaningless for rigid body objects)
- Projections (Only necessary for visualization, non-affine transformation)

How to describe the transformations?

Model- and World-space:

- The *space* in which the object is setup without any modifications is called the model-space.
- The object is moved/rotated during initialization or by applying a velocity in each timestep.
- This moves the object from model-space to world-space.

Handling transformations:

- Store the *position* of the object in a vector $\vec{p}$.
- Store the *rotation* of the object in a quaternion $\vec{q}$ (next session)
- Use a 4×4 matrix M which is created based on the position and quaternion to describe the transformation.
- (for most cases, only a 4×3 matrix is necessary)

Without any modifications of the object, we get the identity matrix.

Translation:

- A translation can be expressed by setting specific values in the last column of our 4*x*4 matrix *M*:

$$M_{translate} = \begin{pmatrix} 1 & & & t_x \\ & 1 & & t_y \\ & & 1 & t_z \\ & & & 1 \end{pmatrix}$$

- To allow a translation simply by applying the matrix *M* to a postion, we extend the objects position $\vec{p}$ by a fourth component which is set to 1 to the homogenous vector $\vec{\vec{p}}$.
- Example: $\vec{p} = (1, 3, -2)^T$

$$\begin{pmatrix} p'_x \\ p'_y \\ p'_z \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & . & . & t_x \\ . & 1 & . & t_y \\ . & . & 1 & t_z \\ . & . & . & 1 \end{pmatrix} \cdot \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} p_x + t_x \\ p_y + t_y \\ p_z + t_z \\ 1 \end{pmatrix} = \vec{\vec{p}}$$

Rotation:

- One way to express 3D Rotations is to by combining 2D rotations.
- E. g. 2D rotation around Z-Axis:

$$M_{rotate} = \begin{pmatrix} \cos\alpha & -\sin\alpha & . & . \\ \sin\alpha & \cos\alpha & . & . \\ . & . & 1 & . \\ . & . & . & 1 \end{pmatrix}$$

- Combining several rotation matrices, we can express every rotation.
- Outlook: This combination has a severe disadvantage (see slides of next session)

Putting everything together

- Now we can express rigid body transformations by applying all transformations to the object's position:

$$M_{translate} \cdot M_{rotate} \cdot \vec{p}$$

- By setting $M = M_{translate} \cdot M_{rotate}$ and using the *associative law* for linear transformations the transformation can be expressed by only a single matrix-vector expression:

$$M \cdot \vec{p}$$

- However, remember that also M has to be computed somehow!

Inverse Transformations

- It's also possible to **transform objects from world-space to the model-space**.
- This becomes handy when intersection tests should be can be easily handled in the space of a model (e. g. plane-box intersection tests)
- The transformation of an object O_1 to the space of another object O_2 can be simply handled by using the inverse of the **model-matrix**:

$$M_2^{-1} \cdot M_1$$

- Hint: read this formula from right to left:
  - Firstly, object 1 is transformed to the world space
  - By applying the inverse transformation for object 2, transforms object 1 to the model-space of object 2

How to transform Vectors?

- A vector at an arbitrary position cannot be transformed simply by applying the model-matrix.
- To transform a vector $\vec{v}$ from one space to another given the point transformation matrix M, the vector $\vec{v}$ can be either transformed by
  - Applying M to the origin point as well as to the vector $\vec{v}$ and subtracting both values to get the appropriate solution or
  - by applying the inverse transpose of M (see next slide)

Inverse-transpose of a matrix (1/2)

- Here we give a short sketch (See `http://www.faqs.org/faqs/graphics/algorithms-faq/` for more details)
- We start by splitting up the matrix M into the $3x3$ matrix L and the translation vector t.
- Then we can describe the transformation of an arbitrary vector $\vec{v}$ by the transformation of its starting and end point $\vec{p}$ and $\vec{q}$:

$$\vec{v}' = (L\vec{p} + t) - (L\vec{q} + t) = L(\vec{p} - \vec{q}) = L(\vec{v})$$

- Note that L is assumed not only to be a rotational matrix - this would tremendously simplify our assumtions since $L_{rotational} = L^{-T}_{rotational}$. However, we stay generic.

Inverse-transpose of a matrix (2/2)

- When transforming both vectors to world-space, their **angle stays the same** which can be expressed by a **dot-product**.
- Since we know, that $\vec{v}$ can be transformed by applying L, we can rewrite the equation to

$$\vec{n}' \cdot \vec{v}' = (N\vec{n}) \cdot (L\vec{v})$$

- By replacing the dot product using the transposed:

$$\vec{n}' \cdot \vec{v}' = (N\vec{n})^T (L\vec{v}) = \vec{n}^T N^T L\vec{v}$$

- Since the angles of the 2 vectors $\vec{n}$ and $\vec{v}$ also have to be equal in model-space, we can assume that $N^T L = I$.
- Thus, we get $N = L^{-T}$ for the matrix transforming normals and vectors to the world-space

Things, you should remember...

- You can simply invert the transformation to transform ”backwards”
- Use the inverse-transpose to transform normals/vectors
-

Outline

- **Some very very basic math**
  - Left-handed system / Right-handed system
  - Cross product
  - Dot product
- **Linear transformations**
  - Object properties
  - Object movement
- **Collisions**
  - Overview
  - Sphere-Sphere
  - Collision Normal
  - Sphere-Plane
  - Collision points

Worksheet schedule for collision tests

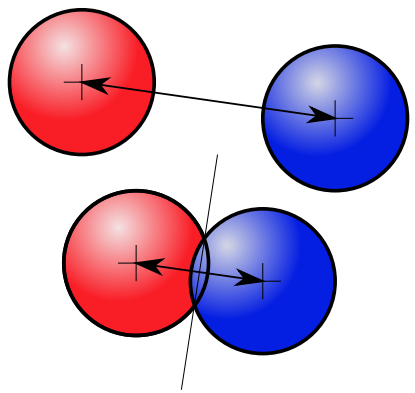
The implementation of the collision tests is splitted up over different worksheets

	Plane	Sphere	Box
Plane	x	-	-
Sphere	WS2	WS2	-
Box	WS4	WS4	WS5

- x: meaningless
- -: Symmetric

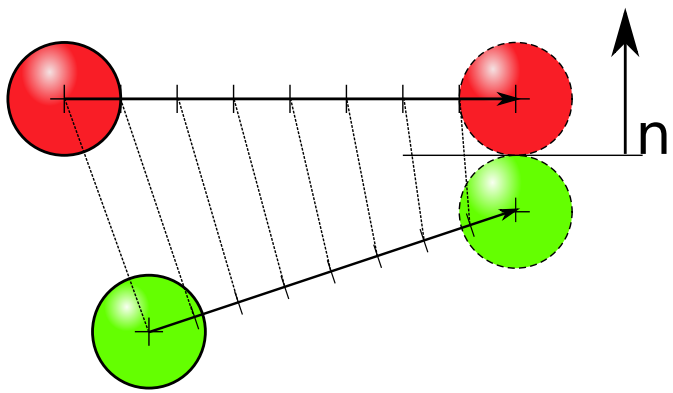
Collision: Sphere-Sphere

- Really easy to compute!!
- Only test whether the distance of the 2 center points is less that the sum of both radii
- But we need more collision data in order to handle collisions



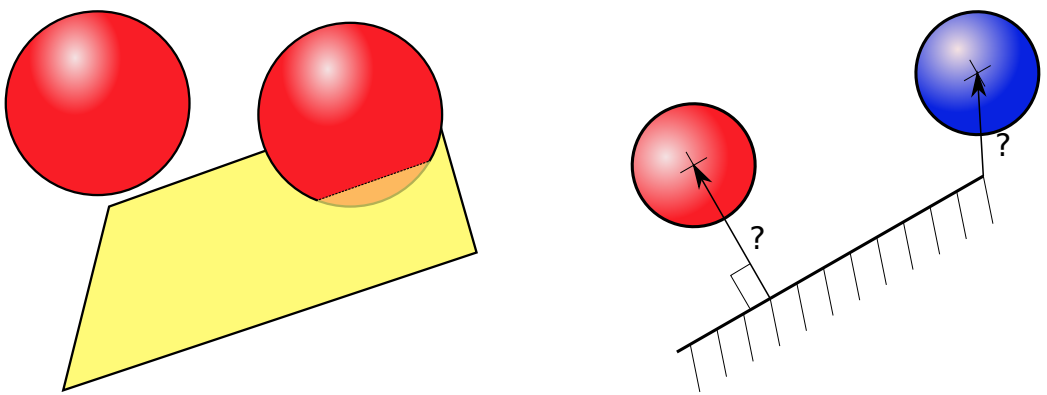
Collision Normal

- We need to know in which **direction** we have to **apply some force** due to the collision.
- For spheres we assume to determine the collision normal when the 2 spheres are very close.
- Thus the normal from the first sphere aiming to the other one is given by computing the normal based upon both sphere's centers.



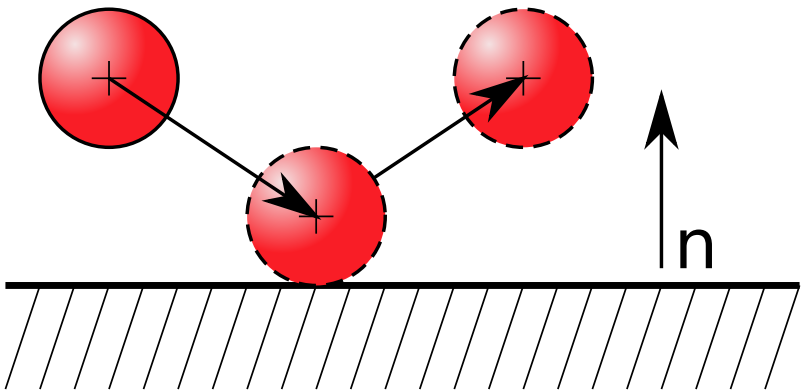
Collision: Sphere-Plane (1/2)

- How should we handle sphere-plane collision tests?
- The Sphere is given by its position and radius
- The Plane is given by its position, size and orientation



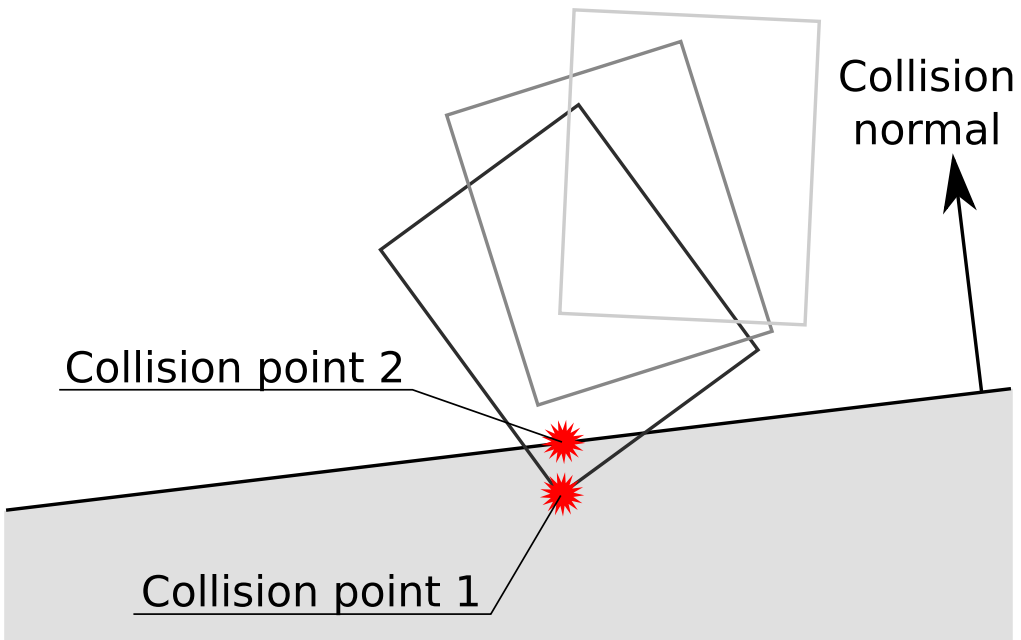
Collision: Sphere-Plane (2/2)

- Transforming the sphere to plane space simplifies our further computations.
- Then computing the intersections of the sphere with the plane as well as with its edges and vertices gets really easy to compute.
- Note that the **collision normal has to be transformed back to world-space**!



Collision points

- Since an interpenetration can occur during a timestep, the collision point for every object has to be determined.
- This collision point gets important when applying a impulse to an object which results in a angular momentum.



Resolving Interpenetration

- If a binary-collision for 2 objects was detected, this interpenetration has to be solved somehow
- The first assumption is, that the interpenetration can be solved by moving the objects **parallel to the collision normal**
- Question: How far should every object be moved?

Resolving Interpenetration (2/2)

- We start by considering 2 extreme cases assuming that resolving the interpenetration depends on the rigid-bodys mass
  - O1 has infinite mass, O2 has mass 1 $\Rightarrow$ Only O2 is moved
  - O2 has infinite mass, O1 has mass 1 $\Rightarrow$ Only O1 is moved
- Linearly interpolating inbetween these 2 extreme cases, the fraction d_i of the interpenetration distance for object i can be computed by:

$$d_i = \frac{m_i^{-1}}{m_i^{-1} + m_{\bar{i}}^{-1}}$$

with m_i^{-1} being the inverse mass of the corresponding object and $\bar{i}$ as the object colliding with object i.