Bounding Enclosures & Unity Colliders

(SENG 463 - Game Programming)

Dr.Çağatay ÜNDEĞER

Research and Innovation Director

SimBT Inc.

e-mail :

cagatay.undeger@simbt.com.tr

cagatay@undeger.com

Outline

- Geometric Objects and Queries
- Bounding Enclosures
  - Axis-aligned bounding boxes
  - General bounding boxes
  - Bounding spheres
  - Bounding ellipsoids

Geometric Objects and Queries

- Large games involve the storage and maintenance of a huge number of geometric objects
- Many of these change dynamically over time
- Game software needs to be able to access this information efficiently.

Geometric Objects and Queries

- Access to these structures are:
- Queries (asking questions about the objects of the database)
- Updates (making changes to these objects)

Geometric Objects and Queries

- Queries typically involve determining what things are close by
- Nearby objects are more likely to have interesting interactions in a game (collisions or attacks).

Geometric Objects and Queries

- In a shooting game, we may be interested in which other players have a line-of-sight to my own entity.
- To decide which are seen and, to shoot at

Geometric Objects and Queries

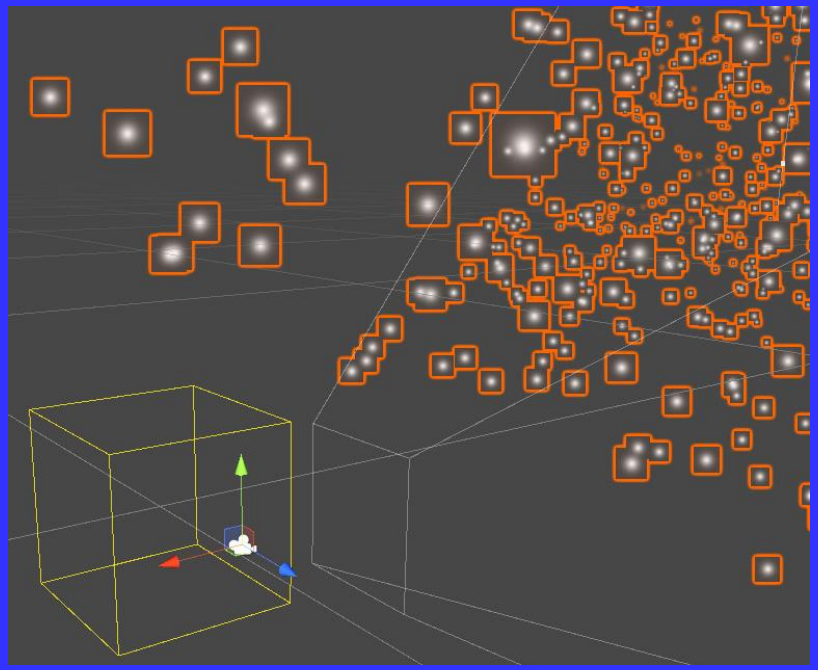
- “Find all law enforcement vehicles within half a mile radius of the player's car", might be a query for a car theft game.
- Such queries involve both geometric properties (half a mile radius) and
- Non-geometric properties (law enforcement)
- Such hybrid queries may involve a combination of multiple data structures.

Bounding Enclosures

- When storing complex geometric objects in a spatial data structure
- Common to first approximate the object by a simple enclosing structure.
- Bounding enclosures are often very valuable to approximate an object as a filter in
- Objects in rendering, collision detection, etc.

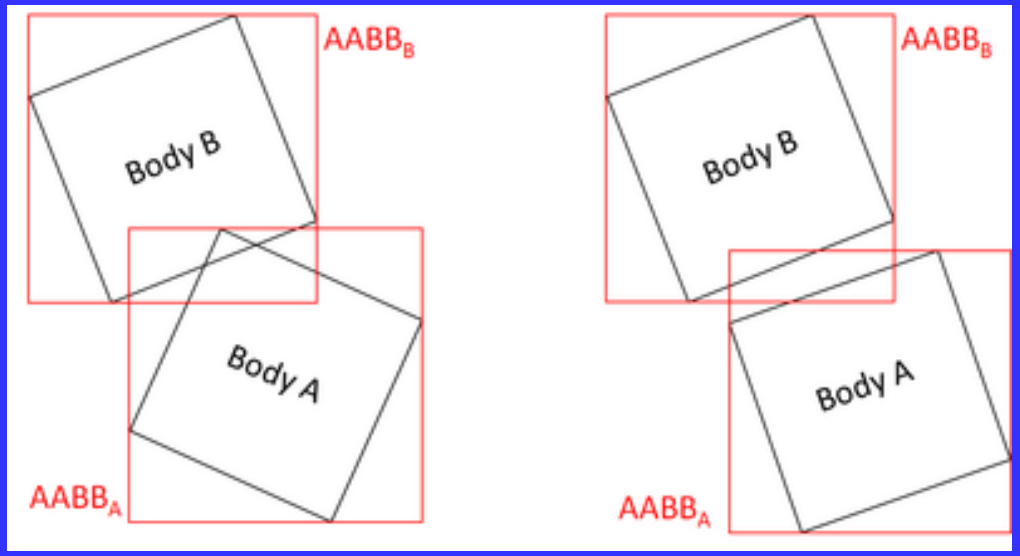
Bounding Enclosures

- For instance, in unity every object has rendering bounding box
- Updated every frame when anything changed
- Rendered objects are found using bounds



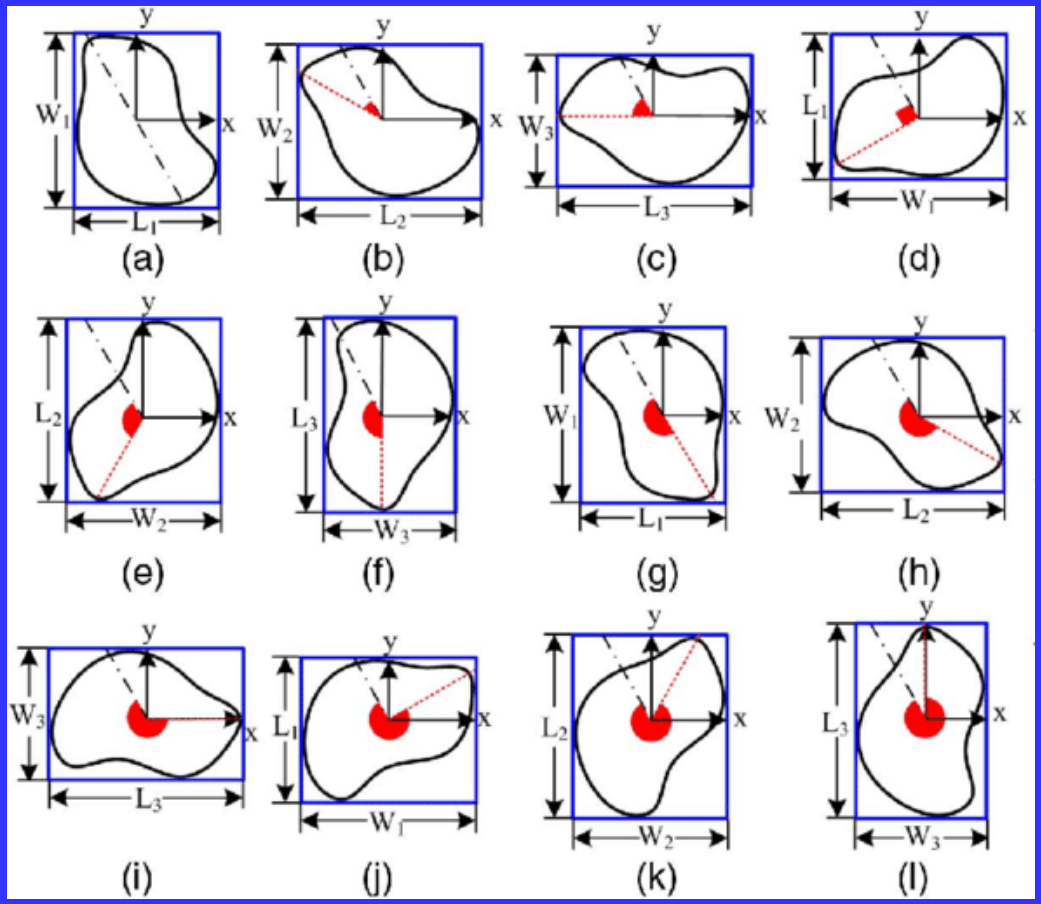
Bounding Enclosures

- If we need collision check then...
- If the bounding enclosures do not collide, then the objects to not collide.
- If they do, then we apply a more expensive intersection test to the actual objects.



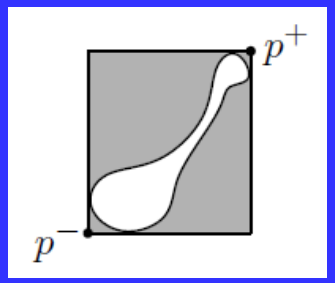
Axis-Aligned Bounding Boxes

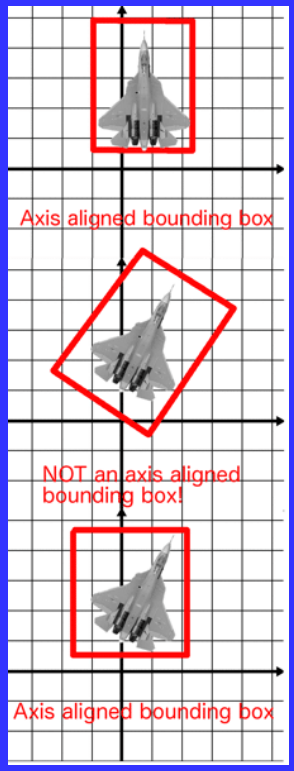
- This is an enclosing rectangle whose sides are parallel to the coordinate axes



Axis-Aligned Bounding Boxes

- Commonly called AABBs
- Very easy to compute
- The corners are based on the minimum and maximum x,y,(z) coordinates.
- An AABB can be represented by two points
  - For example, the lower-left point p- and the upper-right point p+.
- AABBs are preserved under translation, but not under rotation

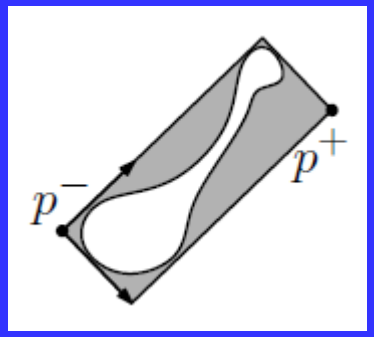




General Bounding Boxes

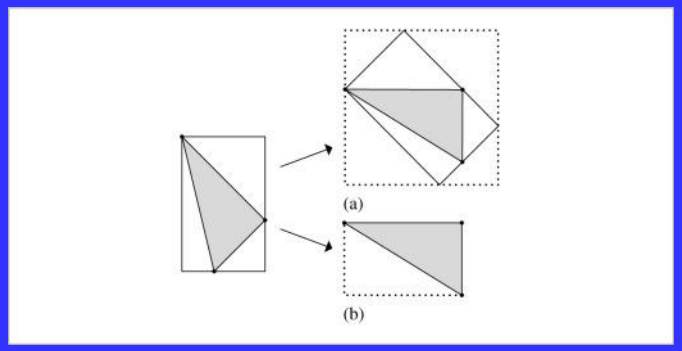
- The principal shortcoming of axis-parallel bounding boxes is:
- That it is not possible to rotate the object without recomputing the entire bounding box.

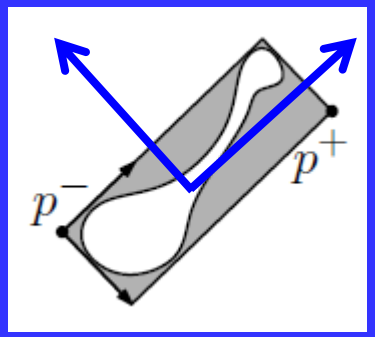
- In contrast, general (arbitrarily-oriented) bounding boxes can be rotated without the
- need to recompute them



General Bounding Boxes

- A natural approach to represent such a box is to describe the box as an AABB but relative to a different coordinate axes.

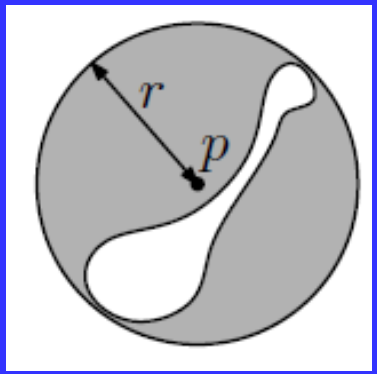




- Computing minimum bounding box is not simple.
- Need AABB for an appropriate rotation
- Determining the best rotation (especially in 3-space) is quite tricky.

Bounding Spheres

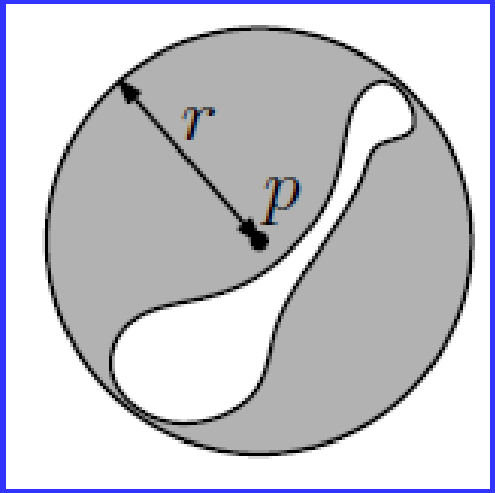
- These are among the most popular bounding enclosures.
- A sphere can be represented by a center point p and a radius r
- Spheres are invariant under rigid transformations,
  - translation and rotation.



- Minimum bounding spheres are tricky to compute exactly.

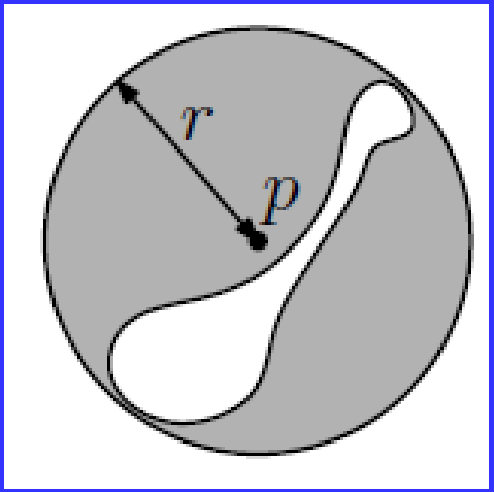
Bounding Spheres

- A commonly used heuristic
- To first identify (by some means?) a point p that lies near the center of the body
- Then set the radius just large enough so that a sphere centered at p encloses the body.



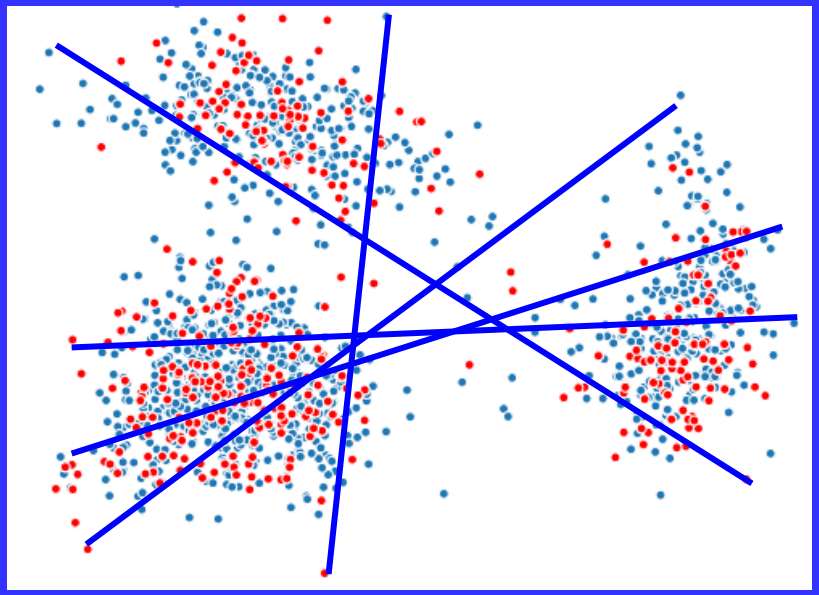
Bounding Spheres

- Identifying the point p is tricky.
- One heuristic is set p to be the matematical / average center of the body.
- Another heuristic is set p to be the center of gravity of the body.



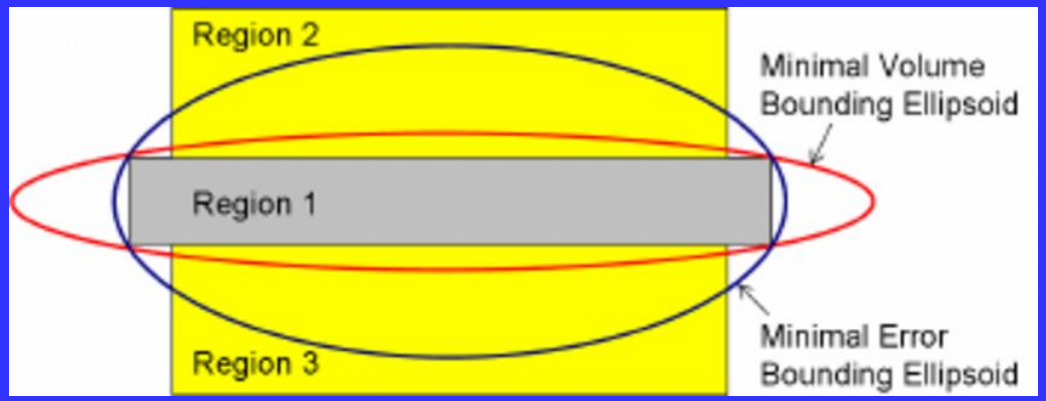
Bounding Spheres

- Another is to compute two points a & b on the body that are farthest apart from each other.
- This is called the diametrical pair.
- Define p to be the midpoint of the diametrical pair segment.



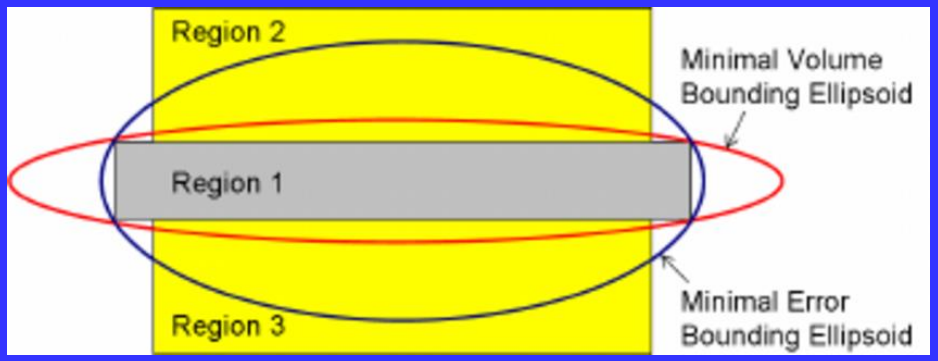
Bounding Ellipsoids

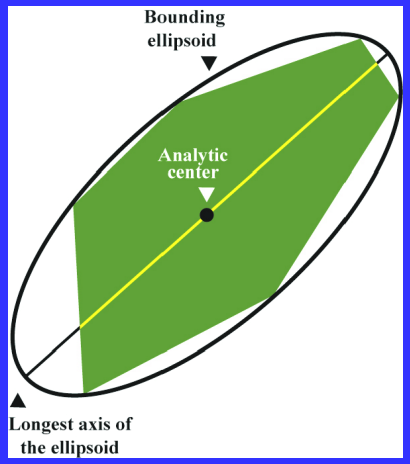
- The main problem with spheres is that some objects are not well approximated by a sphere *(problem also exists with axis-parallel bounding boxes)*
- A ellipse (or generally, an ellipsoid in higher dimensions) is just a sphere under an affine transformation.



Bounding Ellipsoids

- Similar to boxes, ellipsoids may either be axis-parallel or arbitrary.

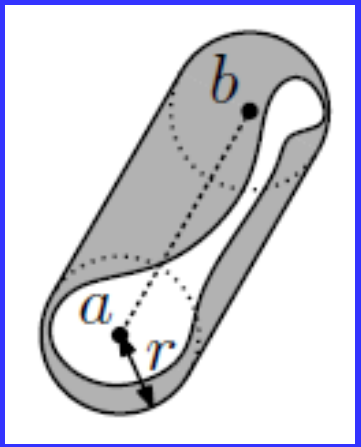




- Minimum bounding ellipsoids are difficult to compute exactly.
- May use the diametrical pair for the principal axis of the ellipse, but this is not generally optimal.

Bounding Capsules

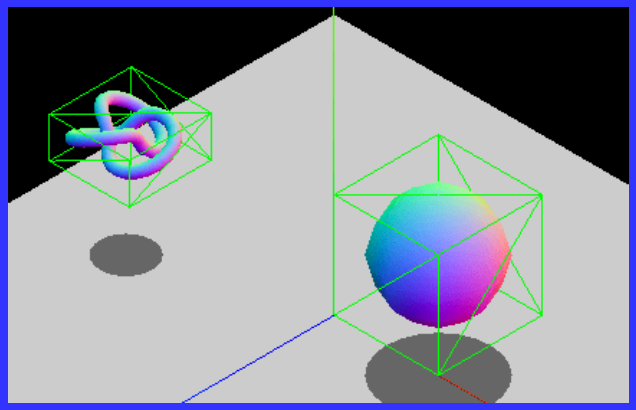
- This shape can be thought of as a rounded cylinder.
- It consists of the set of points that lie within some distance r of a line segment ab

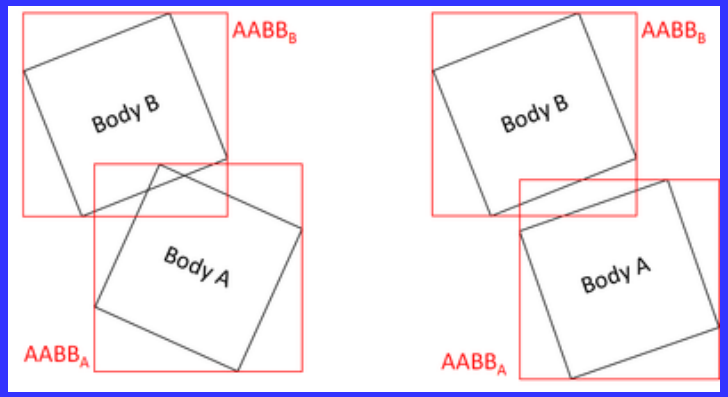


- A line segment is chosen
- All the points that is within distance r to that line segment is inside bounds

Collision Detection

- By enclosing an object within a bounding enclosure,
- Collision detection cost is reduced by predetermining whether two such enclosures may intersect each other or not.
- Note that if we support k different types of enclosure, we need to handle all possible pairs of combinations of collisions.



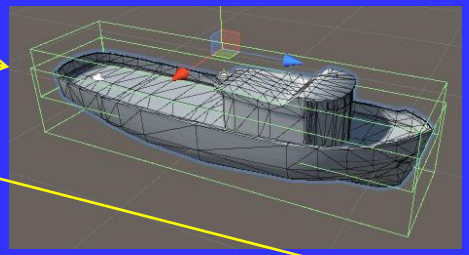


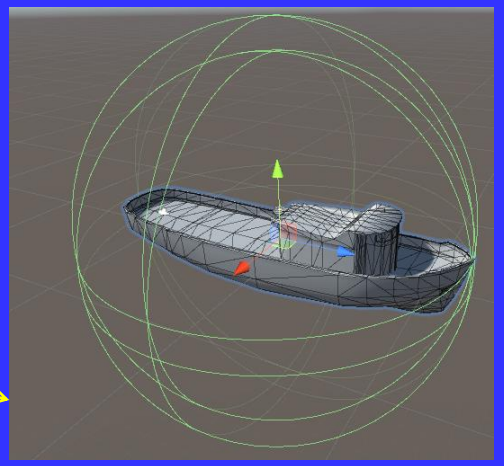
Colliders In Unity

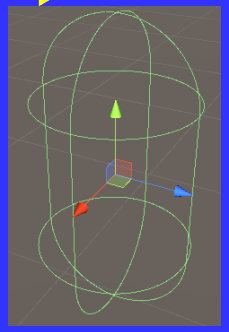
- In Unity, collision detection is not done using real 3D mesh of the objects
- Since collision detection is costly, shape of the collision geometry is chosen by the developer
- Instead of real 3D mesh of object, chosen collider geometries of the object is used for computations

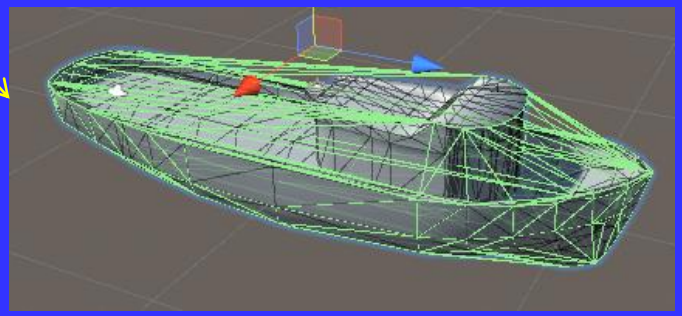
Colliders In Unity

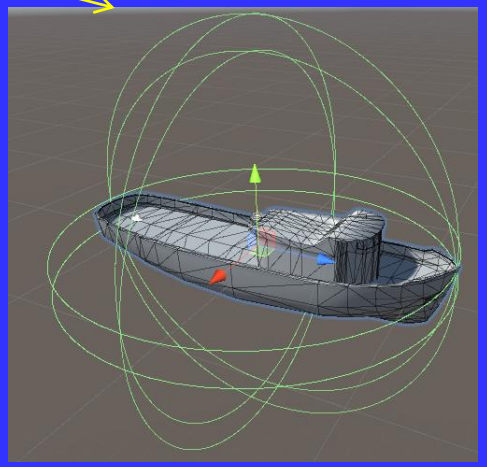
- Oriented colliders are added to objects if required
- If no colliders added, no collision detected
- Standard 3D collider types:
  - Box Collider
  - Sphere Collider
  - Capsule Collider
  - Mesh Collider
    (convex / concave)





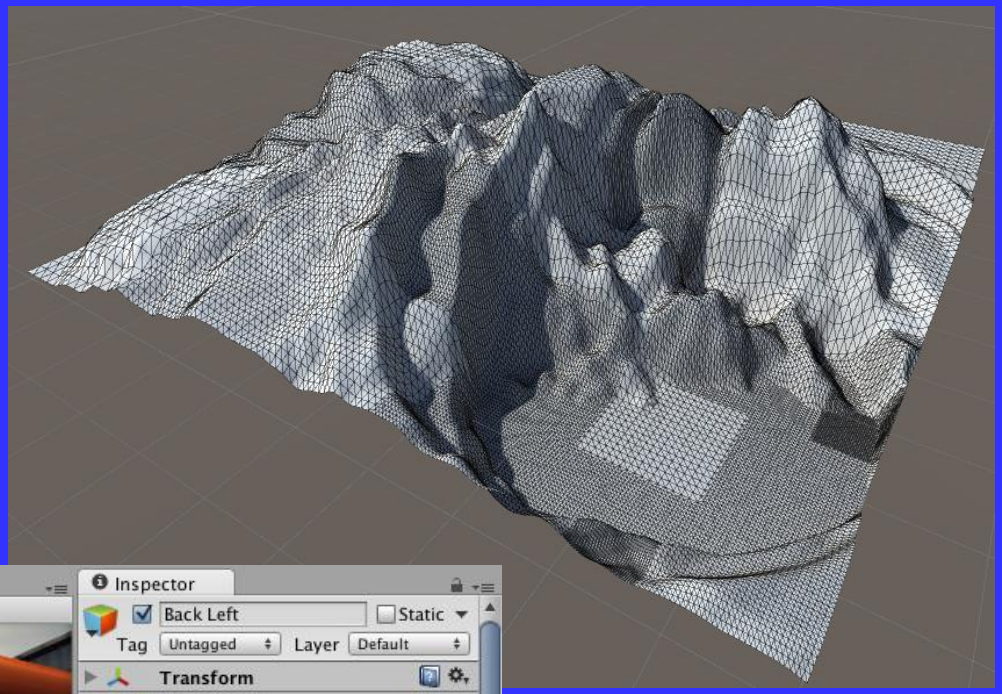




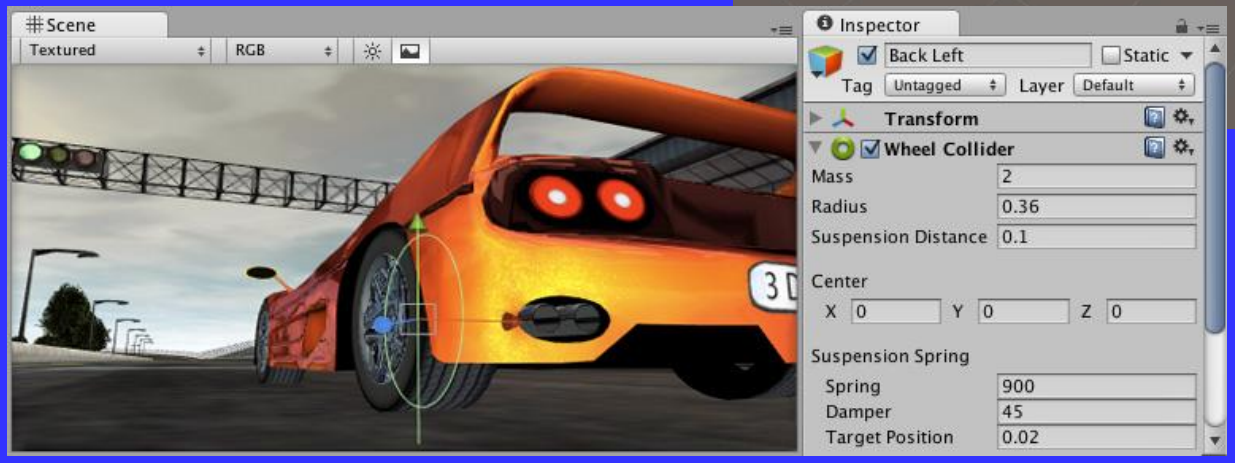


Colliders In Unity

- There are special colliders also:
  - Terrain Collider



  - Wheel Collider

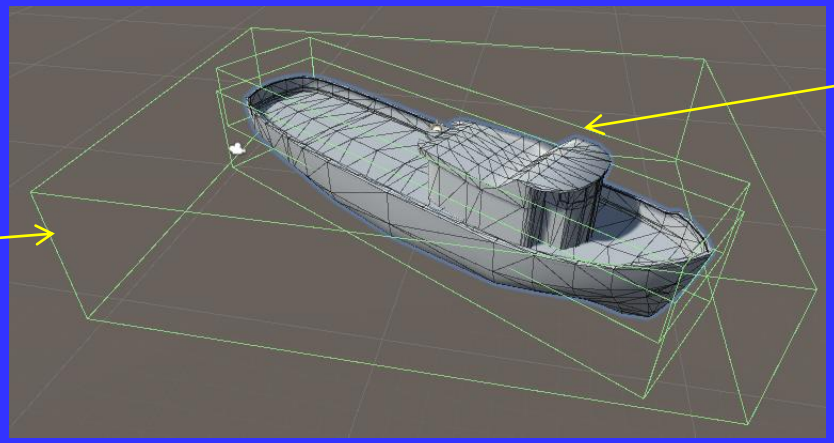


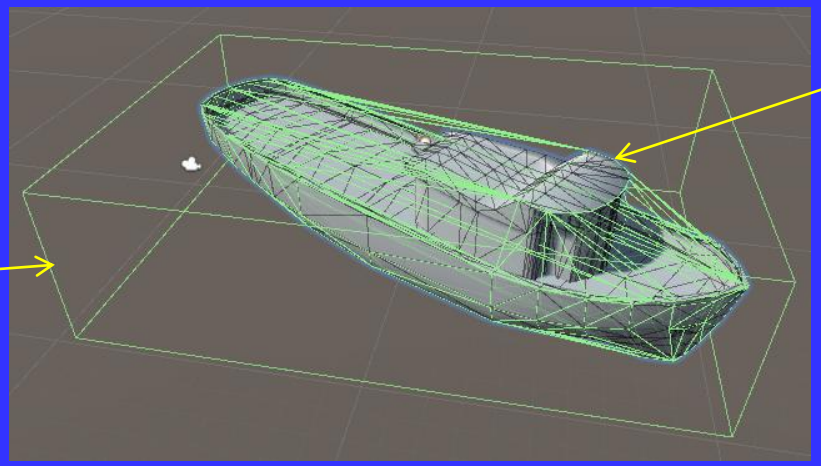
Colliders In Unity

- Even there are simplified colliders on 3D meshes,
  - Unity also automatically generates an Axis-Aligned Bounding Boxes (AABBs) for these meshes
  - Accessed via Collider.Bounds
- Bounds are used for further performance optimization

Colliders In Unity

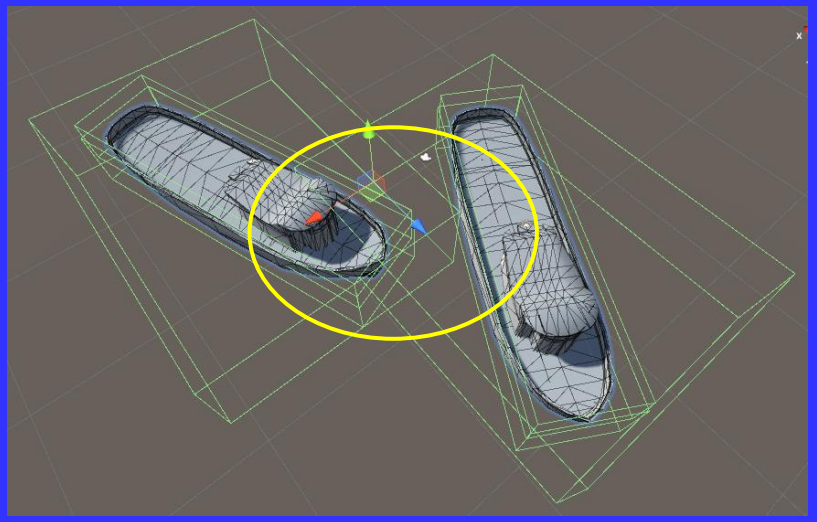
- Bounds are used for further performance optimization

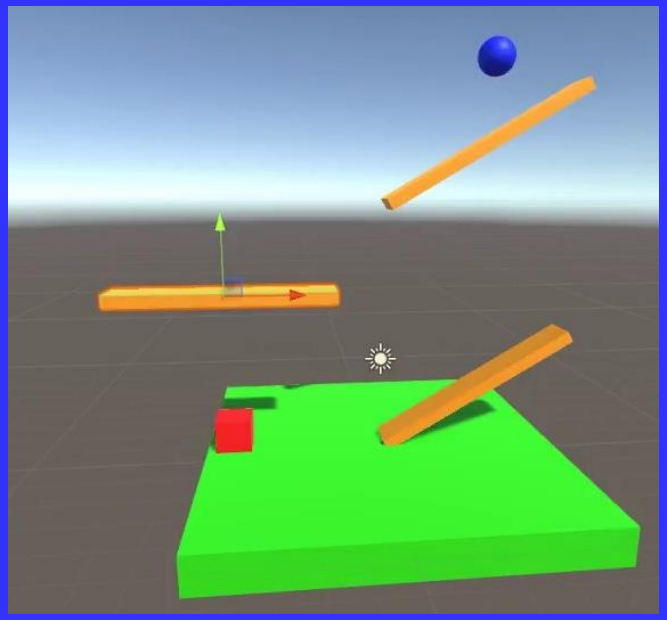


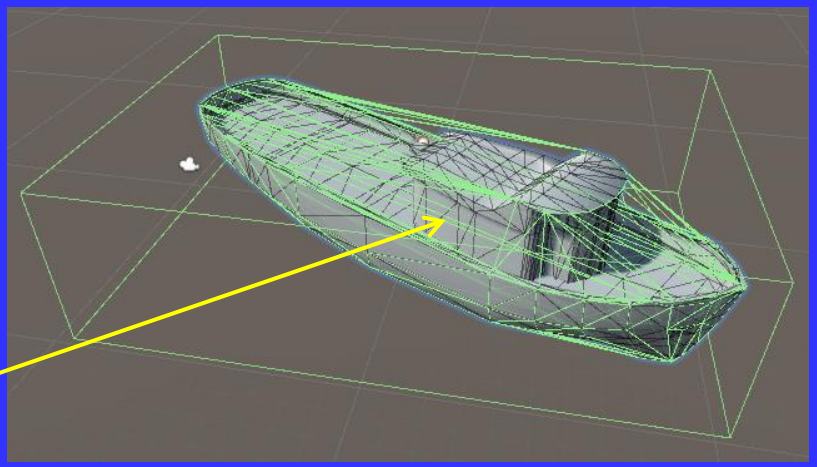


Usage of Colliders In Unity

- Colliders used in:
  - Collision Detections
  - Raycast / Hit Tests
  - Rigid Body Physics







a ray

Raycast In Unity

Physics.Raycast

Declaration

public static bool **Raycast**(Vector3 **origin**, Vector3 **direction**, out RaycastHit **hitInfo**, float **maxDistance**, int **layerMask**, QueryTriggerInteraction **queryTriggerInteraction**);

Parameters

origin	The starting point of the ray in world coordinates.
direction	The direction of the ray.
hitInfo	If true is returned, `hitInfo` will contain more information about where the closest collider was hit. (See Also: RaycastHit).
maxDistance	The max distance the ray should check for collisions.
layerMask	A Layer mask that is used to selectively ignore colliders when casting a ray.
queryTriggerInteraction	Specifies whether this query should hit Triggers.

Returns

bool Returns true when the ray intersects any collider, otherwise false.

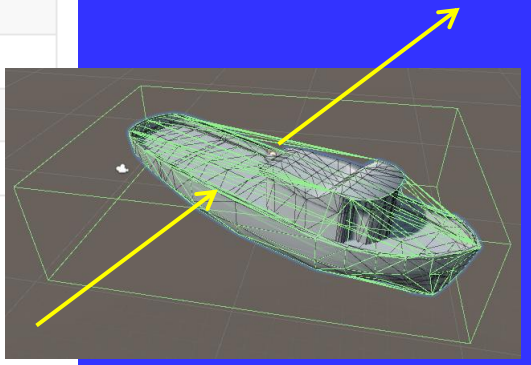
Description

Casts a ray, from point `origin`, in direction `direction`, of length `maxDistance`, against all colliders in the Scene.

You may optionally provide a LayerMask, to filter out any Colliders you aren't interested in generating collisions with.

Specifying `queryTriggerInteraction` allows you to control whether or not Trigger colliders generate a hit, or whether to use the global Physics.queriesHitTriggers setting.

Notes: Raycasts will not detect Colliders for which the Raycast origin is inside the Collider.



Raycast In Unity

```
using UnityEngine;

public class RaycastExample : MonoBehaviour
{
    // See Order of Execution for Event Functions for information on FixedUpdate() and Update() related to physics queries
    void FixedUpdate()
    {
        RaycastHit hit;

        if (Physics.Raycast(transform.position, -Vector3.up, out hit, 100.0f))
            print("Found an object - distance: " + hit.distance);
    }
}
```

```
RaycastHit hit;
// Does the ray intersect any objects excluding the player layer
if (Physics.Raycast(transform.position, transform.TransformDirection(Vector3.forward), out hit, Mathf.Infinity, layerMask))
{
    Debug.DrawRay(transform.position, transform.TransformDirection(Vector3.forward) * hit.distance, Color.yellow);
    Debug.Log("Did Hit");
}
```

Declaration

public Vector3 **TransformDirection**(Vector3 **direction**);

Description

Transforms `direction` from local space to world space.

This operation is not affected by scale or position of the transform. The returned vector has the same length as `direction`.

You should use Transform.TransformPoint for the conversion if the vector represents a position rather than a direction.

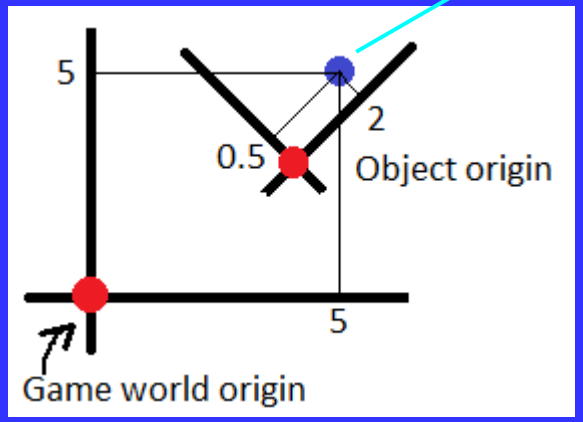
Transformations in Unity

- With Respect to a GameObjects “Transform”:
  - Local to World Transformations:
    - Transform.TransformPoint
    - Transform.TransformDirection
    - Transform.TransformVector
- Inverse computations are:
  - World to Local Transformations:
    - Transform.InverseTransformPoint
    - Transform. InverseTransformDirection
    - Transform. InverseTransformVector

TransformPoint in Unity

- **TransformPoint** transforms position from local space to world space.
- It is affected by **position**, **rotation** and **scale** of game object that you call and also its parent game objects.

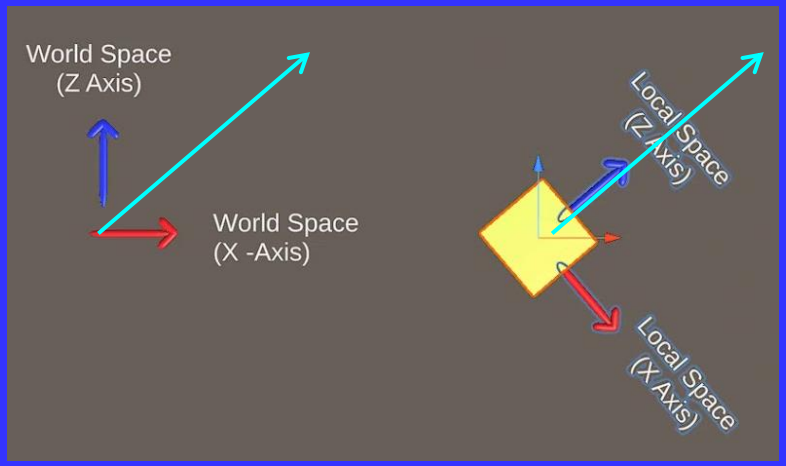
Local coordinate = (2, 0.5, 0)
World coordinate = (5, 5, 0)



TransformDirection in Unity

- **TransformDirection** is used to transform a direction from local space to world space.
- TransformDirection is not affected by position and scale. It is **only affected by rotation** and magnitude is preserved.

World direction = (0.7, 0, 0.7) **Local direction = (0, 0, 1)**

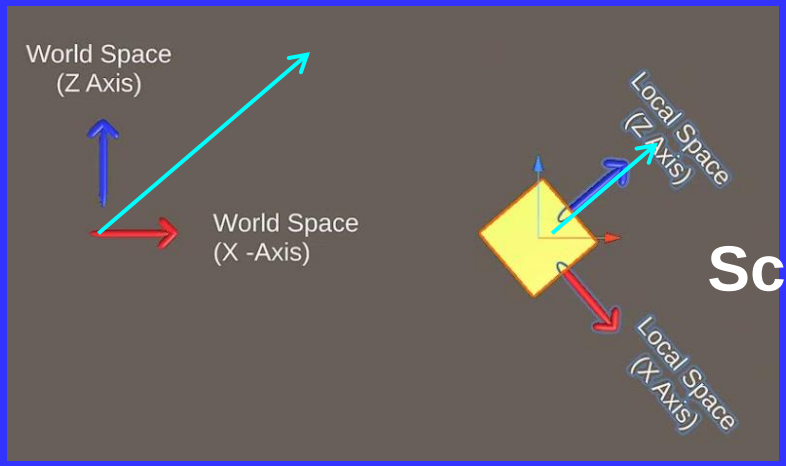


TransformVector in Unity

- **TransformVector** is used to transform a direction from local space to world space.
- **TransformVector** is not affected by position.
- But It is **affected by scale** and magnitude is changed.

World vector = (1.4, 0, 1.4)

Local vector = (0, 0, 1)



Scale = (2, 2, 2)