MVD: Engine Programming

05 - Collisions

alunthomas.evans@salle.url.edu

laSalle ENG
Universitat Ramon Llull

Collision

In which situations do we need to detect collisions?

Collision

In which situations do we need to detect collisions?

Player controller

Non-player movement

Attacks/interaction

Line of site visualisation

Picking

etc.

Collision detection is essential for all games

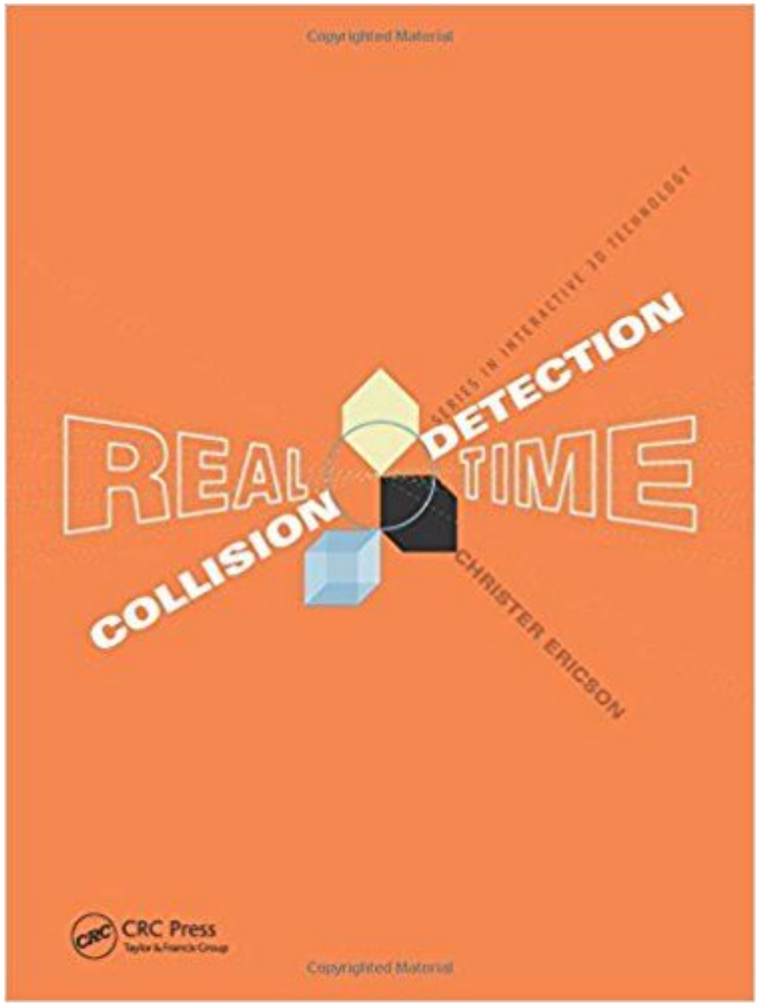
… and is essentially maths

We are going to focus on one useful case

(and leave the rest as 'an exercise for the reader')

This book is 'the bible':

Ericson, C. (2004). *Real-time collision detection*. CRC Press.

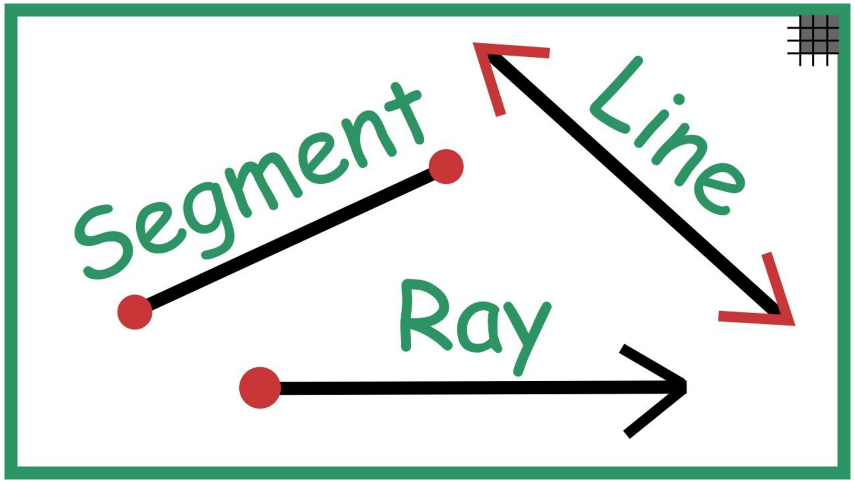


Collision detection occurs between primitives

Can you think of useful primitives for collision detection?

Intersect Segment - Box

Probably the most useful ‘general collision test’ (although Box - Box is also useful)



Why is *segment-box* more useful than *box-box*

What configuration of segments would you need to create an FPS controller, if the world were made of boxes?

Defining Segment and Box

Defining a segment is easy, it’s just two points.

Or is it? Is there a better way

How can we define a box?

Intersecting segment triangle

A box is 6 quads, and a quad is two triangles.

It turns out the most efficient* method of testing rays against boxes is essentially test against all triangles of the box

* I have not done tests, it's what the book says :P

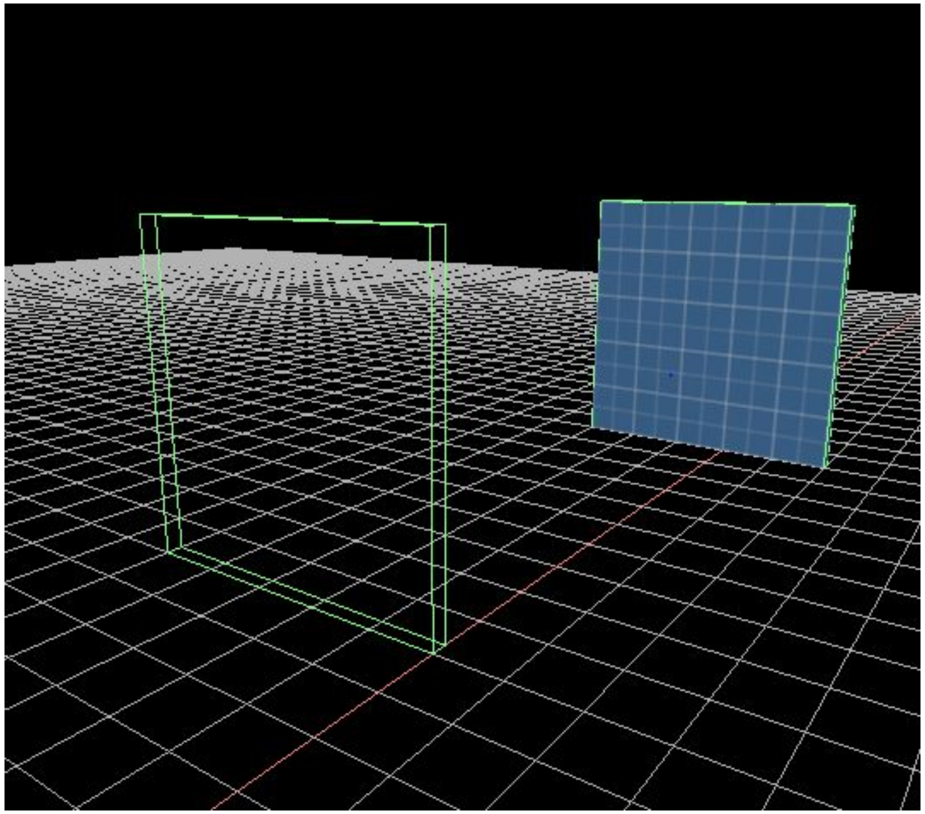
A box collider component in our engine

A new component type: 'Collider'

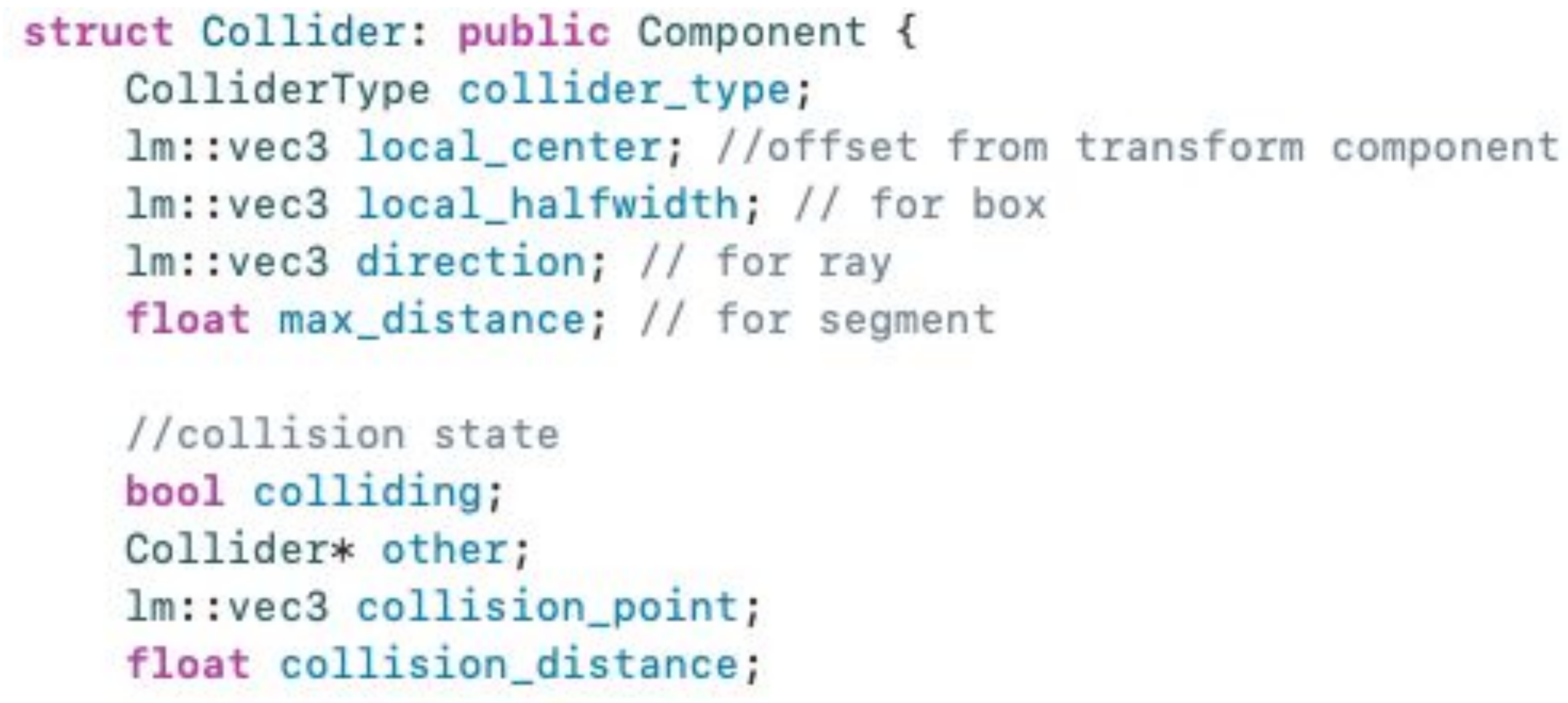
halfwidth dimensions in xyz

center offset in xyz

- why? because we can't guarantee that center of mesh is where transform is



Collider component



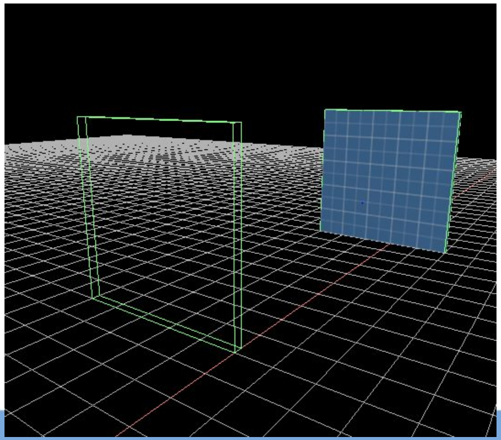
```
struct Collider: public Component {
    ColliderType collider_type;
    lm::vec3 local_center; //offset from transform component
    lm::vec3 local_halfwidth; // for box
    lm::vec3 direction; // for ray
    float max_distance; // for segment

    //collision state
    bool colliding;
    Collider* other;
    lm::vec3 collision_point;
    float collision_distance;
```

Testing all colliders

In a old-fashioned OOP engine structure, we would have to parse the entire scene to find all collidable objects, before calculating collisions

But with our amazing ECS engine, all our colliders are stored in a simple array!



Today: write segment box collision

I give you:

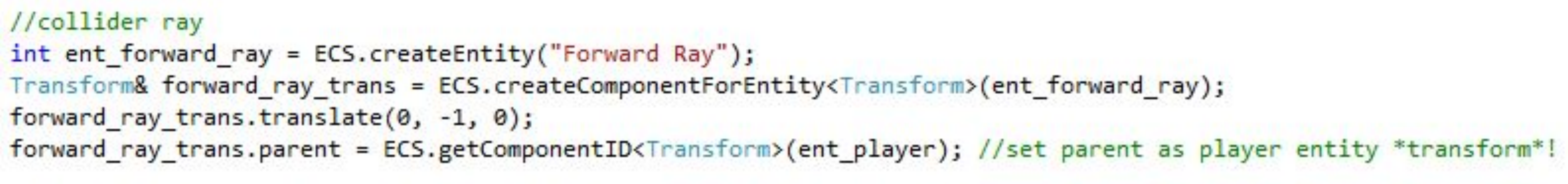
- a scene with two box colliders and one ray
- a ‘Collider’ component (see Component.h)
- a DebugSystem which draws the Collider Component
- code (from Ericson) which calculates the collision between primitives

You give me:

- an function which intersects a ray(segment) and box

Before we look at that… scene hierarchy

This code:

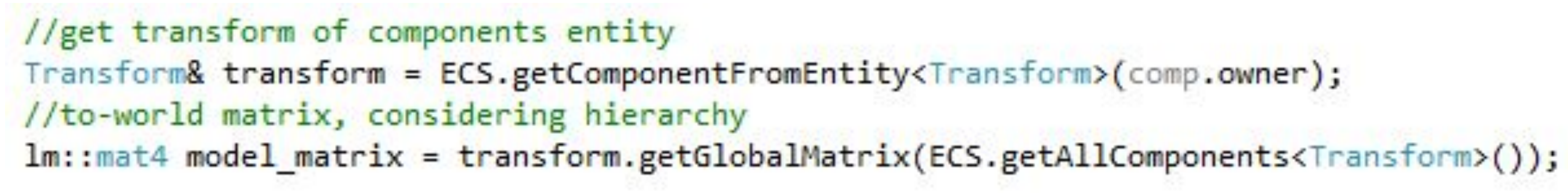


```
//collider ray
int ent_forward_ray = ECS.createEntity("Forward Ray");
Transform& forward_ray_trans = ECS.createComponentForEntity<Transform>(ent_forward_ray);
forward_ray_trans.translate(0, -1, 0);
forward_ray_trans.parent = ECS.getComponentID<Transform>(ent_player); //set parent as player entity *transform*!
```

sets the ray’s transform’s parent to the player transform.

Global Matrix used in Graphics System

The ‘global’ or ‘to world’ matrix is the model matrix of an object multiplied by its parent (multiplied by *its* parent etc)



```
//get transform of components entity
Transform& transform = ECS.getComponentFromEntity<Transform>(comp.owner);
//to-world matrix, considering hierarchy
lm::mat4 model_matrix = transform.getGlobalMatrix(ECS.getAllComponents<Transform>());
```

This is obviously used for drawing, but you also need it for **any algorithm which runs in world space** (e.g. collision detection)

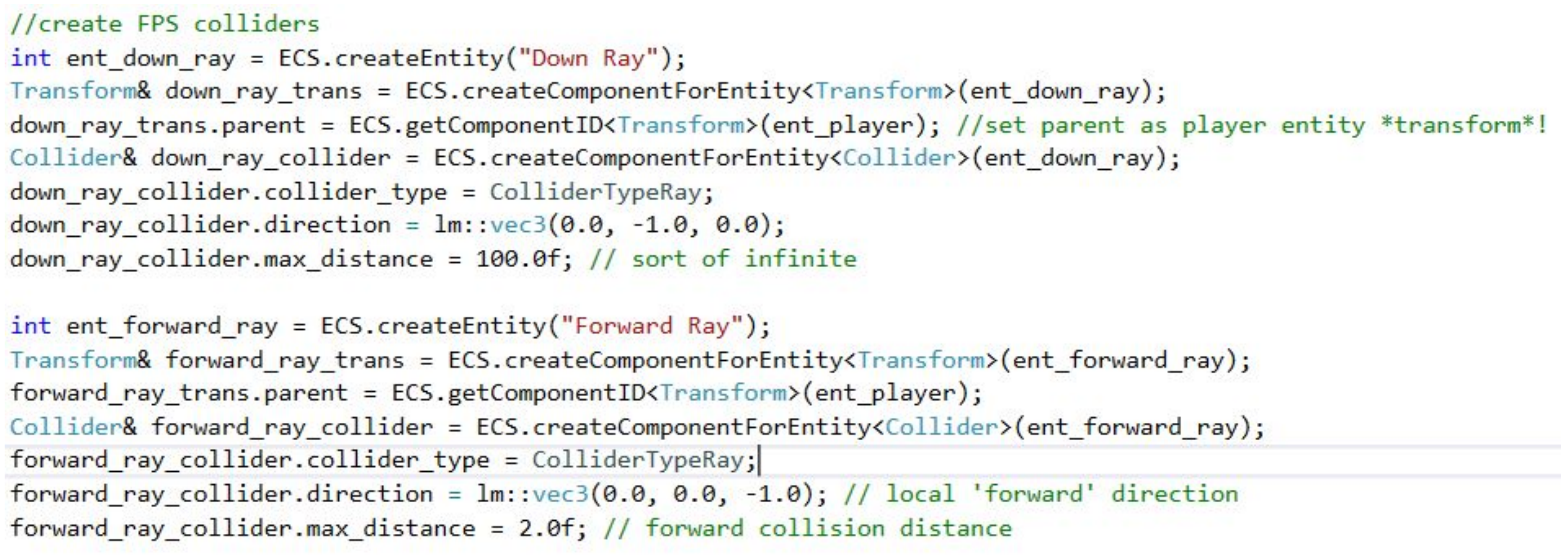
Today: Segment(ray) box collision function

- Calculate corners of box in *local space* (using center offset and halfwidth)
- Get *world position* of both ray/segment start point, and 8 points of box
- rotate ray *direction* using inverse transpose of model matrix (like normal matrix for rendering)
- create segment using ray distance along direction
- use intersectSegmentQuad for 6 faces of box with segment

FPS Control

What colliders do we need for an FPS control?

Example FPS colliders



```
//create FPS colliders
int ent_down_ray = ECS.createEntity("Down Ray");
Transform& down_ray_trans = ECS.createComponentForEntity<Transform>(ent_down_ray);
down_ray_trans.parent = ECS.getComponentID<Transform>(ent_player); //set parent as player entity *transform*!
Collider& down_ray_collider = ECS.createComponentForEntity<Collider>(ent_down_ray);
down_ray_collider.collider_type = ColliderTypeRay;
down_ray_collider.direction = lm::vec3(0.0, -1.0, 0.0);
down_ray_collider.max_distance = 100.0f; // sort of infinite

int ent_forward_ray = ECS.createEntity("Forward Ray");
Transform& forward_ray_trans = ECS.createComponentForEntity<Transform>(ent_forward_ray);
forward_ray_trans.parent = ECS.getComponentID<Transform>(ent_player);
Collider& forward_ray_collider = ECS.createComponentForEntity<Collider>(ent_forward_ray);
forward_ray_collider.collider_type = ColliderTypeRay;
forward_ray_collider.direction = lm::vec3(0.0, 0.0, -1.0); // local 'forward' direction
forward_ray_collider.max_distance = 2.0f; // forward collision distance
```

FPS control

Rotate camera as Free movement

Detect collision on ‘down’ collider

- translate player entity (and camera) to be always ‘FPS_height’ units above collision point

‘Forward’ and ‘Strafe’ dirs are clamped to 0 in y-axisç

Only move in direction if collider in that direction isn’t colliding

FPS Jump

Need to add gravity (if down collider distance is greater than FPS_Height) apply gravity

Need to create FPS_jump_force

- when player is jumping, add translate FPS_jump_force in y axis every frame
- reduce FPS_jump_force every frame until 0