CS354R

DR SARAH ABRAHAM

INTERACTING WITH SIMULATIONS

PHYSICS IN GAMES

- Challenges:
  - Must be able to simulate efficiently with multiple objects
  - Must maintain stability regardless of player input

STATIC AND DYNAMIC OBJECTS

- Static objects do not move during the physics simulation
  - Fewer calculations with simplifying assumptions
  - No mass – only used for collision detection
  - Make as many static objects as possible
- Dynamic objects have physics, collision detection, and collision response applied to them
  - Simulated objects in the scene
  - Controlled by rigid body dynamics
- What about objects the player or game logic controls?
  - Characters, moving platforms, etc?

KINEMATIC OBJECTS

- Kinematic objects can move in the simulation but are not affected by physics
  - Allows for player controls that are not physically based
  - Allows for non-physically based objects to kick off physically based movements in other objects
  - No mass but can simulate dynamic motions

KINEMATIC OBJECTS IN GODOT

- CharacterBody allows for user control or animation control over an object
- When moved in an AnimationPlayer, can estimate linear and angular velocity
  - Rigid bodies can interact with this motion "correctly"
- Can perform collision tests to reconstruct simulated physics interactions:
  - `move_and_collide` moves body along a given vector and returns a `KinematicCollision` if it collides
  - `move_and_slide` moves body along a given vector but will slide based on outside rigid body motions (returns the linear velocity after it collides)

OTHER USEFUL FUNCTIONS

- `move_and_slide_with_snap` moves the body while keeping it attached to surface's slope
- `test_move` checks for collisions without moving the body
- Read the documentation here to learn more:
  - https://docs.godotengine.org/en/4.1/classes/class_characterbody3d.html
  - https://docs.godotengine.org/en/4.1/tutorials/physics/using_character_body_2d.html

GODOT PHYSICS

- Godot Physics is used for both 2D and 3D physics as of Godot 4
- Previously used Bullet (an open source 3D physics library)
  - Used extensively in game and robotics applications
  - Base API is C++
  - PyBullet is Python library for robotics and AI applications
- Godot Physics designed to match Bullet specification

BULLET SIMULATOR CLASS EXAMPLE

```
class Simulator {
protected:
   btDefaultCollisionConfiguration* 
collisionConfiguration;
   btCollisionDispatcher* dispatcher;
   btBroadphaseInterface* overlappingPairCache;
   btSequentialImpulseConstraintSolver* solver;
   btDiscreteDynamicsWorld* dynamicsWorld;
   btAlignedObjectArray<btCollisionShape*> 
collisionShapes;
      std::deque<GameObject*> objList;
```

```
public:
   Simulator();
   ~Simulator();

   void addObject(GameObject* o);
   bool removeObject(GameObject* o);
   void stepSimulation(const float elapsedTime,
      int maxSubSteps = 1, const float fixedTimestep =
1.0f/60.0f);
};
```

```
Simulator::Simulator() {
  collisionConfiguration = new btDefaultCollisionConfiguration();
  dispatcher = new btCollisionDispatcher(collisionConfiguration);
  overlappingPairCache = new btDbvtBroadphase();
  solver = new btSequentialImpulseConstraintSolver();
  dynamicsWorld = new btDiscreteDynamicsWorld(dispatcher,
                    overlappingPairCache,
                       solver,
                    collisionConfiguration);
  dynamicsWorld->setGravity(btVector3(0.0, -0.098, 0.0));
  //Add collision shapes to reuse among rigid bodies
}
```

```
void Simulator::addObject (GameObject* o) {
   objList.push_back(o);
   dynamicsWorld->addRigidBody(o->getBody());
}
```

PHYSICS COLLISION SHAPES

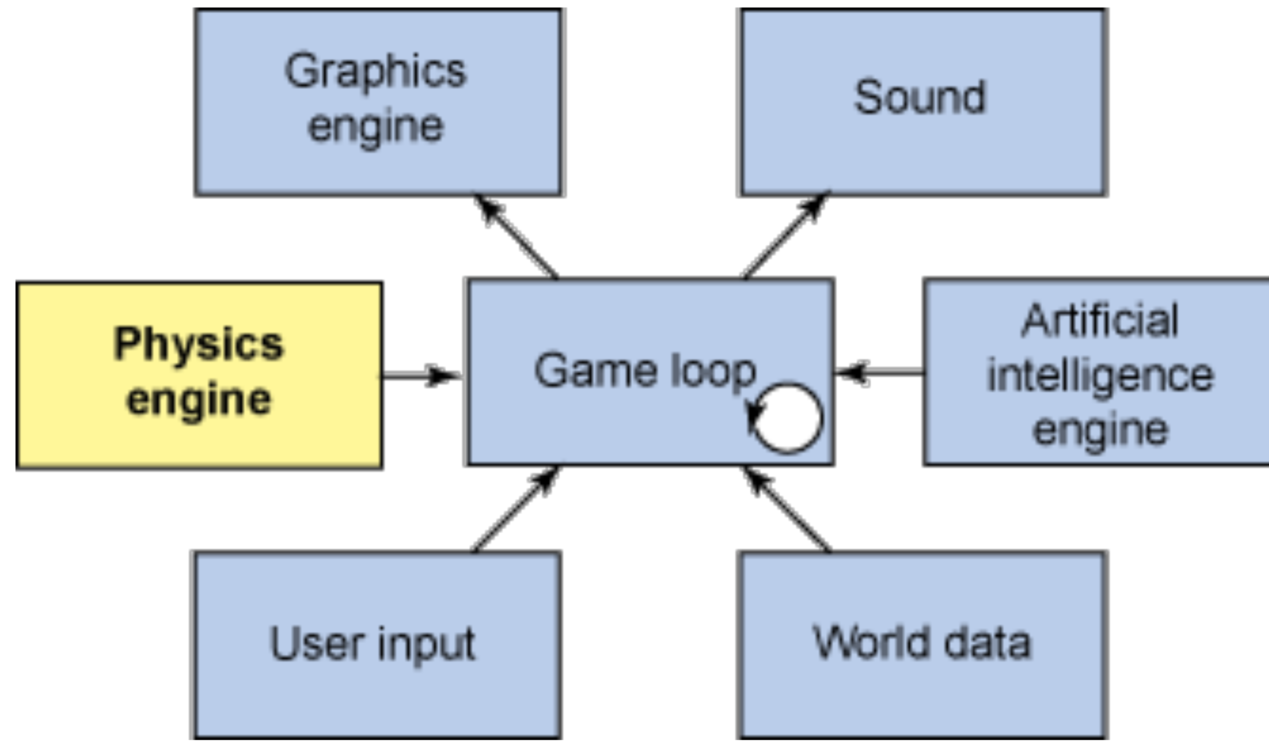
- Collision shapes are bounding boxes that define physics objects
- Reuse of shapes among collision bodies saves memory
- Primitive shapes include spheres, boxes, cylinders, capsules etc
- Mesh shapes include convex hulls, convex triangle meshes, heightfield terrain etc
- Primitive shapes provide an efficiency/accuracy tradeoff
- Meshes better for arbitrary geometry that requires higher degree of accuracy

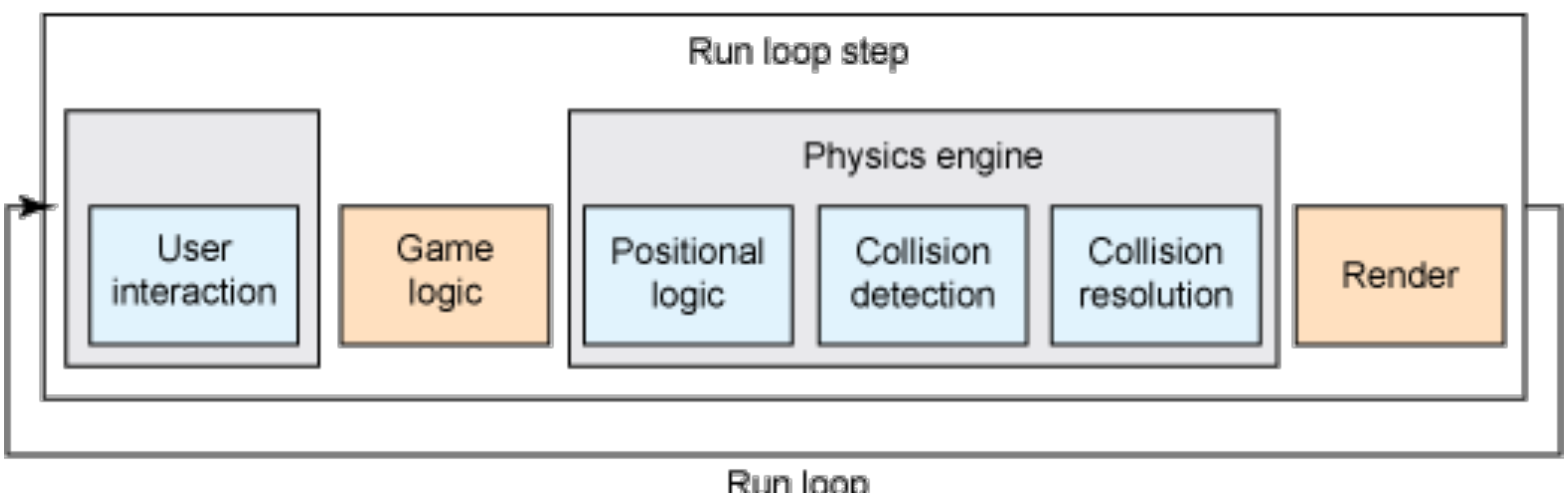
SIMULATION WORLD AND OBJECTS

- `btDefaultCollisionConfiguration` defines default setup for memory and collision
- `btCollisionDispatcher` defines default thread dispatcher (not parallel)
- `btDbvtBroadphase` checks for number of collisions to resolve (can also use `btAxis3Sweep`)
- `btSequentialImpulseConstraintSolver` defines default constraint solver (not parallel)
- `btDiscreteDynamicsWorld` creates a world based on simulation settings
- `btRigidBody` (`btCollisionObject`) manages collision detection for an object:
  - Shape, AABB, and transform

INTEGRATING A PHYSICS ENGINE

- Physics engine called during game loop
- Runs simulation based on a time step
- Provides updated position and orientation





CONNECTING PHYSICS TO THE GAME LOOP

- How do we know when something changes in one or the other?
- What should we do when this change happens?

MULTIPLE REPRESENTATIONS

- Game Object must have visual and physics-based representation
- Visual components:
  - Transform in scene hierarchy
  - Artist mesh
- Physics-based components:
  - Transform in physics simulation
  - Rigid body
  - Collision shape
  - Mass
  - Inertia

BULLET MOTIONSTATES

- MotionStates provide an interface between the simulation and game loop
  - Used for moving objects
  - Movements in simulation are passed to rendered objects in the scene
- Static objects do not need motion states
- Kinematic objects communicate movement from game loop to Bullet
  - Motion states used in reverse

USING MOTIONSTATES

- `btDefaultMotionState` provides basic motion state functionality
- Use motion state to initialize object position when it enters the simulation (`getWorldTransform`)
- Call motion state during simulation to move body in rendering world (`setWorldTransform`)

COLLISION CALLBACKS

- ContactTest tests object against all other objects and calls ContactResultCallback
    - `simulator->getWorld()->contactTest(body, contactCallback);`
- Implement `ContactResultCallback` to determine:
    - Which collisions should result in a hit
    - What information about the hit to store/use

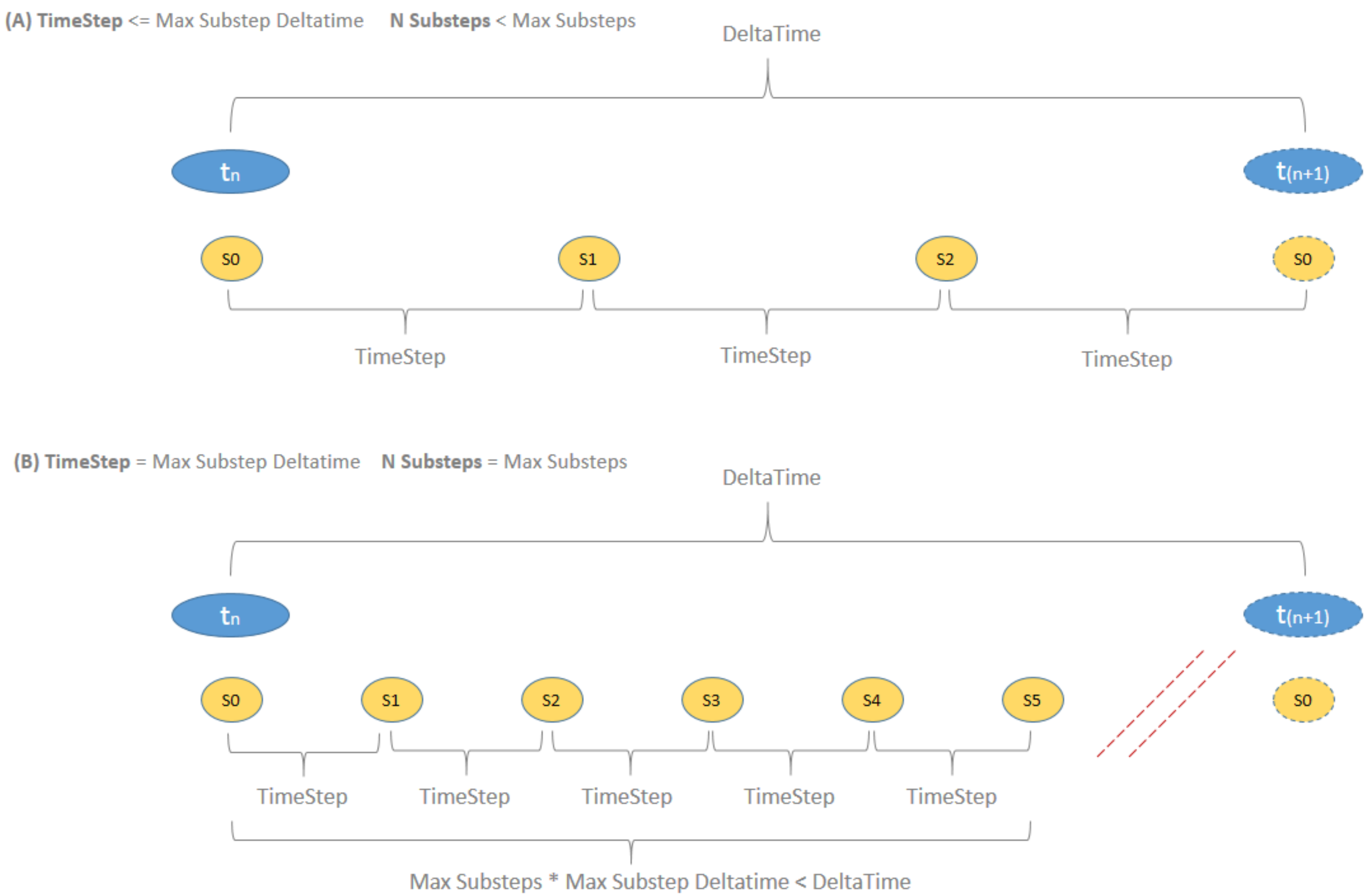
OBJECT SIZE AND MASS

- Physics simulations will struggle with very small and very large objects
- Bullet size recommendations:
  - Minimum object size is 20 cm (0.2 units) in Earth gravity
  - Maximum object size is 5m (5 units) in Earth gravity
- Physic simulations will struggle with large mass ratios
  - Simulations can become unstable when heavy objects rest on light objects
- Bullet mass ratio recommendations:
  - Keep object mass around 1

SIMULATION TIME STEPS

- Simulation time steps should be as decoupled as possible from rendering framerate
  - Simulation should not slow down or speed up based on framerate
- Can decouple by setting simulation to **fixed time step**
  - Simulation will remain more stable
  - Simulation time will not correspond to frame time
- **Substepping** allows physics simulation to adjust to frame rate
  - If frame time is smaller than physics time step, interpolate but do not perform physics simulation
  - If frame time is greater than physics time step, perform multiple physics time steps to match

SUBSTEPPING



http://www.aclockworkberry.com/unreal-engine-substepping/

GHOST OBJECTS AND RAY CASTS

- Ways of detecting interactions without creating collision responses
  - Ghost objects have volume and track objects they are in contact with
  - Ray casts have direction and track objects they have intersected
- Designed to sense scene objects so that program can appropriately respond

EXAMPLE

- https://www.youtube.com/watch?v=qGEpiSRX5ac

ADDITIONAL RESOURCES

- [https://github.com/bulletphysics/bullet3/blob/master/docs/Bullet_User_Manual.pdf]
- [https://docs.godotengine.org/en/3.1/classes/class_kinematicbody.html]
- [https://docs.godotengine.org/en/3.1/classes/class_raycast.html]
- [http://www.aclockworkberry.com/unreal-engine-substepping/]