Game Architecture
Games are Real-Time Software

- must meet time constraints
- ex) air traffic control
Real-time Loops

- "continuously:"
  - update world/simulation
  - interact with world/sim
  - display state of world/sim
Coupled Approach

while True:
    update()
    render()

- want to render as fast as possible
- but, update may not be needed every frame
- motions need to be the same on different platforms
Dual-Threaded Approach

- separate threads, running at different rates
- ex) render 60Hz, update 15 Hz
- use interpolation in renderer for position changes per frame (render 4 frames for each update)
- contention for shared data - locking can cause delays
- exploits multi-core CPUs
Single-Threaded, Decoupled Approach

- need to use interpolation for position update in `render()`

```python
while True:
    if isTimeForUpdate():
        update()
    render()
```
Game Logic

• player update
  • get input - devices
  • restrict motions - collisions, etc
  • update state
• world update
  • passive elements - walls, floors, etc
  • active elements - NPCs, etc
  1. Filter out relevant elements
  2. Update active elements
  3. Generate behaviors
  4. Update world
World Rendering

• for static and simple objects

1. Select visible subset (clip, cull, occlusion test)
2. Select Level-Of-Detail (LOD)
3. Pack geometry - for specific hardware
4. Render geometry
5. Select audible sound sources
6. Pack audio and send to hardware
NPC Rendering

- NPC == non-player character

1. Select visible subset and LOD
2. Animate
3. Pack
4. Render
Player Rendering

- Animate
- Pack
- Render