CMSC 498M: Chapter 2
Game Architecture

Sources:
- Lecture notes by M. Overmars from Univ. of Utrecht.

Overview:
- Basic elements of game design
- Game architecture - basic structural elements and relations
- Sample game program structure and timing
- Scene graphs

Basic Elements of Game Design

The Dimensions of a Typical Game:

Game Play: deals with materials, moves, rules, balance, and winning.
Simulation: deals with the internal mechanics of the virtual world.
Story: deals with the setting, storyline, immersion, dramatic effects and motivation.

Game Space:
- Game genre’s differ in the relative importance of these elements.
- Where do your favorite games fit within this triangle?
Basic Elements of Game Design

Some Thoughts:
- Traditional games tend to fall close to the corners.
- There is a tendency among developers to move towards the center.
- Is this desirable?
  * A realistic world simulation may interfere with game play and balance.
  * Game play emphasizes player control while stories emphasize designer control.

The Ingredients of a Game

Game Ingredients:

- **Core Mechanics**: The rules.
- **Storytelling and Narrative**: Storyline, dramatic effect and motivation, involvement.
- **Interactivity**: How the player perceives the world and how he acts within it.

**Example**: First-Person Shooter:
- **Core mechanics**: What is the effect of shooting?
- **Storytelling and narrative**: Whom are you shooting at and why?
- **Interactivity**: How do you aim? How does the target appear? How do you perceive the effect?
**Game Software Architecture**

![Diagram of Game Software Architecture]

**Architecture Elements: Low-Level**

**Hardware:**

**Physical:**
- Graphics card
- Sound cards
- Input devices (keyboard, mouse, game controllers, steering wheels)

**Drivers:**
- Low level interface

**Hardware abstraction layer:**
- DirectX/DirectSound
- OpenGL/OpenAL
- others...

![Diagram of Architecture Elements: Low-Level]
Architecture Elements: Graphics Engine

Graphics Engine:
  Higher-level interface:
    Tuned to a particular graphics and game type:
    • sprite-based
    • isometric view (fixed aerial viewpoint)
    • full 3D
  Rendering models:
    • Sprites
    • Solids
    • Characters (articulated)
    • ...
  More complicated display aspects:
    • Mini-map
    • Multiple views
    • Overlays
    • Special effects

Architecture Elements: Sound Engine

Sound Engine:
  Function of sound:
    • Effects to enhance reality
    • Ambience/mood (music)
    • Clues about what to do (e.g., follow the voice of a friendly agent)
  Sound formats:
    • Wave (high quality, lots of memory, fast)
    • MP3/OGG (high quality, compressed, slower)
    • MIDI (lower quality, very low storage, adaptable)
  Simultaneous sounds:
    • Mixers
    • Buffer management
    • Streaming sound
  How to create:
    • Get it from the Web
    • Record samples and modify
    • Build your own music studio!
Architecture Elements

User interface:
- Monitors input status (callbacks) and relays inputs to the game data.
- Displays menus and online help.

Configuration system:
- Adapt to hardware specs (requires choices)
- Adapt to player preferences
- Player dependent (stored by player)

Online help:
- Players never read documentation.
- Screen overlays
- Contents:
  - Static
  - Context dependent
  - Player dependent

Architecture Elements: Game Data

Game Data: "A game is a database with a fancy interface"
- Resources
  - graphics models (sprites, characters)
  - sounds, music
  - images, backgrounds, video
  - text
- Level description
- Game status
- Event queue
- User profile

The game components communicate with the support components through the game data.
Architecture Elements: Events

Event handler:

**Event-based:** (typically)
- Events invoke **callbacks**.
- Cause logic and physics engines to **change the game status**.
- New game status is reflected **visually** (redrawing the scene).

**Event Types:**
- **User inputs.**
- **Collisions.**
- **Timers** (controlled by the logic).
- Created by **game entities/objects**.

**Game entities/objects:**
- Entities have a **state**.
- Can **react to/create events**.
- Responses can be **complex** (AI).

---

Architecture Elements: Logic/Physics

**Logic engine:**
- Handles all the **game play**.
- Enforces the **rules**.
- Contains the game **AI**.

**Physics engine:**
- Handles **physical simulation** of the world:
  - collisions
  - rigid-body simulation
  - waves in the sea
- Becoming increasing prevalent, even in **simple games**.
- Separation with logic engine is not always clear.
Simple Game Program structure

Initialization: (on entering a new level)
- Loading resources (art, sound, etc.)
- Intro screen
- Configuration and settings

Game loop:
- Process input events
- Receive network messages
- Update time step
- Run AI (planning) and physics (collision detection and response)
- Update game entities/state
- Send network messages
- Display

Finalization: (on exiting a level)
- Saving results/state
- Trailer

Timing

Based on the display's frame rate:
- Latency hiding (e.g., by pre-computing expensive operations)

Decoupling drawing and processing:
  Semi-decoupled:
  - Event processing cycles at a fixed frequency
    (may be slower than frame rate).
  - Drawing occurs when not busy doing a cycle.
  Fully decoupled:
  - System does small steps, processing few events (and scheduling others
    for future processing).
  - Drawing occurs when frame is needed and not inside a step.
  - Game state must always be valid. (More difficult to implement.)

Maximizing GPU/CPU parallelism:
- Start graphics early so GPU is working in parallel with CPU, not waiting for it.
Scene Management/Scene Graphs

**Scene Management:**
Storing the various entities of your world for efficient access and rendering.

**Motivation:**
- **Culling:** Eliminating objects that are not visible, based on the current camera position.
- **Naïve solution:** Visit all the objects in your database and check them one-by-one. (Very inefficient.)
- **Better solution:** Store the objects in a hierarchical structure. Each node has a bounding box for all its descendents. If the bounding box is not visible, we can eliminate the entire subtree.

**Scene Graph:**
A hierarchical data structure for organizing geometric entities.

---

Scene Graph

**Organization:**
- **Leaves:** Store individual (primitive) objects.
- **Internal nodes:** Store aggregate objects, formed by grouping either primitive objects or other aggregates.

**Example:** A house with 2 rooms. One room has a table and chair. There is a place set at the table.
Scene Graph: Local Coordinates

**Local Coordinates/Transformations:**

- **Local Coordinate System**: Each object is represented relative to its own local coordinate frame.
- **Change of Coordinates**: Each link stores a transformation (4x4 matrix) that converts from the child's coordinate frame to its parent's frame.

**Example**: \((R_1 \cdot T \cdot P) \cdot v\) converts a vector \(v\) on the plate to its coordinates relative to the house.

![Scene Graph Diagram](image)

---

Scene Graph: Compound Motion

**Motion Transformations:**

- **Motion**: These transformations can also be used to represent compound motion.

**Example**:

\((T \cdot L \cdot H \cdot B) \cdot v\) converts vector \(v\) on the thumb to its coordinates with respect to the room.

If we change the transformation \(T\), not only does the torso move, but so do the arms, hand, and fingers (relative to the body).

**Matrix Stack**:

Graphics APIs, such as OpenGL, provide a stack of matrices for drawing scenes based on scene graphs.
Scene Graph: Instancing and Cloning

Instancing:
We can create multiple instances of a single object by sharing nodes, but using different transformations.

Example:
We generate only one model of the arm, but we use two different transformations (one of which involves reflection) to create two instances of this model.
Since only the L and R transformations differ, the two generic arms behave identically otherwise.

Cloning:
Makes a copy of a subtree. This allows the copy to behave independently.

---

Scene Graph: Model Switching

Model Switching:
We select among multiple model instantiations of a single entity by storing multiple child nodes and drawing only one, depending on the entity’s current state.

Example:
A car door in a racing game becomes progressively more damaged as collisions occur.
Different door objects are selected based on the state of the door.
Summary

Topics Covered:
- Basic elements of game design
- Game Architecture - basic structural elements and relations
- Sample game program structure and timing
- Scene graphs