IMGD 3000

Game Engine Introduction

Introduction

• What are the names of some game engines?
• What, exactly, is a game engine?
• How does it work?

What is a Computer Game?

User Perspective

• A goal (or set of goals)
  – Save the Princess (solve puzzles to get sword first)
  – Score points (get power ups)
  – Finish first (unlock next level)
• A set of rules governing play
  – Turn taking, like RPGs
  – Reaction to events, like Tetris’ falling blocks
  – Legal actions
• Visual and audible content (graphics and sound)
• Control and input techniques
  – Button mappings, mouse clicks
  – How you provide input to game world

Computer Perspective

• Set of resources managed to support entertainment (usually) application
• Graphical rendering
• User interface and input
• Event processing
  – Timers, collisions, etc.
• File I/O
• Optional: Networking, AI, Physics, Scripts

Game Code versus Game Engine Code

• Line between game and game engine often blurry
  – E.g. One game, an engine may know how to “draw and orc”
  – E.g. Another game, engine provides rendering and shading, but “orc-ness” defined entirely in user code
• No clear separation since “built-in” parts of game engine are often part of the game
  – E.g. sprite or animation, collision detection …

Game Engine Specificity

• Reusable? Often
  – But many still make one game only
• Efficient? Often
  – Can tune commonly used code
• General purpose? Somewhat
  – Can make more than one game (e.g. mod)
• Often designed with specific genre in mind
  – Some genres with likely very different engine support
    – Arcade (e.g. Tetris)
    – Side-scroller (e.g. Mario)
    – 3d isometric (e.g. Diablo)
    – 1st person (e.g. CoD)
    – MMO/RPG (e.g. Warcraft)
    – Turn-based (e.g. Civ)
    – Story (e.g. Heavy Rain)
• How do you think each may differ?
**Game Engine Components**

- **Substrate**
  - Hardware (PC, Xbox, iPad, ...) and Operating System (Windows 7, iOS, ...)
  - Graphics API (OpenGL, DirectX, Curses)
  - Third-party libraries (STL, Networking)
  - Math libraries (trig, linear algebra)
  - Game engine needs these, and is bound to these but not engine code

- **Core Systems**
  - Memory allocation
  - Engine configuration
  - Parsers (for config files)
  - Debugging and performance (unit testing, profiling, error logging)
  - Startup/shutdown (initialization and final state)

**Example Core System - Structures**

- **Basic data structures**
  - Arrays – fast indexing, fast insertion/deletion at end
  - Lists – slow indexing, fast insertion/deletion in middle
  - Maps (hash tables) – fast searching and insertion
  - (May be provided by standard libraries (e.g. C++ STL))

- **System-specific concepts**
  - System time – converting from OS to game time
  - File system – open, close, read/write, directories and naming

**Example Core System – Object System**

- **Most game engines use objects as foundational representation**
  - Convenient abstraction for programmers and designers
  - Fits with OO design and programming

- **Objects have base values**
  - Location, attributes (e.g. size, mass), velocity
  - Exact attributes often depend upon genre type!

- **Objects know how to react to events**
  - e.g. time elapsed then explode, hit wall then bounce

- **Startup:** Populate world with objects → go!
Example Core System – Object System

- Key functionality → Run-time type information
  - Polymorphic at run-time
  - E.g. Engine wants to make weapon "shoot"
    → Object specific code knows how to do this
  - Note, C++ and Java do this automatically
  - But if C (or some other language), must do yourself

---

Our Focus

- Mainly on the tech stuff
  - How to build core engine components
  - How to use engine to make custom world
  - How to support user interaction
  - How to set rules of play and control
- Less on content
  - Art
  - Sound
  - Game design

---

Example Core System – Object System

- Controllers – most objects can be altered, so associate generic (and then specific) controller

```java
class Weapon {
    virtual void shoot();
}
class Bullet: public Weapon {
    virtual void shoot();
}
Weapon* p = new Bullet(); // invoke Bullet::shoot()!
```

---

Game Engine Architecture

- Have overview of what game engine does, but how to go about designing your own engine?
- Components
  - What are the major components?
    - How to separate game-independent components from game-dependent components?
- Organization
  - How are components defined and organized?
- Structure
  - Assume an object-oriented approach → What class structure should be used for various elements?
  - This class!