IMGD 3000

Game Engine Introduction

Introduction

- Do you know 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 new features)
- 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 techniques
  - Button mappings, mouse clicks

What is a Computer Game? *Computer Perspective*

- Set of resources managed to support entertainment (usually) application
- Graphical rendering
- User interface
- Script handling
- Event processing
  - Timers, collisions, etc.
- File I/O
- Optional: Networking, AI, Physics

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 "orcness" 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

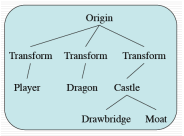
- Reusable? Often
  - But many still make one game only
- Efficient? Often
  - Can tune commonly used code
- General purpose? Somewhat
  - Can make more than one game (e.g. mod)
- *Often designed with specific genre in mind*
- Some genres with likely very different engine support
  - Arcade (e.g. Tetris)
  - Side-scroller (e.g. Mario)
  - 3d isometric (e.g. Diablo)
  - 1st person (e.g. CoD)
  - MMORPG (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)
- Core Systems
  - Memory allocation
  - Engine configuration
  - Parsers (for config files)
  - Debugging and performance (unit testing, profiling, error logging)
  - Startup/Shutdown (initialization and final state)

Game Engine Components

- Representation of the world
  - Game objects
  - Possibly oriented, relative
- Timing is very important
  - Events are time-based
  - Multi-player needs consistency
- Low-level utilities
  - Updating objects, handling resources in/out, logging, memory management, encryption…



Game Engine Components

- Rendering system (Dragonfly – yes)
  - How to display scene
  - Lighting, occlusion, textures, camera, viewport …
  - Special effects (particles)
- Sound system
  - Music and dialog, formats and timing and resources
- Physics
  - How objects may move and/or interact
  - Object physical states (location, velocity, orientation)
  - Bounding volumes and collision detection
- Artificial intelligence
  - "Smart" objects, as opponents or NPC
  - Low-level utilities such as pathfinding

Game Engine Components

- Input management
  - Map device specific commands (e.g. keystroke or mouse click) to generic game-specific command (e.g. left)
- Resource Manager
  - 3d models (skeleton, animations), Textures
  - Loading, decompression
- Online Multiplayer
  - Authentication and registration
  - Game state replication
  - Latency compensation (dealing with lag)
- Gameplay Foundations
  - Static world elements
  - Dynamic world elements
  - Events/messaging

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 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

- Key functionality → Run-time type information
  - Polymorphic at run-time
  - E.g. Engine just wants to make weapon "shoot"
    → object specific code knows how to do this

```
class Weapon {
  virtual void shoot();
};
class AK47: public Weapon {
  virtual void shoot();
};
Weapon* p = new AK47();
p->shoot(); // invokes AK47::shoot()
```

  - Note, C++ and Java do this automatically
  - But if C (or some other language), must do yourself

Example Core System – Object System

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

```
class Controller {          // generic controller
  Object* obj;              // associated game object
  double startTime;         // controller start time
  void setObject(Object* o);
  void initialize(double start);
  double getControllerTime(double gameTime);
};
                            // concrete controller
class ConcreteController: public Controller {
  Transformation getTransformation(double gameTime);
};
class Object {              // game object
  ControllerList* ctls;
  void addController(Controller* c);
  void remController(Controller* c);
  void updateControllers(double gameTime);
  …
};
```

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

Game Engine Architecture

- Have overview of what a 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!