

Videogame engine architecture

Videogames Technology

Objectives

- Introduce the main videogame subsystems
- Deep understanding of the main loop
- Describe different main loop implementation methods

Bibliography

1. Desarrollo de Videojuegos, Arquitectura del Motor de Videojuegos. Capitulo 1, sección 2. UCLM.

Table of Contents

- Videogame engine architecture
 - Overview
 - Conceptual overview of a videogame engine
 - Videogame engine layers
- Videogame models
 - Render loop
 - Game loop
- Game architectures
 - Game architectures
 - Callbacks
 - Events
 - State machine

Videogame engine architecture

Overview

Videogame engines aims to be independent of the game genre

- Increased design complexity

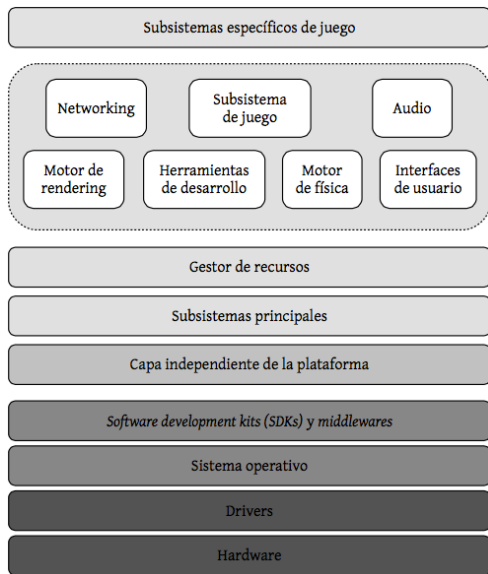
Videogame engines are complex systems \Rightarrow Layered structure

- Layered architectures are common in complex systems
- Handle complexity
- Upper layers use services from the bottom layers
- Lower layers never access upper layers
- Adding layers is simple (well, more or less)
- One layer can be modified independently of the others

Other examples of layered structures: TCP/IP, OSI, operating systems

Videogame engine architecture

Conceptual overview of a videogame engine



Videogame engine architecture

Videogame engine layers (I)

1. **Hardware:** General purpose (PCs) or specific (consoles)
 - The boundaries tend to vanish ...
2. **Drivers:** Interface between processes and hardware
3. **Operating system:** Manages access to hardware
4. **SDK and middleware:** General features not integrated in the OS
 - OpenGL, Direct3D, authentication, etc
5. **Platform independent layer:** Isolates upper layer from the platform
 - Encourages multiplatform games
6. **Main subsystems:** Some basic libraries not dependent on the videogame
 - Mathematical library
 - Datastructures and algorithms
 - Memory management
 - Debugging and logging

Videogame engine architecture

Videogame engine layers (II)

7. **Resource manager:** Unified interface to access videogame resources
 - Many game engines do not implement this
8. **Main subsystems:** General purpose (PCs) or specific (consoles)
 - The boundaries tend to vanish ...
9. **Resource manager:** Manages access to hardware
10. **SDK and middleware:** General purpose libraries
 - OpenGL, Direct3D, etc
11. **Platform independent layer:** Isolates upper layer from the platform
 - Encourages multiplatform games

Videogame engine architecture

Videogame engine layers (III)

12. Game specific subsystems:

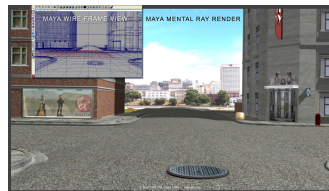
- Physics engine
 - Solid state physics (Video 1) (Video 2) (Video fails)
Havoc, PhysX, Bullet
 - Particle physics (Video 1) (Video 2)
- Collisions engine (usually integrated in physics)
 - I Collision detection
 - II Collision determination
 - III Collision handling
- Rendering engine
- User interface (UI)
- Networking
- Audio



Videogame models

Render loop (I)

- The render loop handles visualization and rendering
- Objectives in 2D games
 - Minimize pixels to draw: Draw only those pixels that have changed
 - Maximize fps
- Objectives in 3D games
 - Camera uses to change everytime: The same technique cannot be used
 - Minimize the number of primitives to draw in each iteration of the render loop



Render example

Videogame models

Render loop (II)

Render loop

```
while (true) {  
    // Update camera, usually according to a  
    // predefined path  
    updateCamera();  
  
    // Update position, orientation and rest  
    // of the state of the entities in the game  
    updateSceneEntitites();  
  
    // Render a frame in a buffer  
    renderScene();  
  
    // Interchange the buffer to visualize the image  
    swapBuffers();  
}
```

Info: http://wiki.wxwidgets.org/Making_a_render_loop

Videogame models

Game loop (I)

The main element in a videogame is the **game loop**

- It is the main control structure in the game
- It controls its execution
- It handles the transitions among states
- The game loop independizes the game execution from the hardware

Classical programs only reacts with user actions

- Videogames are always performing an action
- Game loop implements this easily
- The game engine contains the game loop

Videogame models

Game loop (II)

- There are many subsystems in a videogame
 - Rendering engine (render is to generate an image)
 - Collision detection
 - Collision handling
 - AI subsystem
 - Game subsystem
- Most of these subsystems require periodic updates
- The most critical one is the animation system
 - Frequency: 30 or 60 Hz
 - Synchronized with the rendering subsystem
 - Objective: Provide a good fps rate to generate a realistic experience
- Not all the components are so strict, for instance, AI

Videogame models

Game loop (III)

- There are several ways to implement the game loop
- The easiest one is to have several loops within the game loop
 - Render loop
 - AI loop
 - Multimedia loop
 - Iteration loop

Basic game loop

```
boolean running = true;

while (running) {
    updateGame();
    displayGame();
}
```

Game loop

```
while(running) {
    checkUserInput();
    runAI();
    moveEnemies();
    resolveCollisions();
    drawGraphics(); //Render loop
    playSound();
}
```

Videogame models

Game loop (IV)

```
int main (int argc, char* argv[]) {
    init_game(); // Game initialization

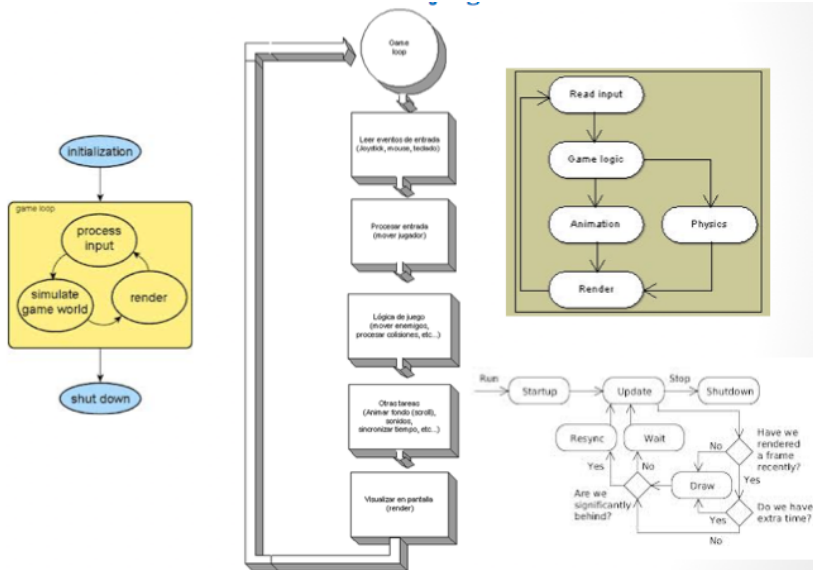
    while (1) { // Game loop
        capture_events(); // Capture events
        if (exitKeyPressed()) break; // Exit
        move_paddles(); // Update paddles
        move_ball(); // update ball
        collision_detection();
        if (ballReachedBorder(LEFT_PLAYER)) {
            score(RIGHT_PLAYER);
            reset_ball();
        }
        if (ballReachedBorder(RIGHT_PLAYER)) {
            score(LEFT_PLAYER);
            reset_ball();
        }
        render();
    }
}
```

Pong game loop example



Videogame models

Game loop (V)



Videogame models

Game loop (VI)

The game loop depends on the platform

- DOS games and some consoles are designed to exploit computational resources
- PC games depend on limitations imposed by the OS
- Games use to be multithreaded to exploit multicore machines

Game architectures

Game architectures

Game loop can be implemented in different ways

- Architectures based on callbacks
- Architectures based on events
- Architectures based on state machines

Most of them implement one or more control loops

Game architectures

Callbacks (I)

- **Callbacks:** Code that is executed to handle an event
 - Function or object
 - Callbacks are used to ``fill" source code
- Related term: **framework**
 - Application partially completed that the developer has to complete

Videogame models

Callbacks (II)

```
void update (unsigned char key, int x, int y) {
    Rearthyear += 0.2;
    Rearthday += 0.2;
    glutPostRedisplay();
}
// More code
int main (int argc, char** argv) {
    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);
    glutInitWindowSize(640, 480);
    glutCreateWindow("Session #04 - Solar System");
    // Define callbacks
    glutDisplayFunc(display);
    glutReshapeFunc(resize);
    glutKeyboardFunc(update);

    glutMainLoop();
    return 0;
}
```

Game architectures

Events

- An event represents a change in the game state
- Two types
 - **External:** Generated by the interactions
Example, The player press a key or moves the joystick
 - **Internal:** Generated by the game logic
Example, NPC respawn
- Most game engines include an event subsystem
 - Closely related to the Observer pattern

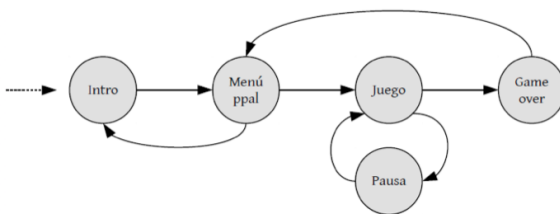
Game architectures

State machine

A game goes through a number of **states**

- Introduction
- Main menu
- Game
- Game over

State machine: A set of states and transitions



Warning: State machines play a mayor role in game AI