Teoria i praktyka programowania gier komputerowych

Wstęp

Piotr Fulmański

Wydział Matematyki i Informatyki,
Uniwersytet Łódzki, Polska

13 października 2011

Spis treści

Structure of a typical game team

Game studios are usually composed of five basic disciplines:

- engineers
- artists
- game designers
- producers
- other stuff
- testers

Structure of a typical game team

Engineers

The engineers design and implement the software that makes the game, and the tools, work. Engineers are often categorized into two basic groups:

- **runtime programmers** (who work on the engine and the game itself)
- **tools programmers** (who work on the off-line tools that allow the rest of the development team to work effectively).

On both sides of the runtime/tools line, engineers have various specialties.

- Some engineers focus their careers on a single engine system, such as rendering, artificial intelligence, audio, or collision and physics.
- Some focus on gameplay programming and scripting,
- while others prefer to work at the systems level and not get too involved in how the game actually plays.

Structure of a typical game team

Artists

- **Concept artists** produce sketches and paintings that provide the team with a vision of what the final game willlook like.
- **3D modelers** produce the three-dimensional geometry for everything in the virtual game world. This discipline is typically divided into two subdisciplines:
  - foreground modelers – create objects, characters, vehicles, weapons, and the other objects that populate the game world;
  - background model – build the world's static background geometry (terrain, buildings, bridges, etc.).
- **Texture artists** create the two-dimensional images known as textures, which are applied to the surfaces of 3D models in order to provide detail and realism.
- **Lighting artists** lay out all of the light sources in the game world, both static and dynamie, and work with color, intensity, and light direction to maximize the artfulness and emotional impact of each scene.

Structure of a typical game team

Artists

- **Animators** imbue the characters and objects in the game with motion.
- **Motion capture** actors are often used to provide a rough set of motion data, which are then cleaned up and tweaked by the animators before being integrated into the game.
- **Sound designers** work closely with the engineers in order to produce and mix the sound effects and music in the game.
- **Voice actors** provide the voices of the characters in many games.
- **Composers** who compose an original score for the game.

Structure of a typical game team

The game designers' job is to design the interactive portion of the player's experience, typically known as *gameplay*. Different kinds of designers work at different levels of detail.

- Some (usually senior) game designers work at the **macro level**, determining the story arc, the overall sequence of chapters or levels, and the high-level goals and objectives of the player.
- Other designers work on **individual levels or geographical areas** within the virtual game world, laying out the static background geometry, determining where and when enemies will emerge, placing supplies like weapons and health packs, designing puzzle elements, and so on.
- Still other designers operate at a **highly technical level, working closely with gameplay engineers and/or writing code** (often in a high-level scripting language). Some game designers are ex-engineers, who decided they wanted to play a more active role in determining how the game will play.

Structure of a typical game team

Producers

Structure of a typical game team

Other stuff

Structure of a typical game team

Testers

What is a game?

Intuition

We probably all have a pretty good intuitive notion of what a game is.

What is a game?

According to Wikipedia
(`http://en.wikipedia.org/wiki/Video_game`)

Definition

A video game is an electronic game that involves interaction with a user interface to generate visual feedback on a video device.

Better definition

Definition

Game is an interactive experience that provides the player with an increasingly challenging sequence of patterns which he or she learns and eventually masters.

According to this, activities of learning and mastering are at the heart of what we call *fun* in case of computer games.

What is a game?

Definition

According to Wikipedia
(`http://en.wikipedia.org/wiki/Video_game`)

Definition

A video game is an electronic game that involves interaction with a user interface to generate visual feedback on a video device.

Better definition

Definition

Game is an interactive experience that provides the player with an increasingly challenging sequence of patterns which he or she learns and eventually masters.

According to this, activities of learning and mastering are at the heart of what we call *fun* in case of computer games.

What is a game?

Our focus – the purposes of this lecture

For the purposes of this lecture, we'll focus on the subset of games that comprise two- and three-dimensional virtual worlds with a small number of players (between one and 16 or thereabouts). Our primary focus will be on game engines capable of producing first-person shooters, third-person action/platform games, racing games, fighting games, and the like.

What is a game?

Video game as a soft real-time simulations

Most two- and three-dimensional video games are examples of what computer scientists would call **soft real-time interactive agent-based computer simulations.** Let's break this phrase down in order to better understand what it means.

What is a game?

Video game as a soft real-time simulations

soft real-time INTERACTIVE agent-based computer simulations

All interactive video games are temporal simulations, meaning that the virtual game world model is **dynamic** – the state of the game world changes over time as the game's events and story unfold. A video game must also respond to unpredictable inputs from its human player(s)–thus **interactive** temporal simulations.

What is a game?

Video game as a soft real-time simulations

soft REAL-TIME interactive agent-based computer simulations

As the final word about interactivity we can say that, most video games present their stories and respond to, player input in **real-time**, making them interactive real-time simulation.

SOFT REAL-TIME interactive agent-based computer simulations

At the core of every real-time system is the concept of a *deadline*. An obvious example in video games is the requirement that the screen be updated at least 24 times per second in order to provide the illusion of motion. A **soft** real-time system is one in which missed deadlines are not catastrophic. Hence all video games are soft real-time systems – if the frame rate dies, the human player generally doesn't! Contrast this with a *hard real-time system*, in which a missed deadline could mean severe injury to or even the death of a human operator.

What is a game?

Video game as a soft real-time simulations

soft real-time interactive AGENT-BASED computer simulations

An agent-based simulation is one in which a number of distinct entities known as ***agents*** interact. This fits the description of most three-dimentsional computer games very well, where the agents are vehicles, characters, fireballs, and so on.

What is a game?

Video game as a soft real-time simulations

soft real-time interactive agent-based COMPUTER SIMULATIONS

In most video games, some subset of the real world (or an imaginary world) is modeled mathematically so that it can be manipulated by a computer. The model is an *approximation* to and a *simplification* of reality (even if it's an imaginary reality), because it is c1early impractical to include every detail down to the level of atoms or quarks. Hence, the mathematical model is a **simulation** of the real or imagined game world. Approximation and simplification are two of the game developer's most powerful tools.

What is a game engine?

A game engine is a system designed for the creation and development of video games. There are many game engines that are designed to work on video game consoles and personal computers. The core functionality typically provided by a game engine includes a

- rendering engine (”renderer”) for 2D or 3D graphics,
- a physics engine or collision detection (and collision response),
- sound,
- scripting,
- animation,
- artificial intelligence,
- networking,
- streaming,
- memory management, threading, localization support, and a scene graph.[1]

[1] http://en.wikipedia.org/wiki/Game_engine

The term ”game engine”

The term ”game engine” arose in the mid-1990s, especially in connection with 3D games such as first-person shooters (FPS). Such was the popularity of Id Software's Doom and Quake games that, rather than work from scratch, other developers licensed the core portions of the software and designed their own graphics, characters, weapons and levels - the ”game content” or ”game assets”. Separation of game-specific rules and data from basic concepts like collision detection and game entity meant that teams could grow and specialize.
Later games, such as Quake III Arena and Epic Games's 1998 Unreal were designed with this approach in mind, with the engine and content developed separately. The practice of licensing such technology has proved to be a useful auxiliary revenue stream for some game developers.
Reusable engines make developing game sequels faster and easier, which is a valuable advantage in the competitive video game industry.

Game engine

The line between a game and its engine is often blurry. Some engines make a reasonably clear distinction, while others make almost no attempt to separate the two.
Arguably a data-driven architecture is what differentiates a game engine from a piece of software that is a game but not an engine. When a game contains hard-coded logic or game rules, or employs special-case code to render specific types of game objects, it becomes difficult or impossible to reuse that software to make a different game. **We should reserve the term *game engine* for software that is extensible and can be used as the foundation for many different games without major modification.**

Engine differences across genres

FPS (First-Person Shooters)

The first-person shooter (FPS) genre is typified by games like Quake, Unreal Tournament, Half-Life, Counter-Strike, and Call of Duty.
First-person games are typically some of the most technologically challenging to build, probably rivaled in complexity only by third-person shooter/action/platformer games and massively multiplayer games. This is because first-person shooters aim to provide their players with the illusion of being immersed in a detailed, hyperrealistic world. It is not surprising that many of the game industry's big technological innovations arose out of the games in this genre.
First-person shooters typically focus on technologies, such as

- efficient rendering of large 3D virtual worlds;
- a responsive camera control/aiming mechanic;
- high-fidelity animations of the player's virtual arms and weapons;
- high-fidelity animations and artificial intelligence for the non-player characters (the player's enemies and allies);
- small-scale online multiplayer capabilities (typically supporting up to 64 simultaneous players), and the ubiquitous "death match" gameplay mode.

Engine differences across genres

Platformers and other third-person games

This genre is typified by games like ....
Platformer is the term applied to third-person character-based action games where jumping from platform to platform is the primary gameplay mechanic.
Same of the technologies specifically focused on by games in this genre include

- moving platforms, ladders, ropes, trellises, and other interesting locomotion modes;
- puzzle-like environmental elements;
- a third-person *follow camera* which stays focused on the player character and whose rotation is typically controlled by the human player ;
- a complex camera collision system for ensuring that the view point never ”clips” through background geometry or dynamic foreground .

Engine differences across genres

Fighting games

This genre is typified by games like ....
Since the 3D world in these games is small and the camera is centered on the action at all times, historically these games have had little or no need for world subdivision or occlusion culling. They would likewise not be expected to employ advanced three-dimensional audio propagation models, for example.
Same of the technologies specifically focused on by games in this genre include

- a rich set of fighting animations;
- accurate hit detection;
- a user input system capable of detecting complex button and joystick combinations;
- crowds, but otherwise relatively static backgrounds;
- high-definition character graphics, including realistic skin shaders with subsurface scattering and sweat effects;
- high-fidelity character animations;
- physics-based cloth and hair simulations for the characters.

Engine differences across genres

Racing games

This genre is typified by games like ....
The racing genre encompasses all games whose primary task is driving a car ar other vehicle on some kind of track. Additionaly some kart racing games, for example, offer modes in which players shoot at one another, collect loot, or engage in a variety of other timed and untimed tasks.
A racing game is often very linear, much like older FPS games. However, travel speed is generally much faster than in a FPS. Therefore more focus is placed on very long corridor-based tracks, or looped tracks, sometimes with various alternate routes and secret short-cuts. Racing games usually focus all their graphic detail on the vehicles, track, and immediate surroundings. However, kart racers also devote significant rendering and animation bandwidth to the characters driving the vehic1es.

Engine differences across genres

RTS (Real-Time Strategy)

This genre is typified by games like Dune II: The Building of a Dynasty, Syndicate, Warcraft, Command & Conquer, Age of Empires, Starcraft. In this genre, the player deploys the battle units in his or her arsenal strategically across a large playing field in an attempt to overwhelm his ar her opponent. The game world is typically displayed at an oblique top-down viewing angle. The RTS player is usually prevented from significantly changing the viewing angle.
Older games in the genre employed a grid-based (cell-based) world construction, and an orthographic projection was used to greatly simplify the renderer. Modern RTS games sometimes use perspective projection and a true 3D world, but they may still employ a grid layout system to ensure that units and background elements, such as buildings, align with one another properly.

Engine differences across genres

MMOG (Massively Multiplayer Online Games)

This genre is typified by games like ...
An MMOG is defined as any game that supports huge numbers of simultaneous players (from thousands to hundreds of thousands), usually all playing in one very large, persistent virtual world (i.e., a world whose internal state persists for very long periods of time, far beyond that of any one player's gameplay session).

Game engine survey

Runtime engine architecture

Hardware layer

Device driver layer

Operating system layer

Third party SDKs and middleware

Most game engines leverage a number of third-party software development kits and middleware for

- data structires and algorithms,
- graphics,
- collision and physics,
- character animation,
- artificial intelligence,
- etc.

Platform independence layer

Most game engines are required to be capable of running on more than one hardware platform. Therefore, most game engines are architected with a platform independence layer. This layer sits atop the hardware, drivers, operating system, and other third-party software and shields the rest of the engine from the majority of knowledge of the underlying platform.

Core systems

Every game engine (as really every large, complex C ++ software application) requires a grab bag of useful software utilities. Here are a few examples of the facilities the core layer usually provides.

- **Memory management.** Virtually every game engine implements its own custom memory allocation system(s) to ensure high-speed allocations and deallocations and to limit the negative effects of memory fragmentation.
- **Math library.**
- **Custom data structures and algorithms.** Unless an engine's designers decided to rely entirely on a third-party package such as STL, a suite of tools for managing fundamental data structures (linked lists, dynamic arrays, binary trees, hash maps, etc.) and algorithms (search, sort, etc.) is usually required. These are often hand-coded to minimize or eliminate dynarnic memory allocation and to ensure optimal runtime performance on the target platform(s).

Resource manager

Present in every game engine in some form, the resource manager provides a unified interface (or suite of interfaces) for accessing any and all types of game assets and other engine input data.

Rendering engine

The rendering engine is one of the largest and most complex components of any game engine. Renderers can be architected in many different ways. There is no one accepted way to do it, although most modern rendering engines share some fundamental design philosophies, driven in large part by the design of the 3D graphics hardware upon which they depend.

The low-level renderer encompasses all of the raw rendering facilities of the engine. At this level, the design is focused on rendering a collection of geometrie primitives as quickly and richly as possible, without much regard for which portions of a scene may be visible. This component is broken into various subcomponents, which are discussed below.
Graphics SDKs, such as DirectX and OpenGL, require a reasonable amount of code to be written just to enumerate the available graphics devices, initialize thern, set up render surfaces (back-buffer, stencil buffer etc.), and so on. This is typically handled by a component that called the graphics device interface (although every engine uses its own terminology). The other components in the low-level renderer cooperate in order to collect submissions of geometrie primitives, such as meshes, line lists, point lists, partic1es, terrain patches, text strings, and whatever else you want to draw, and render them as quickly as possible.

The low-level renderer draws all of the geometry submitted to it, without much regard for whether or not that geometry is actually visible (other than back-face culling and c1ipping triangles to the camera frustum). A higher-level component is usually needed in order to limit the number of primitives submitted for rendering, based on some form of visibility determination.

Modern game engines support a wide range of visual effects, including

- particle systems (for smoke, fire, water splashes, etc.);
- decal systems (for bullet hole s, foot prints, etc.);
- light mapping and environment mapping;
- dynamic shadows;
- full-screen post effects, applied after the 3D scene has been rendered to an offscreen buffer
  - high dynamie range (HDR) lighting and bloom;
  - full-screen anti-aliasing (FSAA);
  - color correction and color-shift effects, inc1uding bleach bypass, saturation and de-saturation effects, etc.

Most games employ some kind of 2D graphics overlaid on the 3D scene for various purposes. These include

- the game's heads-up display (HUD);
- in-game menus, a console, and/or other development tools, which may or may not be shipped with the final product;
- possibly an in-game graphical user interface (GUI), allowing the player to manipulate his character's inventory, configure units for battle, or perform other complex in-game tasks.

Profiling and debugging tools

Games are real-time systems and, as such, game engineers often need to profile the performance of their games in order to optimize performance. In addition, memory resources are usually scarce, so developers make heavy use of memary analysis tools as well. The profiling and debugging layer encompasses these tools and also includes in-game debugging facilities, such as debug drawing, an in-game menu system or console, and the ability to record and play back gameplay for testing and debugging purposes.
However, most game engines also incorporate a suite of custom profiling and debugging tools. For example, they might include one or more of the following:

- a mechanism for manually instrumenting the code, so that specific sections of code can be timed;
- a facility for displaying the profiling statistics on-screen while the game is running;
- a facility for dumping performance stats (for example to a text file);
- a facility for determining how much memory is being used by the engine, and by each subsystem, including various on-screen displays;
- the ability to dump memory usage, high-water mark, and leakage stats when the game terminates and/ar during gameplay;

Collision and physics

Animation

Any game that has organic ar semi-organic characters (humans, animals, cartoon characters, or even robots) needs an animation system. There are five basic types of animation used in games:

- sprite/texture animation,
- rigid body hierarchy animation,
- skeletal animation,
- vertex animation,
- morph targets.

HID (Human Interface Devices)

Every game needs to process input from the player, obtained from various human interface devices including the keyboard and mouse, a joypad, or other specialized game controllers, like steering wheels, fishing rods, dance pads, etc. We sometimes call this component the player I/O component, because we may also provide output to the player through the HID, such as force feedback/rumble on a joypad or the audio.

Audio

Online multiplayer/networking

Many games permit multiple human players to play within a single virtual world. Multiplayer games come in at least four basie flavors.

- Single-sereen multiplayer.
- Split-sereen multiplayer.
- Networked multiplayer.
- Massively multiplayer online games (MMOG).

Gameplay foundation systems

The term gameplay refers to the action that takes place in the game, the rules that govern the virtual world in which the game takes place, the abilities of the player character(s) (known as player mechanics) and of the other characters and objects in the world, and the goals and objectives of the player(s). Gameplay is typically implemented either in the native language in which the rest af the engine is written, or in a high-level scripting language – or sometimes both. To bridge the gap between the gameplay code and the low-level engine systems that we've discussed thus far, most game engines introduce a layer we can call the gameplay foundations layer (for lack of a standardized name). This layer provides a suite of core facilities, upon which game-specific logic can be implemented conveniently.