Game Engine Programming

GMT Master Program
Utrecht University

Dr. Nicolas Pronost

Course code: INFOMGEP
Credits: 7.5 ECTS
Lecture #4

Game engine architecture
History

• At the beginning of game programming, the concept of game engine did not exist
  – software was specialized for application and hardware

• Now game structure is fully featured, reusable and kit-oriented
  – robustness improved in long term
  – faster to develop new games
  – earn money by selling engines
  – but look alike games
The term game engine arose in the mid 90’s
– in reference to the first FPS games (*Doom* and *Wolfenstein*)
– separation between software components, art assets, game worlds and game rules
History

• In the end 90’s, games are designed with reuse and modding
  – examples: *Quake 3 arena* and *Unreal*
  – customizable via scripts
  – secondary revenue for the studio: creating middleware for game industry is viable

*Quake 3 Arena - Id Software*  
*Unreal – Epic Games*
What is a game engine?

• Separation between game and its engine is blurry
  – depends on the development decisions
  – engine should be reserved to describe the reusable parts of a game

• The perfect virtual game engine has not yet been created
  – always requires adjustments to the content
  – genre and platform oriented which is useful if you find the right one for your goals
  – but in the right way thanks to ever-faster hardware allowing to do more in the same amount of time
What is a game engine?

• Architecture patterns of components are emerging
  – graphics and rendering
  – collision and physics
  – animation and AI
  – audio, input and resources managers
  – networking and multiplayer
  – scripting and data-driven systems

• Simplify development process

• Run on multiple platforms
Typical game team

• Engineers
  – design and implement the software (you!)
• Artists
  – produce visual and audio content
• Game designers
  – define the gameplay: story, goals, game world ...
• Producers
  – schedule and human resources
• Publishers and studios
  – own and distribute the product
• Other staff
  – marketing, administrative, IT
Engines across genres

• Engines are typically genre specific, but large overlap
  – user input, visual rendering ...

• Example
  – *Unreal Engine* (originally for FPS) used in
    • *Grimm* (Spicy Horse): action-adv.
    • *Gears of War* (Epic Games): TPS
    • *Speed Star* (Acro Games): race
Genre technology requirements

- **FPS**
  - efficient rendering, physics-based animation and AI of NPCs
- **Platformers & TPS**
  - dynamic world, high fidelity animation of main character, camera collision system
- **Fighting games**
  - animation database, accurate user input, character animation
- **Racing games**
  - LoD, rendering, rigid body physics and deformations
- **Real-time strategy (RTS)**
  - crowds, evolving environment, complex AI
- **Massively multiplayer online games (MMOG)**
  - intensively use of network, data optimization, memory and account management (persistent world), VoIP service
- **And more:** sport games, RPG, serious games, simulations, puzzles and cards, web-based games ...
Commercial game engines

- Id Software’s Quake Engine
  - started in 1992 with Wolfenstein
  - freely available open source (Quake I and II)
  - in C, a bit outdated

- Epic Games’ Unreal Engine
  - started in 1998 with Unreal
  - richest features and easy-to-use tools in UE3

- Valve’s Source Engine
  - drives Half-Life and sequels
  - complete graphics capabilities and tool set

- Microsoft’s XNA Engine
  - development platform
  - to easily create and share games
  - in C#, for PC and Xbox360
Commercial games engines

- C4 Engine
- Torque Game Engine
- 3DGame Studio
- TV3D SDK
- Leadwerks Engine
- Unity
- ShiVa Engine
- Esenthel Engine
- DX Studio
- NeoAxis Engine
- *and more*
Other engines

• In-house engines
  – many studios use their own engine
    • Ex: Ubisoft’s Anvil
  – often platform and genre dependent

• Open source engines
  – GPL and LGPL license
  – Examples
    • Panda3D
    • Yake (OGRE 3D)
    • Crystal Space
    • Blender game
    • Irrlicht
    • and more
Use existing game engine or not?

• Why use an existing game engine
  – Less development time required
  – Less testing and debugging
  – Many features directly available
  – Better focus on the game design

• Why not use an existing game engine
  – No control over the implementation of features
  – Adding features not yet in the game engine might be time consuming
  – Dependent on other licensing scheme for release
  – Other libraries/toolkits linked with the game engine (physics, AI…)
Runtime engine architecture

Your Game

Game Engine API

Graphics engine and render
Collision and physics engine
Animation engine
AI engine
Audio manager
Network and multiplayer
Core and script
...

Hardware Abstraction Layer – DirectX, OpenGL ...

Hardware Layer – sound card, graphics card, physics card, memory, HD ...
Hardware layer

• Physical
  – Graphics card
  – Sound card
  – Physics card
  – Input devices (keyboard, mouse, joysticks, game pads, steering wheels, remote controllers, cameras …)

• Drivers
  – Low level interface
User Interface

• Rather simple
  – Monitors input devices and buffers any data received
  – Displays menus and online help (can nowadays be very complex)

• Should be reusable, especially as a part of a game engine
Graphics engine

• Most graphics engine are built on top of hardware interface libraries
  – Glide: 3D graphics SDK, outdated
  – OpenGL: widely used, multiplatform
  – DirectX: Microsoft’s 3D graphics SDK
  – libgcm+Edge: PlayStation3 graphics interface
Graphics engine

• Higher level interface, tuned to a particular graphics and game type
  – Sprite-based
  – Isometric
  – Full 3D

• Can deal with higher level modeling concepts
  – Sprites
  – Solids
  – Characters (articulated)

• Handles more complicated display aspects
  – Mini map
  – Multiple views
  – Overlays
  – Special effects
Rendering

• Graphics engine is used to model the data
• Rendering performs their visual feedback
  – depends on the graphics hardware (card) and graphics engine
• Takes care of
  – low level and optimized scene graph exploration and rendering
  – visual effects (particle, mapping, dynamic shadows, HDR effect, ...)
  – front end (HUD, menus, GUI, video)
Rendering

- Dynamic shadow (Cry Engine 2)
- Octree representation
- HUD (Monkey Island 4)
- Particle systems
Collision and physics engine

- Handles the simulation of the world
  - Physical behavior (gravity, motion laws ...)
  - Collisions
  - Terrain changes
  - Ragdoll characters
  - Explosions
  - Object breaking and destruction

- Physics is more and more integrated into the gameplay and game subsystems
  - Physics-based animation
  - Interaction with objects using physics

- Limited or non-existent in simple games
Collision and physics engine

- Some SDKs
  - Havok
  - Open Dynamics Engine (ODE)
  - Tokamak
  - PhysX (Nvidia/Ageia)
    - software SDK
    - hardware card (PPU)
Animation engine

- Partially linked to physics engine
- Handles off-line
  - motion capture and retargeting
  - motion editing and annotation
- Handles real-time
  - sprite / texture animation
  - vertex animation by skinning
  - rigid body and skeletal motion
    - generation (ex: reactive character)
    - transition and blending (ex: walk to run)
    - adaptation (ex: reaching constraints, pushing, gazing)
Animation engine

- Animation packages
  - Granny, used in over 2,000 games
  - Havok Animation, extension of Havok SDK
  - Edge, for PS3
  - Endorphin (Maya plug-in) / Euphoria (real-time) include biomechanical models
AI engine

• Behavior and interaction
  – dialogue (scripted or generated)

• Spatial displacements
  – obstacle avoidance, path planning

• Strategies
  – Hiding, attack / defense, grouping

• Decision making
  – scenario based or generated in real-time

• Crowd behaviors
  – simulation of panics, riots, high density areas
AI engine

• Agent-based approach including the loop
  – perception: senses input (view), dialogues...
  – decision: AI core, rules/learning process
  – action: execute (sequence of) operation(s)
AI engine

• Some AI engines
  – AI-implant (Presagis)
    • AI authoring and runtime software solution
  – Kynapse (AutoDesk)
    • customizable game-logic system
  – DirectIA (Masa)
    • generic game AI
  – SimBionic, AISeek,...
Audio manager

• As important as graphics in the game engine
  – Effects to enhance reality
  – Ambience
  – Clues about the gameplay

• Sound formats
  – wave (high quality, memory intensive, fast)
  – mp3 (high quality, compressed, slower)
  – midi (low quality, low storage, adaptable)
  – CD (very high quality, fast, limited to background music)
Audio manager

• Simultaneous sounds
  – Mixers
  – Buffer management
  – Streaming sound

• Special features
  – Positional 3D sound (Dolby Surround)
  – Adaptive music (DirectMusic)

• Some audio managers
  – Quake and Unreal: basics
  – XACT for PC and Xbox360
  – Electronic Arts’ SoundRIOT
  – Sony’s Scream used in PS3 titles
  – IrrKlang, OpenAL, FMOD …
Networking

• To allow multiple players to play together within a shared virtual world
  – Single-screen multiplayer
    • multi devices, one camera
  – Split-screen multiplayer
    • multiple HID and cameras
  – Networked multiplayer
    • multiple computers networked
  – Massively multiplayer
    • central servers, persistent world

Networking games:
- Gauntlet (Atari Games)
- Super Mario Kart (Nintendo)
- Counter Strike (Sierra Studio)
- World of Warcraft (Blizzard)
Networking

• Data have to be transferred between players/computers
  – efficiency is the key
  – memory management critical in MMOG
• Has a profound impact on the design of the game engine
  – on world modeling, rendering, HID, animation...
• Conversion from
  – single player to multiplayer is difficult
  – multiplayer to single player is often trivial
• Networking managers: RakNet, GNet, GNE
Core and script

• The core system usually implements
  – an event system to communicate between objects
  – a scripting system to model the game logic (no rebuilt of the program)

• Gameplay is implemented in the native language of the engine and/or with scripts
  – refers to world actions and rules, character abilities, goal and objectives of the game
Core and script

• Scripting languages in game engines
  – Advantages
    • Easy control of many (or all) features in the game engine
    • Scripting language often provides full OO control
    • Promotes data-driven design
  – Disadvantages
    • Performance
    • Development support tools
    • Learning curve
Common languages used for scripting

- Python
  http://www.python.org

- Lua
  http://www.lua.org

- GameMonkey
  http://www.somedude.net/gamemonkey

- AngelScript
  http://www.angelcode.com/angelscript
Core and script

• What belongs to scripting and what belongs to the engine?
  – Engine
    • Graphics
      – rendering
      – shadows/lighting
      – occlusion culling
    • Physics
      – dynamics
      – collision response
      – raycasting
    • AI
      – pathfinding
      – fuzzy controllers
      – planning
  – Script
    • Graphics
      – time-of-day
      – add/remove lights
      – loading/moving objects
    • Physics
      – object mass/friction
      – collision events
      – raycast events
    • AI
      – path selection
      – decision making
      – goals/objectives
Human interfaces devices (HID)

• To process the I/O from/to the player
  – keyboard, mouse, joystick/pad, controllers

• Driver of the hardware
  – get physical input
  – customizable mapping to actions
  – recognition of chords, sequences and gestures
Profiling and debugging tools

• Game engines often integrates commercial or in-house tools to
  – time parts of the code
  – profile statistics in real-time
  – dump performance stats and memory use
  – control debug statements
  – record events and play them back
Data creation tools

- Not considered as game engine components
- 3D models / geometry data / animation
  - Maya, 3ds Max, Blender, SoftImage, ZBrush ...
- Audio data
  - Sound Forge, Audacity, ...
- Game world data and editor
  - Radiant (Quake), Hammer (Half-Life), UnrealEd (Unreal)
  - games provides more and more in-house world editor
  - very cheap way to extend the content of the game and make its life time longer
- Data are usually processed (simplified and encrypted) after exportation from in-house or commercial software and then loaded in-game
To go further – UU master courses

• Graphics engine and rendering
  – Geometric Algorithms, Computer Vision

• Collision and physics engine
  – Game physics

• Animation engine
  – Motion and manipulation, Computer Animation

• AI engine
  – Games and Agents, Intelligent agents, Common sense reasoning, Multi-agent programming

• Network engine
  – Algorithms and networks
End of lecture #4

Next lecture

*The game loop*