Game Engine Fundamentals

Thanos Tsakiris
Research Associate, CERTH/ITI

What is a Game Engine?

- Above all else … **NOT RESTRICTED TO GAME DEVELOPMENT!**
- **Game = Simulation**
- A game engine is a **framework** comprised of a collection of different **tools, utilities and interfaces** that hide the low-level details of the various tasks that make up the game. A game engine is the **core software** of a game or a simulation and is used to **describe a set of code used** to develop the game. Everything you see on the screen and interact with in a game world or simulation environment is powered by the game engine
- It allows the abstraction of the details of doing common game or simulation related tasks (e.g. rendering, physics, input), so that developers can focus on the aspects that make their games or simulations unique.

Popular Game Engines

- Commercial
  - ID Tech 4 (Quake 4, Doom3)
  - Steam (Half-Life 2, Left4Dead)
  - CryEngine, CryEngine 2 (Far Cry)
  - Unreal Engine
  - LithTech (F.E.A.R)
- Open Source
  - OGRE
  - Delta3D
  - ID Tech 3
  - Irrlicht

Platforms

- Most Game Engines are developed with portability in mind
- Target platforms: PC
  - Wide range of CPUs
  - Wide range of graphics cards
  - Wide range of audio cards
  - Wide range of memory
  - Wide range of I/O devices
  - Wide range of operating systems (Apple OSX, Linux)
  - DirectX, OpenGL graphics subsystems

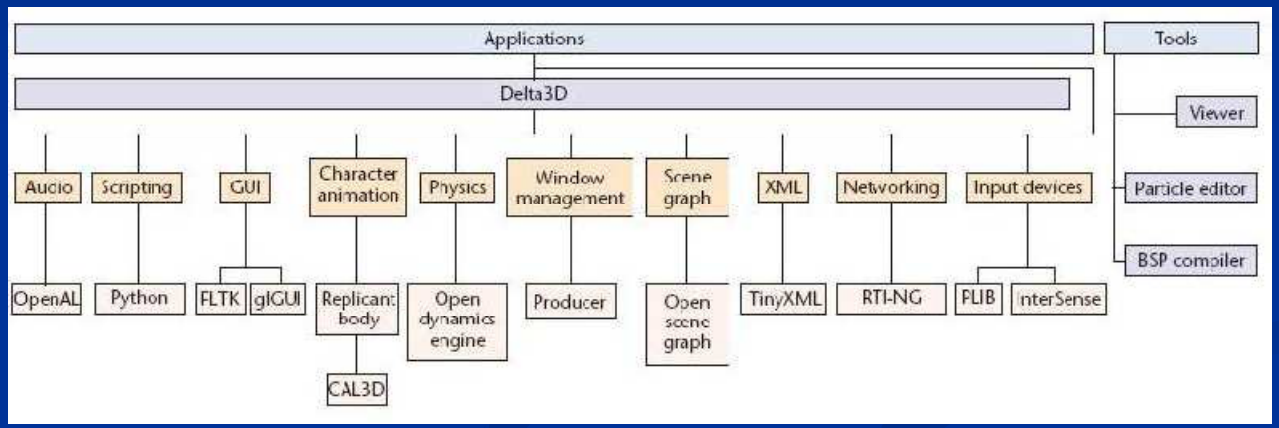
Platforms

- Other platforms:
  - VR Installations (CAVE, HMDs etc)
  - Cell phones, PDAs, etc.
  - Game Consoles:
    - Sony PS1, PS2, PS3, PSP
    - Nintendo Wii, DS
    - Microsoft Xbox, Xbox 360
  - Classic consoles
  - Arcade
  - Location based entertainment (LBE)
  - Interactive theater

Game Engine Framework

Game engines usually consist of libraries and tools that implement:

- Graphics
- Audio
- Input system
- Physics
- Artificial intelligence
- Networking
- Other utilities



General Requirements

- Maintain frame rate: usually 30 or 60 fps
- Never crash
- Tight memory & performance restrictions
- Often must work with unreleased hardware and compilers
- Threading and Multi-Core support
- Minimal configuration

Low Level Systems

- Data structures
- Math routines
- Memory management
- Resources, file I/O
- Input devices
- Widgets, tuning interface
- Performance monitoring

Low Level Components

- Data Structures
  - Lists, trees, arrays, hash tables
  - STL
- Math Routines
  - Vectors, matrices, quaternions
  - Geometry calculations
  - Random numbers
  - Misc. math routines
  - Must run fast and should take advantage of hardware if possible

Memory Management

- Many games use custom memory management routines
- Must avoid fragmentation
- Layered memory management
- Paging
- Garbage collection
- Graphics card memory management

Resources & File IO

- Fast loading
- Paging
- Parsing
- File formats
- XML
- Compression
- Resource packing

Input Devices

- Control pads, joysticks
- Keyboard, mouse
- Special hardware (haptic, 6-DOF)
- Force feedback
- Microphone
- Camera
- Configuration
- Button mapping
- Calibration

Performance Monitoring

- Time is a critical resource
- Various pieces of hardware, each with their own timing & performance characteristics: CPU, graphics, audio, IO
- Many sophisticated profilers exist
- In-game budgets & warnings
- In-game graphing
- Debugging tools for thorough analysis

Mid Level Systems

- Rendering
- Audio
- Text
- Collision detection
- Physics
- Scripting
- Networking
- Character animation
- Cinematic playback

High Level Systems

- Scene management
- User control
- Camera
- AI (artificial intelligence)
- Game logic
- Game flow
- Lighting, visual effects
- HUD
- Front end (user interface)

Graphics Subsystem

- Rendering
- Scene management
- Culling
- Level of detail
- Terrain rendering
- Character Skinning
- Particle Engine
- Effects (Sky, Water, Vegetation, Fog)

Rendering

Rendering

- Layer on top of hardware
- Common APIs: OpenGL, Direct3D, PS2
- Render polygonal meshes (display lists)
- Lighting
- Graphics state
- Matrix & viewing transformations
- Shaders

Audio Subsystem

- 3D spatialization: panning, Doppler, Dolby Surround, HRTF (head related transfer functions)
- Manage sound priorities (voices)
- Reverb, effects
- MIDI
- Music
- Dynamic music
- Stream off CD / DVD (multiple streams)
- Voice

Tools

- Code Development Tools
  - Compilers (Visual C++, SN Systems, CodeWarrior, GNU)
  - Debugger
  - Profiler
  - Editor
  - Revision control (CVS, SourceSafe, SVN)
  - Integrated development environment (IDE)
  - C++, Assembly, Scripting Languages
  - Graphics languages: pixel & vertex shaders…
  - Design analysis tools
  - Documentation, standards

Tools

- Middleware
  - Getting more and more popular and trusted
  - Rendering: RenderWare, NDL, Intrinsic, OGRE, OpenSceneGraph
  - Physics: ODE, Havok, PhysX, Newton, MathEngine
  - XNA, Bink, FMOD, ScaleForm

Art Production Tools

- 3D Modeling & Animation (Maya, 3D Studio)
- Exporting Modules
- Asset management (AlienBrain)
- Paint (2D & 3D) (Photoshop, Z-Brush, DeepPaint)
- Scanning (2D, 3D)
- Motion capture
- In-game tools & editors

Audio Tools

- Recording
- Composing (ProTools)
- Sound effects (Reason)
- Spatial Audio Configuration tools
- In-game tools

Game Design Tools

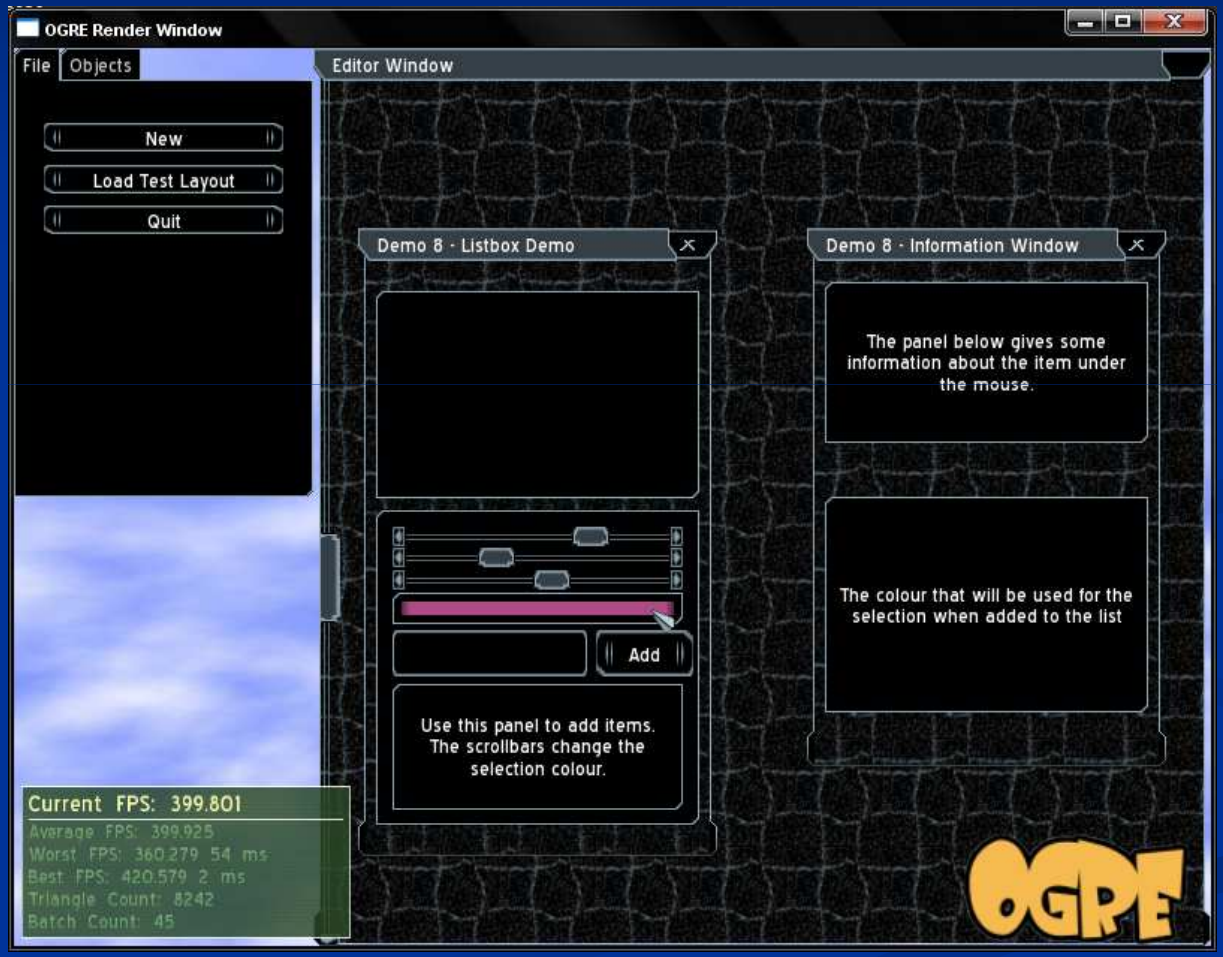
- In-game tools
- Level layout
- Prototyping tools (Director, Flash)
- Design tools
- GUI Tools

GUI Generation (CEGUI)

- To create specialised graphical user interfaces we use libraries that allow their use under a multitude of graphics rendering architectures (e.g. OpenGL, DirectX etc)
- The CEGUI library (Crazy Eddie's GUI - http://www.cegui.org.uk) is one of the most complete open-source libraries that is compatible with a great number of 2D and 3D rendering and game engines (e.g. OGRE, Delta3D, Dark Basic, Torque, Crystal Space 3D etc.)
- The main characteristic of the CEGUI library is the ability for complete customisation of component graphics and the ability to have different gui schemes for different purposes at the same time (e.g. a GUI scheme for menus, one for the main game etc., each one with different graphics and functions)
- Also, the GUI parameterisation is done in the most part in XML scripts which allows for a great deal of GUI customisation without changing the main game's codebase.

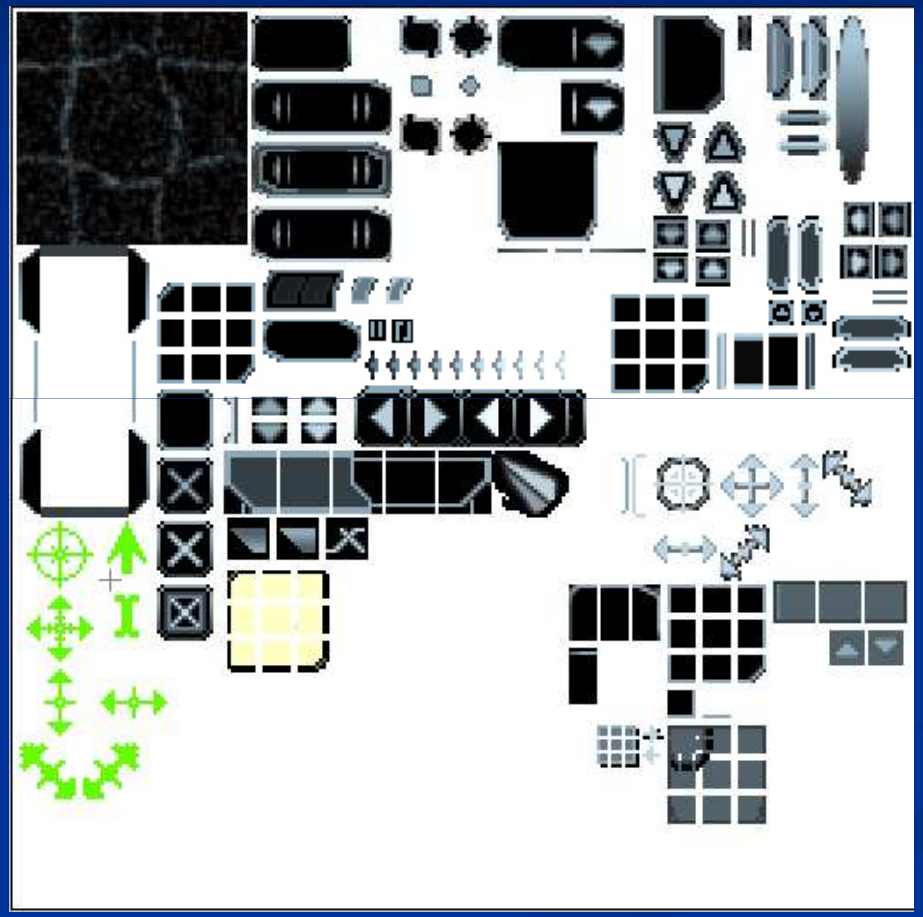
GUI Generation (CEGUI)

- GUI components example:



GUI generation (cont.)

- GUI Imageset example

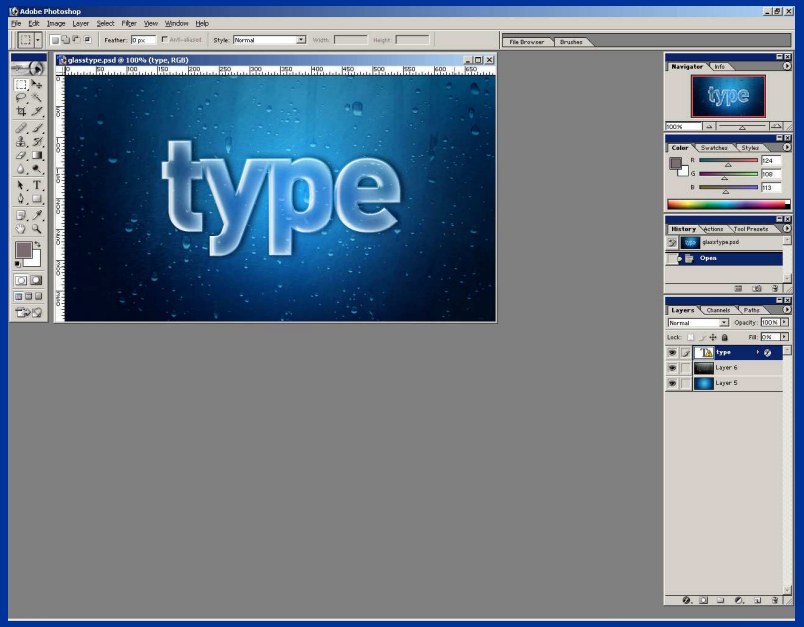


Differences between Game Interfaces - WIMP

- Application Interfaces are based mainly on the WIMP paradigm (Windows, Icons, Mouse, Pointer)
- In contrast, game control interfaces are based on direct manipulation of objects, the player, vehicles etc. while WIMP elements are reduced to a secondary role (selection menus, object inventories etc.)
- Usage of special interaction devices in games (Joystick, Joypad, Steering wheel etc.)

Differences between Game Interfaces – WIMP (continued)

- WIMP applications: Reusage capability of ready-made components (buttons, drop-down menus *κλπ*)
- Games: Custom-made components are required most of the time (special icons, pointers, 3D objects *κλπ*.)



WIMP interface example

Game Interface example

Input Handling

- In general applications, the use of conventional controls is usually handled by a ready-made API and each component (button, text box, menu each) allows the attribution of a function to a certain component action
- On the other hand, although input APIs used in games allow for some of these facilities, the majority of functions each user input performs is up to the game programmers to implement since it involves custom actions.
- Input handling in most games relies on regular update each time the screen is redrawn or at time intervals chosen by the developer (Frame Rate Dependent – Frame Rate Independent)
- In the case of frame rate dependent input control, the developer must take care so that the screen refresh rate does not affect the input handling (lag)
- In a frame rate independent architecture, the programmer must keep separate threads of execution for each function performed by the game (graphics, physics, AI, input etc), or maintain a system of timers for each function that is independent from each other

Input handling

- Context-dependent Handling

Every input can have a different function depending on the circumstances: e.g., in an RPG game:

- Left-click on an enemy results in a different action depending on weather the user controls a sword or a pistol
- Left click on the terrain will move the player character with a different speed depending on the type of terrain (road, rocks etc.)
- Left click on an object can either be translated as acquisition of the object (near the object) or motion towards the object (far from the object) based on the player character's distance to the object

Multimodal Interfaces

- The majority of game systems use conventional interaction devices (keyboard, mouse, joystick, joypad)
- In the last 2-3 years there are efforts to make game interaction more natural e.g. Nintendo WiiMote:

- WiiMote is not a multimodal solution but a more advanced motion interaction using 6DOF infrared tracking (as in VR systems)

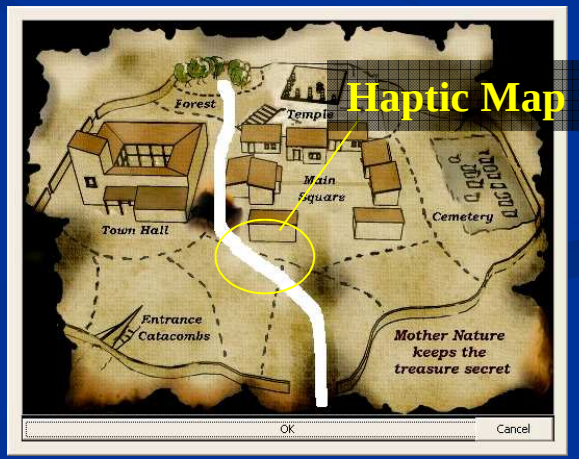
Multimodal Interfaces (cont.)

- Similar technique but with optical tracking is Sony's EyeToy system and Microsoft's upcoming Project Natal

- This system allows the player to perform natural motions as well as to be directly projected within the game space without using an avatar.

Multimodal Game interfaces (cont.)

- Multimodal in game applications are still in research
- By multimodal we mean systems where the user can use 2 or more interaction modes (e.g.. Natural motion tracking and Speech recognition, haptic interaction and gaze-tracking etc.)



Multimodal Game interfaces (cont.)

- Haptic interaction devices aimed at games:

FPS Vest

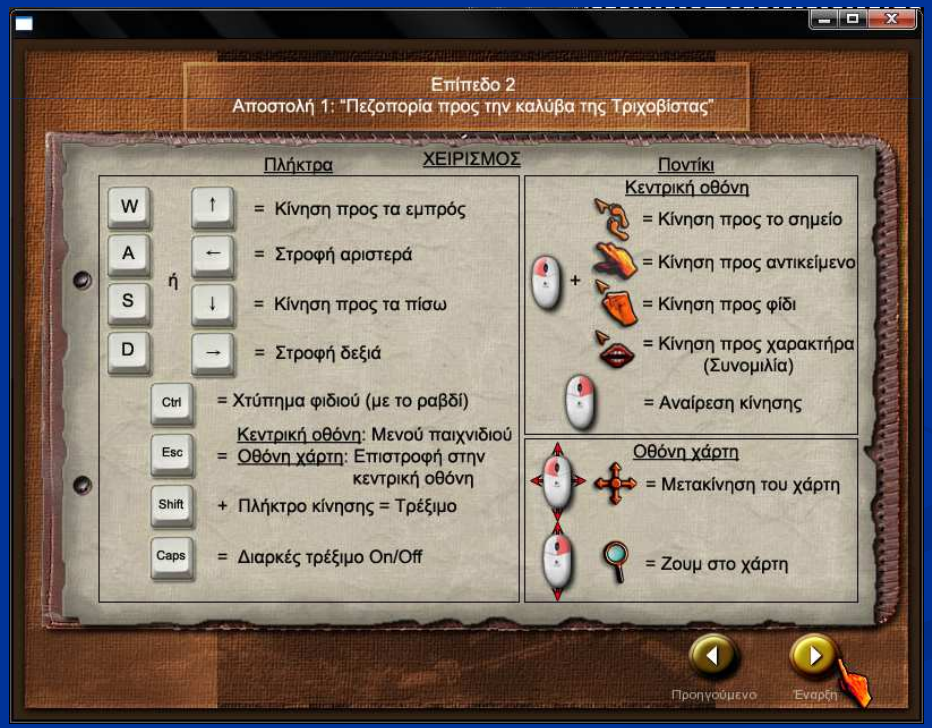
Novint Falcon

Interaction

- Interaction within a game or a simulation is the most important ingredient to the overall user satisfaction
- Each game/simulation has its own requirements that depend on the genre, the controls supported, the response time of the interaction, how realistic is the interaction response etc.
- Each application's target group defines the interaction options to a great extent.
- Whenever possible the ability for the user to customize and modify interaction parameters is imperative. (e.g. mouse motion speed, inverting up/down axis, multi-modal control etc.)

Interaction (cont.)

- Interface example: 3rd person Action Adventure
- Basic supported actions: Character motion, weapon use, object acquisition object use, inventory control, NPC dialogue (Non-Player Characters), character status
- User requirements: Motion supported both by keyboard (up/down/left/right or WASD buttons) and mouse



Interaction (cont.)

- **Character motion:** This function is handled differently when keys are used than when mouse is used
- Keys perform these actions:
  - Up/down: increase/decrease motion speed forward and backward respectively affecting the animation accordingly
  - Left/Right: increase torque left/right respectively
- Mouse motion actions:
  - Click on the terrain moves the character to the spot (double-click = run)
  - Click on an object moves the character to the spot and the object is acquired
  - Click on a character moves the player to the spot and starts a dialogue with the character

  Main Difference: When using the mouse the application has to handle the motion parameters automatically (speed change, rotations, collisions, obstacle avoidance etc.)

Interaction (cont.)

- **Character dialogue**: An example of using WIMP components
- Modal dialogue windows
- The creation of a dialogue control scripting system where depending on the dialogue the game is affected accordingly

Interaction (cont.)

- **Object acquisition**
- Basic questions: Which objects are considered active and can be acquired and how are they acquired?
- Usage of Collision materials so we know which are active and which are decorative
- Usage of Collision detection so we know when the character is close enough to acquire an object
- Player notifications when an object is acquired (WIMP)

Interaction (cont.)

- **Inventory Control- Object usage**
- Basic questions: Which actions are supported in the inventory? How can objects be used? How many objects fit in the inventory? How many objects of the same type can the player have? What happens when 2 objects are combined to form a new one? How do objects affect the character when used?
- The inventory GUI also contains the list of dialogues played so far

Game Engines in Research

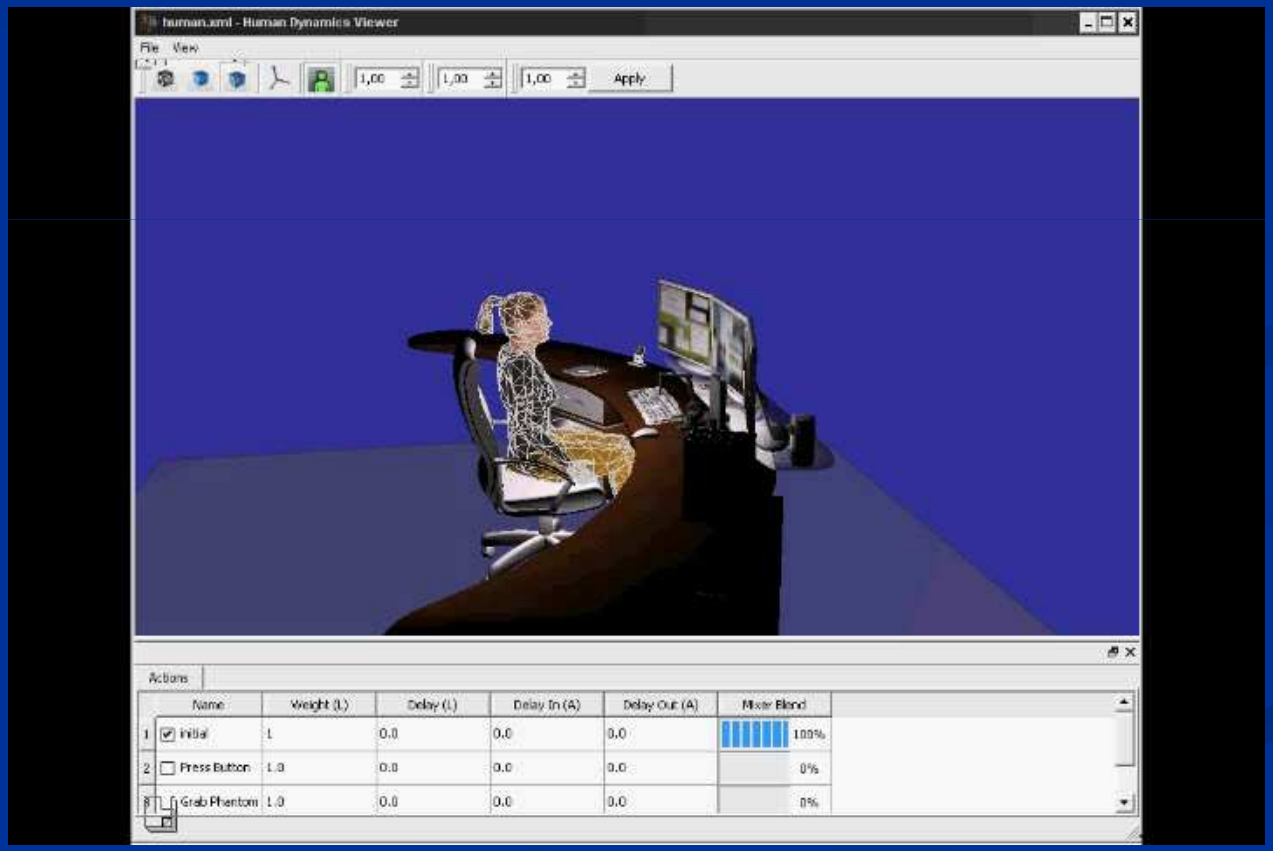
- VR is dead – Long live Game Engines!
- Any research project that requires the implementation of simulation and 3D visuals can utilize a game engine for a great part of the development
- Examples:
  - SIMILAR NoE 06: A multimodal game for users with disabilities (OGRE)
  - VR@Theatre: Generation and Simulation of a theatre play (OGRE)
  - Virthualis: Human factors simulation in VR (Delta3D)

Game Engines in Research

- CERTH/ITI has implemented more than 10 projects using a game engine (OGRE) including a full game for the Museum of Macedonian Struggle

Game Engines in Research

- We are now beginning implementation of the simulation platform for the VERITAS project to facilitate Physical Simulation of Accessibility within a 3D environment using the Delta3D game engine



Game Engines in Research