IMGD 4000
Technical Game Development II
Game Engines

Robert W. Lindeman
Associate Professor
Interactive Media & Game Development
Human Interaction in Virtual Environments (HIVE) Lab
Department of Computer Science
Worcester Polytechnic Institute
gogo@wpi.edu
Pedagogical Goal

- Your technical skills should not be tied to any particular game engine
  - Just like your programming skills should not be tied to any particular programming language!
- Use the best tools for each job
- ... or the tools you were given 😊
Definition

Game Engine

A series of modules and interfaces that allows a development team to focus on product game-play content, rather than technical content.

[Julian Gold, OO Game Dev.]

☐ But this class is about “the technical content” ! 😊
Buy Versus Build

- Depends on your needs, resources and constraints
  - Technical needs (e.g., “pushing the envelope” ?)
  - Financial resources (e.g., venture capital ?)
  - Time constraints (e.g., 1 mo. or 2 yr. ?)
  - Platform constraints (e.g., Flash ?)
  - Other factors (e.g., sequel ?)

- Most games commonly built today with some sort of “engine layer”
Choices:
It’s a Jungle Out There!

- **361** 3D engines reviewed at devmaster.net/devdb/engines

- We are **not** going to try to review them all here!

R.W. Lindeman - WPI Dept. of Computer Science
Interactive Media & Game Development
Many Evaluation Dimensions/Features

<table>
<thead>
<tr>
<th>Filter Results</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Keywords</strong></td>
</tr>
<tr>
<td>unity</td>
</tr>
<tr>
<td><strong>Name</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Developer name</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Status</strong></td>
</tr>
<tr>
<td>any</td>
</tr>
<tr>
<td><strong>Platforms</strong></td>
</tr>
<tr>
<td>Windows, Linux, Mac OS X, Solaris</td>
</tr>
</tbody>
</table>

| Licenses       |
| GPL, LGPL, ZLIB, MIT |

| Languages written in |
| C/C++, Java, C#, Visual Basic 6 |

| Languages supported |
| C/C++, Java, C#, Visual Basic 6 |

| Features       |
| General, Lighting, Shadows, Texturing |

If there’s a feature term here you don’t know, you should look it up!
Types of Engine Architectures (Roughly)

- **Monolithic** (e.g., Unreal Engine)
- **Modular**
  - Extensible IDE (e.g., Unity)
  - Open Class Library (e.g., C4 or Dragonfly?)
Monolithic Engines (e.g., Unreal)

- “Old style”
  - Typically grew out of specific game
- Tend to be genre-specific
- Difficult to go beyond extensions/
  modifications not anticipated in (e.g.,
  scripting) API
- Proven, comprehensive capabilities
Modular Engines
(e.g., C4 and Unity)

- “Modern”
  - Often developed by *game engine company* (relatively new category)
- Use *object-oriented* techniques for greater modularity
- Much easier to extend/replace components than monolithic engines
Modular: Extensible IDE’s (e.g., Unity)

- GUI-oriented development process
  - More accessible for novice/casual programmers
  - More “art asset friendly”

- Comprehensive asset management

- Limited (controlled) exposure of internals
  - Prevents abuse
  - Prevents some extensions
Modular: Open Class Library (e.g., C4)

- Code-oriented development
- Very carefully layered
- Allows maximum modifiability
- Often open source (e.g., jME)
- Not as accessible for novices and “casual” programmers
Dragonfly

Log Manager

Manager

Resource Manager

World Manager

Graphics Manager

Input Manager

Game Manager
# C4 Architecture

## Base Services

<table>
<thead>
<tr>
<th>Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Manager</td>
<td>Low-level memory operations.</td>
</tr>
<tr>
<td>File Manager</td>
<td>Low-level file access.</td>
</tr>
<tr>
<td>Resource Manager</td>
<td>Loading resources, defining custom resource types.</td>
</tr>
<tr>
<td>Time Manager</td>
<td>Time access, deferred events, timers, date.</td>
</tr>
<tr>
<td>Math Library</td>
<td>Vectors, matrices, quaternions, random numbers.</td>
</tr>
<tr>
<td>Utility Library</td>
<td>Lists, maps, arrays, strings, graphs, trees, smart links</td>
</tr>
</tbody>
</table>

## System Managers

<table>
<thead>
<tr>
<th>Manager</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphics Manager</td>
<td>Renderables, textures, shaders, post-processing.</td>
</tr>
<tr>
<td>Sound Manager</td>
<td>Playing sounds, sound flow, audio streaming.</td>
</tr>
<tr>
<td>Network Manager</td>
<td>Low-level networking access.</td>
</tr>
<tr>
<td>Input Manager</td>
<td>Input devices, actions, control configuration.</td>
</tr>
<tr>
<td>System Utilities</td>
<td>Event handling, threads, mutexes, variables, logging.</td>
</tr>
</tbody>
</table>

C4 Architecture (cont.)

Large-Scale Architecture

- **World Manager**
  Scene graph, nodes, objects, model animation.

- **Controller System**
  Controllers, scripts, functions.

- **Physics Manager**
  Rigid bodies, joints, force fields.

- **Message Manager**
  High-level networking access.

- **Effect Manager**
  Particle systems, emitters, panels, markings.

- **Interface Manager**
  User interface widgets, windows, menus.

- **Movie Manager**
  Playing movies into interface widgets.

Plugin Modules

- **World Editor**
  The World Editor plugin API.

- **Browser Plugin**
  Displaying interactive web browsers in panels.

- **Logitech Plugin**
  Image output on the Logitech G15 keyboard.

http://www.terathon.com/docs/
“Best” Engine Choice is Relative to the Situation

- Similar issues of needs, resources, and constraints (as in buy vs. build)
  - Platform, programming language constraints
  - Cost constraints (commercial: $ to $$$)
  - Specific technical features required (e.g., MMO)
  - Previous experience of staff
  - Support from developers, user community (e.g., forums)
  - Pedagogical goals (e.g., this course)
Choice of Unity for IMGD 4000

- Unity 3D

  - Much better for artists (trivial importing) than, e.g., C4
  - Programming in C# (good for structure and robustness)
  - Debugger support with Monodevelop
  - Very popular
  - Could help you get an internship

http://www.unity3d.com
But...

C4 would also be a solid option

- Written in C++
  - Builds on your Dragonfly experience

- We have a full source-code license

- Better control over lighting
  - Games look “better”

- Requires more work on your part

- Maybe next year...
Detailed Feature Comparisons

- C4/Unity from DevMaster.net
- Dragonfly from my/your knowledge

Caveats:
- Info is not audited
- Let’s not get bogged down in the details
  - The idea is to get overall sense of emphasis
General Features

- Object-Oriented Design, Plug-in Architecture, Save/Load System:
- Extremely clean class hierarchy for scene graph nodes, including geometries, cameras, lights, sounds, zones, portals, triggers, markers, and special effects
- General state serialization support for saving worlds
- Quick save and quick load capabilities
- Separation between per-instance and shared data
- Support for pack files and a virtual directory hierarchy
- Skinable GUI's

- Object-Oriented Design, Plug-in Architecture, Save/Load System:
- Professional FPS controller ready to drop in (and tune)
- Streamed loading for the Unity Web Player
- Unity asset server / asset source code version control
- Cross-platform web player content, the Unity Web Player is available for both Mac OS X and Windows users and works with all browsers
- Standalone executables for both Mac OS X and Windows
- Mac OS X Dashboard Widgets
- iPhone Publishing is available as add-on product
- Streaming Asset Bundles: the ability to stream in any asset (terrain, mesh, etc) into the game
Scripting

- Graphical script editor for easy artist/designer access
- Games can easily define custom script components, and these automatically appear in the editor
- Controllers can advertise custom function calls that can be accessed from scripts
- Scripts support variables, looping, and conditional execution, all shown in a concise graphical manner

- Uses the Mono and supports JavaScript, C# and Boo, interoperable (to a certain extent) and JIT'ted to native code
- Complete scripting documentation
- Source-level debugging

R.W. Lindeman - WPI Dept. of Computer Science
Interactive Media & Game Development
Built-in Editors

- Full-featured integrated cross-platform world editor
- Graphical shader editor
- Graphical script editor
- Interface panel editor
- Animation cue editor

- Editor provides zero-cost asset pipeline: save a file and it updates automatically
- Editor Extensibility: Create completely custom editor windows, and entirely new tools and workflows.
- Asset Server that provides version control capabilities for Unity projects
- Optimized for use with large projects
- New Server view integrated into the Unity user interface
- Updates, commits, and graphical version comparisons are all done inside the Unity editor.
- Procedural tree creator
Physics

- Rigid body simulation
- Real-time cloth simulation
- Real-time large-scale fluid surface simulation
- Continuous collision detection
- Contact forces and friction
- Precise buoyancy forces
- Customizable force fields
- Joints (spherical, universal, discal, revolute, cylindrical, and prismatic)
- Full networking support

- Basic Physics
- Collision Detection
- Rigid Body
- Vehicle Physics
- Powered by the PhysX Engine, which also supports particle physics
- Cloth simulation

R.W. Lindeman - WPI Dept. of Computer Science
Interactive Media & Game Development
Lighting

- Point lights, spot lights, directional lights
- Inter-zone lighting analysis
- Fully dynamic lighting
- Ambient lighting volumes
- True volumetric light shafts
- Screen Space Ambient Occlusion (SSAO)

- Per-vertex
- Per-pixel
- Lightmapping Beast-Lightmapping
Shadows

- Fully dynamic lighting
- Stencil shadows
- Cascaded shadow mapping with smooth transitions
- Projected light source shadows
- All types of shadows can be combined

- Projected planar
- Blob shadows
- Realtime dynamic soft shadows
- Shadows are dynamic, optimized, and allow self-shadowing (only available in Unity Pro)
Texturing

- Multitexturing
- Bump mapping
- Parallax mapping
- Horizon mapping
- Diffuse, specular, normal, gloss, emission, occlusion maps
- Emission glow and specular bloom effects
- Surface markings / decals on arbitrary geometry

- Basic
- Bump mapping
- Procedural
Shaders

- Large variety of standard shaders can be created with material attributes
- Complex custom shaders can be created with graphical shader editor
- Comprehensive bump/normal mapping capabilities
- Parallax bump texture mapping
- Horizon bump shadow mapping
- Bumpy reflection and refraction
- Cook-Torrance microfacet shading
- Cube environment reflections
- Realistic water shading

- Vertex
- Pixel
- High Level
- Unity comes with an extensive library of 40 shaders including Vertex Lit, Diffuse, Glossy, Bumped, Bumped Specular, Reflective, Self-illuminating, a Toon (Cell) shader, and 9 different particle shaders.
- Everything falls back gracefully on low-end GFX cards
- Parallax shaders
- GLSL support (in addition to Cg)
Scene Management

- Portals
- Visibility graph
- Occlusion culling
- Level of detail
- Fully lit impostors for massive forests
- Mirrors and remote portals
- Object instancing
- External scene referencing
- Game level save/load system

- Occlusion culling
- Umbra occlusion culling
Animation

• Skeletal animation
• Hierarchical animation blending system
• Forward kinematics
• Inverse kinematics

• Keyframe Animation
• Skinned character animation
• Procedural Characters and Animation: the ability to stitch multiple body parts into one character, and reassign bones to different characters. The entire skinned animation system is now scriptable.
Meshes

- Dynamic level-of-detail
- Skinning
- Constructive Solid Geometry (CSG) operations
- Import from Collada, OpenGEX

Mesh Loading
- Skinning Native importing from Cinema 4D, Maya, Cheetah3D, Blender.
- Also support for Collada, FBX, 3DS, OBJ formats
Special Effects, etc.

- Full-scene cinematic motion blur
- Distortion effects such as heat haze and shockwaves
- High-quality and high-performance procedural fire effects
- Volumetric fog with multiple density functions
- Interactive in-game interface panels
- Fully extensible particle systems
- Voxel-based blob (metaball) particle systems
- Voxel terrain (Cliffs, overhangs, caves, arches)
- Transvoxel algorithm for dynamic seamless level-of-detail

- Rendering Terrain engine with full editor tools, dense foliage, lightmapping and more
- Rendering with Shader Replacement: This makes it simple to make incredible eye-candy like depth-of-field, soft particles, thermal goggles, etc.
- Lens Flares
- Particle System
- Motion Blur
- Sky, Water, Mirror
- Color correcting filter, grayscale, sepia, and twirl
- Skinnable in-game GUIs
Networking

- Client-Server
- Fast, reliable network implementation using UDP/IP
- Solid fault tolerance and hacker resistance
- Advanced security measures, including packet encryption
- Automatic message distribution to entity controllers
- Cross-platform internet voice chat

- Client-Server
- Build on Raknet
- Supports .NET library and asynchronous WWW API
- Multiplayer Networking (advanced NAT punch-through, delta compression, easy to set up)
Sound & Video

- 2D & 3D sound
- Streaming sounds
- Doppler shift and other frequency effects
- Reverberation
- Atmospheric effects
- Directional sounds with cone attenuation
- Obstruction attenuation applied to direct and reflected paths
- Audio capture for voice chat

- 2D & 3D Sound
- Streaming Sound
- Streaming video and audio based on FMOD, includes sound effects (Reverb Zones, Various Filters: Low Pass Filter, High Pass Filter, Echo Filter, Distortion Filter, Reverb Filter, Chorus)