Game Engines

Technical Game Development II

Professor Charles Rich
Computer Science Department
rich@wpi.edu

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”

Types of Engine Architectures (Roughly)

- **Monolithic** (e.g., Unreal Engine)
- **Modular** (e.g., C4 Engine)
- **Tool Kit** (e.g., jME)

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)

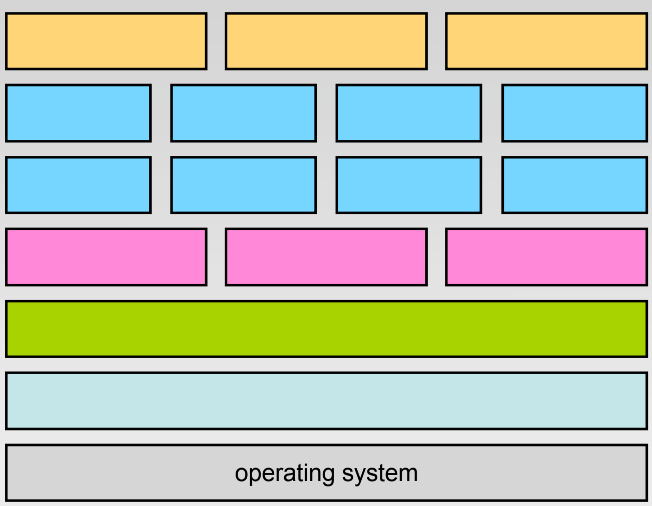
- “modern”--often developed by game engine company
- use object-oriented techniques for greater modularity
- much easier to extend/replace components than monolithic engines
- architecture a bit more “bundled” (IDE-like) than tool-kit engines (see next)

Tool Kit Engines (e.g., jME)

- highly object-oriented
- designed for maximum modifiability
- typically open source
- may not be as complete or mature

Basic Game Engine Architecture Blocks

ai
animation
camera
controllers
effects
game
geometry
graphics



gui
input
level builder
modeler
physics
scene graph
sound
renderer

Choices: "It's a Jungle Out There"

- 284 3D engines reviewed at DevMaster.net

Most Reviewed Open Source Engines

1. OGRE
2. Irrlicht
3. Crystal Space
4. Panda3D
5. jME
6. Reality Factory
7. The Nebula Device 2
8. RealmForge
9. Blender Game Engine
10. OpenSceneGraph

Most Reviewed Commercial Engines

1. Torque Game Engine
2. TV3D SDK 6.5
3. 3DGameStudio
4. C4 Engine
5. Unity
6. NeoAxis Engine
7. DX Studio v2.2
8. 3Impact
9. Beyond Virtual
10. Deep Creator

Latest Engines XML

- Reactor 3D Engine
- Demoniak3D
- Q 2.0 Technology
- CSP Game Engine
- Lightning Engine
- AGE
- E76 game engine
- Ston3D Engine
- Awakening
- Dawntec 3D Engine

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

Many Evaluation Dimensions/Features

[DevMaster.net]

General Info

Graphics API
OpenGL | DirectX | Glide | Software | Other

Operating Systems
Windows | Linux | MacOS | Solaris | SunOS | HP/UX | FreeBSD | Irix | OS/2 | Amiga | DOS | Xbox | Playstation | GameCube | GBA | PSP | N-Gage | BeOS | Xbox360 | PS2 | PS3 | Nintendo Wii | Nintendo DS

Programming Language
C/C++ | Java | C# | D | Delphi | Pascal | BASIC | Ada | Fortran | Lisp | Perl | Python | Visual Basic 6 | VB.NET

Status
Alpha | Beta | Productive/Stable | Inactive

Misc
Documentation

General Features
Object-Oriented Design | Plug-in Architecture | Save/Load System | Other

Game Features

Networking System
Client-Server | Peer-to-Peer | Master Server

Tools & Editors
Scripting | Built-in Editors

Sound & Video
2D Sound | 3D Sound | Streaming Sound

Physics
Basic Physics | Collision Detection | Rigid Body | Vehicle Physics

Artificial Intelligence
Pathfinding | Decision Making | Finite State Machines | Scripted | Neural Networks

Graphics Features

Lighting
Per-vertex | Per-pixel | Volumetric | Lightmapping | Radiosity | Gloss maps | Anisotropic | BRDF

Shadows
Shadow Mapping | Projected planar | Shadow Volume

Texturing
Basic | Multi-texturing | Bumpmapping | Mipmapping | Volumetric | Projected | Procedural

Shaders
Vertex | Pixel | High Level

Rendering
Fixed-function | Stereo Rendering | Raytracing | Raycasting | Deferred Shading | Render-to-Texture | Voxel | Fonts | GUI

Scene Management
General | BSP | Portals | Octrees | Occlusion Culling | PVS | LOD

Animation
Inverse Kinematics | Forward Kinematics | Keyframe Animation | Skeletal Animation | Morphing | Facial Animation | Animation Blending

Meshes
Mesh Loading | Skinning | Progressive | Tessellation | Deformation

Surfaces & Curves
Splines | Patches

Special Effects
Environment Mapping | Lens Flares | Billboarding | Particle System | Depth of Field | Motion Blur | Sky | Water | Fire | Explosion | Decals | Fog | Weather | Mirror

Terrain
Rendering | CLOD | Splatting

If there's a feature term here you don't know, you should look it up!

Best Choice is Relative to Situation

- Similar issues of needs, resources and constraints (as in buy vs. build)
  - platform, programming language constraints
  - cost constraints (commercial run $ 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 C4 and jME for This Course

- C4 Engine http://www.terathon.com/c4engine
  - modular
  - C++ language (industry standard)
  - previous experience in IMGD 3000
  - good TA support (TJ)
  - reasonable cost
  - technically sophisticated
  - good support community (forum)

Choice of C4 and jME for This Course

- jME (jMonkeyEngine) http://jmonkeyengine.com
  - tool kit
  - Java language
    - "up and coming", especially for mobile
    - ties in with Darkstar assignment
    - much less error-prone than C++
  - *pedagogical goal:* experience with two substantial engines (Game Maker doesn't count)
  - free, open source
  - technically sophisticated
  - good support community (forum)

C4 and jME Comparison

- Architecture
- Guided Tour of Tutorial Examples
- Feature comparison

Terathon Software

C4 ENGINE

Base Services

Memory Manager
- Allocating memory
- Organizing multiple heaps

File Manager
- Reading and writing files
- Asynchronous file access
- Building file lists

Resource Manager
- Resource loading
- Multiple resource catalogs
- Packing utilities

Time Manager
- Frame timing
- Deferred events and timers

Math Library
- Vectors, matrices, quaternions
- RGB colors, complex numbers
- Linear algebra routines
- Polygon manipulation

Utility Library
- Arrays, lists, trees
- Smart links, link targets
- Reference-counted objects
- Strings

System Managers

Graphics Manager
- Renderable objects
- Light objects
- Camera objects
- Texture maps
- Shading attributes

Sound Manager
- Playing sounds
- Streaming music

Network Manager
- Low-level network access
- Establishing connections
- Domain name resolution

Input Manager
- Enumerating input devices
- Assigning input actions

System Utilities
- Multithreading
- Persistent variables
- Console commands
- Logging tools

Large-Scale Architecture

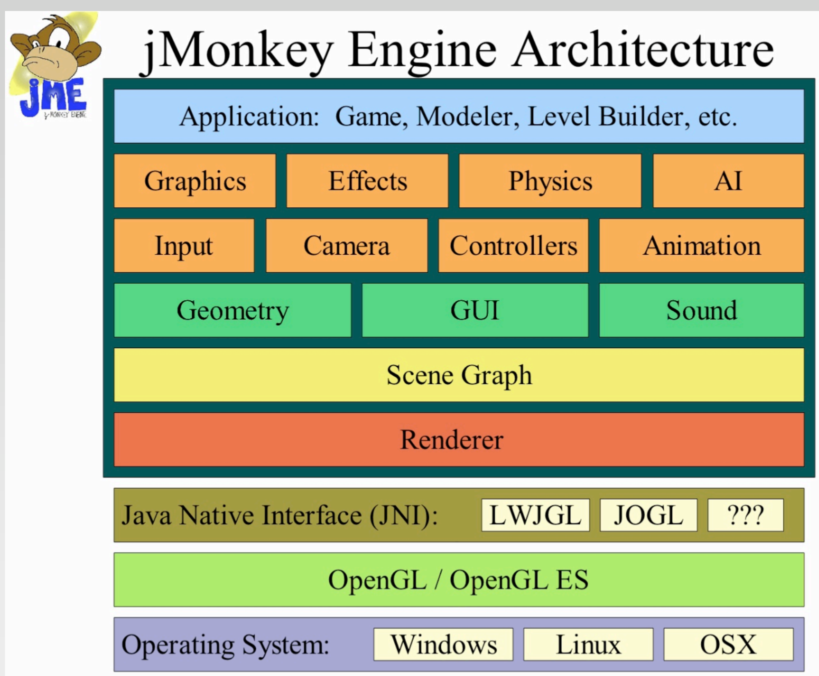
World Manager
- Node hierarchy
- Geometries, lights, cameras
- Sound sources
- Zones and portals
- Triggers, markers
- Entities

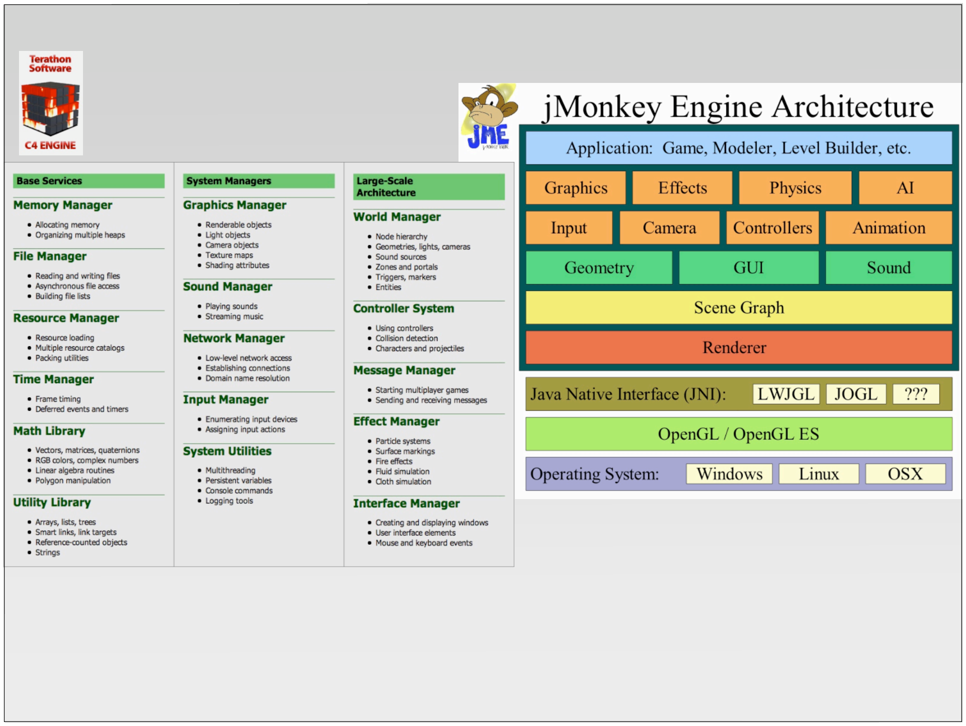
Controller System
- Using controllers
- Collision detection
- Characters and projectiles

Message Manager
- Starting multiplayer games
- Sending and receiving messages

Effect Manager
- Particle systems
- Surface markings
- Fire effects
- Fluid simulation
- Cloth simulation

Interface Manager
- Creating and displaying windows
- User interface elements
- Mouse and keyboard events





Guided Tour of Tutorial Examples

- Why are we doing this?
  - ***not*** to save you the trouble of reading the documentation!   (You will need to anyways :-)
    - leaving out many details (e.g., error checking)
    - reordering for clarity (e.g., combining .h and .cpp files)
  - ***not*** interested in low-level C++ vs. Java coding differences
  - goal is to understand the *design space* of game engines by looking closely at different choices made
  - more generally, *thoughtful reading* of other people's code is an important skill for software developers
    - paying close attention to modularity and architecture

```
module C4::Application *ConstructApplication(void)  // called by C4 engine
{ return (new Game); }

class Game : public Application {

  private: EntityRegistration ballEntityReg; // for World Editor
           MovementAction *forwardAction; // typical input control

  Game() :
    ballEntityReg(kEntityBall, "model/Ball", kEntityPrecache, kControllerBall)
  {
    ballEntityReg.SetEntitySize(0.125F, 0.125F, 0.125F);
    ballEntityReg.SetEntityColor(ColorRGB(0.0F, 1.0F, 0.0F));
    TheWorldMgr->SetWorldConstructor(&ConstructWorld);
    // create and register movement actions
    forwardAction = new MovementAction(kActionForward, kSpectatorMoveForward);
    TheInputMgr->AddAction(forwardAction);
  }

  class MovementAction : public Action {

    void Begin(void)
    {
      GameWorld *world = static_cast<GameWorld *>(TheWorldMgr->GetWorld());
      SpectatorCamera *camera = world->GetSpectatorCamera();
      camera->SetSpectatorFlags(camera->GetSpectatorFlags() | movementFlag);
    }
  };

  World *ConstructWorld(const char *name, void *data) // called by TheWorldMgr
  { return (new GameWorld(name)); }
};
```

```
class GameWorld : public World {

  private: SpectatorCamera spectatorCamera;

  GameWorld(const char *name) :
      World(name),
      spectatorCamera(2.0F, 1.0F, 0.3F) {}

  WorldResult Preprocess(void)
  {
    Zone *zone = GetRootZone();
    const Marker *marker = zone->GetFirstMarker();
    while (marker)  // find World Editor marker for camera placement
      {
        MarkerType type = marker->GetMarkerType();
        if (type == kMarkerLocator)
          {
            if (static_cast<const LocatorMarker *>(marker)->GetLocatorType() == kLocatorSpectator)
              {
                spectatorCamera.SetNodePosition(marker->GetWorldPosition());
                const Vector3D direction = marker->GetWorldTransform()[0];
                float azimuth = Atan(direction.y, direction.x);
                float altitude = Atan(direction.z, Sqrt(...));
                spectatorCamera.SetCameraAzimuth(azimuth);
                spectatorCamera.SetCameraAltitude(altitude);
              }
          }
        marker = marker->ListElement<Marker>::Next();
      }
    SetCamera(&spectatorCamera); // set world's camera for rendering
    return (kWorldOkay);
  }
};
```

```
public abstract class AbstractGame { // in com.jme.app

    protected DisplaySystem display;

    public final void start() {
      initSystem();
      initGame();
      while (!finished && !display.isClosing()) {
        InputSystem.update();
        update();
        render();
      }
    }
}

public class SimpleGame extends AbstractGame {

    public static void main(String[] args) { // called by JVM
      new SimpleGame().start();
    }

    protected Camera camera;
    protected InputHandler input;
    protected LightState lightState;
    protected Node rootNode;

    protected final void update() {
      timer.update(); // recalculate frame rate
      float tpf = timer.getTimePerFrame();
      input.update(tpf); // check for key/mouse events
      rootNode.updateGeometricState(tpf, true);
    }

    protected final void render() {
        display.getRenderer().clearBuffers();
        display.getRenderer().draw(rootNode);
    }
```

```
    protected final void initSystem() {
      display = DisplaySystem.getDisplaySystem(properties.getRenderer());
      display.createWindow(...);
      camera = display.getRenderer().createCamera(...);
      camera.setFrustumPerspective(...);
      camera.setFrame(...);
      camera.update();
      display.getRenderer().setCamera(camera);
      // setup input controls
      input = new FirstPersonHandler(camera);
    }

    protected final void initGame() {
      rootNode = new Node("rootNode");
      // create ZBuffer
      ZBufferState buf = display.getRenderer().createZBufferState();
      buf.setEnabled(true);
      buf.setFunction(ZBufferState.CF_LEQUAL);
      rootNode.setRenderState(buf);
      // set up basic default light
      PointLight light = new PointLight();
      light.setDiffuse(new ColorRGBA(1.0f, 1.0f, 1.0f, 1.0f));
      light.setAmbient(new ColorRGBA(0.5f, 0.5f, 0.5f, 1.0f));
      light.setLocation(new Vector3f(100, 100, 100));
      light.setEnabled(true);
      // attach light to a lightState and the lightState to rootNode
      lightState = display.getRenderer().createLightState();
      lightState.setEnabled(true);
      lightState.attach(light);
      rootNode.setRenderState(lightState);
      // attach example box to root node
      rootNode.attachChild(new Box("my box",new Vector3f(0,0,0),new Vector3f(1,1,1)));
      // update geometric and rendering information
      rootNode.updateGeometricState(0.0f, true);
      rootNode.updateRenderState();
    }
}
```

Some Observations from Code Tour

- Code is overall more similar than different
  - systematic separation of node vs. state (to allow reuse of state desriptions)
    - C4: Light/LightObject, etc.
    - jME: Light/LightState, etc.
  - controllers associated with nodes for response to events

Some Observations from Code Tour

- Examples of how C4 more bundled, IDE-like:
  - C4 makes heavier use of singleton “managers”
    - C4 has single root node in WorldManager
    - any jME program can call updateGeometricState on any node
  - World editor more tightly integrated
    - “markers” installed in world editor and searched for by game initialization

Detailed Feature Comparisons

- From DevMaster.net
- Caveats:
  - Info may not be up-to-date (especially for jME)
  - I have added a few comments of my own
  - 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
- External scene graph referencing from within another scene graph
- Support for pack files and a virtual directory hierarchy
- Skinable GUI's

Modular OO based design with abstract interfaces for all low level APIs:
- 3D Text Generation
- Binding system for input controls
- Support for using jME in a Java Applet
- New Importer and Exporter System giving a standard framework for loading and saving jME scenegraphs
- A Binary Format implementation for the new import/export system that is more compact and faster than standard Java serialization
- Control Binding Management

Scripting

Graphical script editor

Efforts underway to add scripting extensions:
- Current JVM's include JavaScript and LiveConnect (easy api between Java and JS) [CR]

Builtin-Editors

- Full-featured integrated cross-platform world editor
- Interface panel editor
- Complete built-in windowing system
- Powerful and intuitive interface design
- Advanced surface attribute manipulation and material management

Level editor considered separate project:
- e.g., MonkeyWorld3D [CR]

Physics

Basic Physics, Collision Detection, Rigid Body:
- Built-in character controller.
- Built-in projectile controller.
- Real-time fluid surface simulation.
- Real-time cloth simulation.

Collision Detection:
- Triangle accurate collision detection

Physics considered separate project:
- e.g., jME Physics interface to ODE (Open Dynamics Engine) [CR]

Lighting

Per-vertex, Per-pixel, Lightmapping, Radiosity, Gloss maps, Anisotropic:
- Support for fully dynamic infinite, point, and spot lights
- Gloss-mapped specular reflections
- Ambient radiosity
- Projected cube and spot textures
- Cook-Torrance microfacet shading

Per-vertex, Lightmapping

Shadows

Shadow Mapping, Projected planar, Shadow Volume:
- All shadows are rendered in real time at global scale
- Three types of shadows are seamlessly combined in one world
- True penumbral soft shadows for area light sources

Shadow Volume:
- Z-Pass shadow volumes

Shadows

Shadow Mapping, Projected planar, Shadow Volume:
- All shadows are rendered in real time at global scale
- Three types of shadows are seamlessly combined in one world
- True penumbral soft shadows for area light sources

Shadow Volume:
- Z-Pass shadow volumes

Texturing

Basic, Multi-texturing, Bumpmapping, Mipmapping, Projected:
- Comprehensive bump mapping capabilities
- Enhanced parallax mapping
- Ambient occlusion channels
- Emission/glow maps
- Horizon mapping
- Realistic water shading

Basic, Multi-texturing, Mipmapping, Procedural:
- Support for simple texture based dot3 bump mapping

Shaders

Vertex, Pixel, High Level:
- Extensive support for vertex programs and pixel shaders

Vertex, Pixel, High Level:
- Support for OpenGL Vertex Programs.
- Support for OpenGL Fragment Programs
- Support for GLSL [*** we will use *** -CR]

Scene Management

General, Portals, Occlusion Culling, LOD:
• Efficient large-scale visibility determination
• Advanced inter-zone lighting analysis at runtime
• Special support for mirrors and remote portals
• Object instancing and external scene referencing
• Scene data can be imported from Collada format

General, Octrees, LOD:
• Scene graph based architecture

Animation

Skeletal Animation, Animation Blending:
• Full skeletal hierarchy support for deformable meshes
• Powerful hierarchical animation blending system

Keyframe Animation, Skeletal Animation:
• A Skin and Animatable Bone System enabling realistic representation of models and motion

Meshes

Mesh Loading, Progressive:
- Support for the Collada scene format, enabling models to be imported from 3D Studio MAX, Maya, XSI, Blender, and other content creation packages

Mesh Loading, Skinning:
- Handles it's internal format (.jme) and exports to ASE, 3DS, MD2, MD3, Milkshape, Obj and Collada
- Support for importing files in the COLLADA format
- New extension providing the ability to generate 3d meshes from text

Special Effects

Environment Mapping, Lens Flares, Billboarding, Particle System, Motion Blur, Sky, Water, Fire, Decals, Fog, Mirror:
- Cube environment mapping
- Environment-mapped bump mapping
- Fully extensible particle systems
- Surface markings on arbitrary geometry
- Bump-mapped (fully lit) surface markings
- Real-time fire and electrical effects
- Transparent warping effects (heat haze, etc.)
- Bumpy reflection and refraction
- Postprocessed glow
- Fog volumes
- Full-scene cinematic motion blur
- Interactive in-game interface panels

Environment Mapping, Lens Flares, Billboarding, Particle System, Sky, Water, Fire, Explosion, Fog:
- Cloth Simulation
- Water, with configurable reflection, refraction, wave generation and more
- Bloom, with configurable intensity, blurring, resolution and more-Dot3 Bumpmapping

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

Networking viewed as separate project:
• e.g., see Darkstar
[CR]

Sound and Video

2D Sound, 3D Sound, Streaming Sound:
• Fully spatialized 3D sound effects
• Unlimited streaming music channels with seamless looping and concatenation
• Doppler shift and other frequency effects
• High-precision sound travel delay
• Atmospheric absorption effects
• Reverberation with multiple simultaneous environments
• Directional sounds with cone attenuation
• Obstruction attenuation applied to direct and reflected paths
• Frequency-dependent volume settings for all effects
• Permeation system determines how far sounds travel through interiors
• Apple's QuickTime technology can be used to play movies or soundtracks from numerous formats

3D Sound:
• OpenAL support with 3D position

Rendering

Fixed-function, Render-to-Texture:
- Antialiasing (up to 8x)
- Bilinear and trilinear filtration
- Anisotropic filtration (up to 16x)
- Vertical Sync control

Fixed-function, Render-to-Texture, Fonts, GUI:
- Rendering system supports both rendering to a screen context as well as rendering to a texture.
- Implements a Rendering Queue that automatically sorts opaque, transparent and screen objects and renders them in the correct order
- Multipass rendering system
- Supports rendering into a web-page via applets
- FBO support
- Support for rendering to Framebuffer Objects

WPI IMGD 4000 (D 08) 41

Summary Ratings (5 star scale)

Overall:	4.5	(48 votes)
Features:	4.0	
Ease of Use:	4.0	
Stability:	4.5	
Support:	4.5	

Enjoy them!

Overall:	4.0	(28 votes)
Features:	4.0	
Ease of Use:	4.0	
Stability:	4.0	
Support:	4.5	

WPI IMGD 4000 (D 08) 42