IMGD 3000 - Technical Game Development I: Game Engine Structure

by

Robert W. Lindeman

gogo@wpi.edu

The User Experience

- You spawn into an outdoor scene
  - Flag waving
  - Waterfall
  - Trees
  - Rocks
  - A bridge
  - A satellite dish
- You shoot at the rocks
  - A projectile
- Animate vs. inanimate objects

The Engine Experience

- Engine must provide support for your world
  - Load the scene objects
  - Place inanimate objects
  - Place you
  - Make the flag wave, the water fall
  - Make your projectile fly/hit/disappear
  - Show you everything

High-Level Engine Code

- Basic game loop:

```
InitializeObjects( );
while( gameNotFinished )  {
   // Handle user input
   // (mouse, keyboard, gamepad, etc.)
   // Update objects in the world
   // Render the World
}
```

Digging Deeper: Initialization

```
ResourceResult GameWorld::Preprocess( void )  {
  ResourceResult result = World::Preprocess( );
  if( result != kResourceOkay ) return( result );
  SetCamera( &spectatorCamera );
  playerCamera = &firstPersonCamera;
  spawnLocatorCount = 0;
  CollectZoneMarkers( GetRootZone( ) );
  const Marker *marker = GetFirstSpectatorLocator( );
  if( marker )  {
    // Initialize spectatorCamera to the marker's
    // position and direction.
  else  {
    spectatorCamera.SetNodePosition( Point3D( 0.0F, 0.0F, 1.0F  ));
  }
  return( kResourceOkay );
}
```

Digging Deeper: User Input

- C4 defines a *singleton* called **`TheInputMgr`**
- Singleton?
- The input manager dispatches actions to your code
  - You need to
    - subclass the **`Action`** class
    - Define **`Begin()`** and **`End()`** methods
    - Bind the action to the instance you want to use

Game Engine Flow

- Load program
- Initialize variables
- Load mission/level information
- Place objects/NPCs into world
- Schedule events
- Start clock
- Spawn player
- Handle events
  - Generated by player(s), NPCs, or timers

Multiplayer: Server

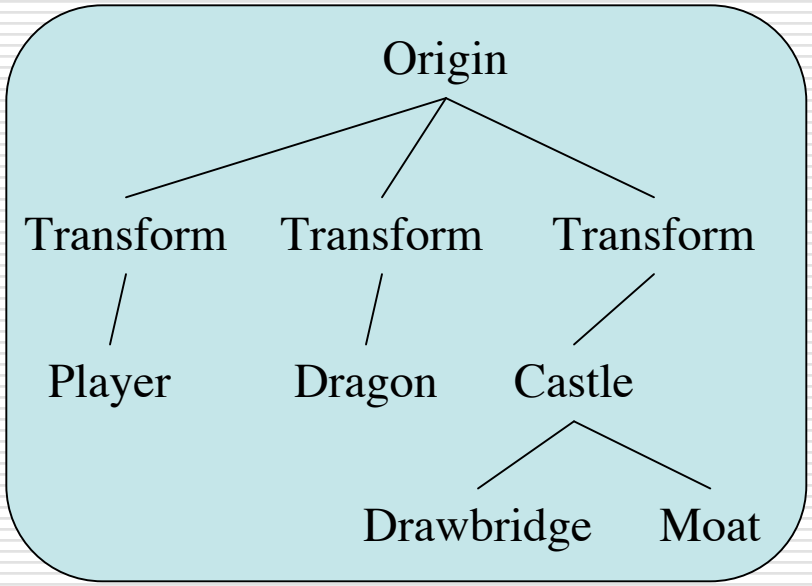
- Start server
  - Like previous slide
  - Events include clients joining
- Spawn player
- Receive updates from clients
- Update global state
  - Maintain the world state
- Disseminate state changes
  - To clients
  - To other servers

Multiplayer: Client

- Load client code
- Search for a server
  - Choose wisely!
- Establish connection
- Receive current game state
- Render game to user
- Receive
  - Input from user
  - Updates from server

Game Engines

- Scene graph
  - Representation of the world
  - Includes characters
- Timing is very important
  - Events
    - Time-based
    - Multi-player
  - Synchronization
- Database of objects
- Networking
  - Between Server and clients
  - Between Servers



Game Graphics

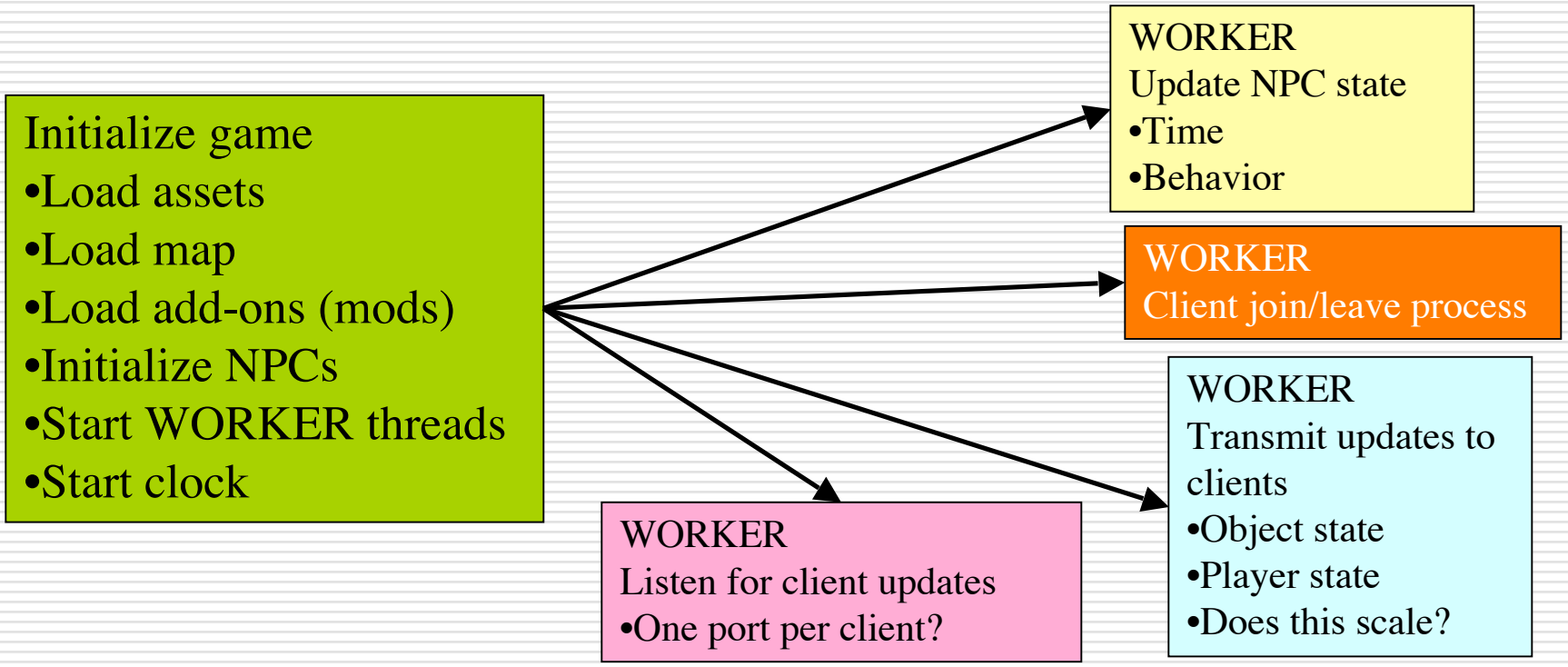
- Different from other media
  - Need to process and display @ 30 fps
  - Dynamic scenes
- Graphics Processing Units (GPUs) are now programmable
  - Need to understand how to program for them
  - nVidia's cg programming language, OpenGL 2.0 extensions, GLSL
  - Stream-processing model
  - Data must be packed into textures
  - Limited control support
    - Loops, stack data structures
- Good jobs here!

Physics

- Need to consider how fast you can compute
  - Scalable in the number of objects?
  - Scalable in the types of objects?
    - Cloth?
    - Hair?
    - Water?
- Three main types of objects
  - Point masses
  - Rigid bodies
  - Soft bodies
- Life is a combination of physics and freewill
  - How do we balance these?

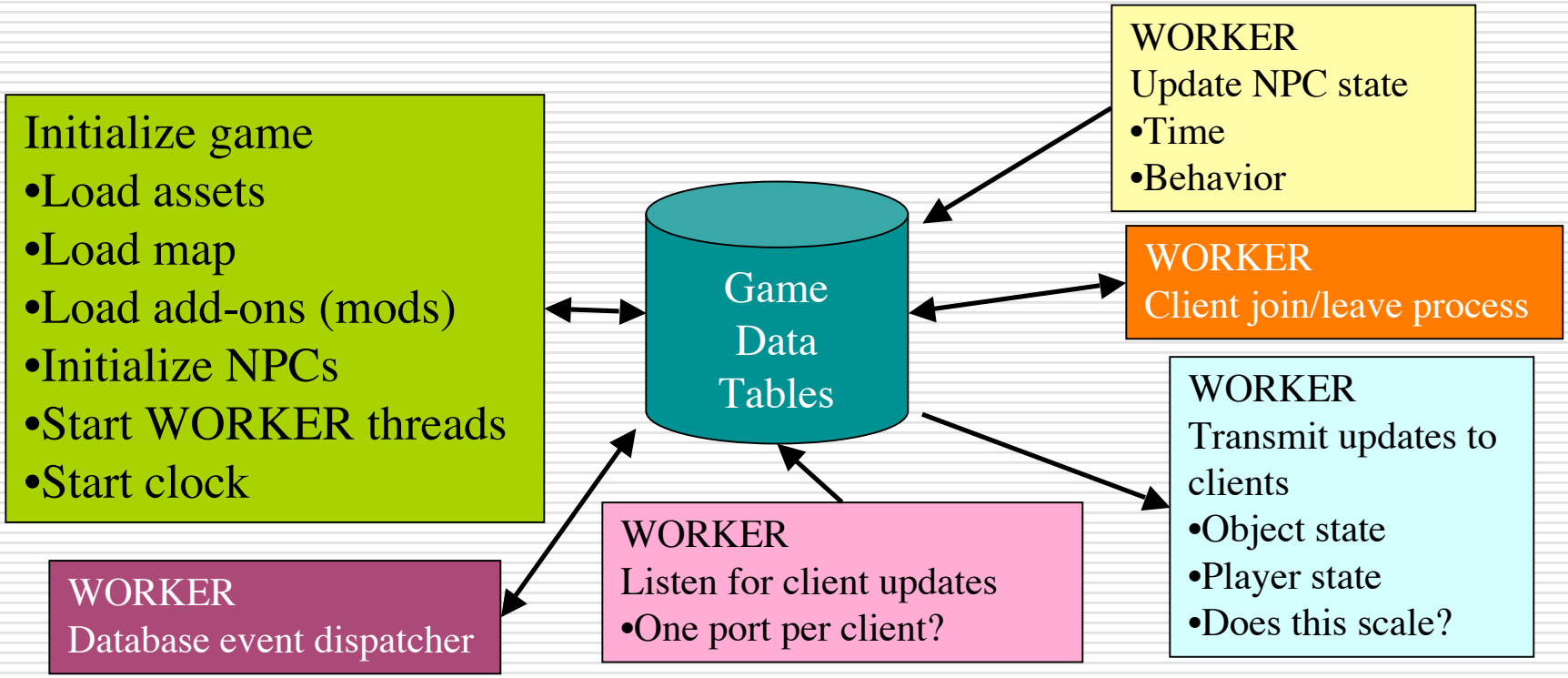
Server Details

- Server performs multiple tasks concurrently
  - Each WORKER is a separate thread
  - How do they coordinate efforts?



Server Coordination

- Each worker has tables of interest
  - Workers sleep until table data changes
  - Database dispatcher monitors tables, wakes workers



Even More Server Details

- For this to work, you need
  - Threads
  - Inter-process/thread Communication
    - Sockets
    - Shared memory
  - Some way of doing timing
    - Callback
    - Interrupt handler
  - An efficient data store
- In order to do it well, you also need
  - Thorough understanding of systems programming
  - A very good design, and lots of it!
  - You should have seen this in CS-3013: OS, and CS-2303: Systems Programming Concepts

Client/Server Approach

- Requires messages to be passed
  - Network could be bottleneck
  - Server could be bottleneck
- Lag is bad
  - Example: the player you shoot at is "magically" not there anymore by the time the projectile gets to him
- Inconsistent state is bad
  - Who grabbed that object first?

Client/Sever Programming

- Make it easy on the programmer
  - Hide the fact that things are being sent to server
- Make "surrogates" for server objects
  - Underlying system does actual communication
- How can we make a system really scalable to 1000s of users?
  - How is this done in gaming systems?

Graphical User Interface

- Provides access to
  - Game menus (*e.g.*, save, load, boss)
  - Player status (*e.g.*, health, current speed)
  - Maps
    - Current play location
    - Location of "persons of interest"
    - Location of "goals"
  - Non-Player Character (NPC) dialog
  - Player-to-player chat

Rob, stop here…

- I said STOP!

C4 Scene Graph

- Everything in the scene is part of the scene graph
- The scene graph is created (loaded) at initialization
- At runtime, your game will manipulate the nodes in the graph
  - Update transformations (positions/orient.)
  - Add nodes (e.g., projectiles)
  - Delete nodes (e.g., health packs)

Traversing the Scene Graph

- In C4, the root node is called the "infinite zone"
  - All game elements must be part of a zone
- You can access the root node with the **`World::GetRootNode()`** function
- Move through (traverse) the tree with
  - **`GetFirstSubNode()`**
  - **`GetNextNode()`**
  - **`GetPreviousNode()`**
  - etc.
- Look at the **`Tree`** class
- More on scene graphs later

More on Nodes

- Search the C4 API for "hierarchy"
  - Shows Node class hierarchy
- A *transform* is a matrix representing the object's position, orientation, and scale
- Two notions of a *transform*
  - ***Local transform*** is relative to the immediate parent node in the scene graph
  - ***World transform*** is the absolute position in world space
- Moving an object means updating its transform

Game Loop, Revisited

- Can expand "Update objects in the world" to:
  - Starting at the root node in the scene graph, traverse from parent to child nodes recursively
  - For each node, if certain conditions are met, call some function to update the transform
- But how do you specify what code to call, and under what conditions?

Controllers

- One way to change a node's transform is through the use of a controller
- The Node class has **`Set/GetController()`** methods
- Controller class has **`Move()`** method
  - This is what is called during traversal
  - This is where you put your transform update code
  - Actually, you can update the transform of *any* nodes from this method!

Controllers (cont.)

- As with many things, the controller class makes heavy use of inheritance
  - **`CollectableController`**
  - **`DoorController`**
  - **`LightningController`**
  - **`RotationController`**
  - **`CharacterController`**
  - **`RocketController`**
- Everything that has some kind of behavior has a controller assigned to it
  - Swinging lights? **`PendulumController`**

Controllers (cont.)

- What's the difference between the **`Move()`** and **`Travel()`** controller methods?
  - Movement code goes in **`Move()`**, and tells C4 where you want your object to go.
  - **`Travel()`** is used to apply any corrective movement caused by things like collisions
- The collision system tries to move each object, and checks for collisions
  - In **`Travel()`**, if a collision happened, handle it. If not, set the position to the final position calculated in **`Move()`**.
  - You need to tell the sysem what to do once a collision happens in the **`Travel()`** method.

Final Notes on Controllers

- If you want to animate something
  - Make sure that the associated node has a controller assigned to it
  - Add your code to update the transform in the **`Move()`** method of the controller

Geometry and Nodes

- Geometry (mesh) information is not contained directly in the node
  - It is stored in a GeometryObject
  - See `Set/GetObject()` methods for geometry nodes
  - Separating them allows for instancing, saving memory
    - Each instance has its own transforms

C4 Engine Structure

- Layered structure
  - Base Services
  - System Managers
  - Large-Scale Architecture
  - Plugin Modules
  - Application (e.g., your game)

http://www.terathon.com/c4engine/architecture.php

C4 Base Services

- File Manager
- Memory Manager
- Time Manager
- Resource Manager
- Math Library
- Utility Library
- System Utilities

C4 System Managers

- Sound Manager
- Rendering Core
- Display Manager
- Graphics Manager
- Input Manager
- Network Manager

C4 Large-Scale Architecture

- Interface Manager
- Message Manager
- Effect Manager (fluid, cloth, particles)
- Scene Graph
- Animation System
- Controller System
- World Manager
- Plugin Manager

C4 Plugin Modules

- Import Tools (Collada, TGA files)
- World Editor
- Application Module
- Media players
  - Model viewer
  - Texture viewer
  - Font generator
  - Sound player
  - Movie player