Lecture 9

Memory Management: High-Level Overview
Gaming Memory (Last Generation)

- Playstation 3
  - 256 MB RAM for system
  - 256 MB for graphics card
- X-Box 360
  - 512 MB RAM (unified)
- Nintendo Wii
  - 88 MB RAM (unified)
  - 24 MB for graphics card
- iPhone/iPad
  - 1 GB RAM (unified)
Gaming Memory (Current Generation)

- Playstation 4
  - 8 GB RAM (unified)

- X-Box One
  - 8 GB RAM (unified)
  - 5 GB for games

- Nintendo Wii-U
  - 2 GB RAM (unified)
  - 1 GB only for OS

- iPhone/iPad
  - 1 GB RAM (unified)
Why Not Virtual Memory?

- **Secondary storage** exists
  - Consoles have 500 GB HD
  - iDevices have 64 GB Flash
- But **access time** is slow
  - HDs transfer at ~160 MB/s
  - Best SSD is ~500 MB/s
- Recall **16 ms** per frame
  - At best, can access 8 MB
  - Yields uneven performance
Memory Usage: Images

- Pixel color is 4 bytes
  - 1 byte each for r, b, g, alpha
  - More if using HDR color

- Image a **2D array** of pixels
  - 1280x1024 monitor size
  - 5,242,880 bytes ~ 5 MB

- More if using **mipmaps**
  - Graphic card texture feature
  - Smaller versions of image
  - Cached for performance
  - But can double memory use
But My JPEG is only 8 KB!

- Formats often **compressed**
  - JPEG, PNG, GIF
  - But not always TIFF
- Must **uncompress** to display
  - Need full pixels to uncompress
  - In RAM or graphics card
- Only load when needed
  - Texture loading is primary I/O operation in high-end games
  - Many tricks to optimize
  - The cause of “texture popping”
Traditional Memory Organization

- **Stack**: Function parameters, Local variables, Return values (Easy to Track)
- **Heap**: Objects created via `new`, Allocations with `malloc` (Problems!)
- **Program Data**: Program Code, Static Variables (Easy to Track)

Memory Management
Traditional Memory Organization

High Address

Stack

Function parameters
- Local variables
- Return values

Free Space

Dedicated to process.

Consists of machine addressable space.

Leverages Virtual Memory

Heap

Objects created via `new`
- Allocations with `malloc`

Program Data

Program Code
- Static Variables

Low Address
Mobile Memory Organization

Device Memory

Heap

Device Memory

Stack

Program Data

Stack

Program Data

Stack

Program Data

Stack

Program Data
How Do Apps Compete for Memory?

- Active app takes what it can
  - Cannot steal from OS
  - OS may *suspend* apps

- **App Suspension**
  - App quits; memory freed
  - **iOS**: 5 min (or so) on exit
  - **Android**: If needed

- Suspend apps can *recover*
  - OS allows limited paging
  - Page out on suspension
  - Page back in on restart
How Do Apps Compete for Memory?

- Active app takes what it can
  - Cannot steal from OS
  - OS may *suspend* apps

- **App Suspension**
  - App quits; memory freed
  - **iOS**: 5 min (or so) on exit
  - **Android**: If needed

- Suspend apps can *recover*
  - OS allows limited paging
  - Page out on suspension
  - Page back in on restart

Can override in **iOS 7**
How Do Apps Compete for Memory?

- Active app takes what it can
  - Cannot steal from OS
  - OS may *suspend* apps

- **App Suspension**
  - App quits; memory freed
  - **iOS**: 5 min (or so) on exit
  - **Android**: If needed

- Suspend apps can *recover*
  - OS allows limited paging
  - Page out on suspension
  - Page back in on restart

Can override in **iOS 7**

You must code this! Otherwise, data is **lost**.
State Management in iOS 7

- **Active**
  - Running & getting input

- **Inactive**
  - Running, but no input
  - Transition to suspended

- **Background**
  - Same as inactive
  - But apps can stay here
  - **Example**: Music

- **Suspended**
  - Stopped & Memory freed
State Management in iOS 7

- **Active**
  - Running & getting input

- **Inactive**
  - Running, but no input
  - Transition to suspended

- **Background**
  - Same as inactive
  - But apps can stay here
  - **Example**: Music

- **Suspended**
  - Stopped & Memory freed

Write handlers to process entering, leaving each state.
iOS State Handling

- **applicationDidBecomeActive:**
  - Your app became (resumed as) the foreground app.
  - Use this to recover memory state.

- **applicationWillResignActive:**
  - Your app will switch to inactive or background.
  - Stop the game loop and page out memory.

- **applicationDidEnterBackground:**
  - Your app is in the background and may be suspended.

- **applicationWillEnterForeground:**
  - Your app is leaving the background, but is not yet active.
Android State Handling

Activity launched
  - onCreate()
  - onStart()
  - onResume()
  - onRestart()

Activity running
  - onPause()
  - onStop()
  - onDestroy()

App process killed
  - User navigates to the activity

Apps with higher priority need memory
  - The activity is no longer visible

Activity running
  - User returns to the activity

Activity launched
  - onCreate()
  - onStart()
  - onResume()
  - onRestart()

All method in Application class

Memory Management
Android State Handling

Activity launched
  - onCreate()
  - onStart()
  - onResume()

Activity running
  - onPause()
  - onStop()
  - onDestroy()

User navigates to the activity
App process killed
  - Apps with higher priority need memory

Another activity comes into the foreground
  - User returns to the activity

User navigates to the activity
The activity is no longer visible

User navigates to the activity
The activity is finishing or being destroyed by the system

All method in Application class

Reload memory
Android State Handling

All method in Application class

Page out memory

Memory Management
Memory Organization and Games

Inter-Frame Memory
Carries over across frame boundaries

Update

Draw

Intra-Frame Memory
Recovered each frame
Memory Organization and Games

Inter-Frame Memory
Carries over across frame boundaries

Heap or Stack?
Does it matter?

Intra-Frame Memory
Recovered each frame

Draw
# Distinguishing Data Types

## Intra-Frame
- **Local computation**
  - Local variables (managed by compiler)
  - Temporary objects (not necessarily managed)

- **Transient data structures**
  - Built at the start of update
  - Used to process update
  - Can be deleted at end

## Inter-Frame
- **Game state**
  - Model instances
  - Controller state
  - View state and caches

- **Long-term data structures**
  - Built at start/during frame
  - Lasts for multiple frames
  - May adjust to data changes
### Distinguishing Data Types

**Intra-Frame**

- **Local computation**
  - Local variables
    - Local variables (managed by compiler)
    - Local variables (not necessarily managed)
  - Transient data structures
    - Built at the start of update
    - Used to process update
    - Can be deleted at end

**Inter-Frame**

- **Game state**
  - Model instances
  - Long-term data structures
    - Built at start/during frame
    - Lasts for multiple frames
    - May adjust to data changes

---

**Local Variables**

**Object Fields**
## Distinguishing Data Types

<table>
<thead>
<tr>
<th>Intra-Frame</th>
<th>Inter-Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Local computation</strong></td>
<td><strong>Game state</strong></td>
</tr>
<tr>
<td>- Local variables</td>
<td>- Model instances</td>
</tr>
<tr>
<td>(managed by compiler)</td>
<td>- Scene and caches</td>
</tr>
<tr>
<td>(not necessarily managed)</td>
<td>- <strong>Object Fields</strong></td>
</tr>
<tr>
<td><strong>Transient data structures</strong></td>
<td><strong>Long-term data structures</strong></td>
</tr>
<tr>
<td>- Built at the start of update</td>
<td>- Built at start/during frame</td>
</tr>
<tr>
<td>- Lasts for multiple frames</td>
<td>- Lasts for multiple frames</td>
</tr>
<tr>
<td>- May adjust to data changes</td>
<td>- <strong>e.g. Pathfinding</strong></td>
</tr>
<tr>
<td><strong>Local Variables</strong></td>
<td><strong>e.g. Collisions</strong></td>
</tr>
<tr>
<td><strong>Object Fields</strong></td>
<td><strong>e.g. Pathfinding</strong></td>
</tr>
</tbody>
</table>
## Handling Game Memory

### Intra-Frame
- Does not need to be paged
  - Drop the latest frame
  - Restart on frame boundary
- Want size reasonably **fixed**
  - Local variables always are
  - Limited # of allocations
  - Limit new inside loops
- Often use **custom allocator**
  - GC at frame boundaries

### Inter-Frame
- Potential to be paged
  - Defines current game state
  - May just want level start
- Size is more **flexible**
  - No. of objects is variable
  - Subsystems may turn on/off
  - User settings may affect
- **OS allocator** okay, but…
  - May want to use a **budget**
Budgeting Memory

Module or Subsystem

Low Memory Behavior

Med Memory Behavior

High Memory Behavior

Design each of these separately
Budgeting Memory

- Module chooses approach
  - Based on priority setting
  - Based on previous attempts

- At each allocation checks
  - Is there enough memory?
  - Conflict with other module?
  - Has my priority changed?

- If it must **downgrade**
  - Release all allocations
  - Start over completely
Budgeting Memory

- Module chooses approach
  - Based on priority setting
  - Based on previous attempts

- At each allocation checks
  - Is there enough memory?
  - Conflict with other module?
  - Has my priority changed?

- If it must **downgrade**
  - Release all allocations
  - Start over completely
Budgeting Memory

- Module chooses approach
  - Based on priority setting
  - Based on previous attempts

- At each allocation checks
  - Is there enough memory?
  - Conflict with other module?
  - Has my priority changed?

- If it must **downgrade**
  - Release all allocations
  - Start over completely

Memory Management
Communication Can Be Sophisticated

- Module (Priority 1) → High Memory
- Module (Priority 2) → Medium Memory
Communication Can Be Sophisticated

Module (Priority 1)

Module (Priority 2)

Hitting Limit

Stop it!

Medium Memory
Communication Can Be Sophisticated

Module (Priority 1)

Module (Priority 2)

Your turn!

Stop it!

Medium Memory

Hitting Limit

Medium Memory
Probably Need a Separate Controller

- Module Memory Controller
- Monitors
- Module (Priority 1) -> High Memory
- Module (Priority 2) -> Medium Memory
Can Budget Speed as Well

Module or Subsystem

Fast Algorithm
Medium Algorithm
Slow Algorithm

Check each *iteration* instead of *allocation*
Dynamic Loading

- Most game data is *spatial*
  - Only load if player nearby
  - Unload as player moves away
  - Minimizes memory used

- Arrange memory in *cells*
  - Different from a memory pool
  - Track player visibility radius
  - Load/unload via outer radius

- **Alternative**: loading zones
  - Elevators in *Mass Effect*
Dynamic Loading in Assassin’s Creed
Implementing Dynamic Loading

- Part of serialization model
  - Level/save file has the cells
  - Cell *addresses* in memory
  - Load/page on demand
- Sort of like virtual memory
  - But paging strategy is spatial
Dynamic Loading Challenges

- **Not same** as virtual memory
  - Objects unloaded do not exist
  - Do not save state when unload
  - Objects loaded are new created

- Can lead to *unexpected states*
  - “Forgetful” NPCs
  - Creative *Assassin’s Creed* kills

- **Workaround**: Global State
  - Track major game conditions
  - **Example**: Guards Alerted
  - Use to load objects in standard, but appropriate, configurations

See Piazza for There is No Spoon
Custom Allocators for Intra-Frame

Pre-allocated Array (called Object Pool)

- Instead of new, get object from array
  - Just reassign all of the fields
  - Use Factory pattern for constructor

- Delete all objects at frame end
  - Just reset free pointer to start
  - Do not worry about freeing mid frame
Custom Allocators for Intra-Frame

Pre-allocated Array     (called Object Pool)

Start                  Free                End

• Instead of `new`, get object from array
  • Just reassign all of the fields
  • Use **Factory pattern** for constructor

• Delete all objects at frame end
  • Just reset free pointer to start
  • Do not worry about freeing mid frame

Easy if only one object type to allocate
Next Time: Language-Specific Details