Lecture 12

Memory Management
Gaming Memory (Current Generation)

- **Playstation 4**
  - 8 GB RAM (unified)

- **X-Box One (X)**
  - 12 GB RAM (unified)
  - 9 GB for games

- **Nintendo Switch**
  - 3 GB RAM (unified)
  - 1 GB only for OS

- **iPhone/iPad**
  - 2 GB RAM (unified)
  - Better than an XBox 360
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 show
  - Need space to uncompress
  - In RAM or graphics card
- Only load when needed
  - Loading is primary I/O operation in AAA games
  - Causes “texture popping”
But My JPEG is only 8 KB!

- Formats often **compressed**
  - JPEG, PNG, GIF
  - But not always TIFF

- Must **uncompress** to show
  - Need space to uncompress
  - In RAM

- Only load when needed
  - Loading is primary I/O operation in AAA games
  - Causes “texture popping”

**Sounds** have a similar problem
Loading Screens

Rebel Mages

WHEREAS THE CIRCLE WAS ESTABLISHED NOT MERELY TO PROTECT THE WORLD FROM MAGES, BUT ALSO TO ALLOW MAGES TO PRACTICE THEIR ART SAFELY AND WITHOUT FEAR, AND,

WHEREAS UNDER LORD SEEKER LAMBERT’S COMMAND, THE TEMPLARS SWEAR TO PROTECT ALL PEOPLE—INCLUDING MAGES—FROM THE HARMFUL EFFECTS OF MAGIC, HAVE INSTEAD PERSECUTED MAGES WITH SUCH BIASED JUDGMENT AS TO WORSEN THE PROBLEMS THEY WERE MEANT TO MITIGATE, AND,

WHEREAS THE RITE OF TRANQUILITY, INTENDED AS A TOOL OF LAST RESORT TO STOP UNCONTROLLED MAGES FROM HURTING THEMSELVES OR OTHERS, HAS INSTEAD BEEN USED FOR PUNITIVE AND POLITICAL PURPOSES TO SILENCE DISSENT AND INHIBIT CIVILIZED DISCOURSE, AND,

WHEREAS ANDRASTE HERSELF INTENDED THE RELATIONSHIP BETWEEN MAGE AND TEMPLAR TO BE ONE OF PRACTITIONER AND PROTECTOR, NOT PRISONER AND
Problems with Asset Loading

- **How to load assets?**
  - May have a lot of assets
  - May have large assets

- **Loading is blocking**
  - Game stops until done
  - Cannot draw or animate

- **May need to unload**
  - Running out of memory
  - Free something first
Problems with Asset Loading

- How to load assets?
  - May have a lot of assets
  - May have large assets

- Loading is **blocking**
  - Game stops until done
  - Cannot draw or animate

- May need to **unload**
  - Running out of memory
  - Free something first

---

Memory Management
Minimal animation/feedback while loading assets
Solution: Asynchronous Loader

Game Thread

- Update
- Draw

Second Thread

- Asset Loader
- Specify Asset
- Notify done

Update and draw simple animations until assets loaded
Solution: Asynchronous Loader

- Also an asset manager
  - Each asset given a key
  - Can access asset by key
  - Works like a hash table
Solution: Asynchronous Loader

Game Thread

Update

Draw

Specify Asset

Notify done

Second Thread

Asset Loader

- Not always a good idea
- May need OpenGL utils
- **Example:** Textures
- Limited to main thread
Alternative: Iterative Loader

Game Thread

Update

Draw

Asset Manager

Initialize

Update

Access
Alternative: Iterative Loader

- Uses a time budget
  - Give set amount of time
  - Do as much as possible
  - Stop until next update

- Better for OpenGL
  - Give time to manager
  - Animate with remainder
  - No resource contention

- LibGDX approach
  - CUGL is asynchronous

Asset Manager

- Initialize
- Update
- Access
Alternative: Iterative Loader

- Uses a time budget
  - Give set amount of time
  - Do as much as possible
  - Stop until next update
- Better for OpenGL
  - Give time to manager
  - Animate with remainder
  - No resource contention
- LibGDX approach
  - CUGL is asynchronous
Aside: When Do We Load Assets?

Choice affects **design** & **ownership** of the asset manager.
Assets Beyond Images

• AAA games have a lot of 3D geometry
  • Vertices for model polygons
  • Physics bodies per polygon
  • Scene graphs for organizing this data

• When are all these objects created?
  • At load time (filling up memory)?
  • Or only when they are needed?

• We need to understand memory better
Traditional Memory Organization

Dedicated to process.
Consists of machine addressable space.

Leverages Virtual Memory

Program Data
Heap
Free Space
Stack

Function parameters
Local variables
Return values

Objects created via new
Allocations with malloc

Program Code
Static Variables

High Address
Low Address

Memory Management
Mobile Memory Organization

Device Memory

- Stack
- Program Data
- Stack
- Program Data
- Stack
- Program Data
- Stack
- Program Data

Heap
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
  - Done only as 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
  - Done only as needed

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

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

All methods in **Application** class
Android State Handling

All methods in **Application** class

Reload memory
Android State Handling

All methods in Application class

Page out memory
CUGL is Simplified Android Model

- **onStartup()**
  - Initialized and now active

- **onSuspend()**
  - Sent to background
  - Gives you chance to save
  - Also time to pause music

- **onResume()**
  - Returns to app to active
  - Allows you to restore state

- **onShutdown()**
  - Stopped & memory freed
CUGL is Simplified Android Model

- **onStartup()**
  - Initialized and now active

- **onSuspend()**
  - Sent to background
  - Gives you chance to save
  - Also time to pause music

- **onResume()**
  - Returns to app to active
  - Allows you to restore state

- **onLowMemory()**
  - Warning memory is low
  - Gives you chance to unload
  - Else app will shut down

- **onShutdown()**
  - Stopped & memory freed
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

<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>- Sprites and caches</td>
</tr>
<tr>
<td>- Temporary objects</td>
<td>- Long-term data structures</td>
</tr>
<tr>
<td>(not necessarily managed)</td>
<td>- Built at start/during frame</td>
</tr>
<tr>
<td><strong>Transient data structures</strong></td>
<td>- Lasts for multiple frames</td>
</tr>
<tr>
<td>- Built at the start of update</td>
<td>- May adjust to data changes</td>
</tr>
<tr>
<td>- Used to process update</td>
<td></td>
</tr>
<tr>
<td>- Can be deleted at end</td>
<td></td>
</tr>
</tbody>
</table>
### Distinguishing Data Types

#### Intra-Frame
- **Local computation**
  - Local variables
    - (managed by compiler)
  - Local objects
    - (not necessarily managed)
- **Transient data structures**
  - Built at the start/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

**Local Variables**
e.g. Collisions

**Object Fields**
e.g. Pathfinding
# 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…
  - Recycle with **free lists**

---

**Topic of Next Lecture**
Advanced: Spatial 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*
**Advanced: Spatial 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*
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*
Spatial Loading in Assassin’s Creed

Memory Management
Implementing Spatial 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

In RAM

On Disk
Spatial 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
Next Time: Low-Level Details