Building your Game

CMSC425.01 Fall 2019

Sit at the same table as last class

Administrivia

- Get started with Unity
  - Install Unity
  - Find references
- Project 1
  - Variation on Roll-A-Ball tutorial
- Today – Questions, rather than activities

Game systems this semester

Processing

- Interactive version of Java
- Used to illustrate concepts
- Not a game engine but has rich libraries
- https://processing.org

Unity

- Full game engine
- Used for projects and assignments
- https://unity3d.com

Today’s questions

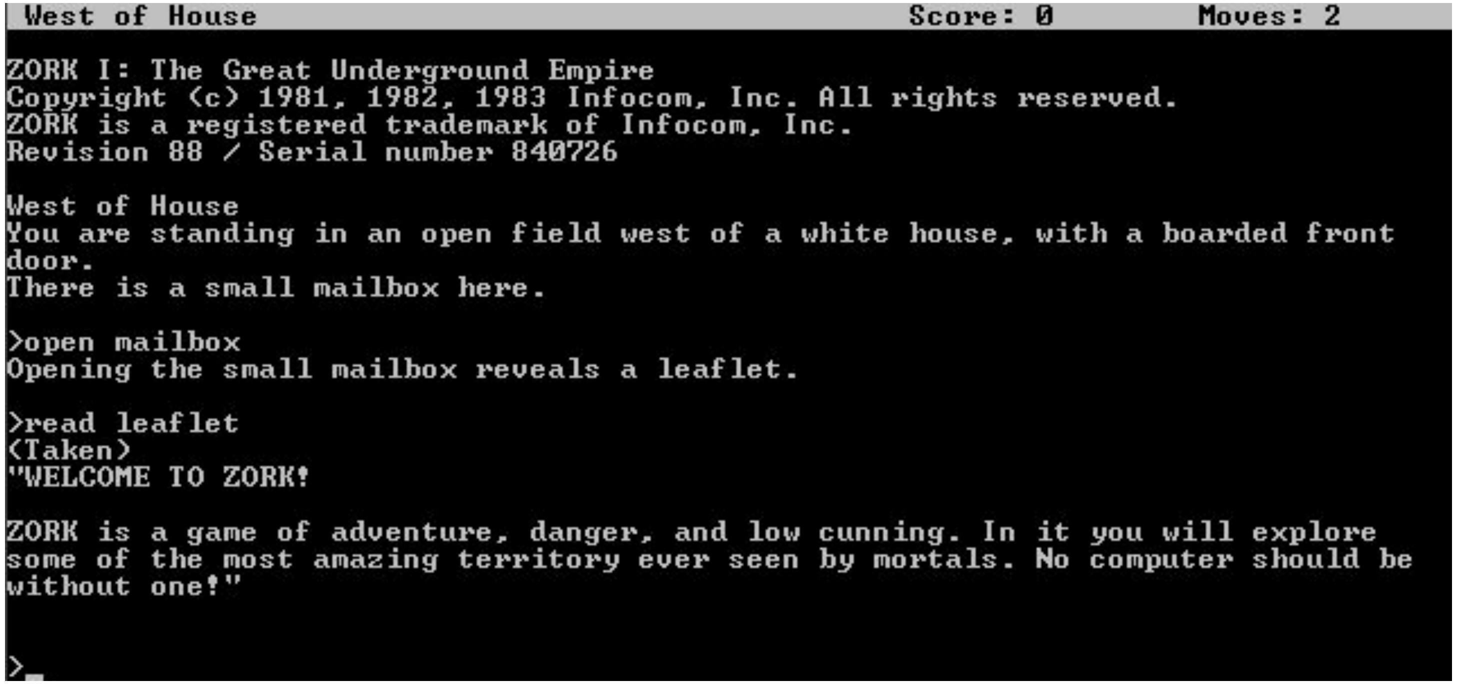
How do you build a
real time, interactive game?

What are the key elements
of a game engine?

Game 1: Zork

- Early text based game
- Text of places and objects
- Simple command language
- Navigation by text

- **Q:** Can we abstract and write a text game engine?



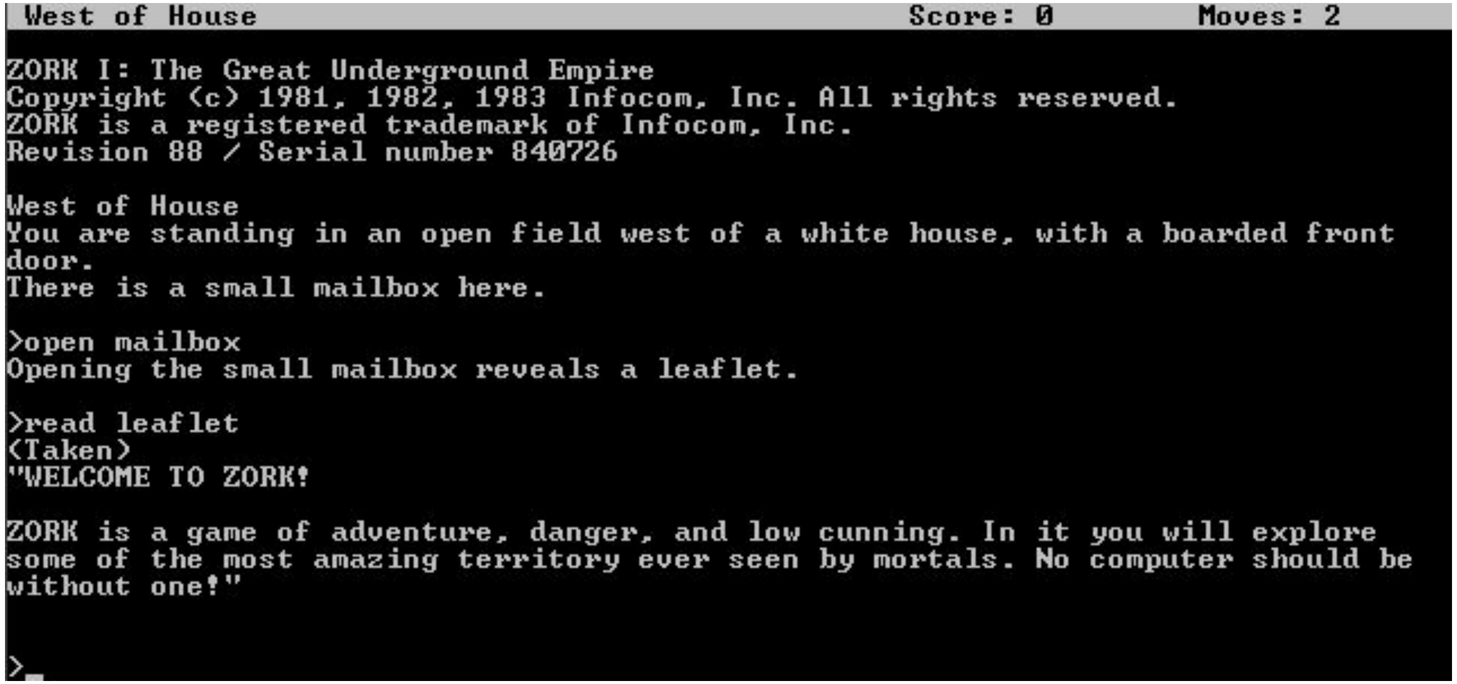
```
 West of House                                          Score: 0         Moves: 2

ZORK I: The Great Underground Empire
Copyright (c) 1981, 1982, 1983 Infocom, Inc. All rights reserved.
ZORK is a registered trademark of Infocom, Inc.
Revision 88 / Serial number 840726

West of House
You are standing in an open field west of a white house, with a boarded front
door.
There is a small mailbox here.

>open mailbox
Opening the small mailbox reveals a leaflet.

>read leaflet
(Taken)
"WELCOME TO ZORK!

ZORK is a game of adventure, danger, and low cunning. In it you will explore
some of the most amazing territory ever seen by mortals. No computer should be
without one!"


>_
```

Game 1: Zork

- Early text based game
- Text of places and objects
- Simple command language
- Navigation by text

- **Q:** Can we abstract and write text game engine?
- Yes

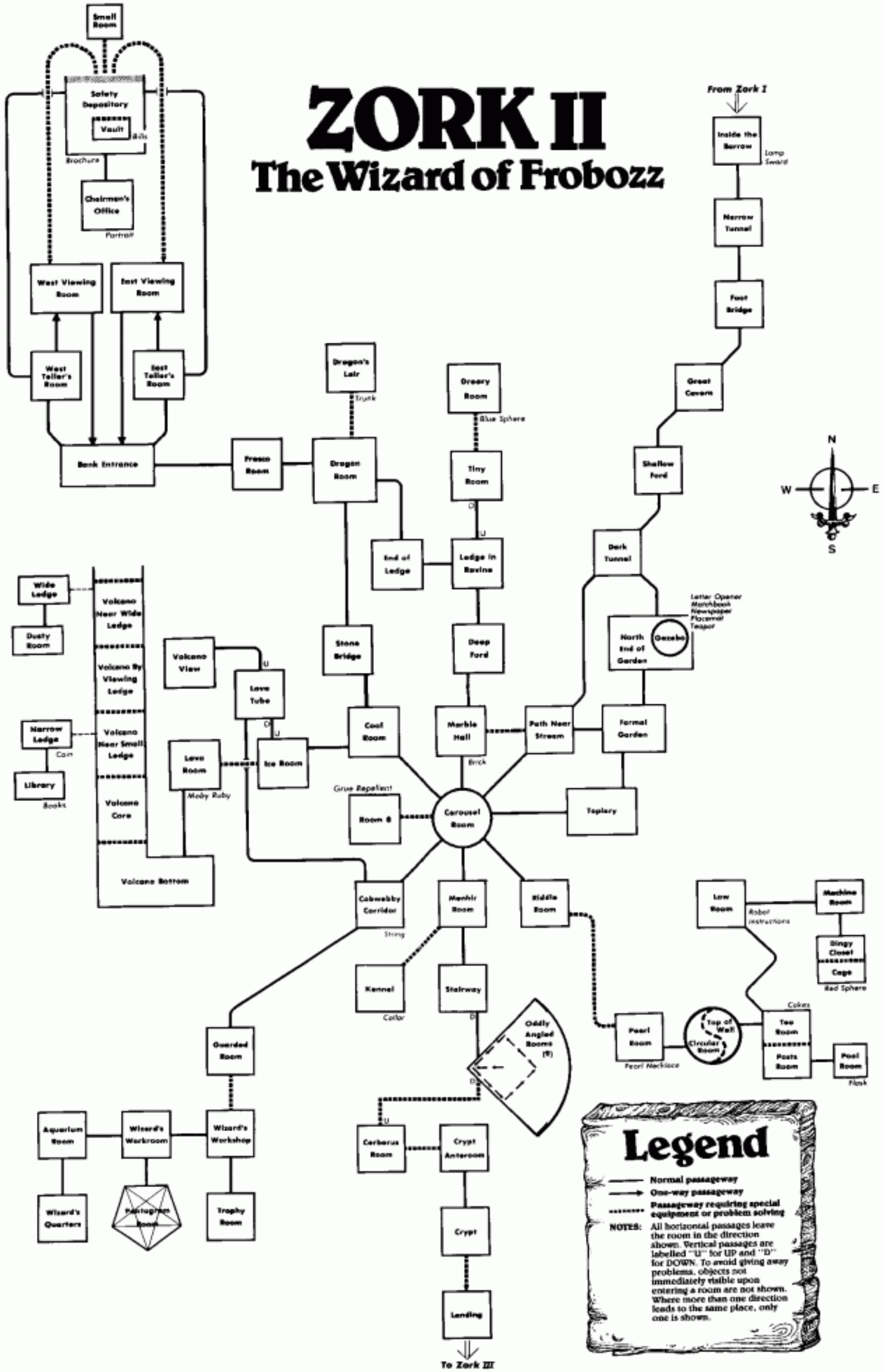
Need:
Code engine
Command parser
Text file descriptions
Graph of locations
User item bag
Read/parse/do loop



```
 West of House                                      Score: 0        Moves: 2

ZORK I: The Great Underground Empire
Copyright (c) 1981, 1982, 1983 Infocom, Inc. All rights reserved.
ZORK is a registered trademark of Infocom, Inc.
Revision 88 / Serial number 840726

West of House
You are standing in an open field west of a white house, with a boarded front
door.
There is a small mailbox here.

>open mailbox
Opening the small mailbox reveals a leaflet.

>read leaflet
(Taken)
"WELCOME TO ZORK!

ZORK is a game of adventure, danger, and low cunning. In it you will explore
some of the most amazing territory ever seen by mortals. No computer should be
without one!"


>_
```

Game 1: Zork

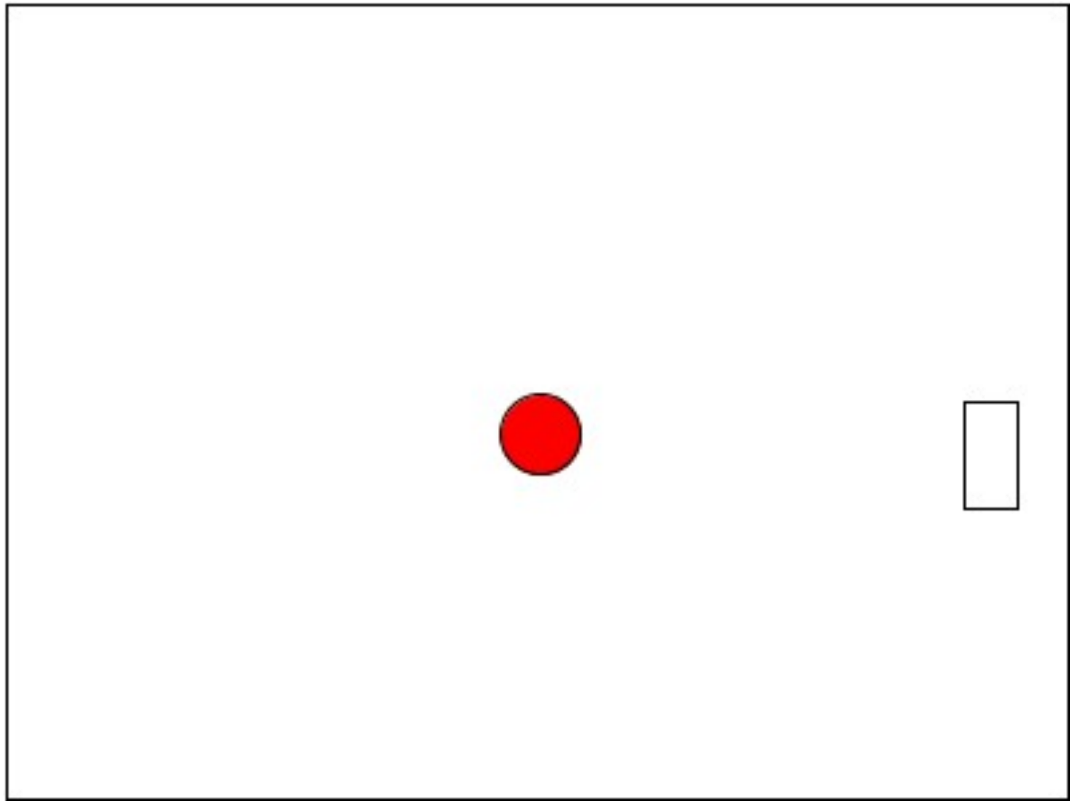
- "Interactive fiction"
- Existing text engines:
  - Adrift, Inform, Quest
- Why care?
- Emphasis on story and language, not glitz
- The skeleton of a game



Game 2: Pong!

- Flatworm of interactive games
- Simple, but complete interactive game

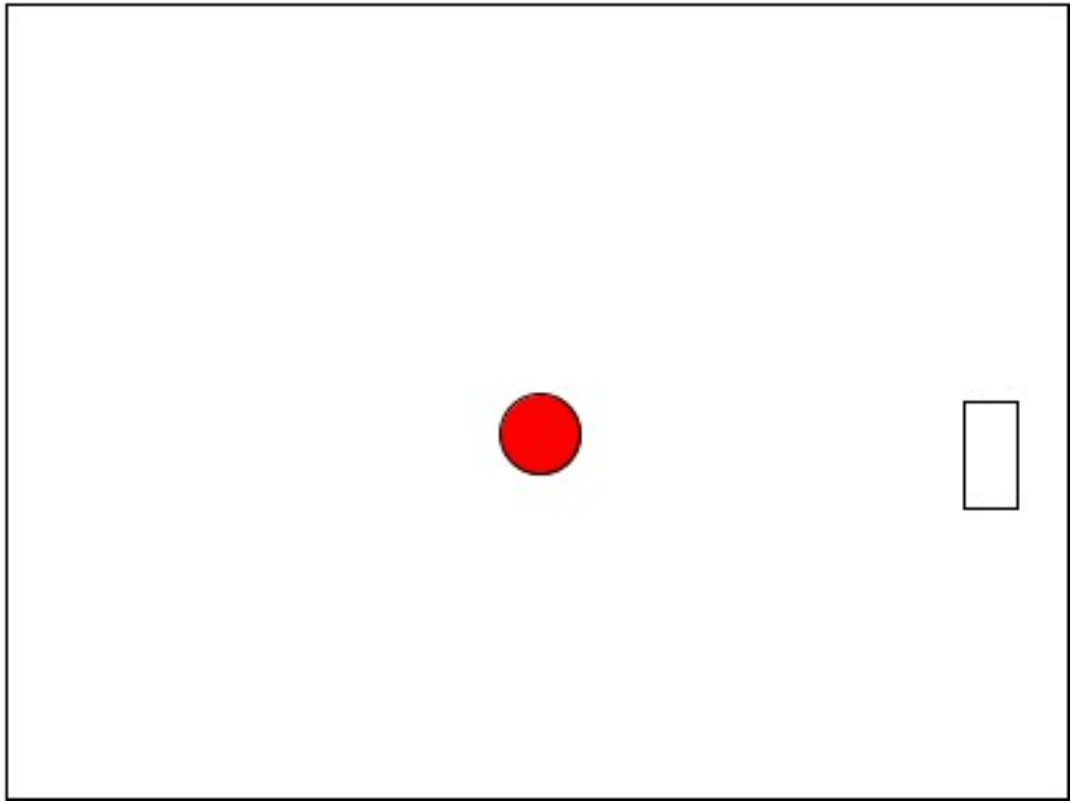
- Example in Processing



Game 2: Pong!

- **Q:** How would you code this?

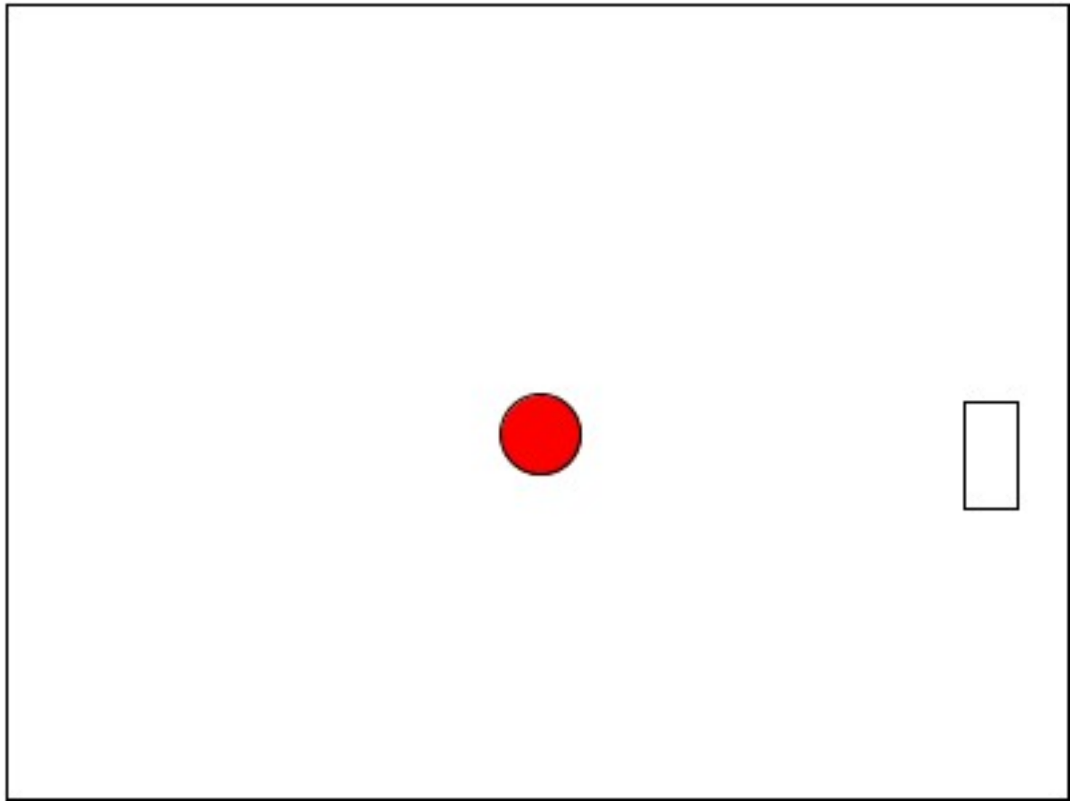
- What elements needed?



Game 2: Pong!

- Basic game loop

```
Initialize
do
   update ball (physics)
   update paddle (user input)
   if (collide) do something
   draw stuff
until done
Clean up
```



Game 2: Pong!

- Basic game loop

```
Initialize
do
    update ball (physics)
    update paddle (user input)
    if (collide) do something
    draw stuff
until done
Clean up
```

- **Update ball**
- Very simple physics

```
x += dx;
y += dy;
```

- Can add acceleration

```
dx += ddx;
```

https://processing.org/examples/bouncingball.html

Game 2: Pong!

- Basic game loop

```
Initialize
do
   update ball (physics)
   update paddle (user input)
   if (collide) do something
   draw stuff
until done
Clean up
```

- **Update paddle**
- Poll device – interrogate

```
if (keyPressed && keyCode == DOWN)
    py = constrain(py+2,0,height);
```

Two form of user/system input

- **Poll device**

- Initiate in your code
- Read fixed memory location updated by system

- **Event driven**

- Initiated by system
- Not under your control
- You write **callback** routine to service event (or **event handler**)

```
void mousePressed() {
  save("image.jpg");
}
```

Basic event program in Processing

```
void setup() {
  size(400,400);
}
void draw() {
}
void mousePressed() {
  ellipse(mouseX,mouseY,20,20);
}
void keyPressed() {
  save("pic.jpg");
}
```

- setup – called once on program start
- draw – called every frame (rate adjustable)
- mousePressed – called once when mouse is pressed
- keyPressed – called once when key is pressed

Game 2: Pong!

- Basic game loop

```
Initialize
do
    update ball (physics)
    update paddle (user input)
    if (collide) do something
    draw stuff
until done
Clean up
```

- **if (collide) do something**
- If hit wall or paddle, take action

```
if (pong.hitLeft()) {
  pong.reverseX();
  }
```

Game 2: Pong!

- Basic game loop

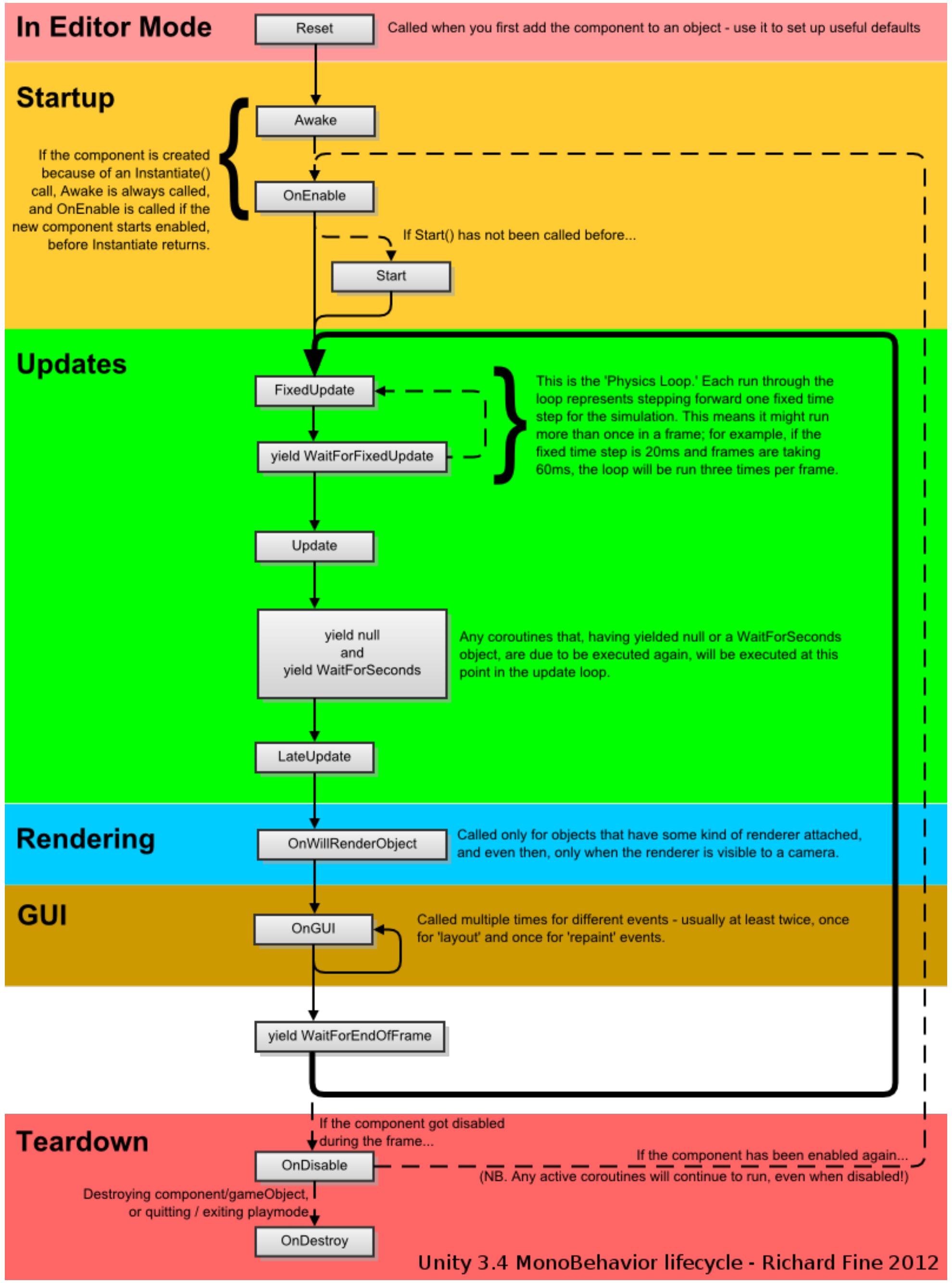
```
Initialize
do
    update ball (physics)
    update paddle (user input)
    if (collide) do something
    draw stuff
until done
Clean up
```

- **draw stuff**
- Draw the arena, paddle and ball

```
// draw ball
color c = color(255,0,0); // red(RGB)
fill(c);
ellipse(x,y,radius,radius);
```

Unity game loop

```
Initialize game
do
  Physics (+collision)
  Input
  Game logic(new)
  Rendering
  GUI rendering
loop
Clean up
```



In Editor Mode

Reset

Called when you first add the component to an object - use it to set up useful defaults

Startup

Awake

OnEnable

If the component is created because of an Instantiate() call, Awake is always called, and OnEnable is called if the new component starts enabled, before Instantiate returns.

If Start() has not been called before...

Start

Updates

FixedUpdate

yield WaitForFixedUpdate

This is the 'Physics Loop.' Each run through the loop represents stepping forward one fixed time step for the simulation. This means it might run more than once in a frame; for example, if the fixed time step is 20ms and frames are taking 60ms, the loop will be run three times per frame.

Update

yield null and yield WaitForSeconds

Any coroutines that, having yielded null or a WaitForSeconds object, are due to be executed again, will be executed at this point in the update loop.

LateUpdate

Rendering

OnWillRenderObject

Called only for objects that have some kind of renderer attached, and even then, only when the renderer is visible to a camera.

GUI

OnGUI

Called multiple times for different events - usually at least twice, once for 'layout' and once for 'repaint' events.

yield WaitForEndOfFrame

Teardown

If the component got disabled during the frame...

OnDisable

If the component has been enabled again...

(NB. Any active coroutines will continue to run, even when disabled!)

Destroying component/gameObject, or quitting / exiting playmode

OnDestroy

Unity 3.4 MonoBehavior lifecycle - Richard Fine 2012

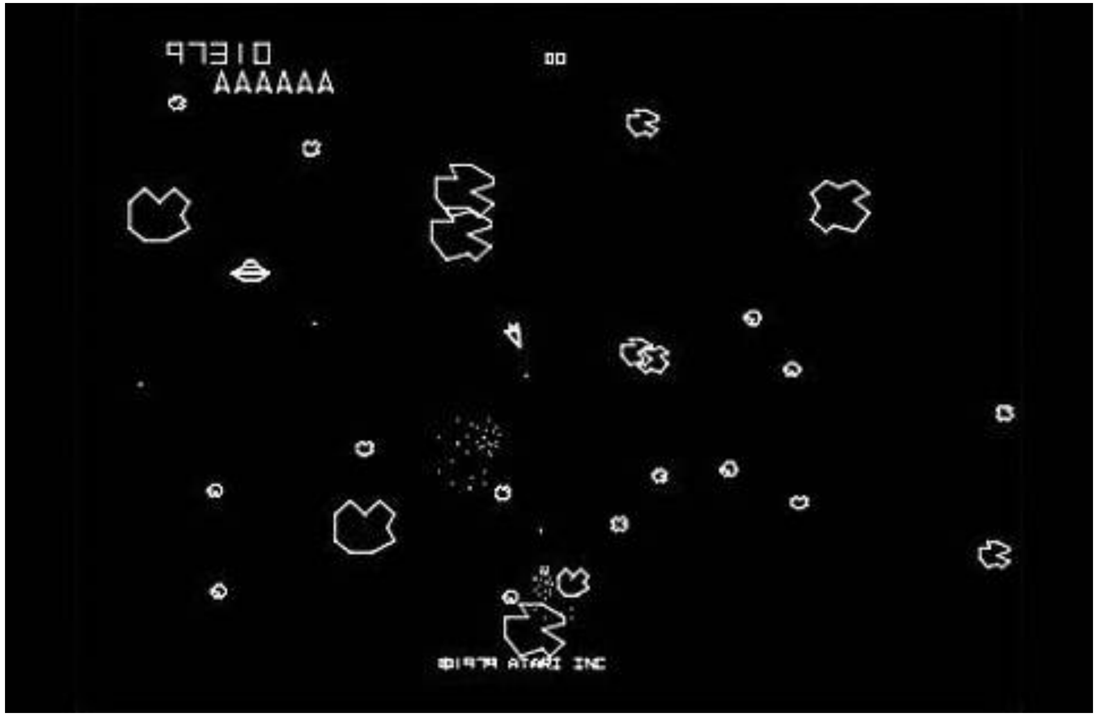
Time!

- Frame time (not constant)
  - Things executed every frame
  - Most important is rendering of scene
- Physics time
  - Steps in physics simulation
  - May run faster than frame time to get physics right (avoid big steps)
- Real time
  - System clock
  - For syncing music, video, other things that need real time

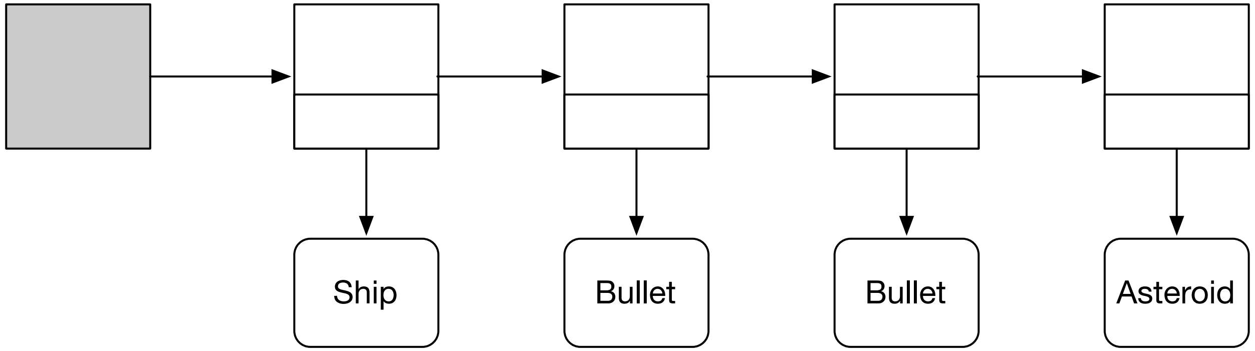
Game 3: Asteroids!

- More objects
  - Ship
  - Bullets
  - Asteroids
  - Enemy ship
  - GUI: Score, remaining ships

- **Q:** How upgrade our Pong game?



Game 3: Asteroids!



- Big change: more objects
  - Ship
  - Bullets
  - Asteroids
  - Enemy ship
  - GUI: Score, remaining ships
- **Q:** How upgrade our Pong game?
  - Object list

- List of game objects
- In loop
  - Update all
  - Interact! (time expensive)
  - Render all

Game 3: Asteroids!

- More objects
  - Ship
  - Bullets
  - Asteroids
  - Enemy ship
  - GUI: Score, remaining ships

- **Q:** How upgrade our Pong game?
  - Object list

Object hierarchy

- **Q:** How design inheritance hierarchy for Asteroid game objects?

Unity – not OOP, but Entity-Component

- More like interfaces in Java

- Bullet
  - Implements Draw (Bullet shape)
  - Implements BallasticMotion
  - Owns Collider component
- Asteroid
  - Implements Draw (Asteroid shape)
  - Implements BallasticMotion
  - Owns Collider component

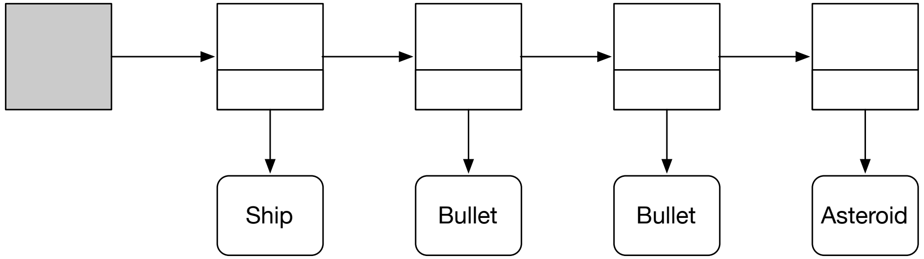
- Ship
  - Implements Draw (Ship shape)
  - Implements UserControlledMotion
  - Owns Collider component
  - Owns Shoot component

- Score
  - Implements Draw (Score shape)
  - No collider component, no motion

Scene graph vs. Object list

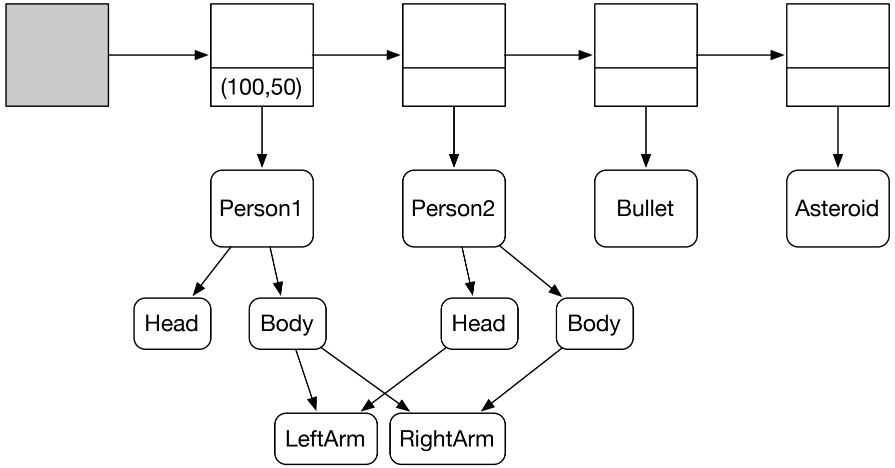
Object list in Asteroids

- All objects are simple, no articulated motion



Scene graph

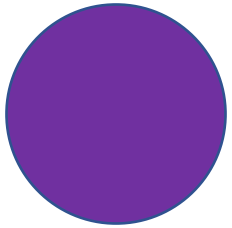
- Directed graph, compound objects
- May share subparts
- Subparts have own displacements



Model View (MV) and rendering

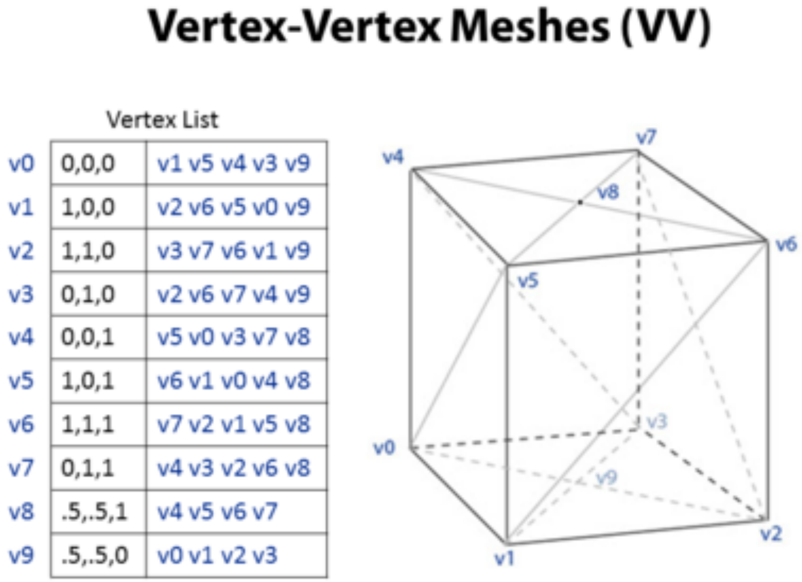
- ***Model* of object stored**
- Circle: (x,y,r,color)
  - Location x,y
  - Radius r

- ***View* of object rendered**



Model View (MV) and rendering

- ***Model* of object stored**
- In 3D
    - Store list of vertices and polygons

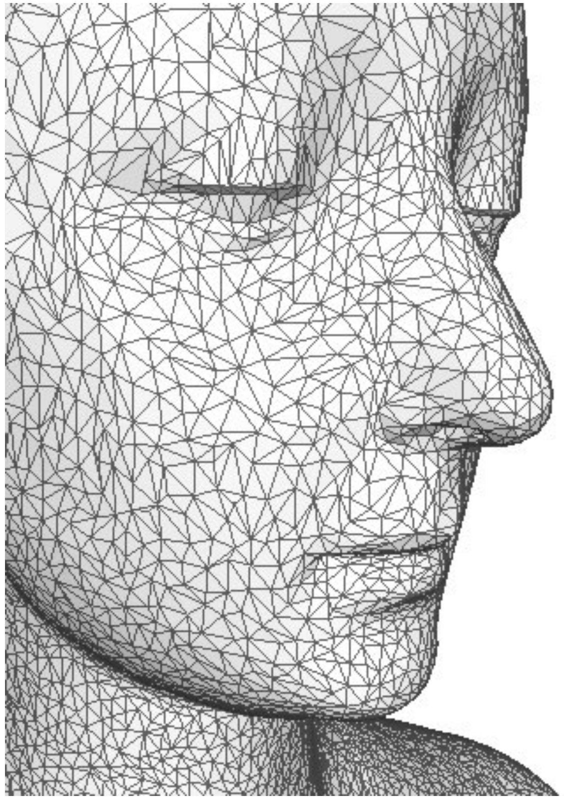


Vertex-Vertex Meshes (VV)

Vertex List

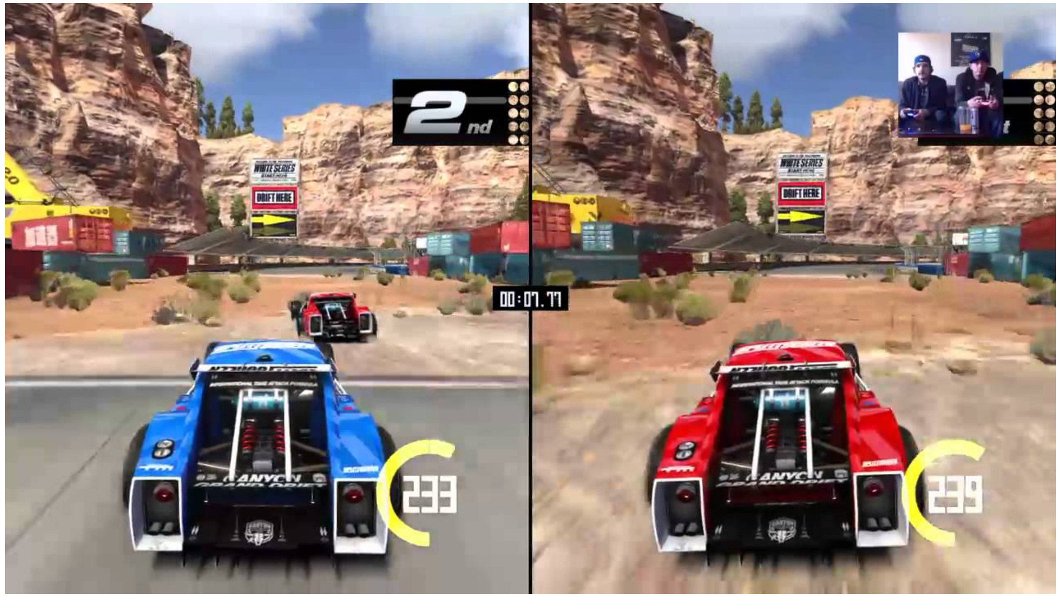
v0	0,0,0	v1 v5 v4 v3 v9
v1	1,0,0	v2 v6 v5 v0 v9
v2	1,1,0	v3 v7 v6 v1 v9
v3	0,1,0	v2 v6 v7 v4 v9
v4	0,0,1	v5 v0 v3 v7 v8
v5	1,0,1	v6 v1 v0 v4 v8
v6	1,1,1	v7 v2 v1 v5 v8
v7	0,1,1	v4 v3 v2 v6 v8
v8	.5,.5,1	v4 v5 v6 v7
v9	.5,.5,0	v0 v1 v2 v3

- ***View* of object rendered**
- Render object in 3D (using GPU)

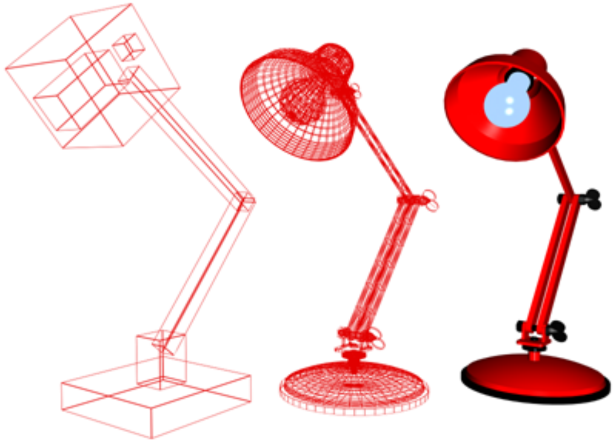


Independence of model and view

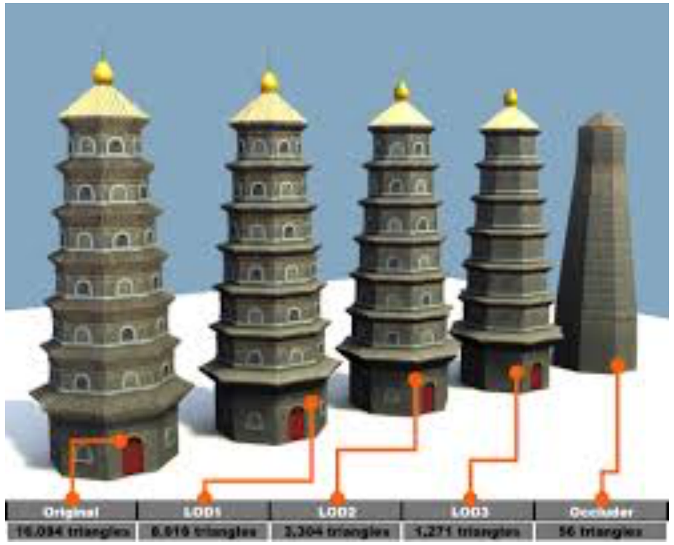
- ***Can render 3D model***
- From different viewpoints
  - Eg, split screen simultaneously
  - Change of perspective



- In different ways

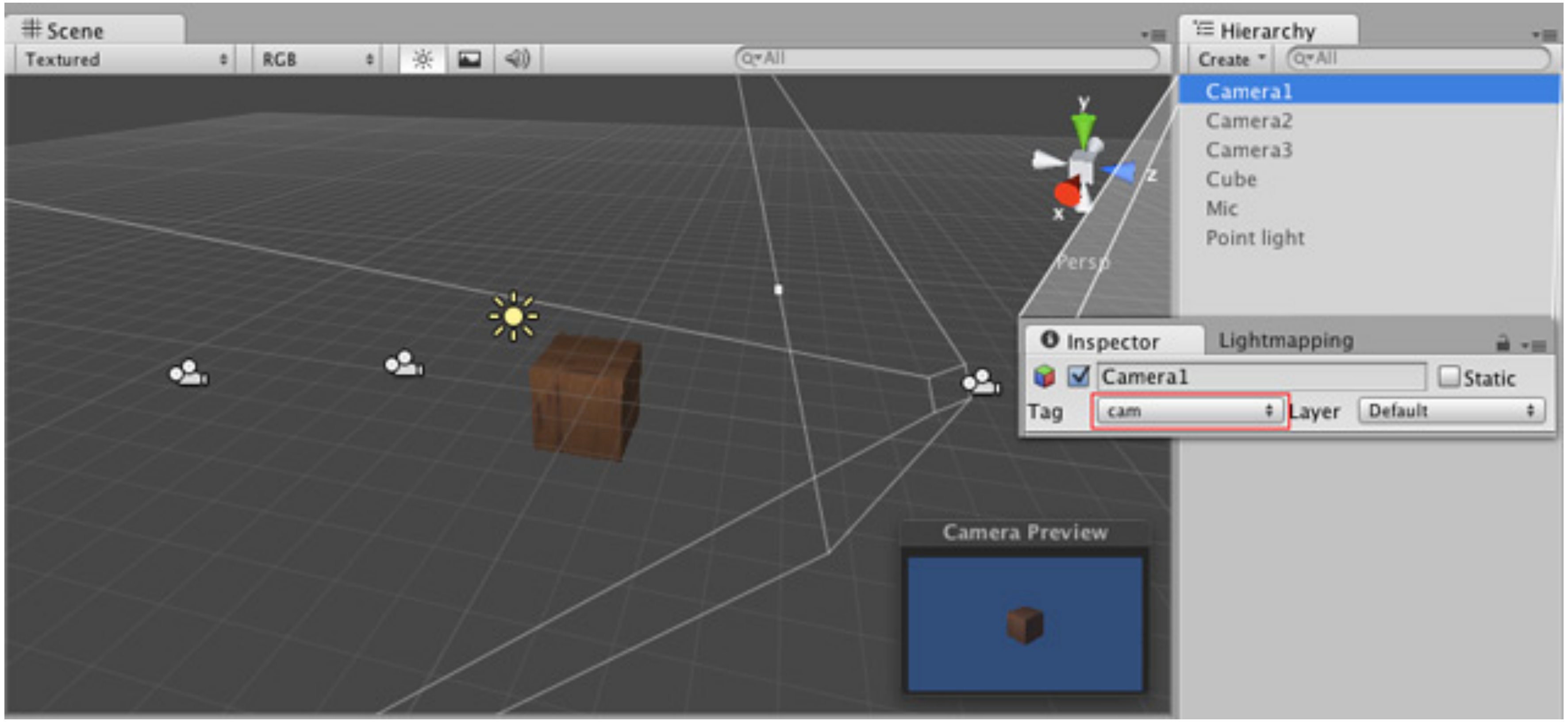


- At different levels of detail
- (far objects, less LOD)



Not all game objects are rendered (visible)

- Cameras/lights can move & behave but aren't rendered in game
- Model, no view except in mock up

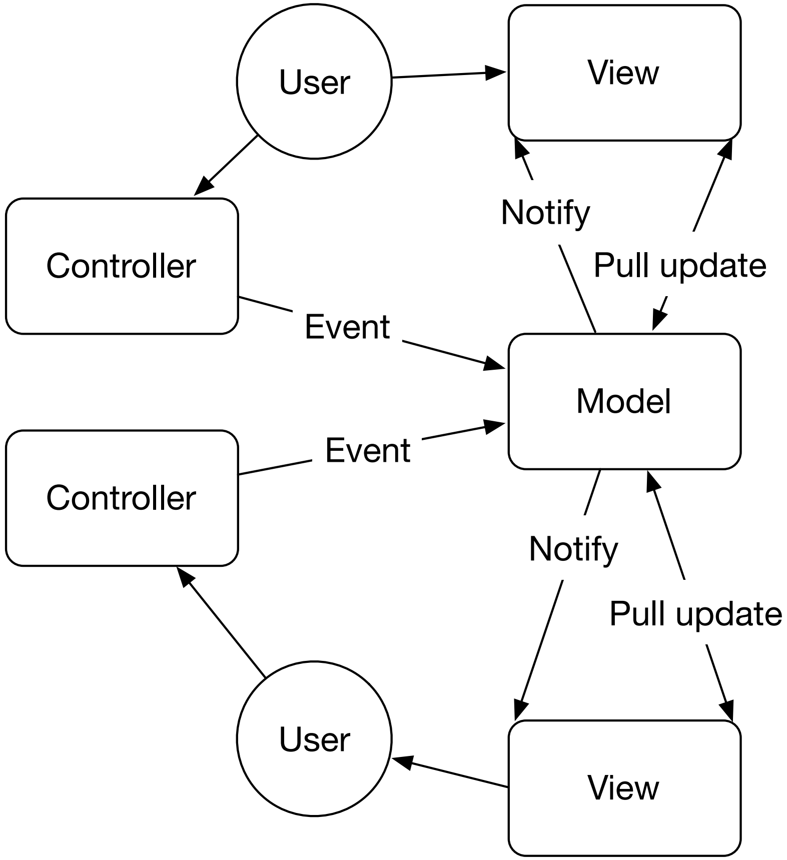


☑ Camera1 □ Static

Tag cam Layer Default

Model View Controller (MVC) program

- Multi-user game
- Shared Model/Database
- Different Views
- Coordinated controllers
- (BTW – this could be Accounting system, any multi-user app)



Rendering!

- Convert 3D polygonal model to 2D image

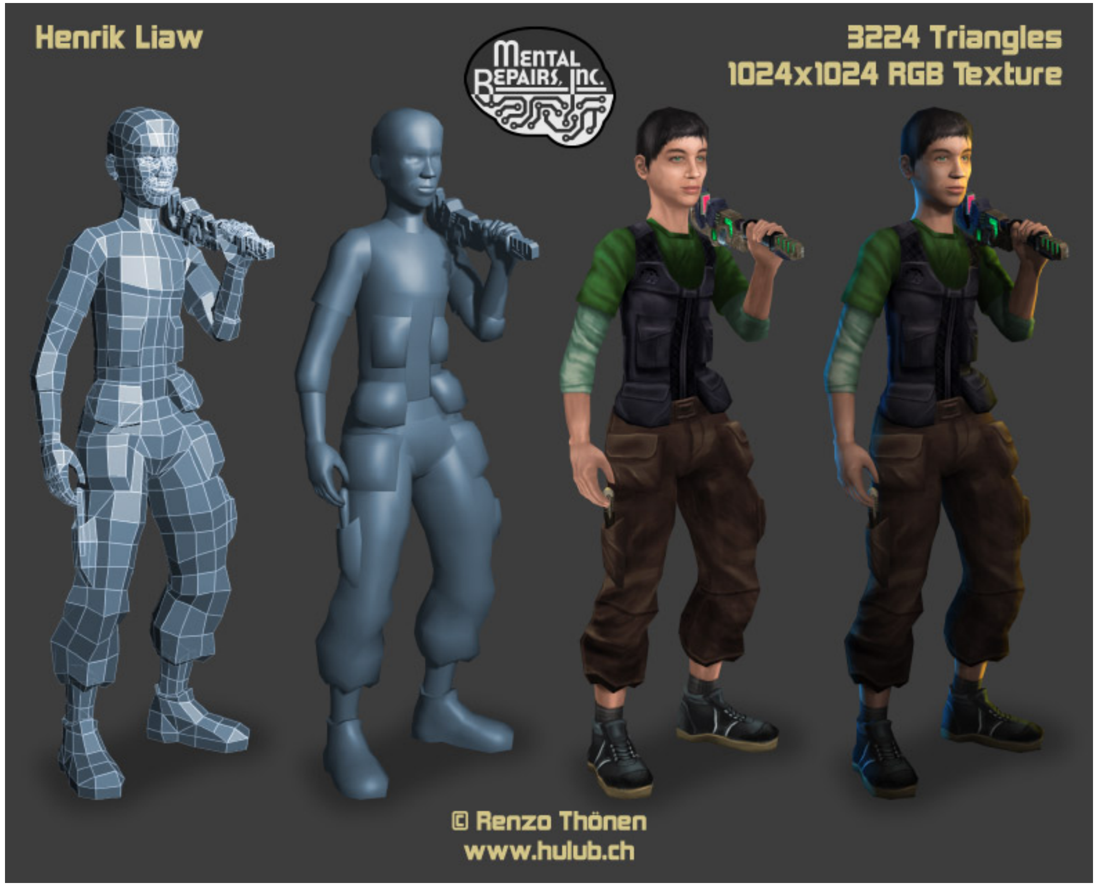
- Do it well
- Do it cheaply
- Do it fast

- How?

Rendering

- Step 1: Elements of model

- Geometry: polygonal mesh
  - 3D points
  - Topology (graph structure)
- Appearance: color
  - Texture
  - Procedural shader
- Articulation/motions

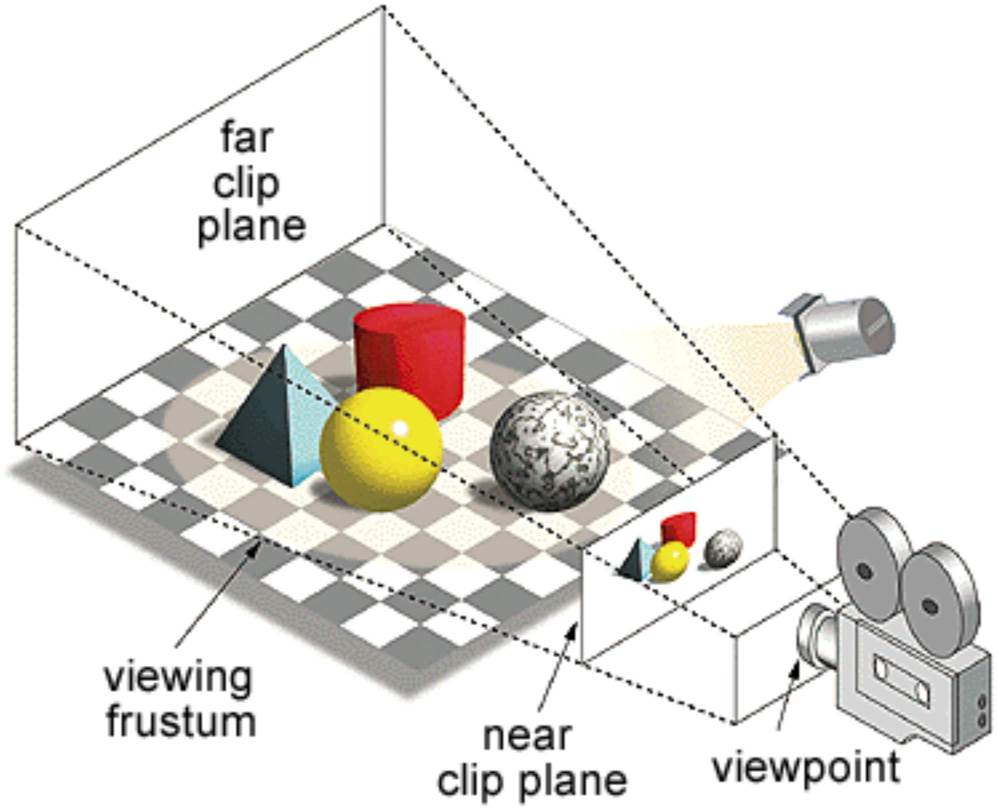


Rendering

- Step 2: Scene elements

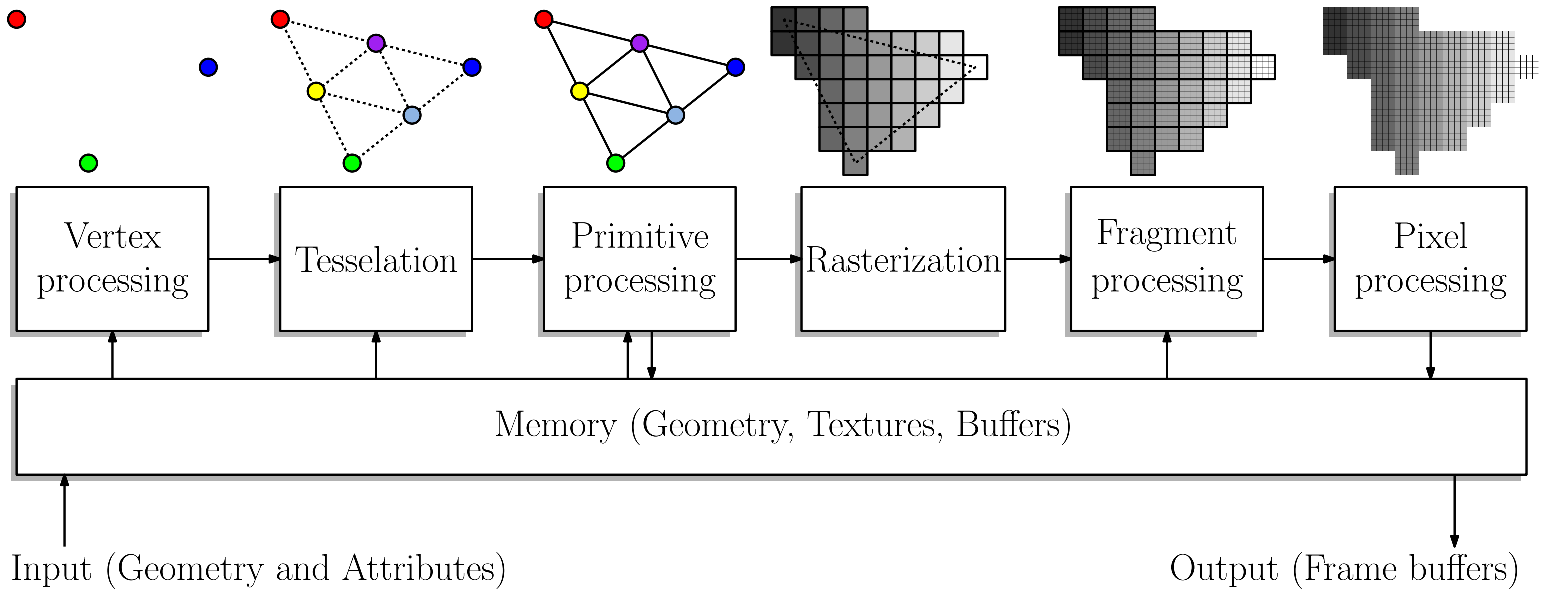
- Figures plus

- Camera
- Lights
- Skybox



Rendering

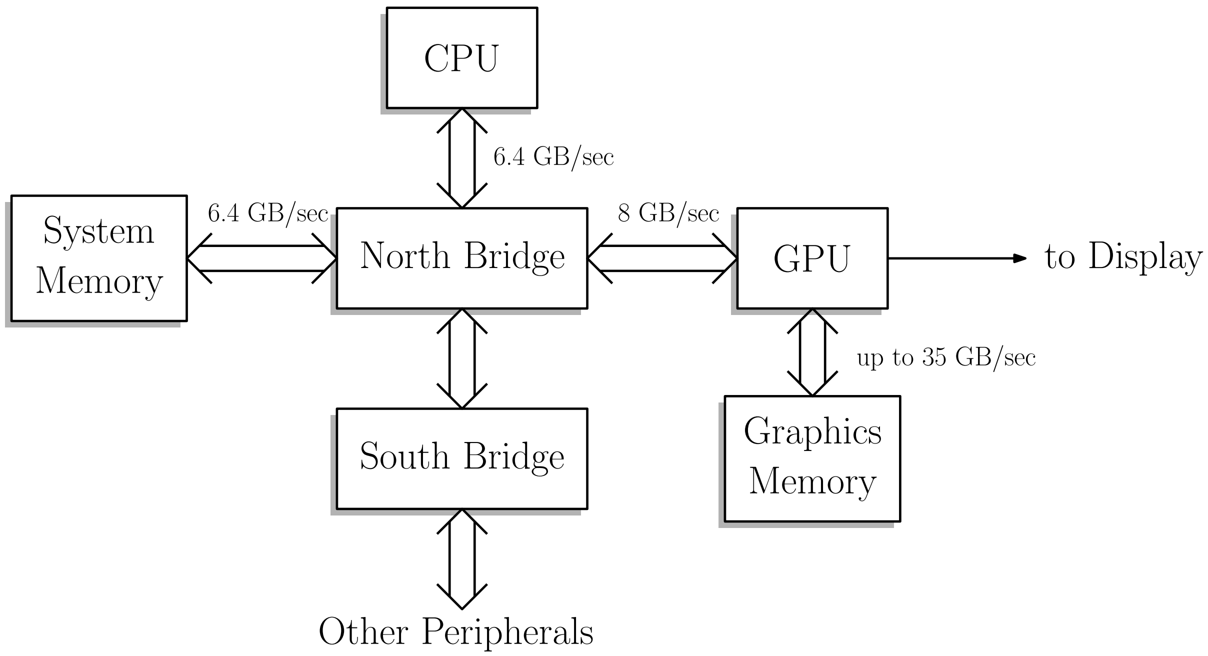
- Step 3: software graphics pipeline
- In 3D compute interaction between lights, model camera (math!)
- In 2D do low level rendering to display triangles with color



Rendering

- Step 4: hardware pipeline

- Push (immense data) to GPU
- Use dedicated bus (north bridge)
- Use GPU memory to pre-load textures, models, send only recent motion data



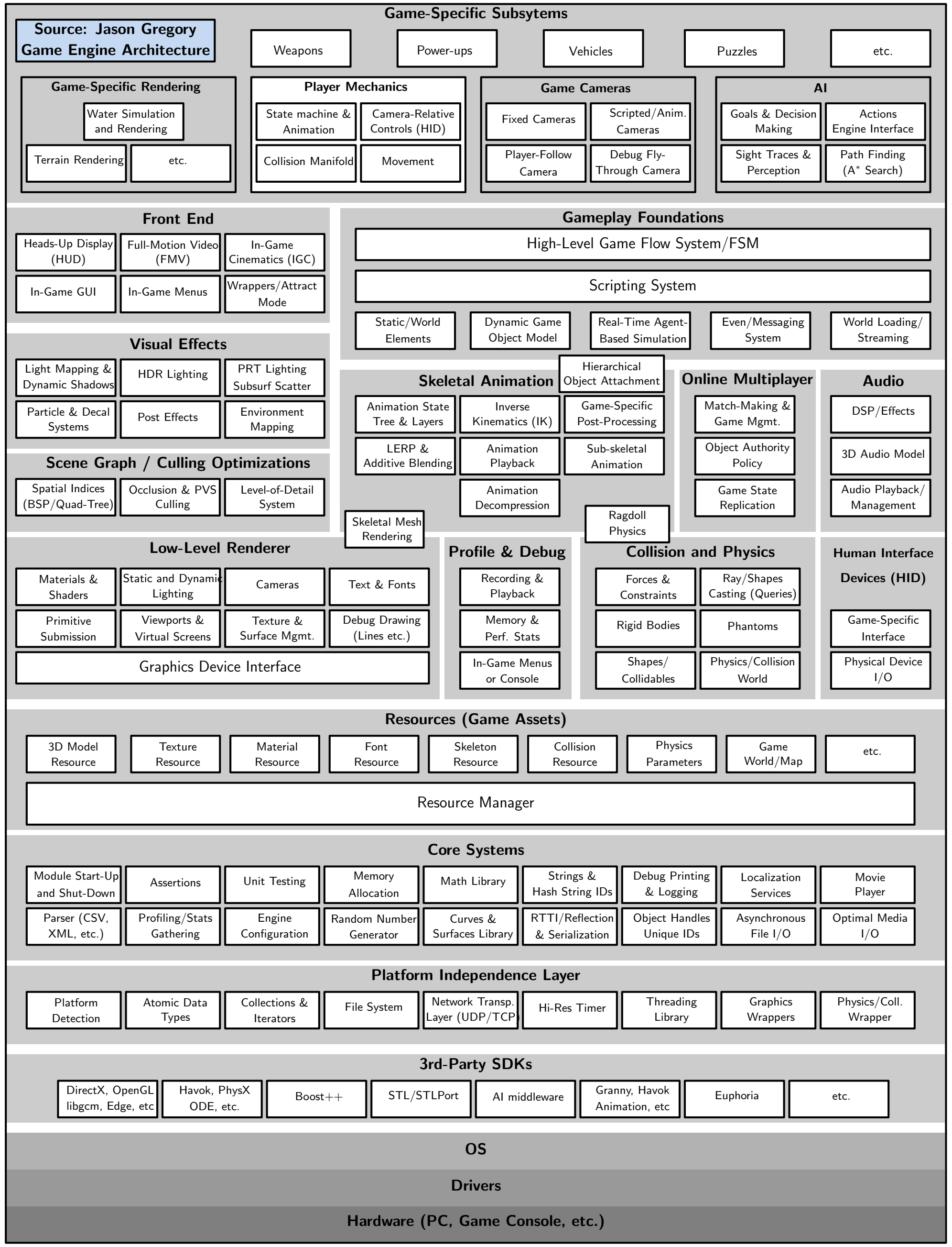
Summary

- After today you should be able:

1) Explain the separation between game engine and game logic, assets
2) Outline and explain a basic game loop and its stages
3) Distinguish polling and events for user (network, game) input
4) Read and explain an event driven real time graphics program
5) Explain the different clocks used in a game
6) Describe an object list, and a scene graph, for game objects
7) Differentiate between OOP and Entity-Component systems
8) Explain the elements of Model-View-Controller systems (MVC)

Putting it all together

- Key elements of typical game engine?
- Lots of parts to full system!
- Don't memorize diagram, but get high level view
  - Gameplay (high level game loop)
  - Model/scene graph+asset management
  - Physics/collision
  - Player/network interface
  - GUI
  - Rendering



Game-Specific Subsytems

Source: Jason Gregory
Game Engine Architecture

Weapons | Power-ups | Vehicles | Puzzles | etc.

Game-Specific Rendering: Water Simulation and Rendering; Terrain Rendering; etc.

Player Mechanics: State machine & Animation; Camera-Relative Controls (HID); Collision Manifold; Movement

Game Cameras: Fixed Cameras; Scripted/Anim. Cameras; Player-Follow Camera; Debug Fly-Through Camera

AI: Goals & Decision Making; Actions Engine Interface; Sight Traces & Perception; Path Finding (A* Search)

Front End: Heads-Up Display (HUD); Full-Motion Video (FMV); In-Game Cinematics (IGC); In-Game GUI; In-Game Menus; Wrappers/Attract Mode

Gameplay Foundations: High-Level Game Flow System/FSM; Scripting System; Static/World Elements; Dynamic Game Object Model; Real-Time Agent-Based Simulation; Even/Messaging System; World Loading/Streaming

Visual Effects: Light Mapping & Dynamic Shadows; HDR Lighting; PRT Lighting Subsurf Scatter; Particle & Decal Systems; Post Effects; Environment Mapping

Scene Graph / Culling Optimizations: Spatial Indices (BSP/Quad-Tree); Occlusion & PVS Culling; Level-of-Detail System

Skeletal Animation: Hierarchical Object Attachment; Animation State Tree & Layers; Inverse Kinematics (IK); Game-Specific Post-Processing; LERP & Additive Blending; Animation Playback; Sub-skeletal Animation; Animation Decompression; Skeletal Mesh Rendering; Ragdoll Physics

Online Multiplayer: Match-Making & Game Mgmt.; Object Authority Policy; Game State Replication

Audio: DSP/Effects; 3D Audio Model; Audio Playback/Management

Low-Level Renderer: Materials & Shaders; Static and Dynamic Lighting; Cameras; Text & Fonts; Primitive Submission; Viewports & Virtual Screens; Texture & Surface Mgmt.; Debug Drawing (Lines etc.); Graphics Device Interface

Profile & Debug: Recording & Playback; Memory & Perf. Stats; In-Game Menus or Console

Collision and Physics: Forces & Constraints; Ray/Shapes Casting (Queries); Rigid Bodies; Phantoms; Shapes/Collidables; Physics/Collision World

Human Interface Devices (HID): Game-Specific Interface; Physical Device I/O

Resources (Game Assets): 3D Model Resource; Texture Resource; Material Resource; Font Resource; Skeleton Resource; Collision Resource; Physics Parameters; Game World/Map; etc.; Resource Manager

Core Systems: Module Start-Up and Shut-Down; Assertions; Unit Testing; Memory Allocation; Math Library; Strings & Hash String IDs; Debug Printing & Logging; Localization Services; Movie Player; Parser (CSV, XML, etc.); Profiling/Stats Gathering; Engine Configuration; Random Number Generator; Curves & Surfaces Library; RTTI/Reflection & Serialization; Object Handles Unique IDs; Asynchronous File I/O; Optimal Media I/O

Platform Independence Layer: Platform Detection; Atomic Data Types; Collections & Iterators; File System; Network Transp. Layer (UDP/TCP); Hi-Res Timer; Threading Library; Graphics Wrappers; Physics/Coll. Wrapper

3rd-Party SDKs: DirectX, OpenGL libgcm, Edge, etc; Havok, PhysX ODE, etc.; Boost++; STL/STLPort; AI middleware; Granny, Havok Animation, etc; Euphoria; etc.

OS

Drivers

Hardware (PC, Game Console, etc.)

Readings

- David Mount's lectures
- This class:
- **"Computer Game and Graphics System Architectures"**
- Next class:
- "**Intro to Unity**"

- Pong code on web site – optional to read or run, but Processing is fun

- Other readings

- Unity manual
- Michael Kissner Gamasutra

Next: Moving on to Unity

- Will refine and explain these ideas through the semester

- You should
  - Install Unity
  - Do Roll-a-Ball tutorial
  - Start working on Project 1

- Ideas from today apply Unity