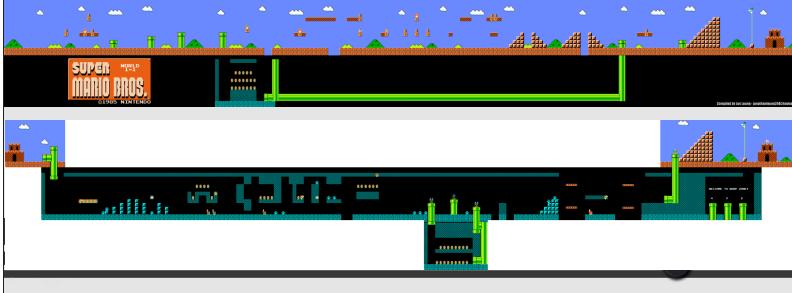


BASIC GAME ENGINE



Early Game Engine

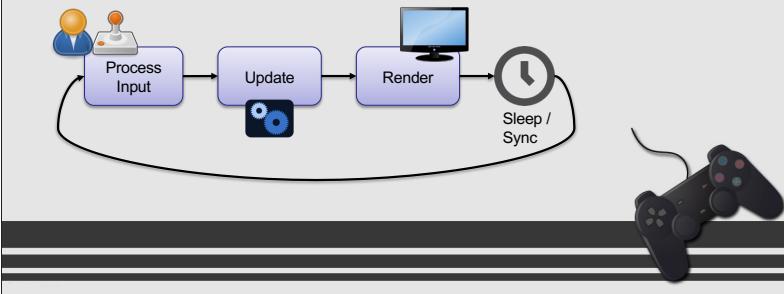
- Computational kernel that runs the game
- Operates on internal data models
- Some level of reusability (at least within the game, e.g. levels)



Super Mario Bros. 1st 2 levels tile sets

The Kernel

- Frame-based
- Closed-loop where player is part of the overall system



We make assumption that player is only acting/considering info within the system and isn't influenced by outside information



GAME ENGINES HUMAN PERCEPTION



Human Perception

- Visual Stimulus Reaction: 0.2s
- Auditory Stimulus Reaction: 0.16s

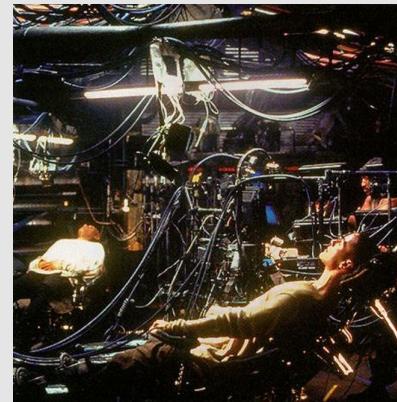


Similar for self-stimulus (e.g. move physically and expect world view to change in 0.2s)

Render bug from one of the Assassins Creed games

Fool the Senses

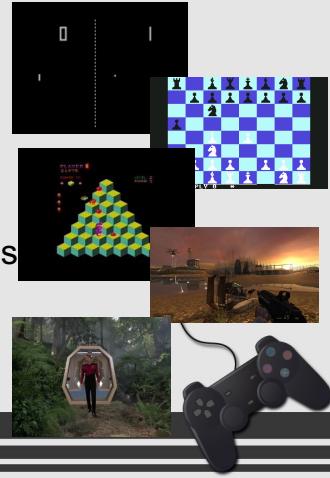
- Sensory Illusions
- Vision
- Hearing
- Other senses?



The Matrix

Goals in “Fooling the Senses”

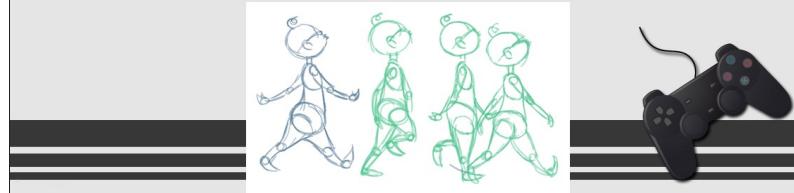
- Object Representation
 - Permanence
- Relationships
 - Spatial Constraints
- Cause and Effect
- Real world representations
 - Objects
 - physics
- Immersive Simulation



Pong, Sargon II, Qbert, Half Life 2, Holodeck

Animation

- **Consecutive similar images appear to be persistent shapes that move/change if image change rate is “fast enough”**
- 10 FPS enough for sense of spatial presence, Movies (from camera) 24 FPS, Games aim for ≥ 30 FPS



Flicker fusion threshold > 50 Hz (consider fluorescent lighting)

About twice the 0.2 visual stimulus rate, consecutive similar images blend together into a continuous perception of motion



GAME ENGINES SIMULATION CONCEPTS



Pursuit of Realism



Models of Human Observable Macroscopic Phenomenon

versus

Unified Simulation



What is more realistic: Minecraft or GTA5?

Models of Human Observable Macroscopic Phenomenon

Often disparate models that must be synchronized

(e.g. surface rendering + digital audio + Newtonian physics + animation system)

Limited interaction of elements within the simulation

Illusions

Computation dedicated to only the observable phenomenon

Storytellers have more direct control of consequences

Unified Simulation

Emergent observable phenomenon

High computational costs

There is no Theory of Everything (yet)

Would be difficult for a storyteller to control or constrain

A Matter of Minutes



A Matter of Minutes episode from the New Twilight Zone:
https://en.wikipedia.org/wiki/A_Matter_of_Minutes

https://www.youtube.com/watch?v=gXMw_ECmcPw

Or

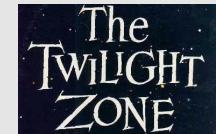
<https://www.youtube.com/watch?v=e9XFRfeGBVI>

Or

<https://www.dailymotion.com/video/x7xgrp>

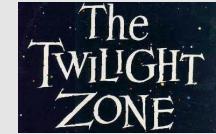
Simulation Concepts?

- What are some of the things that the Twilight Zone got right?



Simulation Concepts?

- What are some of the things that the Twilight Zone got right?
- Simulation of time slices
- Rebuild for every time slice
- Only build what you need
- Glitch out of game world
- Objects disappearing or pop-in/pop-up



A Frame is Frozen in Time

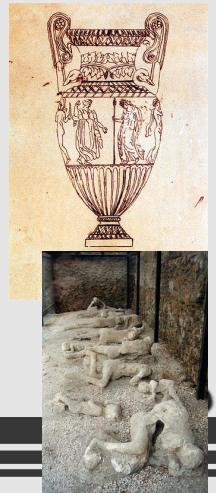
- Almost everything in a frame update shares a common reference time (from the start of frame processing)
- Visuals rebuilt from scratch
- Only simulate what is needed to create the illusion



https://en.wikipedia.org/wiki/High-speed_photography

Why is frame time fixed/frozen?

- Consistent output (especially visual) as all game objects animate by the same amount of time
- Potentially avoid race conditions in logic that dictates object interactions

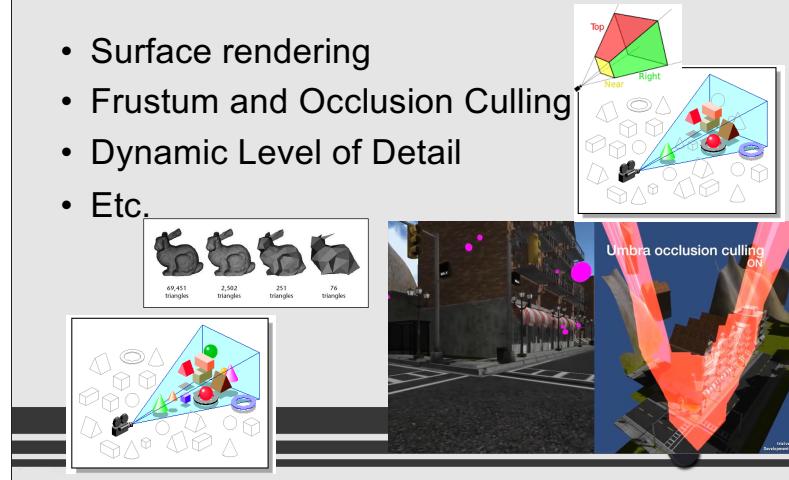


https://en.wikipedia.org/wiki/Ode_on_a_Grecian_Urn

https://en.wikipedia.org/wiki/Eruption_of_Mount_Vesuvius_in_79_AD

Only Simulate What is Needed

- Surface rendering
- Frustum and Occlusion Culling
- Dynamic Level of Detail
- Etc.



Umbra's culling demo. Used in Unity

Note that Frustum Culling isn't only for reducing render complexity. In fact it is primarily for avoiding rendering anomalies during projection

A Tree in Hyrule Forest...



"If a tree falls in a forest and no one is around to hear it, does it make a sound?"

Simulation analog: If a tree falls in Hyrule Forest and no one is around to hear it, does it even exist?



A game tends to only simulate what can be seen in the given moment. But this can create problems and be at odds with things like object permanence, etc.

Zelda: Breath of the Wild

Side Effects of Detail Management

- Pop-in/Pop-up of objects and characters
- Un-activated characters
- Characters teleport when you aren't looking/out of range
- Glitch out of levels
- Disappearing objects
- See inside solids



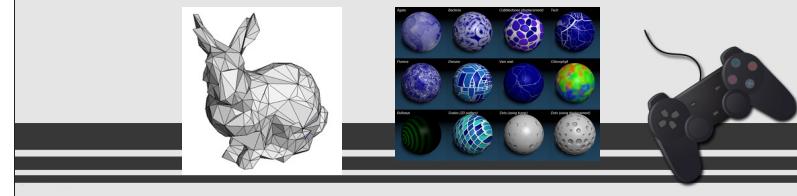
Zelda: OoT

Video game with waterfall pop up – Maybe Far cry?

Louis XIV and Gardens of Versailles

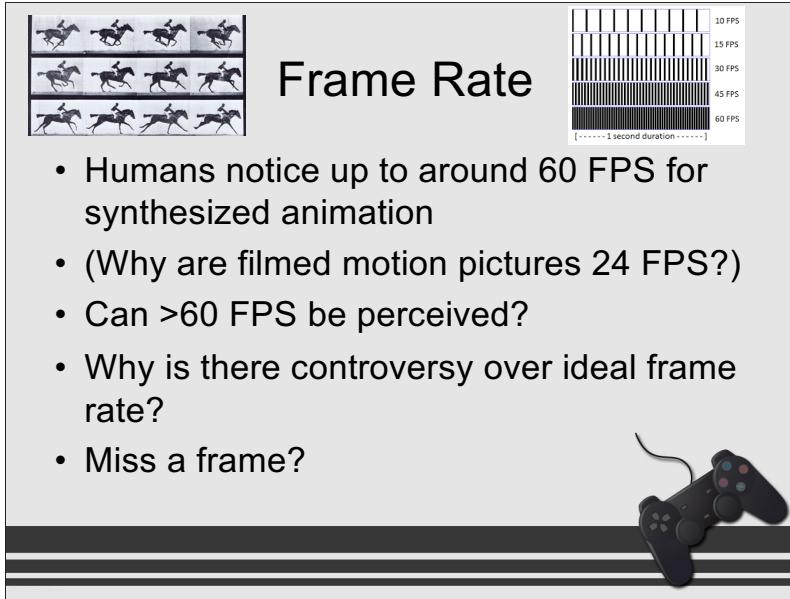
Game Simulation: Focus on Rendering Surfaces

- Only model what is seen
- Model surfaces and surface material light interactions
- What's inside? Nothing!
- Limited or incomplete modeling of matter
- Susceptible to rendering artifacts



Examples beyond just looking at surface: Subsurface scattering, volumetric rendering

Artifacts come from inconsistent physics modeling related to camera position (and clipping planes) and object positions



- Humans notice up to around 60 FPS for synthesized animation
- (Why are filmed motion pictures 24 FPS?)
- Can >60 FPS be perceived?
- Why is there controversy over ideal frame rate?
- Miss a frame?

It may be possible to devise experiments where higher framerates can be noticed by game player, perhaps indirectly by looking for motion blur, display flicker, aggregate assessment of game play performance (confounded by input sampling, LCD refresh/ghosting, vsync misses, interaction of pipeline and/or cache induced latency with target rate, etc.)

Some interesting comments from John Carmack @ Oculus <http://oculus rift-blog.com/john-carmacks-message-of-latency/682/>

Demo App

<https://frames-per-second.appspot.com>

GAME ENGINES FRAME RATE DEMO



[Demo App](#)

<https://frames-per-second.appspot.com>

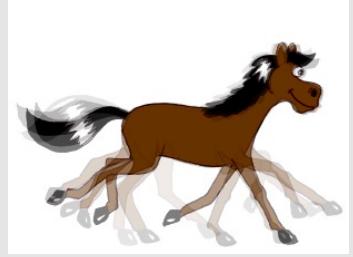


GAME ENGINES SIMULATION CONCEPTS (CONTINUED)



Why Frames?

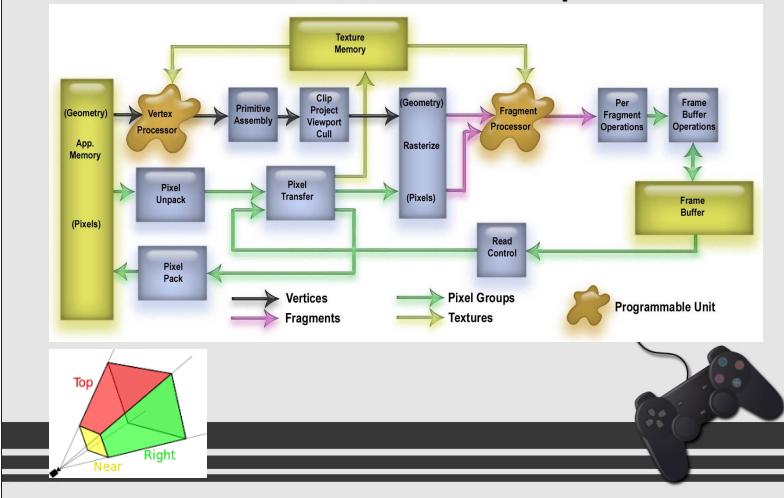
- Intuitive
- Built on what we know (geometric methods)
- Entrenchment (classic celluloid animation, algorithms, application of sampling theory, raster display, etc.)
- Compatible with human perception
- Audio frames?



Digital audio is continuous and is instead implemented as a circular buffer fed incrementally from the main loop. Audio can only be updated at the frame rate though and incurs a latency related to the size of the circular buffer

https://en.wikipedia.org/wiki/Onion_skinning

Canonical Render Pipeline



Geometric raster based rendering

Massive parallelization

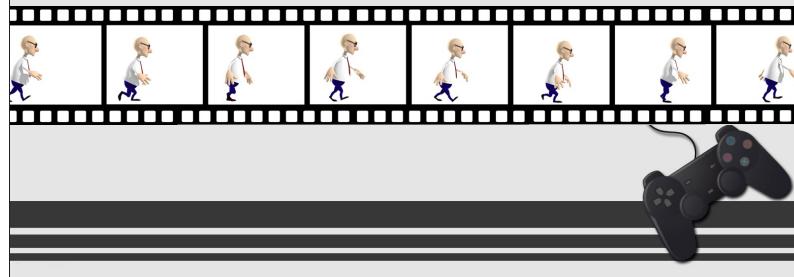


GAME ENGINES SIDE QUESTS: BREAKING WITH TRADITION

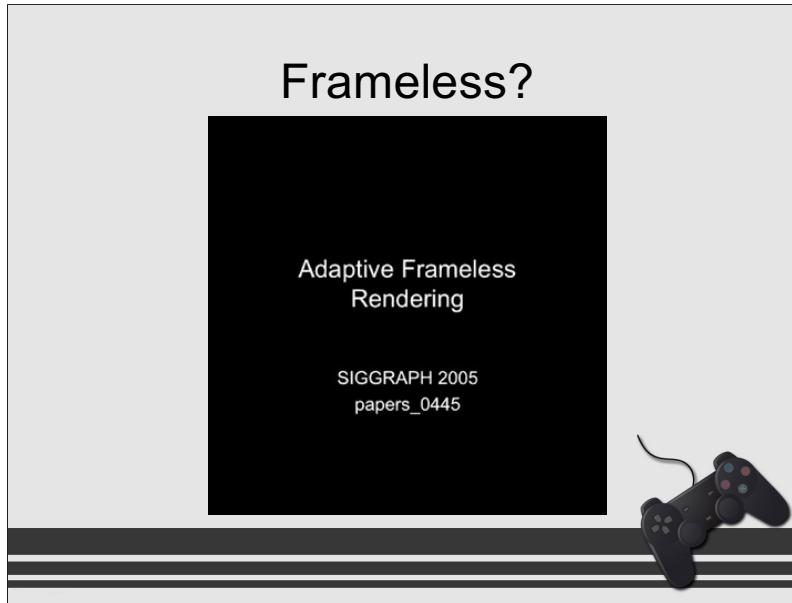


Side Quest: Can We Simulate Without Frames?

- How?
- Pros?
- Cons?



Continuous Rendering?



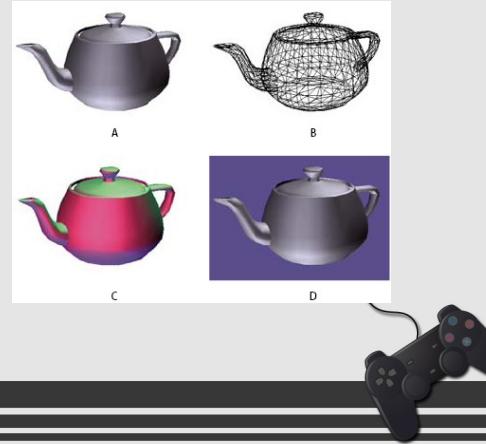
Note the high frequency artifacts. A side effect of raytracing over time
One could rasterize conventionally but frustum clipping is prohibitive

https://www.youtube.com/watch?v=JNeYI_3WIMs

Note that the corresponding paper wasn't actually accepted at SIGGRAPH, even though it's in the video title

Side Quest: Move away from surface rendering?

- How?
- Pros?
- Cons?



Particle-based Rendering (in 2D)



Jelly in the Sky – everything is a particle

One might also consider voxels, though most voxel techniques rely on mapping the voxels to surfaces of some sort

<https://www.youtube.com/watch?v=SyCSEAfGGwU>

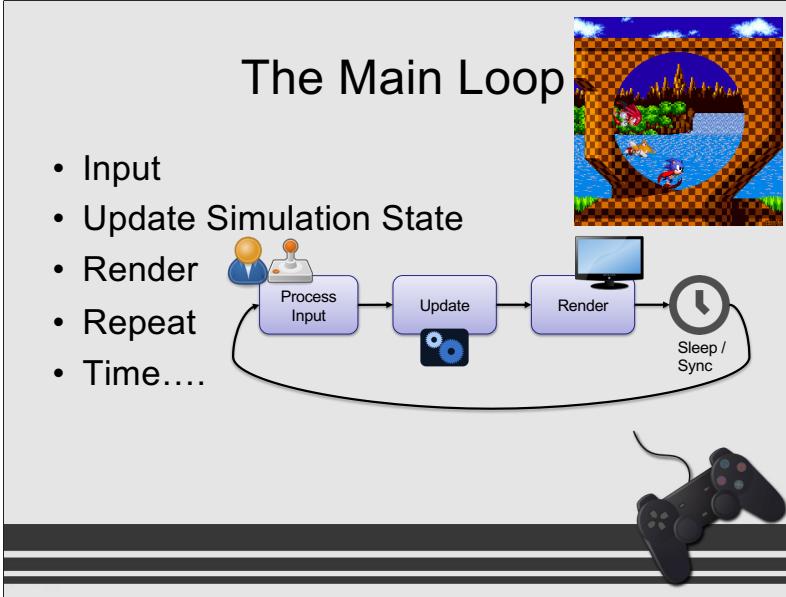


GAME ENGINES SYNCHRONIZING REAL TIME WITH SIMULATION

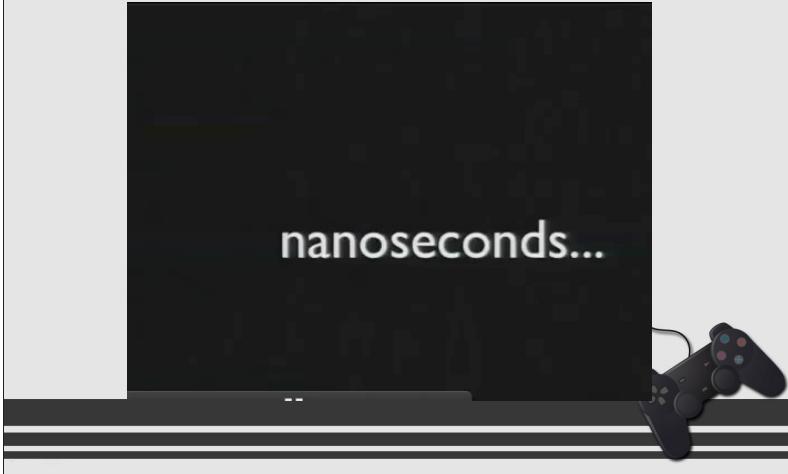


The Main Loop

- Input
- Update Simulation State
- Render
- Repeat
- Time....



Rear Admiral Grace Hopper's Nanoseconds



<https://www.youtube.com/watch?v=JEpsKnWZrJ8>

Time to Work

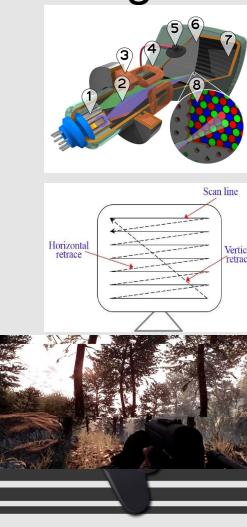
- 60 Hz (frames per second)
- 0.01667 seconds per frame
- 16.67 milliseconds per frame
- 16666.67 microseconds per frame
- Can you get everything done before the next frame?
- You usually want to get done before the next frame and sleep. Why?



One of the Lemmings games

Display's Impact on Timing Schedule

- Legacy of Cathode Ray Tube fixed update rate
- Screen Tearing – Updates of game loop and display not aligned
- Stuttering – Game loop is (v)synced with display but misses a frame boundary occasionally
- Adaptive VSync? - Nvidia



https://en.wikipedia.org/wiki/Cathode-ray_tube

Yesterday's News Today!

- Holy Grail < 20 ms latency (we aren't there)
- Frame-based nature introduces latency
- Rendering based on old data
- User input -> painfully long time -> Draw screen
- Pipelining good for throughput, bad for latency
- Caching good for throughput and smoothing jitter, but bad for latency
- Can we reduce latency?



Paperboy!

Latency Really A Concern?

- Depends on type of game
- **Latency can be recognized if response takes longer than visual stimulus reaction! (0.2s)**
- VR <- very big deal
- FPS
- Game Feel style of game
- Indirect Control (real time strategy)
- Turn based <- much less important



Reduce Latency



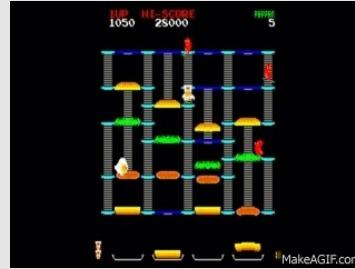
- Higher Frame Rate
- Adaptive VSync
- Take care with pipelining and cache coalescing
- Commercial VR driving latency reduction
- Input prediction (Kalman Filter)
- Relaxed frustum culling (also good for stereo rendering) *coupled with...*
- Late (secondary) input update to camera pose in rendering pipeline
- Direct Memory Address, wider mem buses, etc.



Of course input devices must be able to provide measurements as often as the framerate and can have their own pipeline latency

What to do When?

- Input
- Visuals
- Physical Simulation
- Audio
- AI
- Time for Sleep?



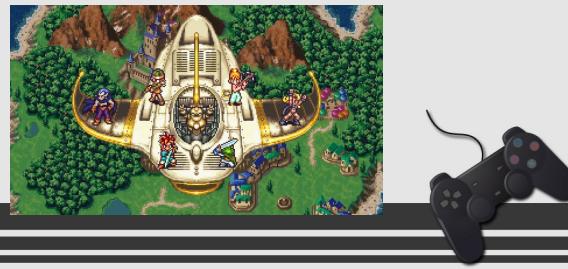
Physical simulation requires more frequent updates typically

Audio is essentially doing its own thing in effectively a separate precisely-timed process. Not too big a deal for one shot sounds (so long as latency is not too bad). However, dynamic sounds can be a problem.

Burgertime

Time Dependency

- Frame rates are not constant.
Repercussions?



Once upon a time, frame rate didn't matter so much. MSDOS was effectively a real-time system (no multi-tasking). Also, games were often sprite based, with a fixed amount of graphical objects onscreen (so processing load was effectively constant). Games could simplify calculations for animations by just counting frames. Provided that everyone had the same clock speeds on their CPU, this approach worked fine for 2D sprite-based games, which have pretty constant processing demands.

A side effect of this, is that newer computers ran the games too fast. Utilities such as "Mo'Slo" allowed a partial fix

Chrono Trigger

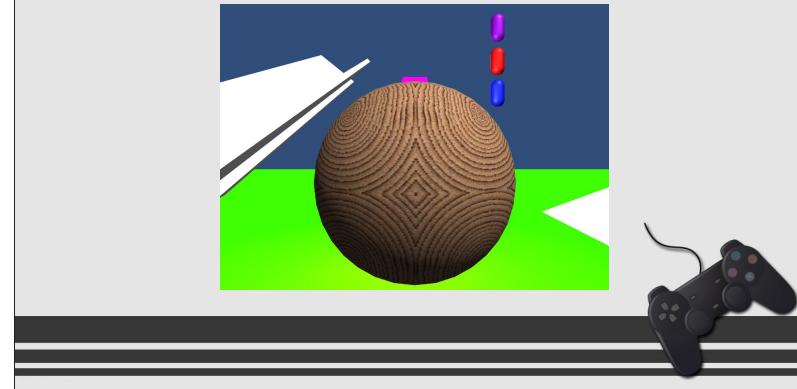
GAME ENGINES

TIME DEPENDENCY DEMO



Time Dependency

- Unity Demonstration



Once upon a time, frame rate didn't matter so much. MSDOS was effectively a real-time system (no multi-tasking). Also, games were often sprite based, with a fixed amount of graphical objects onscreen (so processing load was effectively constant). Games could simplify calculations for animations by just counting frames. Provided that everyone had the same clock speeds on their CPU, this approach worked fine for 2D sprite-based games, which have pretty constant processing demands.

A side effect of this, is that newer computers ran the games too fast. Utilities such as "Mo'Slo" allowed a partial fix

Chrono Trigger



GAME ENGINES SYNCHRONIZING REAL TIME WITH SIMULATION – TIME DEPENDENCY

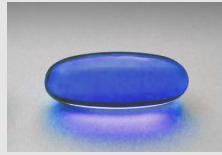


Time Dependency – Animation of translation/rotation

- Constant translation/rotation “dumb mode”
 - $\text{newPos} = \text{oldPos} + \text{constTranslation}$
- Time-dependent translation/rotation
 - $\text{newPos} = \text{oldPos} + \text{velocity} * \text{deltaTime}$
- Constant translation/rotation with **fixed update** cycle
 - $\text{newPos} = \text{oldPos} + \text{constTranslation}$



“Dumb Mode”



- **Disadvantages**

- Oblivious to advance of real-world time external to simulation
- This only works well if simulation takes a near constant amount of time to update each frame
- Maybe ok for classic 2D games on fixed hardware, but very difficult to guarantee for modern 3D games
- Animation/effects can be inconsistent
- Varying gameplay experience depending on hardware and/or scene complexity

- **Advantages**

- Low overhead (only need to add constant offset for translation/rotation!)



https://en.wikipedia.org/wiki/Red_pill_and_blue_pill#/media/File:Red_and_blue_pill.jpg

Time-Dependent Mode



- **Advantages**

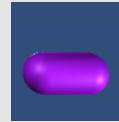
- GOs aware that real-world time may advance differently relative to simulation frames
- Gameplay experience normalized across scene complexity and/or diff. hardware

- **Disadvantages**

- Extra computation to compute time-dependent effects
- Tiny deltaT's can cause FP rounding errors if you aren't careful



Fixed Update



- Hybrid approach to time-based updates
- GameObjects update as if framerate is constant
- External manager determines fixed update schedule



Fixed Update

- A common misconception is that a fixed update process runs in parallel perfectly following a real-world clock
- But generally, NOT the case!



Fixed Update

- Fixed Update Mgr keeps track of a target elapsed time and decides how many times to run fixed update cycles per frame
- $\text{totalDeltaTime} = \text{remainingDTPrevFrame} + \text{deltaTime}$
- $\text{numFixedUpdatesThisFrame} = \text{Floor}(\text{totalDeltaTime} / \text{fixedUpdatePeriod})$
- $\text{remainingDTPrevFrame} = \text{totalDeltaTime} - \text{fixedUpdatePeriod} * \text{numFixedUpdatesThisFrame}$



Fixed Update

- Fixed Update framerate *higher* than framerate?
 - Frames have more than one fixed update cycles per frame on average!
- Fixed Update *slower* than framerate?
 - Some frames have no fixed updates!



Fixed Update

- **Advantages**
 - Compensates for real-world time elapse
 - GameObjects can be oblivious to time sync issues
- **Disadvantages**
 - Runaway computational load when frame rate drops (lower framerate leads to more fixed updates, leads to lower framerate...)
 - Can only be applied to a subset of all update objects/tasks
 - User can only respond to state changes reflected in full frame updates (not fixed updates)
 - Sometimes difficult to coordinate fixed update with normal update logic (some GOs update in both cycles)
 - Jitter



Fixed Update Special Cases

- Fixed Update manager may need to prioritize certain objects if too much impact on frame rate (e.g., not update some)
- In cases where fixed update is slower than framerate, it may be desirable to interpolate/extrapolate visual objects to resolve jitter (Unity can do this)



Reasons to Use Fixed Update

- Can simplify time-based computations
 - Ex: Constant velocity displacement only needs an Add
- Simulation unstable if update rate too low
 - Physics simulation (Unity does this)
- Avoid expensive computation with a lower fixed update rate than frame rate
 - can be interpolated with less computation
 - Mobile gaming (battery power)
- Avoid FP rounding errors from extremely small deltaT



What in a game has time dependency?

- Pretty much everything!
- Game object movement including velocity and acceleration
- Animation Systems
- Physics simulation
- AI decision making
- Probabilistic behavior
- Etc.



GAME ENGINES

THE MODERN GAME ENGINE



Modern Game Engine

- Software framework supporting the creation, development, and deployment of a game.



Influences on GEs

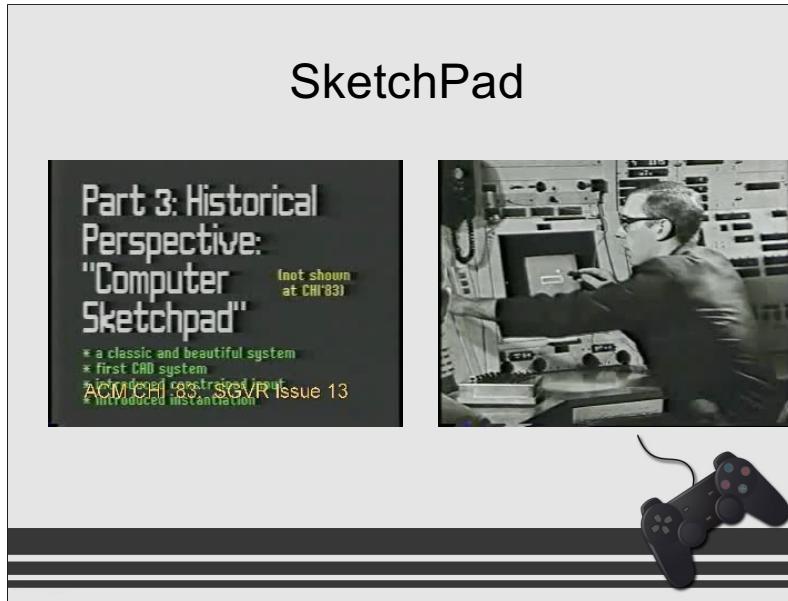
- Ivan Sutherland's SketchPad
- HyperCard



SketchPad late 1950's/early 1960's

HyperCard 1987

Vannevar Bush's Memex from "As We May Think"



Ivan Sutherland's SketchPad

Early CAD like software. Defining constraints (e.g. declarative)

https://www.youtube.com/watch?v=USyoT_Ha_bA

Part 2

<https://www.youtube.com/watch?v=BKM3CmRqK2o>

Geometric Manipulations

Referential objects (like prefabs)

Declaratively define graphical data objects via constraints

Interface metaphors

- Paper (but 2 miles wide! And multiple workspaces)
- Pen
- Mechanical Drafting Plan View (top, sides, perspective, all synchronized in real-time)

State machine visual programming



https://www.youtube.com/watch?v=AmeUt3_yQ8c

Similar to modern web development

Event-based programming

WYSIWYG – What You See Is What You Get Editor

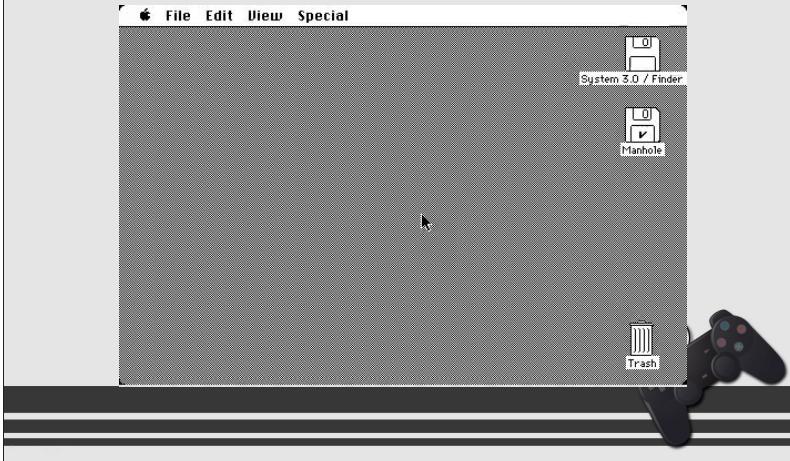
Integrated Development Environment – Interface Designer coupled with code writing (HyperTalk)

Foreground/background (shared background)

Some primitive animation (e.g. Juggler)

Save state between cards

HyperCard Game Example



The Manhole - https://en.wikipedia.org/wiki/The_Manhole

<https://www.youtube.com/watch?v=YyOTq1EpV5o>

Early Internal Game Engines

- Z-machine
by Infocom
- SCUMM



Ease of authoring and Cross-platform big motivating factors

Clear emphasis on non-expert constraint authoring

Note that these are highly genre specific

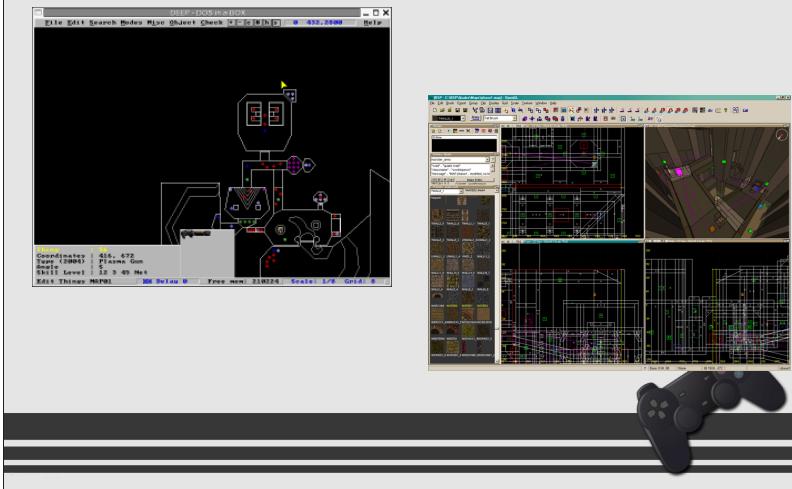
Early 3rd Party Game Engines



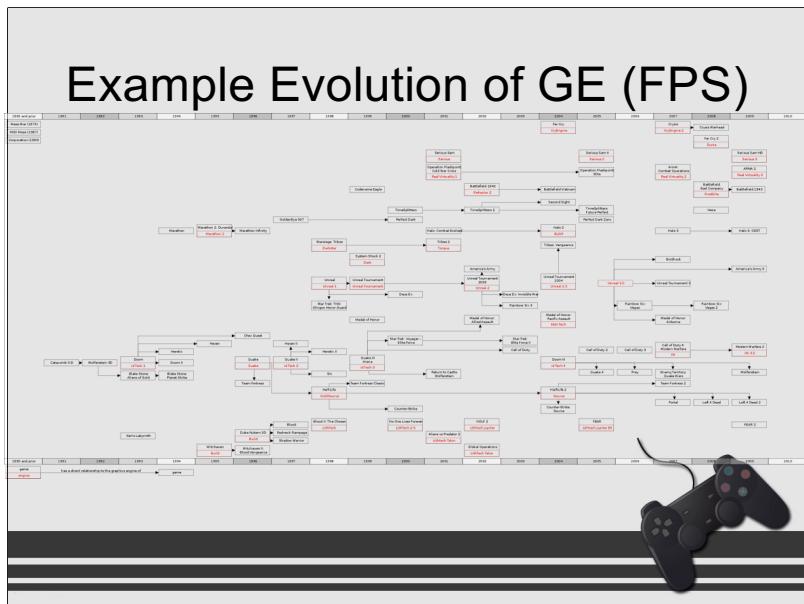
Adventure Game Maker and RPG Maker 95

Appeal to kids that want to make games

The Rise of FPS Games



Doom Editor and a Quake Editor



From wikipedia

Game Engine Features

- Creation is often declarative
- Some procedural extension of tools and underlying computational kernel
- Platform Abstraction
- Integrated Development Environment (IDE)
- WYSIWYG (e.g. live editing)
- Asset Mgt/Content Pipeline/Workflow Support
- Often Standalone
- Genre flexibility



Zelda Spirit Tracks

GE Components

- Computational Kernel (Main Loop)
- Input Management
- Rendering Engine
- Physics Engine
- Audio Engine
- Networking Module
- Scheduling Updates...



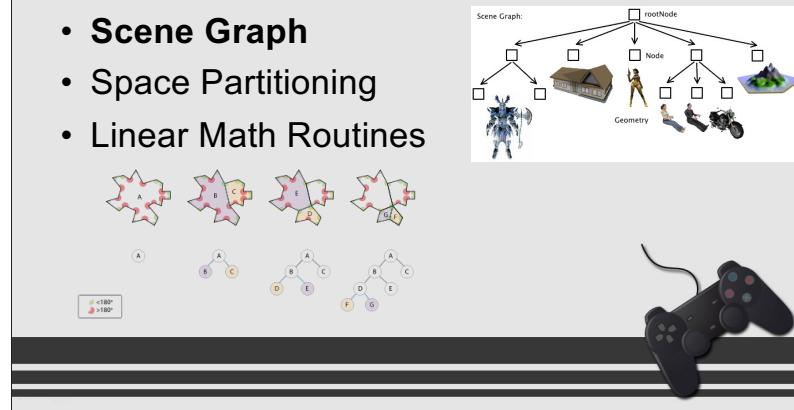
Input Management

- Low Latency Desired
- Abstraction/Mapping of common input controls across variety of hardware



Graphical Rendering Engine

- Canonical Render Pipeline
- **Scene Graph**
- Space Partitioning
- Linear Math Routines

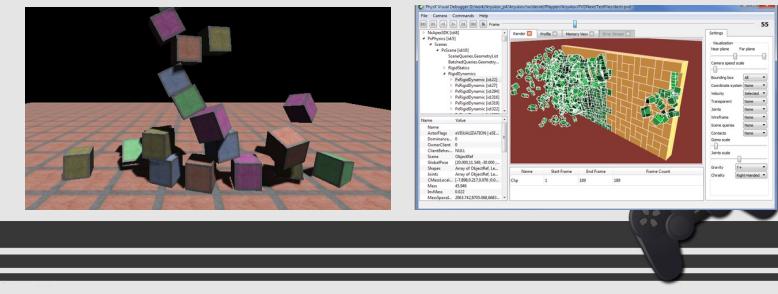


2D Binary Space Partitioning (BSP) figure shown

Also, Scene Graph example

Physics Engine

- Constraint Solver
- Simultaneous “world” synchronized with graphical and audio representations



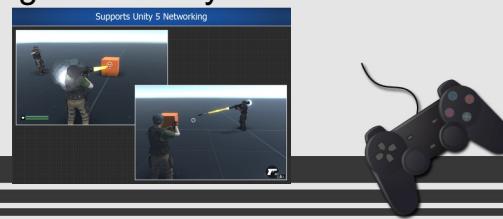
Artificial Intelligence Module

- Path Planning
- Behavior Implementation
- Behavior Planning
- Time is on a different scale
- But still time dependencies!!! (e.g. account for FPS)



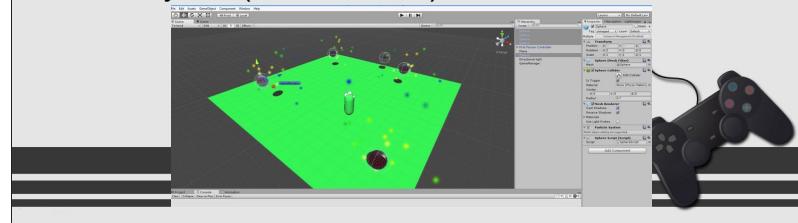
Networking Module

- Platform Abstraction (big- little endian, etc.)
- Event Synchronization (input and/or game object state)
- Priority Management of synchronization
- Prediction



Event Based Architecture

- Message-driven (queues)
- User input
- Constraint Solver Triggers
 - Time based
 - Physics (collision, etc.)



Event Based Architecture

- Supports Modularity/Loose Coupling
- Emitters don't need to know anything about consumers
- Event streams can be readily dispatched via network
- Downside: Can be difficult to read existing event-based code without executing and following events in debugger





GAME ENGINES SUMMARY



What is a Game Engine? (Revisited)

- A closed-loop sensory simulation meant to convince a game player that a virtual world exists and can be interacted with in real time
- A simulation based on a rapid sequence of frames (like frozen slices of time)
- A constraint solver declaratively defined by the game designer and further extendable via event-based callbacks/handlers (events generated by the constraint solver, user input, or connected system)
- A set of interactive tools supporting creation, development, and deployment of a game

