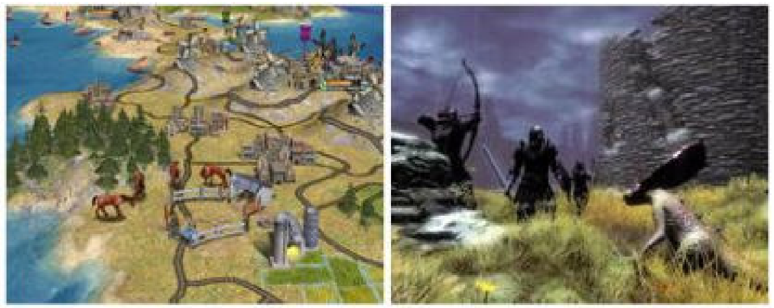
ENJMIN

ECOLE NATIONALE
DU JEU ET DES MEDIAS
INTERACTIFS NUMERIQUES

Gaming Technologies

Gamebryo Element™
3D GRAPHICS ENGINE
AND TOOLS



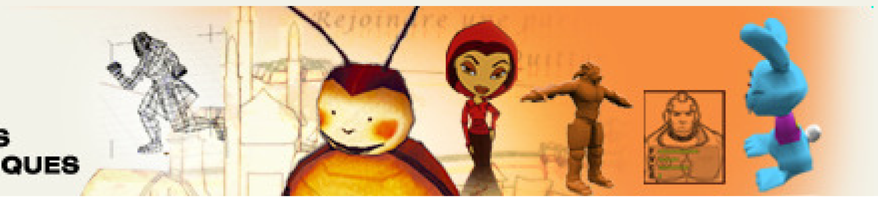
Alexandre Topol

ENJMIN
Conservatoire National des Arts & Métiers

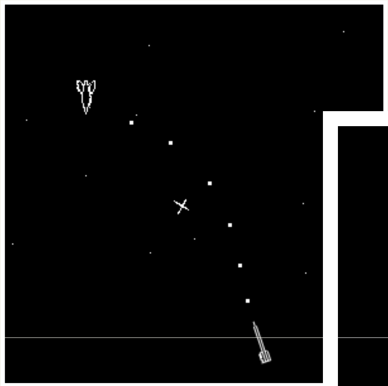
Qui suis-je ?

- Ingénieur (2000) et docteur (2002) en Informatique
- Ancien programmeur Ubisoft (en 1996)
- Enseignant au CNAM et à l'ENJMIN
  - Programmation 3D
  - Langages de shading
  - Physique
  - Animation
  - IHM
  - …
- Chercheur au laboratoire CEDRIC

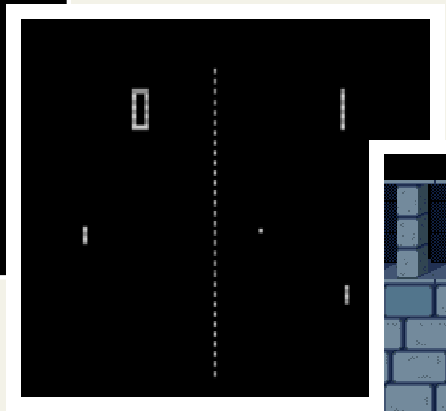
© Ubisoft – POD - 1997



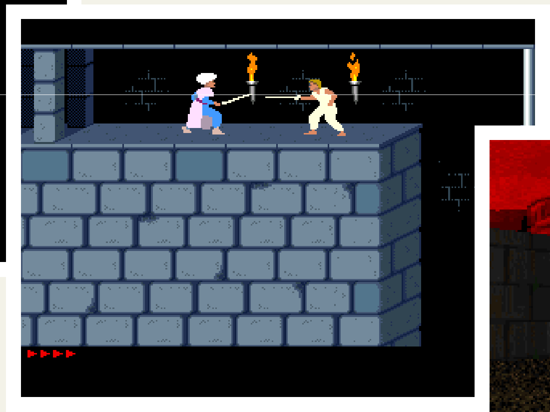
50 years of evolution ...



1961
Spacewar!
Steve Russel
(MIT)



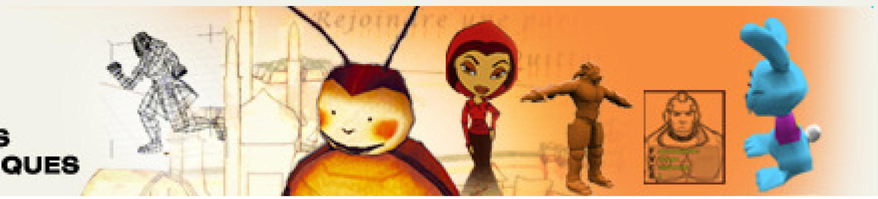
1972
PONG
Nolan Bushnell
(Atari)



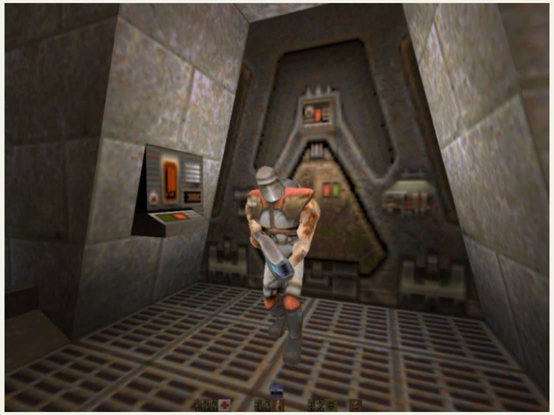
1989
Prince of Persia
Jordan Mechner
(Brøderbund)

1993
Doom
John Carmack
(Id Software)

ECOLE NATIONALE
DU JEU ET DES MEDIAS
INTERACTIFS NUMERIQUES



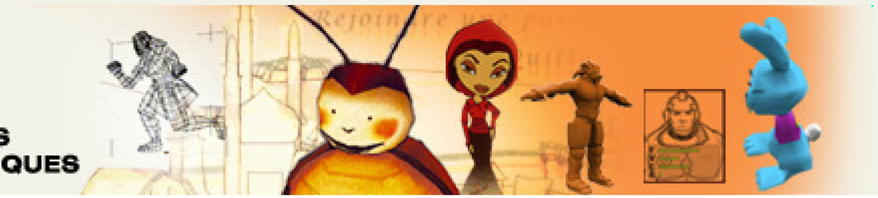
... of games ...



1997
Quake 2
Id Software
(Activision)

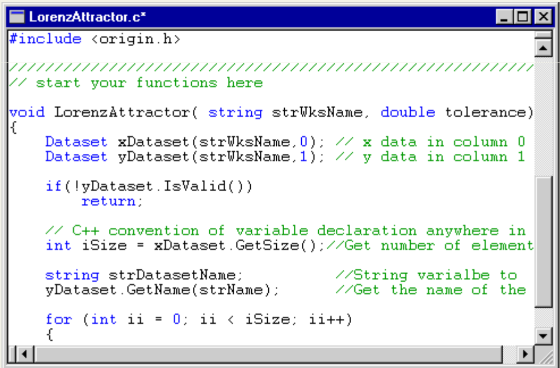
2004
Far Cry
Crytec
(UbiSoft)

2008
Assasin's Creed
(UbiSoft)

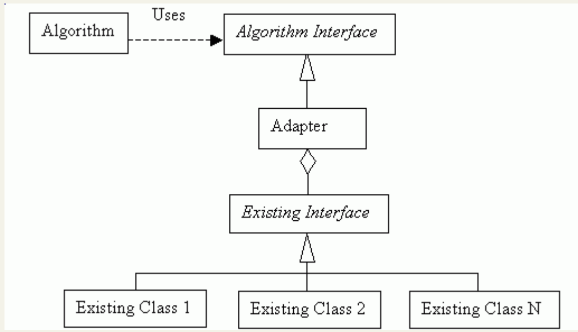


... and programming ...

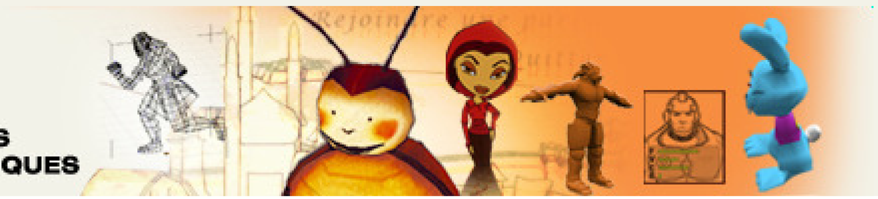
1960
BASIC



1972
C

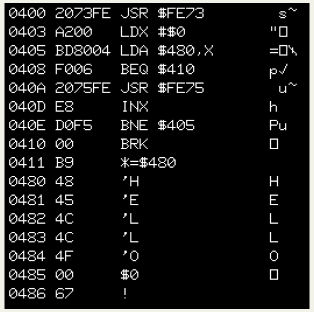


1983
C++

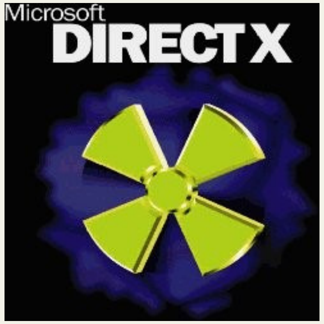


... methods

1993
Mods

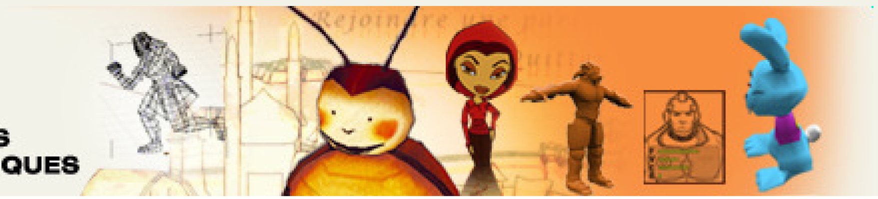


1995
3dfx ASM



1995
DirectX 1.0

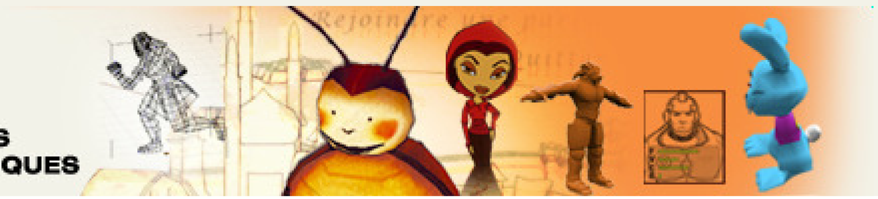
2002
Renderware



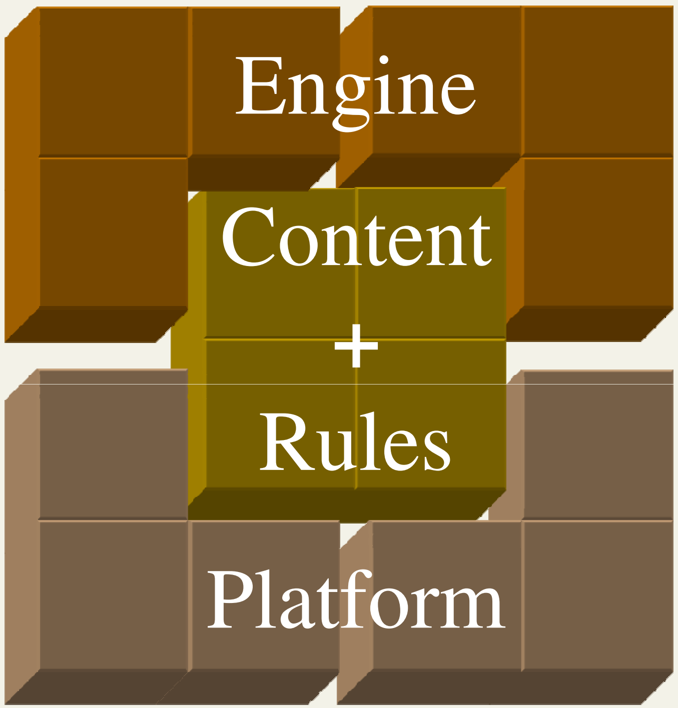
50 years of evolution

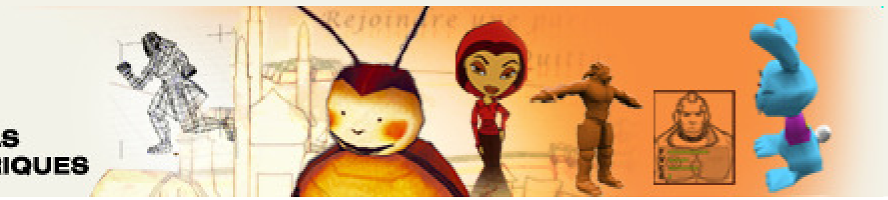
- Why these evolutions ?
  - For programming languages: need of more expressive power
  - For game programming methods : need of less development time
- How did we get there ?
  - To get closer to the heart of the game :

the gameplay

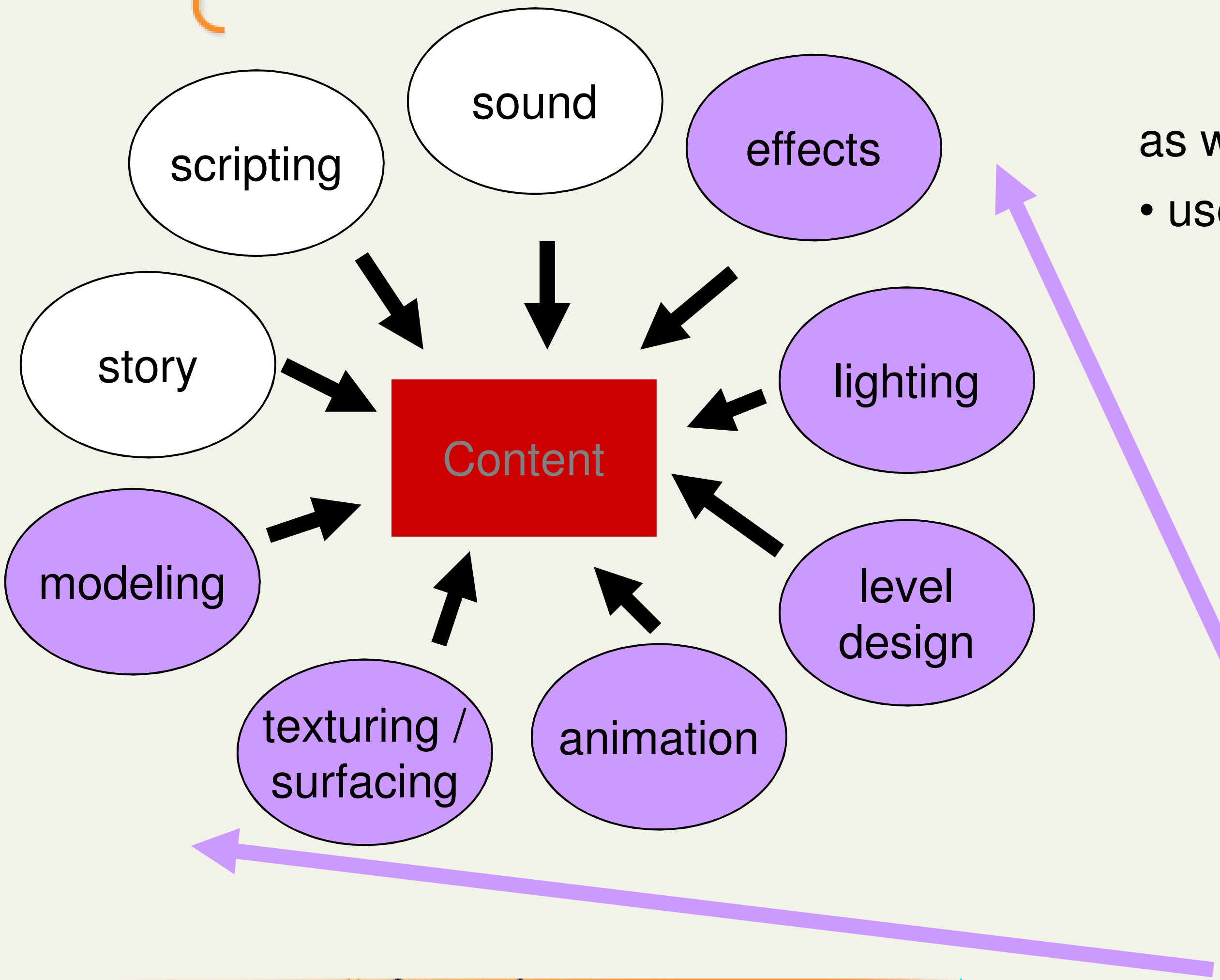


Game Components





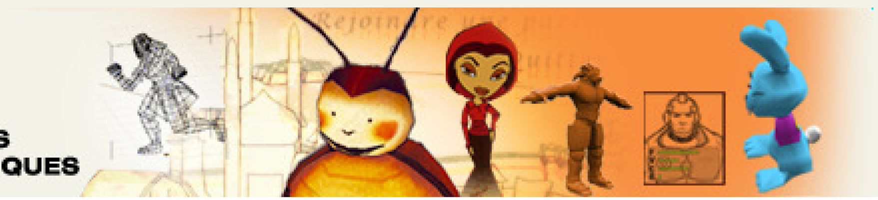
Content Components



as well as …

- user interface

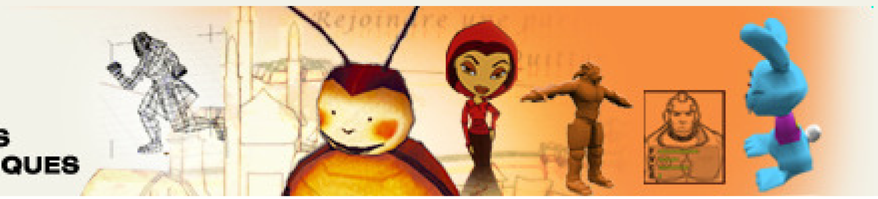
visual technologies



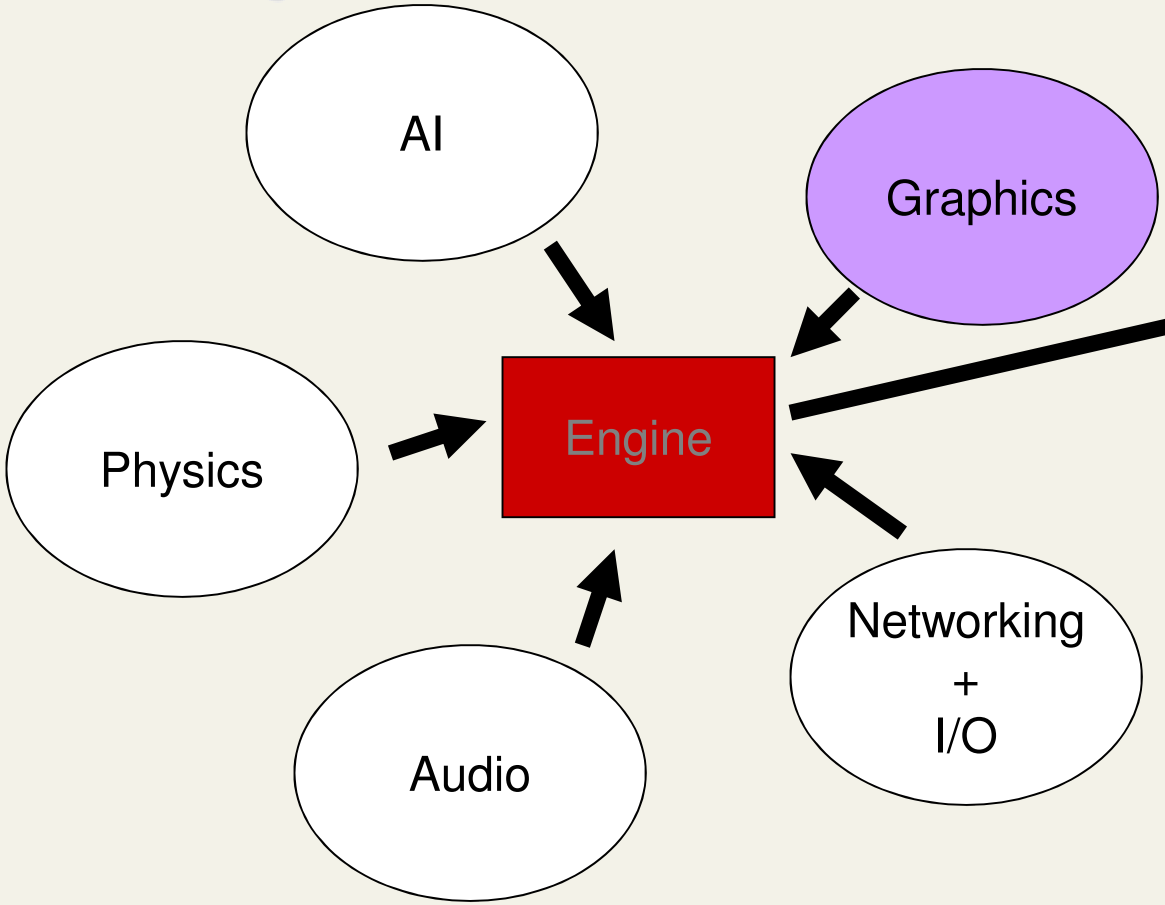
ECOLE NATIONALE
DU JEU ET DES MEDIAS
INTERACTIFS NUMERIQUES

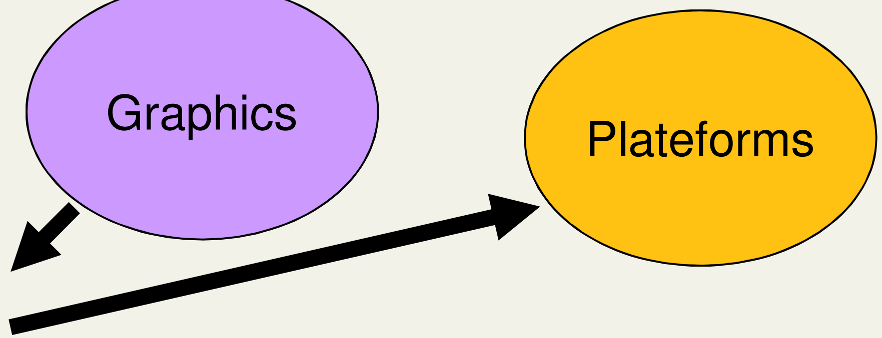
Content Creation

- Method driven for some
- Tool driven for others
- Combination of tools forms “Art Pipeline”
- Character Pipeline
  - 3D Model/Sculpting
  - Animation/Rigging
  - Skinning
  - Texturing
  - Lighting/Shading
- An artist often specializes on one part of pipeline
- Some tools in Game Engines (Emergent AnimationTool)



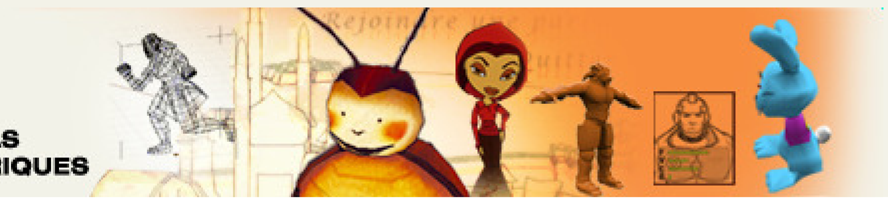
Game Engine Components



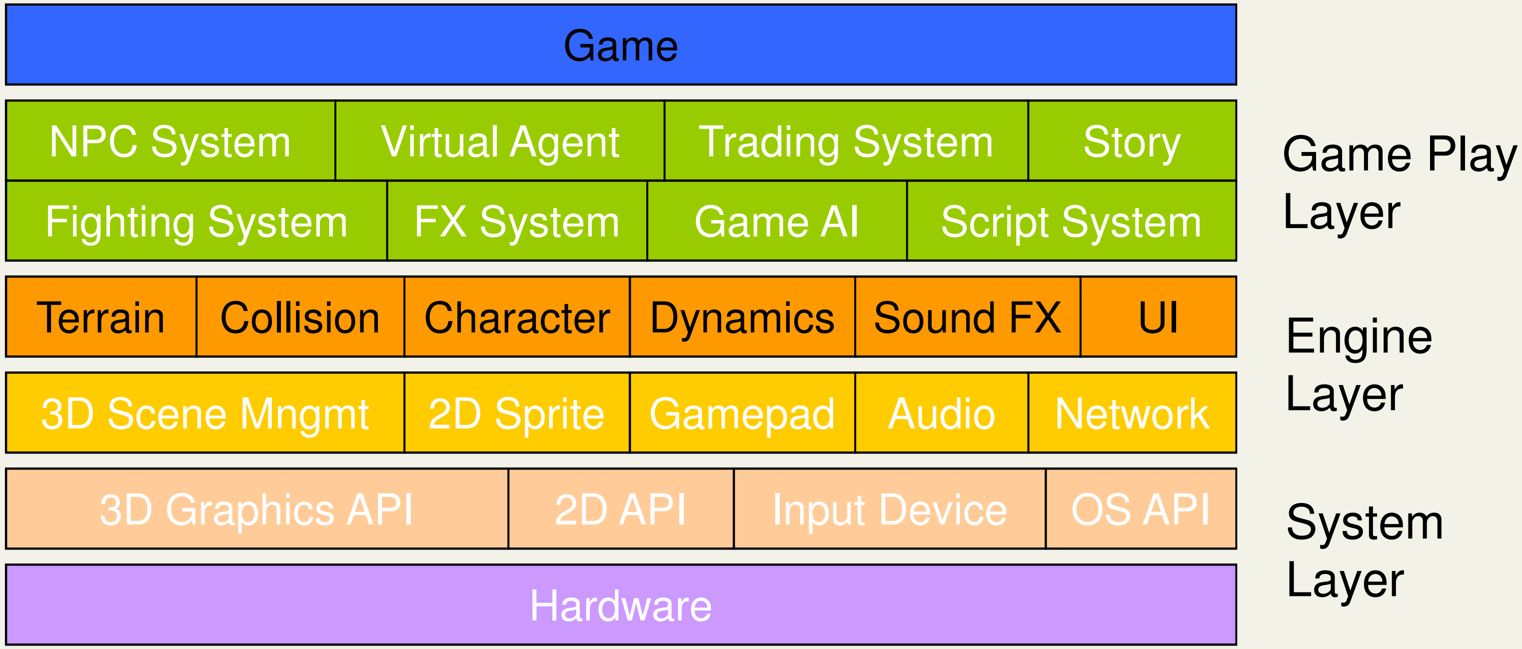


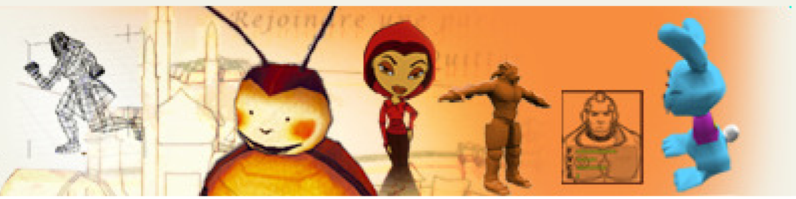
as well as …

- installation
- patching
- resource management
- multithreading
- disk i/o
- state save/restore



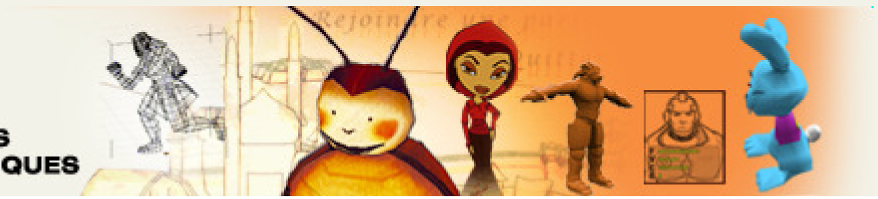
Game Engines





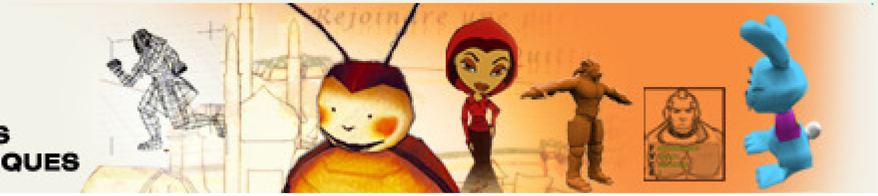
Games Engines Vs Σ(APIs)

- For instance : DirectX or { Ogre3D + OSG + Bullet + Wwise… }
- Each element is tightly embedded
- Same structures for different aspects
  - No need to dupicate objects
  - Same bounding boxes are used for both collision detection and object culling
- Most important: higher levels are gameplay oriented
- High level tools to design levels
  - Graphical & audio aspects
  - Triggers for audio or AI script
  - Physics parameters
  - …



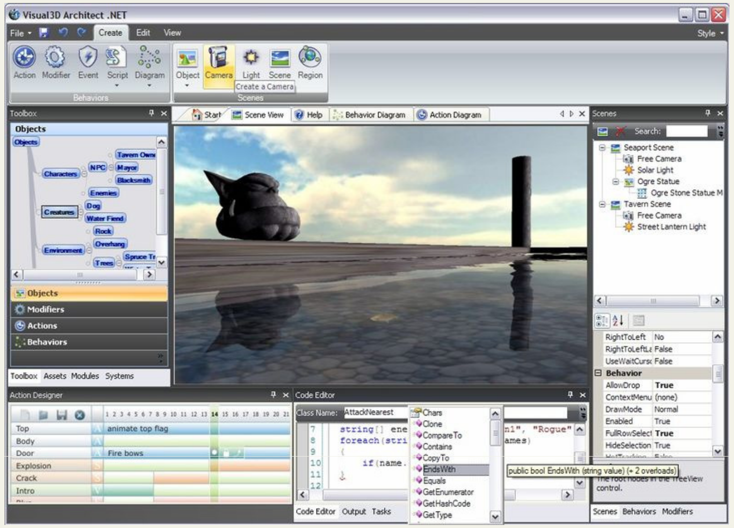
Game Engines

- Studio chooses to build or buy
  - Quake Source, Unreal engines
  - Renderware, Gamebryo middlewares
  - Often come with great authoring tools (level editors, etc.)
- Componentized software
- May buy specialized components
  - How many people can write a physics engine?
  - And how many studios can afford writing one ?
  - Video codecs, etc
- Engine may be optimized for game genre/style
  - Includes special features to answer special needs
  - Terrain vs indoor scenes, camera management, …



Game Engines

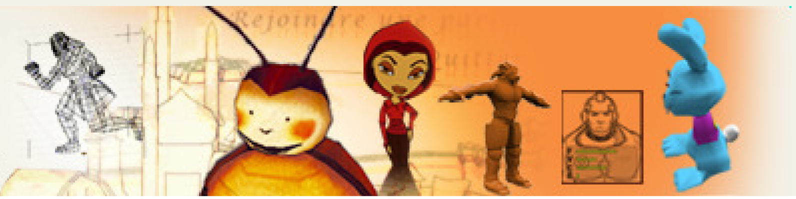
- 3D graphics tools
- Physics engine
- Audio
- Animation
- Character “AI”



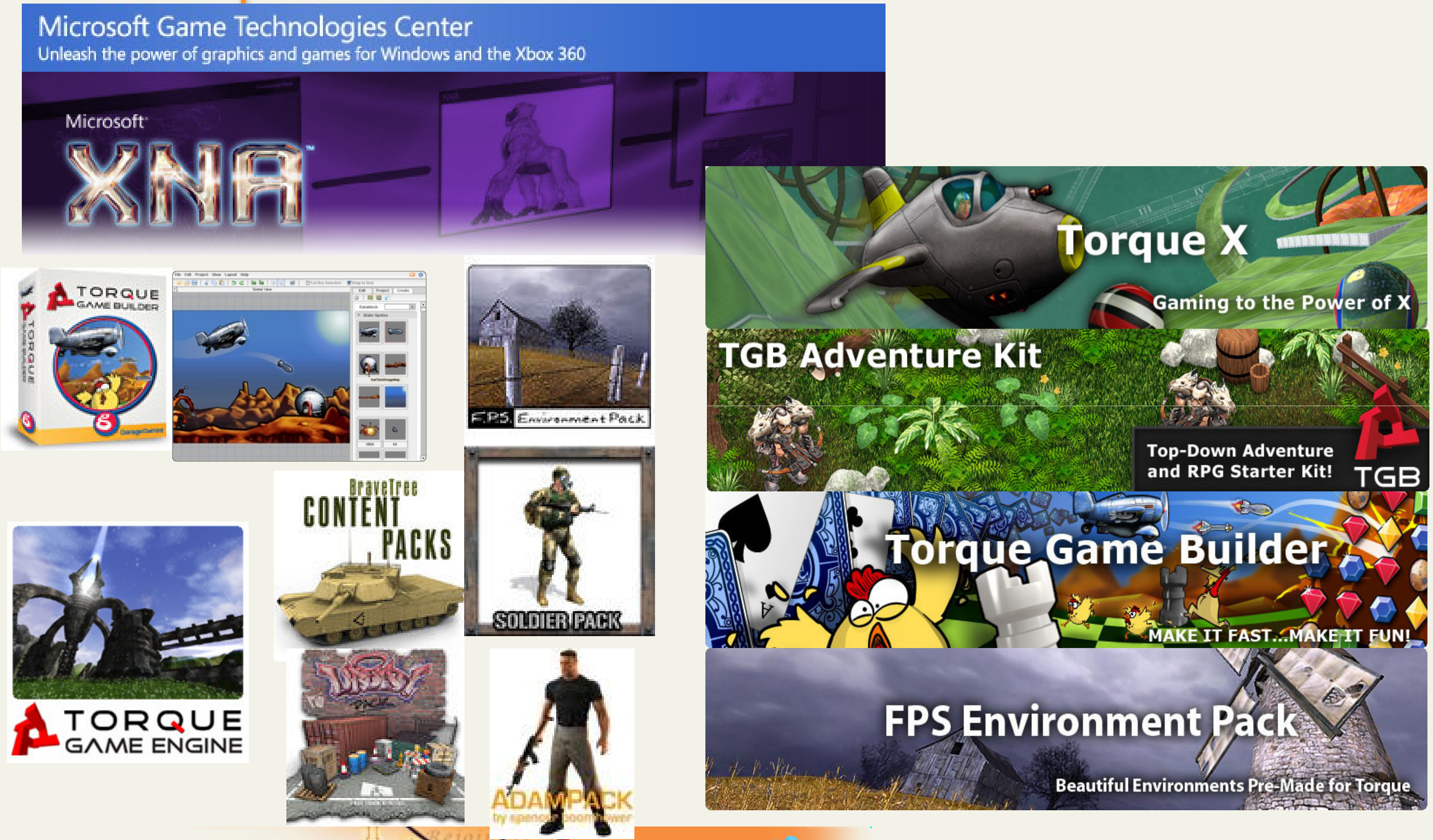
Visual3D Architect .NET Screenshot RealmWare Corporation

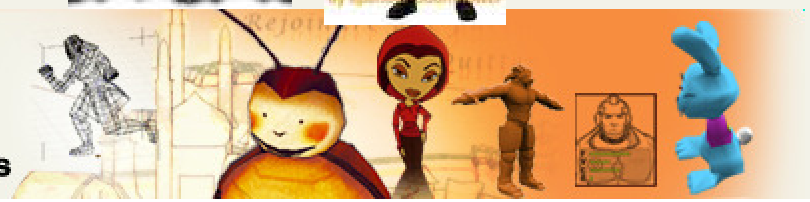
Cost: ranges from open source (CrystalSpace) to $100K+ (Unreal Engine)

- Typically tailored to a particular kind of game
  - First person shooter (FPS)
  - Real time strategy (RTS)
  - massively multiplayer online role-playing game (MMORPG)



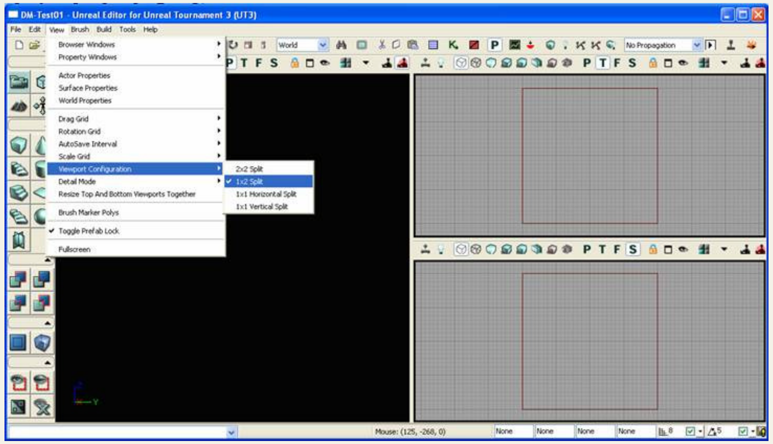
Game Engines

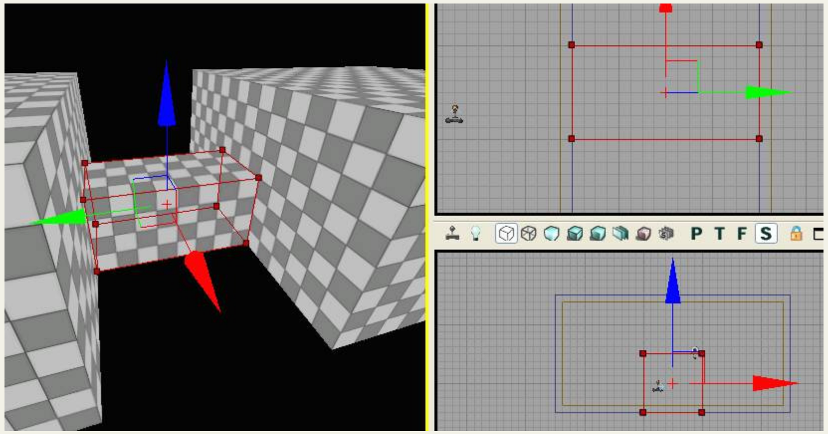


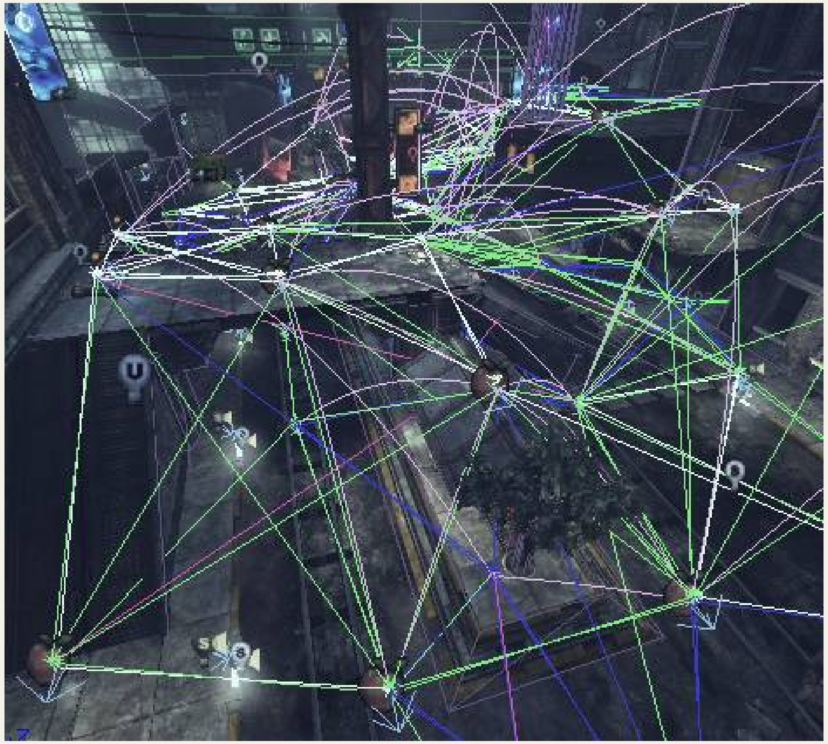


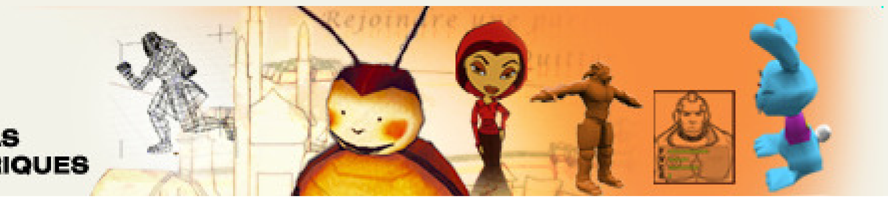
Game Engines

Unreal Engine 3







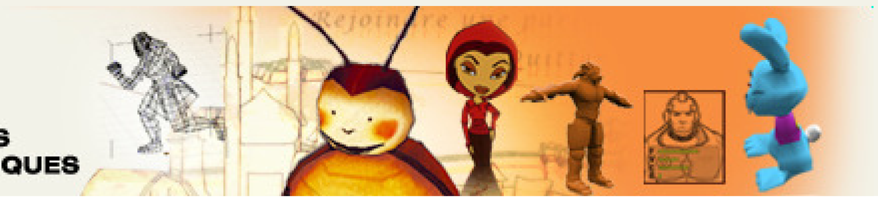


Game Engines

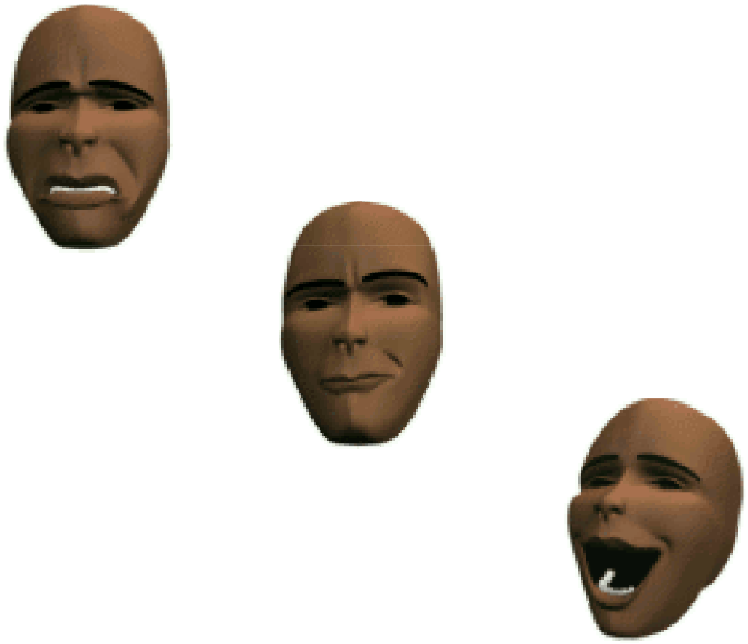
- *Engines:*
  - *Unity3D*
  - *UDK (vs Unreal Engine)*
  - *Cry Engine*
  - *Torque* ***– Low cost set of engines (2D, 3D, 3D+Shaders), large dev community***
  - *3D Game Studio* ***– Hundreds of games, C-script, many libraries of pre-made games***
  - *OGRE* ***– Scene-oriented, 3D engine, open source, Basic Physics***
  - *Crystal Space* ***– Used for Modeling and Simulation, Physics engine, True 6DOF***
  - ***Many others at http://www.devmaster.net/engines/***

- *Terrain Tools:*
  - *L3DT –* ***"Plugable" Terrain Generation engine, low cost, importing into Torque***
  - *Terragen 2 –* ***Amazing photorealistic terrains and terrain imagery – More real than real***

- *"Mod" tools:*
  - *Return to Castle Wofenstein / Enemy Territory -* **Based upon an older version of the Quake engine.**
  - *Quake III -* **One of the most heavily modified game ever. id has announced they will make the game code open source.**
  - *Counter Strike -* **A great starting point for tactical & law enforcement sims and FPS**
  - *Counter Strike: Source -* **A rebuild of the original but to use the Source engine.**



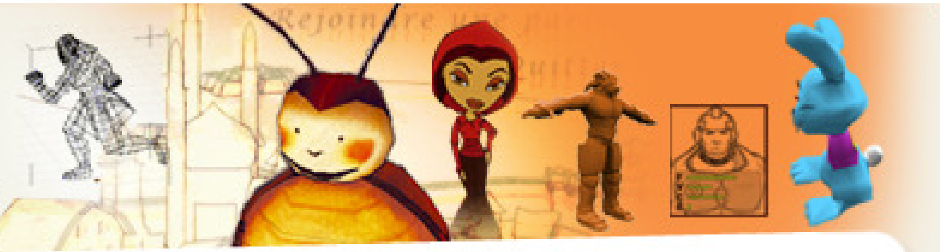
Game Engines Features (Rendering)



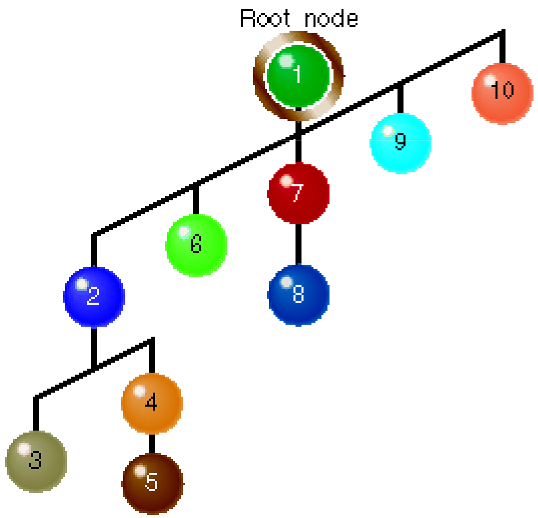
Alexandre Topol

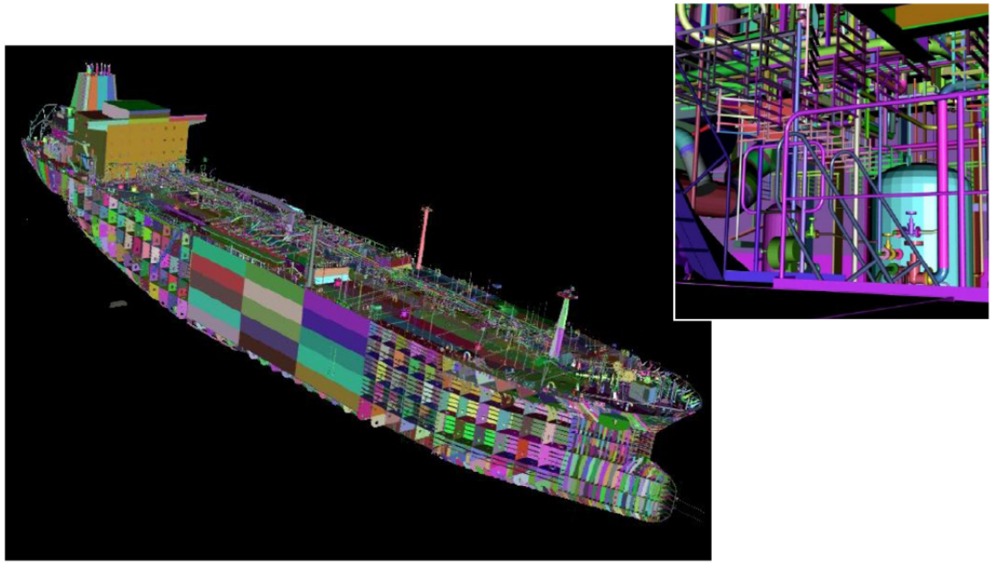
ENJMIN
Conservatoire National des Arts & Métiers

ECOLE NATIONALE
DU JEU ET DES MEDIAS
INTERACTIFS NUMERIQUES



Game Engines Features (almost Rendering)





Alexandre Topol

ENJMIN
Conservatoire National des Arts & Métiers

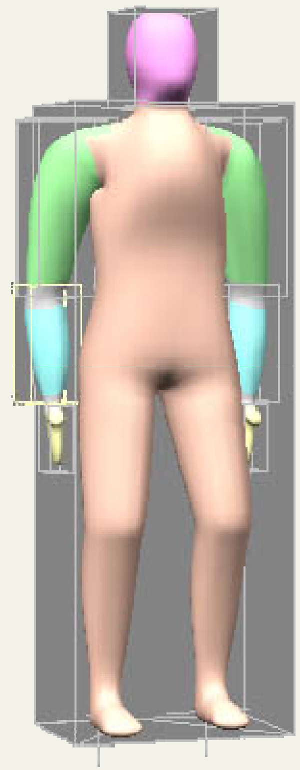
Visibility Management

- Render cost should be proportional to what is seen !
- And not proportionnal to the complexity of the scene

- Don't render objects that won't be seen:
  - Object outside viewing frustrum (➔ is clipping good ?)
  - Objects behind others (➔ is ZBuffer good ?)
- Don't render unnecessary details:
  - Details too small to be performed
  - ➔ LOD
  - ➔ Pixel culling

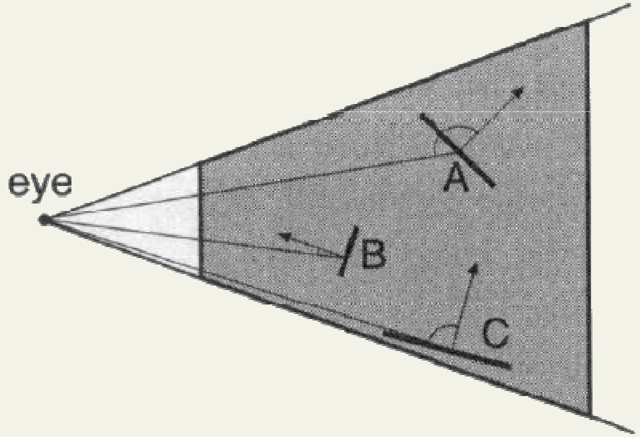
Culling

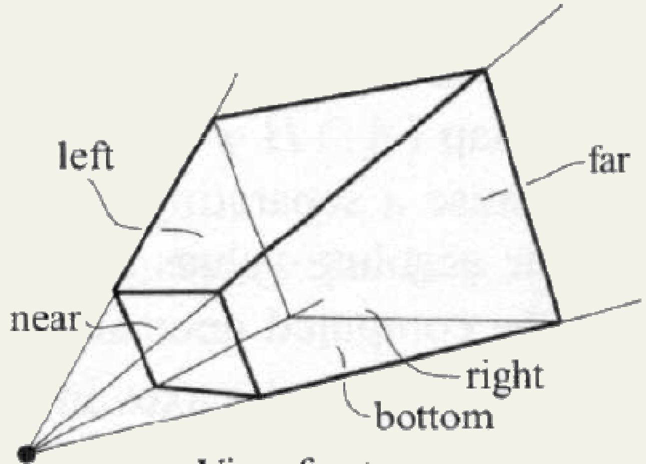
- Culling = throw away non-visible things
- General strategies:
  - Multi phase testing
  - First cheap and coarse
  - Then gradually increase cost/precision
  - Based on hierarchical bounding volumes



Basic culling

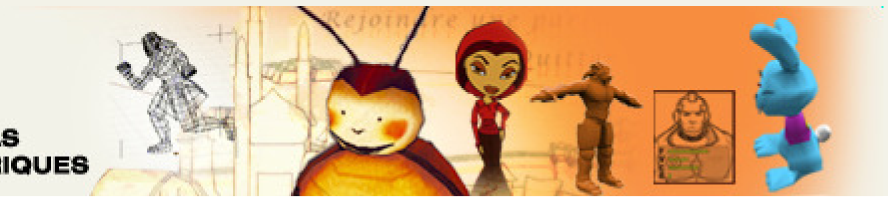
- Backface culling
- Polygon culling (clipping)





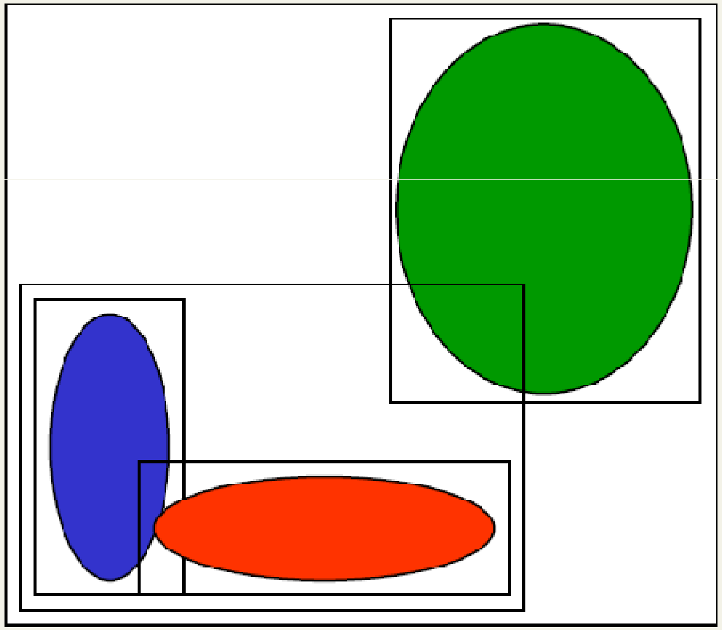
Scene graph

- Cascading transformations and behaviors
- Basic algorithm
  - From top node
  - Render all children
- With visibility management on:
  - Start from root node
  - For each node
    - Get bounding box
    - if visible:
      - Go down hierarchy
    - esle:
      - Cull



Bounding Volume Hierarchies

- BVHs

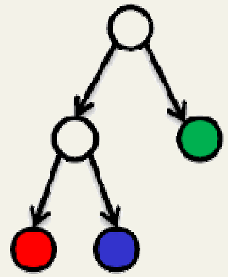


if bv cullable

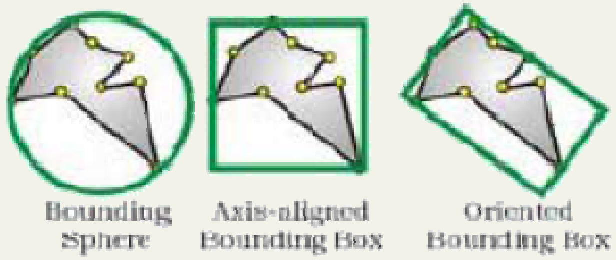
  cull all content

else for each child

  recurse

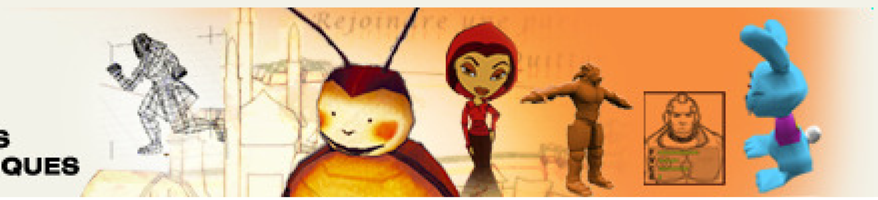


Bounding volumes



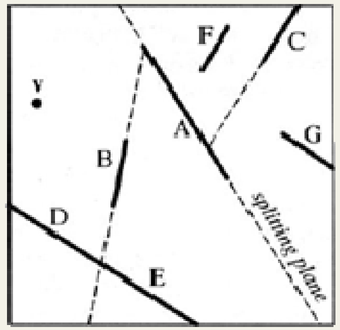
- BS – bounding sphere
  - BS vs BS is the easiest way to compute intersections
- AABB – Axis Aligned Bounding box
  - Boxes given a in the same global coordinate system
- OABB – Object Aligned Bounding Box
  - Boxes given in a local coordinate system

	Test	**Create**	**Update**	**Fit**
Sphere	Very cheap	Cheap	Free	Poor
AABB	Cheap	Very cheap	Medium	Medium
OBB	Medium	Expensive	Free	Very good

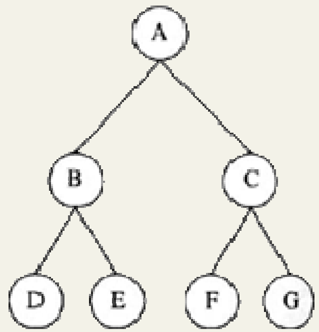


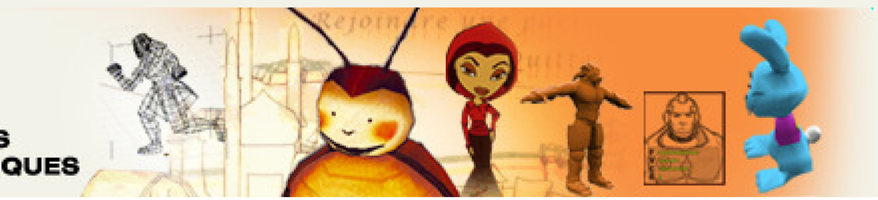
BSP Trees

- Binary Space Partitioning Trees
- A 3D "binary search tree"
- Good for terrains



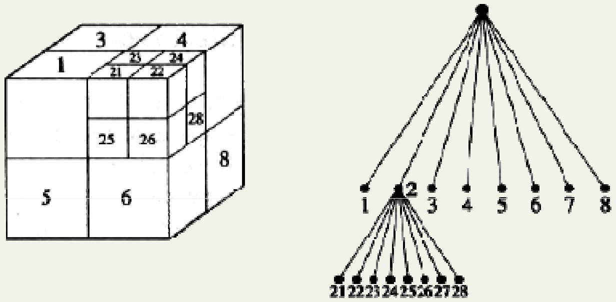
- BSP construction
  - Choose a plane defined by a random polygon
  - Classify all polygons as in, behind or in front
  - Split polygons if necessary
  - Recurse

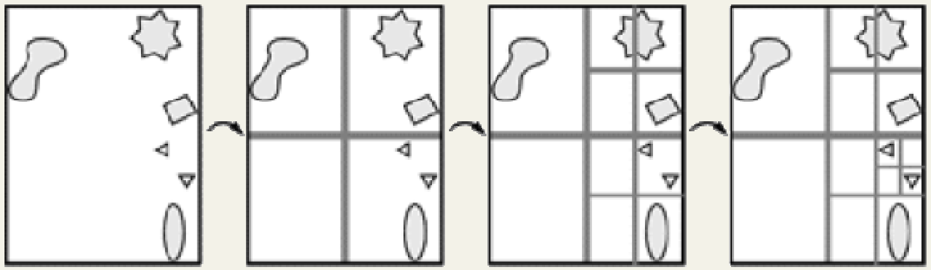




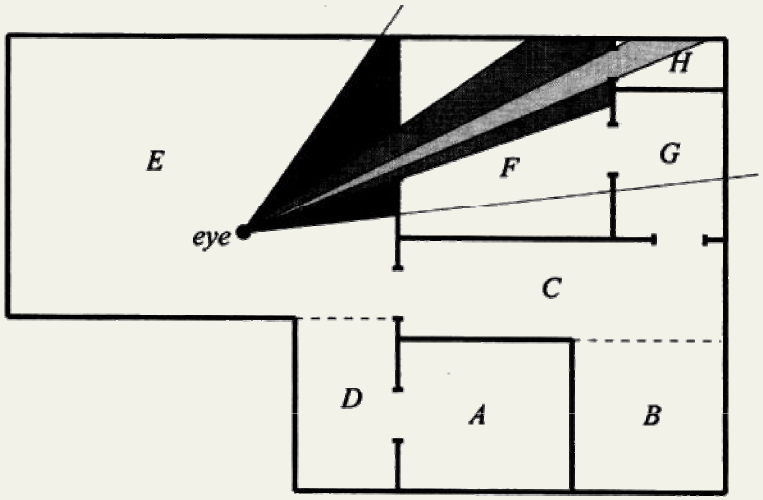
Octrees

- 3D "cube trees"
- Special case of BSP

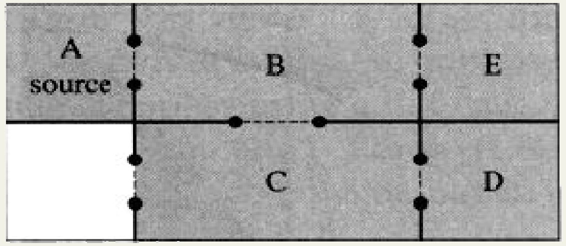




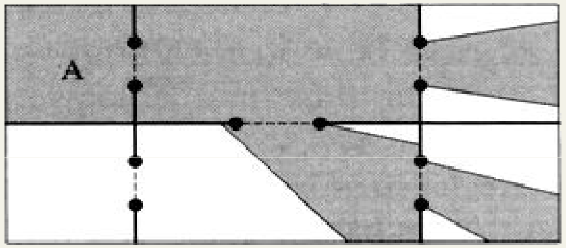
Portals & PVS



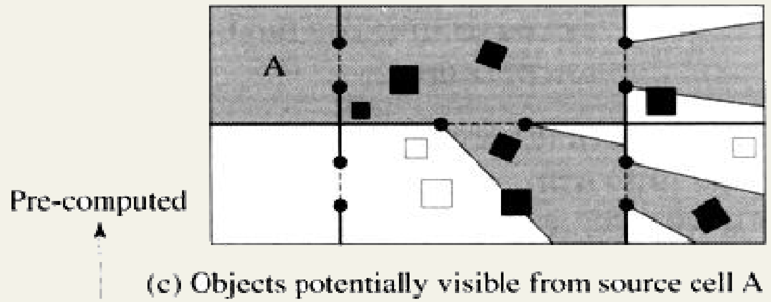

```
render( currentRoom, viewPort )

render( room, clipArea ):
  for all polygons
    if not portal
      render
    else
      render( portal.nextRoom,
              clipArea ∩ portal.silhouette )
```



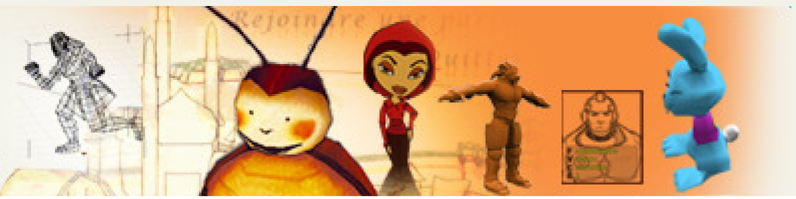
(a) Cells visible from source cell A – cell to cell visibility



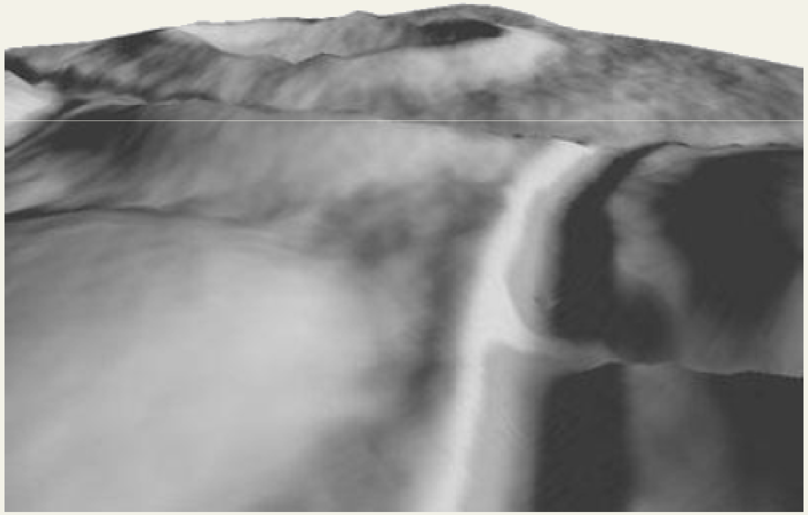
(b) The region potentially visible to a viewer positioned in A – cell region visibility

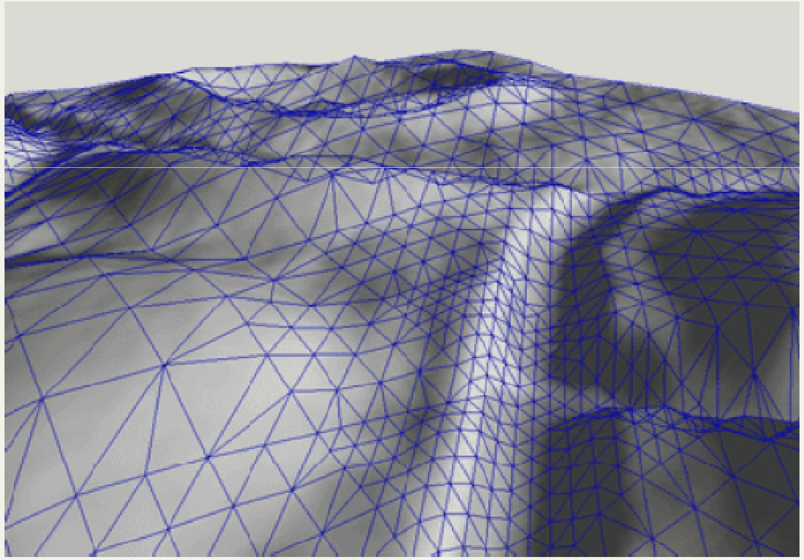


(c) Objects potentially visible from source cell A – the PVS



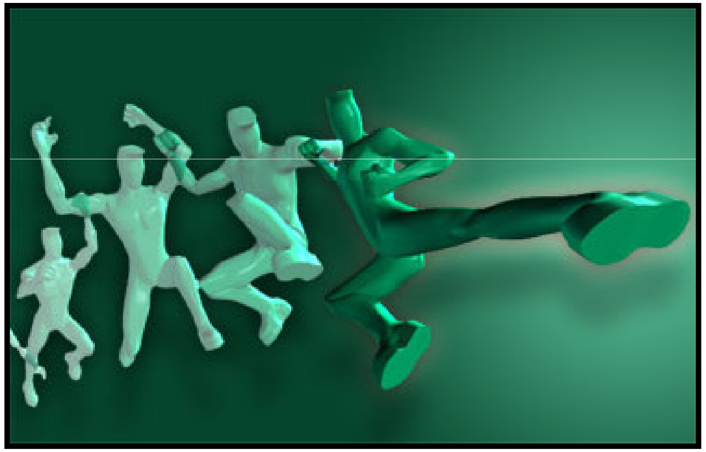
LOD





ECOLE NATIONALE
DU JEU ET DES MEDIAS
INTERACTIFS NUMERIQUES

Game Engines Features (besides Rendering)



Alexandre Topol

ENJMIN
Conservatoire National des Arts & Métiers

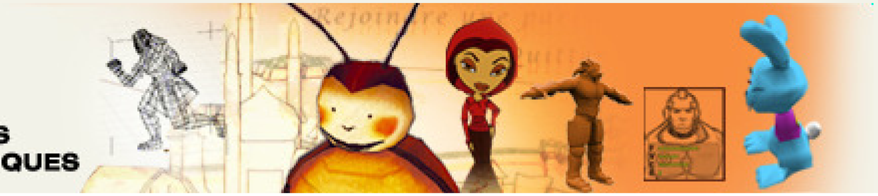
Interactive Programs

Battlefield 2

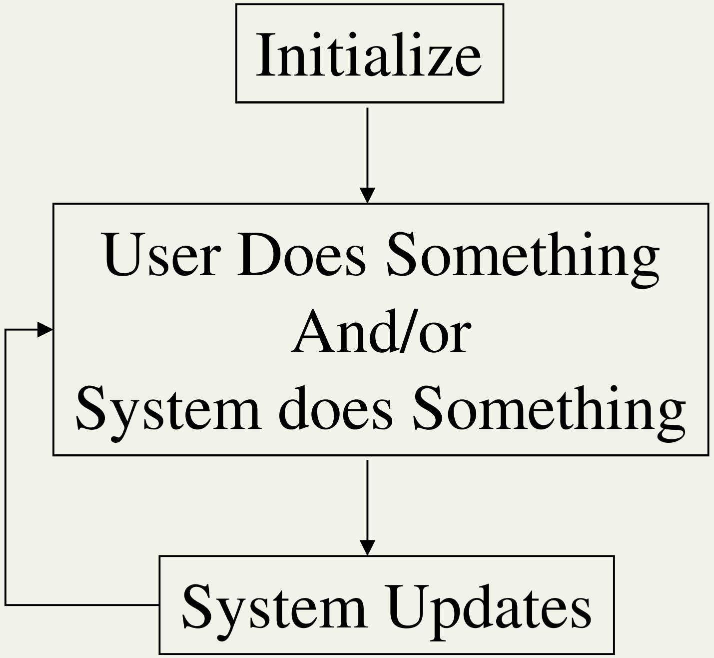
- Games are interactive programs
- Moreover, they are typically immersive in some way
- What are the important features of an interactive program?
- Which features are particularly important for immersive software like games?

Important Features

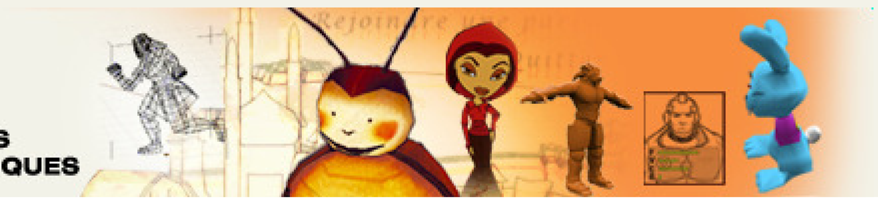
- User controls the action
  - Control is “direct” and “immediate”
- Program provides constant feedback about its state
  - The user must know and understand what is happening
  - The user must receive acknowledgment that their input was received



Interactive Program Structure



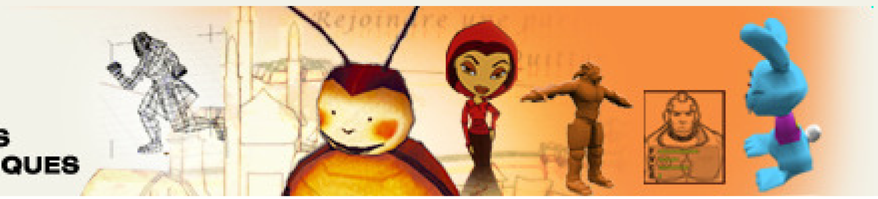
- Event driven programming
  - Everything happens in response to an event
- Events come from two sources:
  - The user
  - The system
- Events are also called *messages*
  - An event causes a message to be sent…



ECOLE NATIONALE
DU JEU ET DES MEDIAS
INTERACTIFS NUMERIQUES

User Events

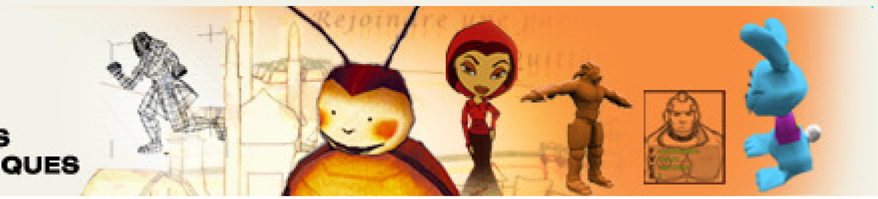
- Usually the OS manages user input
  - Interrupts at the hardware level …
  - Get converted into events in queues at the windowing level …
- It is generally up to the application to make use of the event stream
- User interface toolkits have a variety of methods for managing events
- The game engine gives easier manners to deal with user inputs
- There are two ways to get events: You can ask, or you can be told



Polling for Events

```
while ( not done )
        if ( e = checkEvent() )
                process event
        …
        draw frame
```

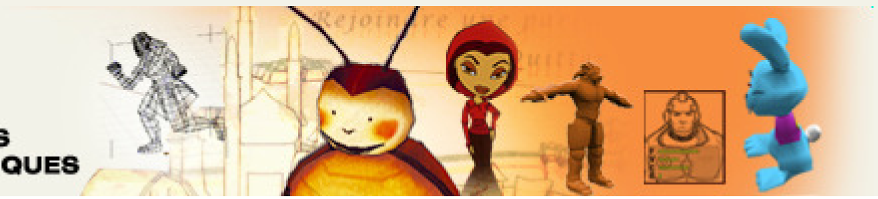
- Most game engines provide a *non-blocking* event query
  - Does not wait for an event, returns immediately if no events are ready
- What type of games might use this structure?
- Why wouldn't you always use it?



Waiting for Events

```
while ( not done )
        e = nextEvent();
        process event
        …
        draw frame
```

- Most game engines provide a blocking event function
  - Waits (blocks) until an event is available
- On what systems is this better than the previous method?
- What types of games is it most useful for?

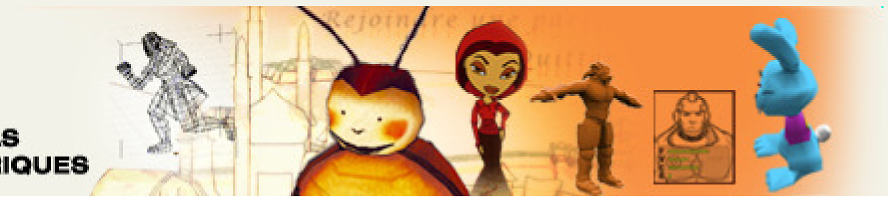


Real-Time Loop

- At the core of games with animation is a real-time loop:

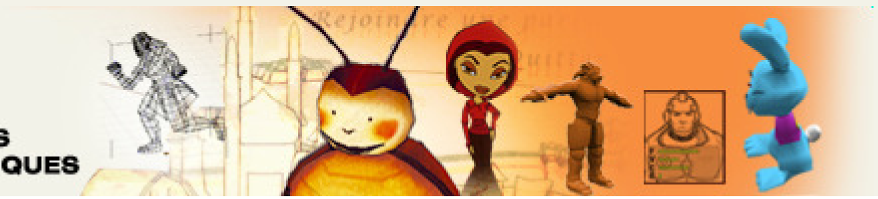
```
while ( true )
        process events
        update animation
        render
```

- What else might you need to do?
- The number of times this loop executes per second is the frame rate
  - # frames per second (fps)

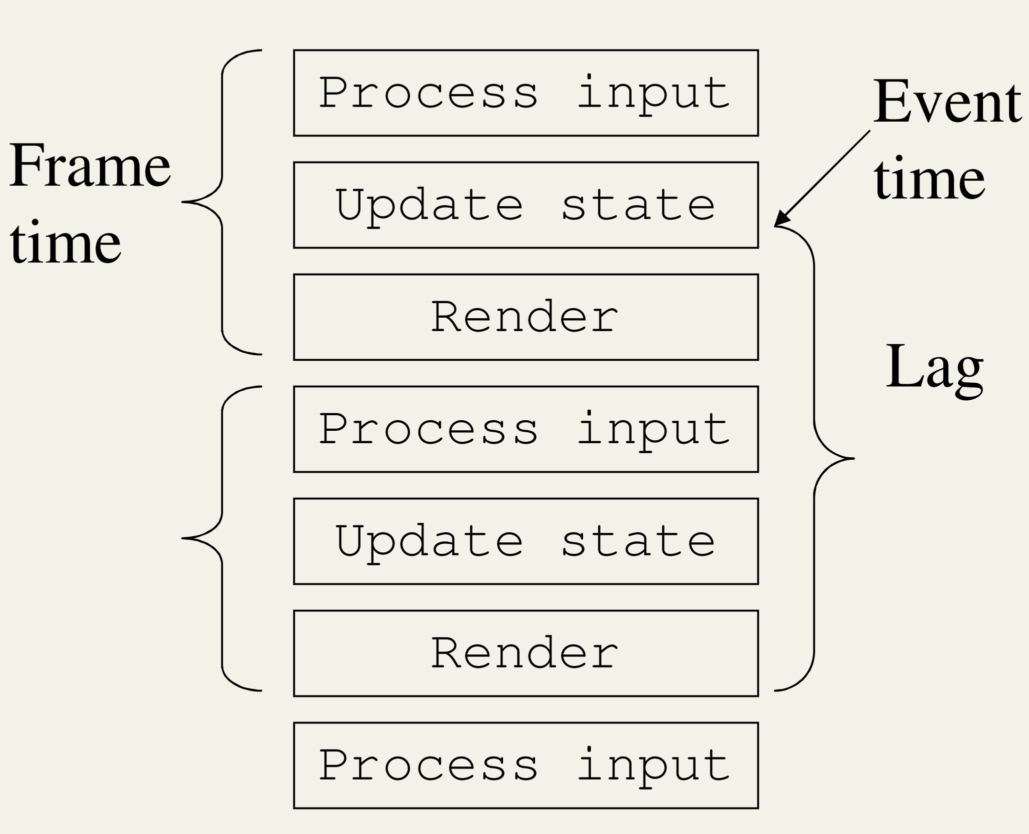


Lag

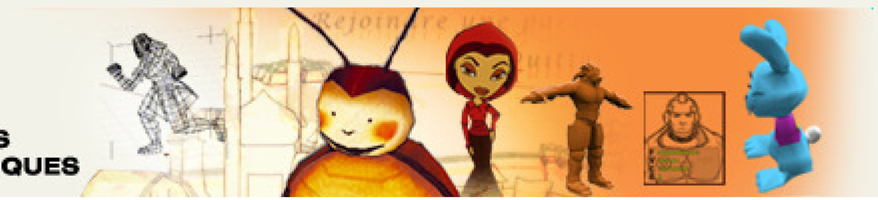
- *Lag* is the time between when a user does something and when they see the result - also called *latency*
  - Too much lag and causality is distorted
  - With tight visual/motion coupling, too much lag makes people motion sick
  - Too much lag makes it hard to target objects (and track them, and do all sorts of other perceptual tasks)
- High variance in lag also makes interaction difficult
  - Users can adjust to constant lag, but not variable lag
- From a psychological perspective, lag is the important variable



Computing Lag

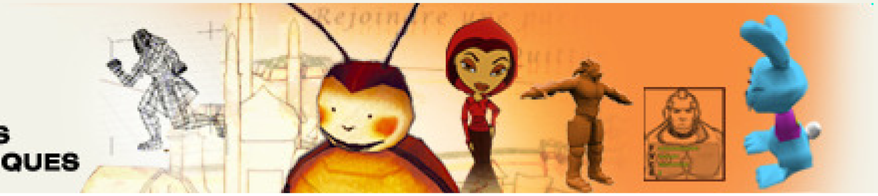


- Lag is NOT the time it takes to compute 1 frame!



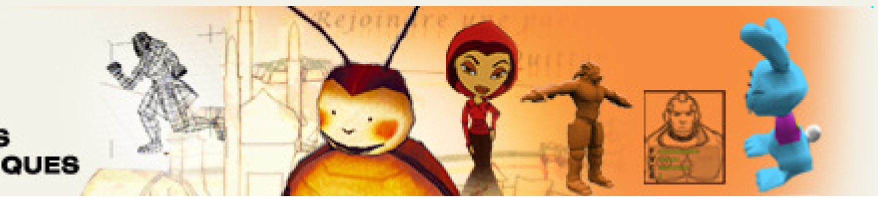
Reducing Lag

- Faster algorithms and hardware is the obvious answer
- Designers choose a frame rate and put as much into the game as they can without going below the threshold
  - Part of design documents presented to the publisher
  - Threshold assumes fastest hardware and all game features turned on
  - Options given to players to reduce game features and improve their frame rate
- There's a resource budget: How much time is dedicated to each aspect of the game (graphics, AI, sound, …)



Decoupling Computation

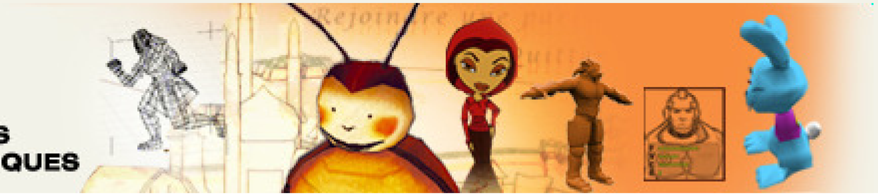
- It is most important to minimize lag between the user actions and their direct consequences
  - So the input/rendering loop must have low latency
- Lag between actions and other consequences may be less severe
  - Time between input and the reaction of enemy can be greater
  - Time to switch animations can be greater
- Technique: Update different parts of the game at different rates, which requires decoupling them
  - For example, run graphics at 60fps, AI at 10fps
  - Very common in real games



Animation

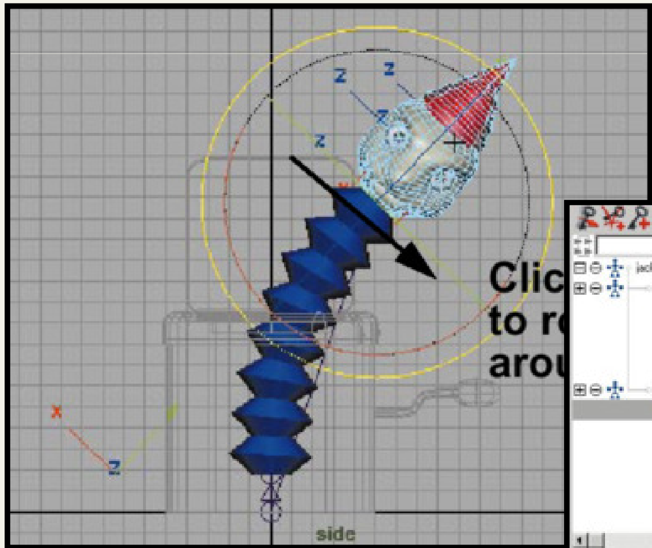
- Key-frame animation
  - Specification by hand
- Motion capture
  - Recording motion
- Procedural / simulation
  - Automatically generated
- Combinations
  - *e.g.* mocap + simulation

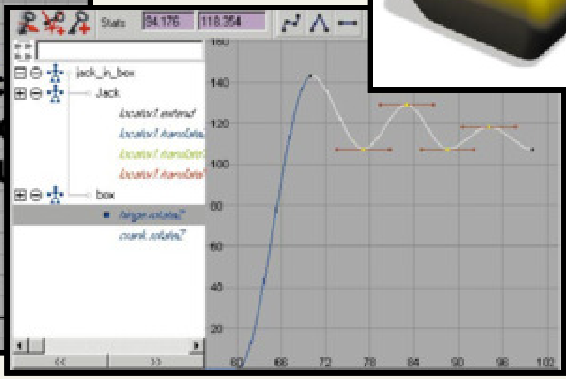
Jeff Lew

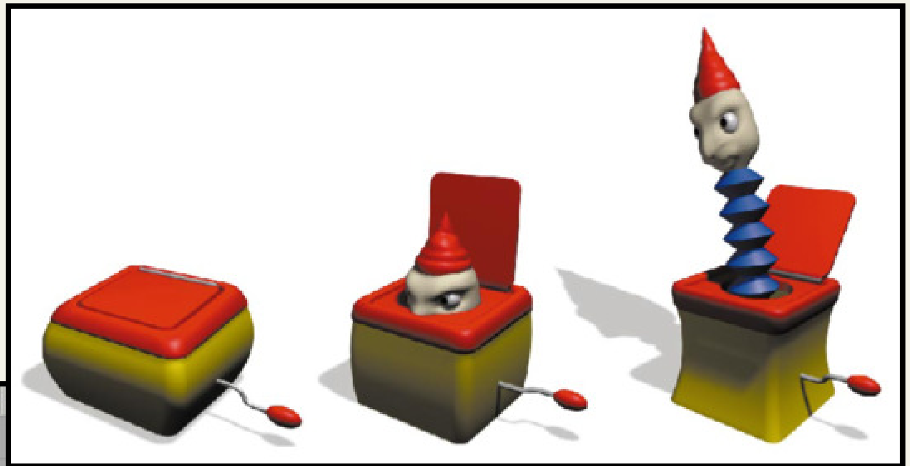


Key-framing (manual)

- Requires a highly skilled user
- Poorly suited for interactive applications
- High quality / high expense
- Limited applicability



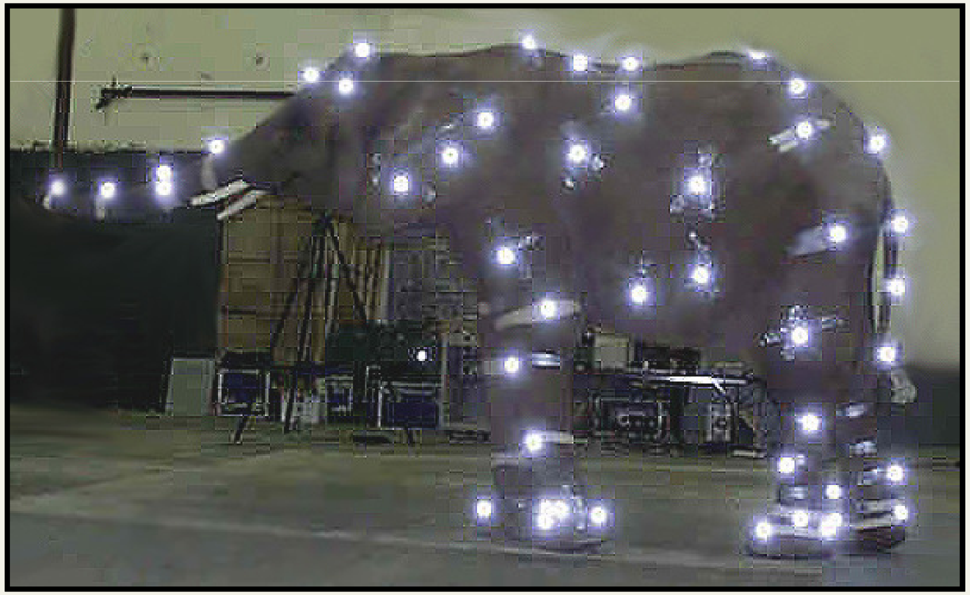


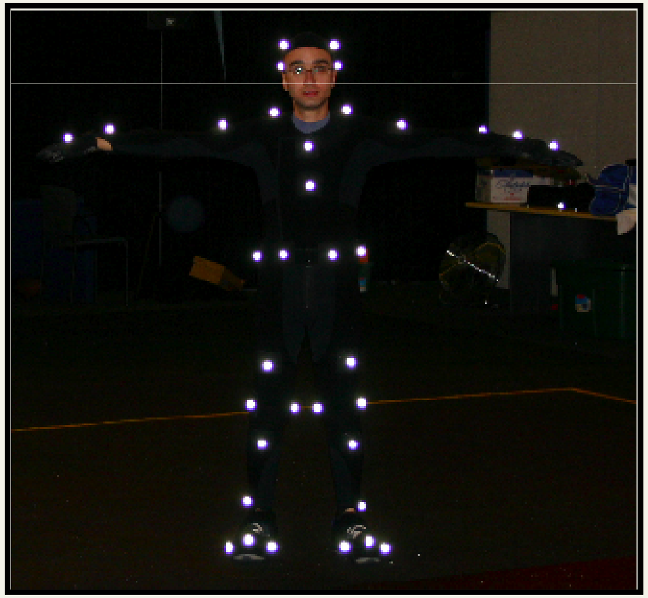


From *Learning Maya 2.0*

Motion Capture (recorded)

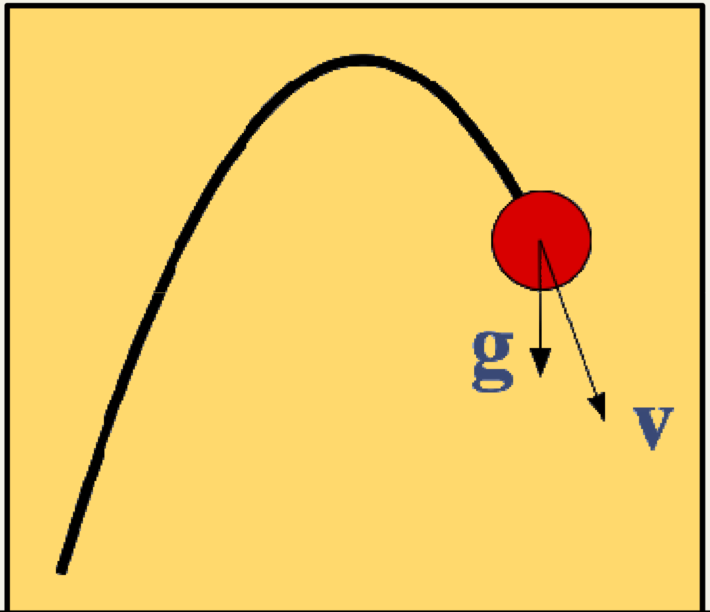
- Markers/sensors placed on subject
- Time-consuming clean-up
- Reasonable quality / reasonable price
- Manipulation algorithms an active research area



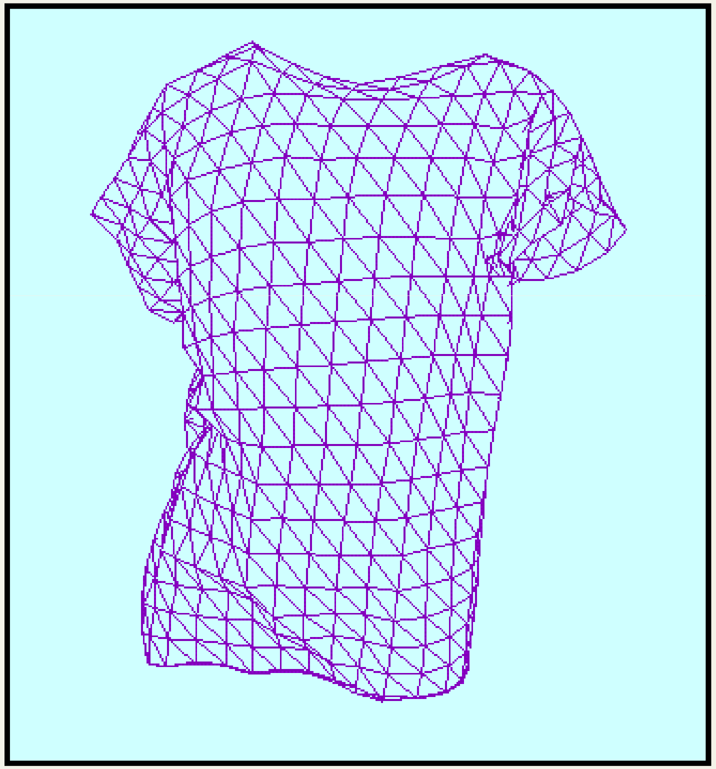


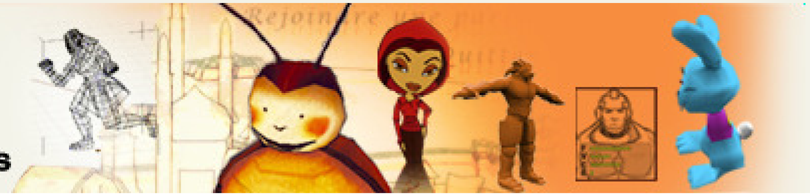
Simulation

- Generate motion of objects using numerical simulation methods



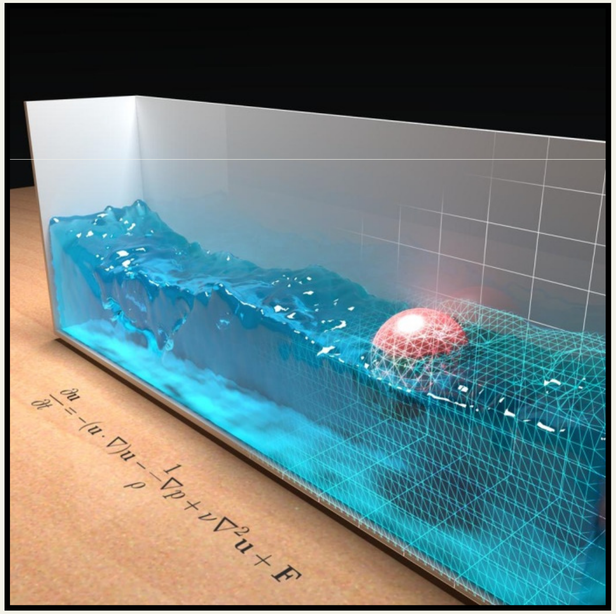
$$\mathbf{x}^{t+\Delta t} = \mathbf{x}^{t} + \Delta t\,\mathbf{v}^{t} + \frac{1}{2}\Delta t^{2}\,\mathbf{a}^{t}$$





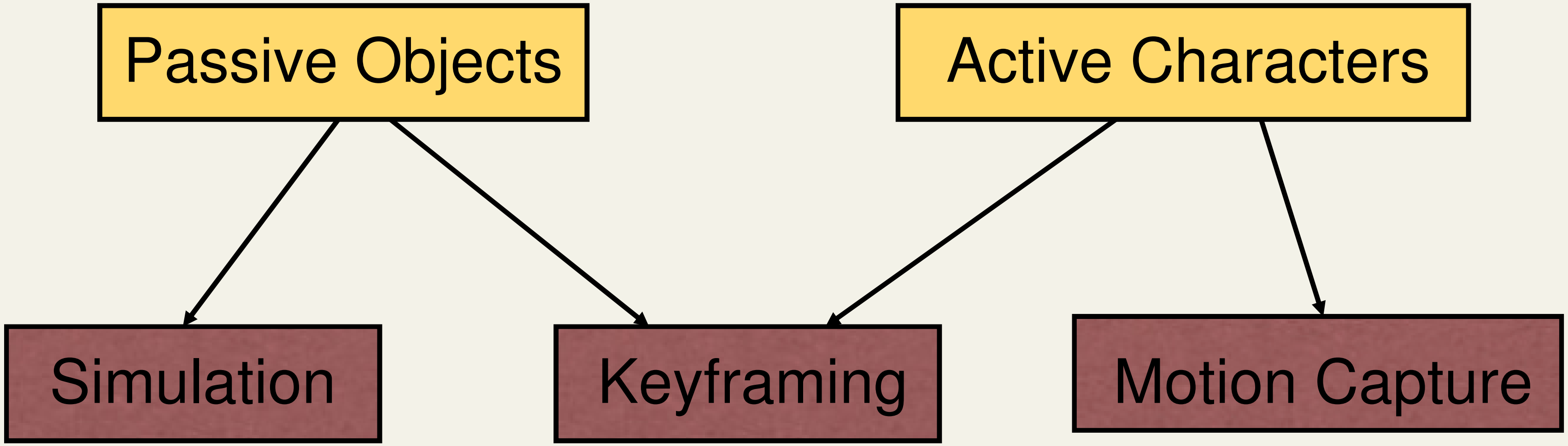
Simulation

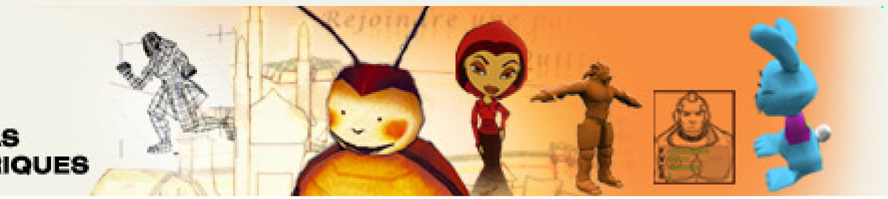
- Perceptual accuracy required
- Stability, easy of use, speed, robustness all important
- Control desirable



IGG

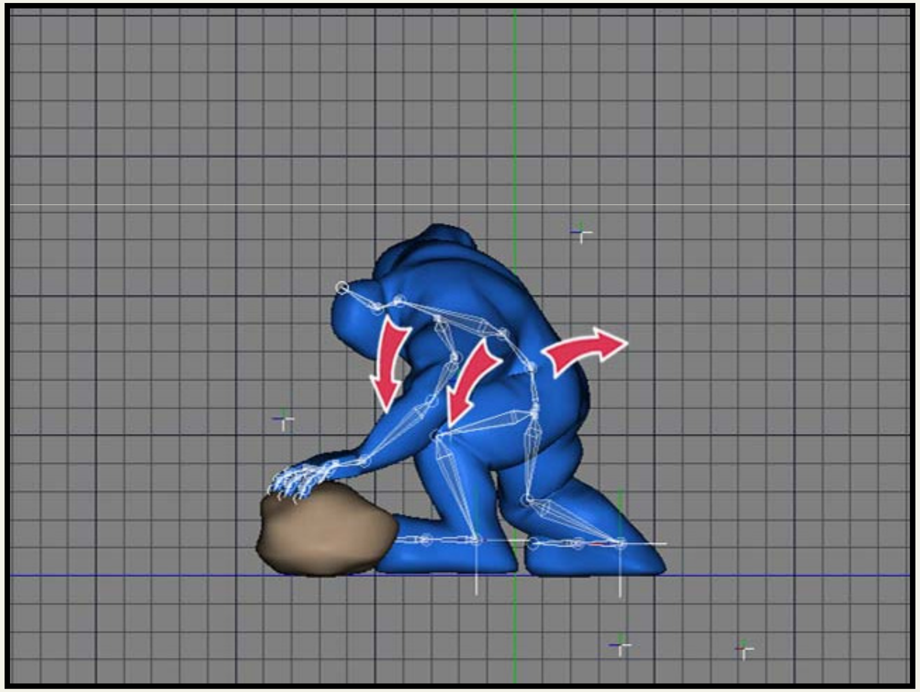
Animation


Passive Objects
Active Characters
Simulation
Keyframing
Motion Capture



Character Animation

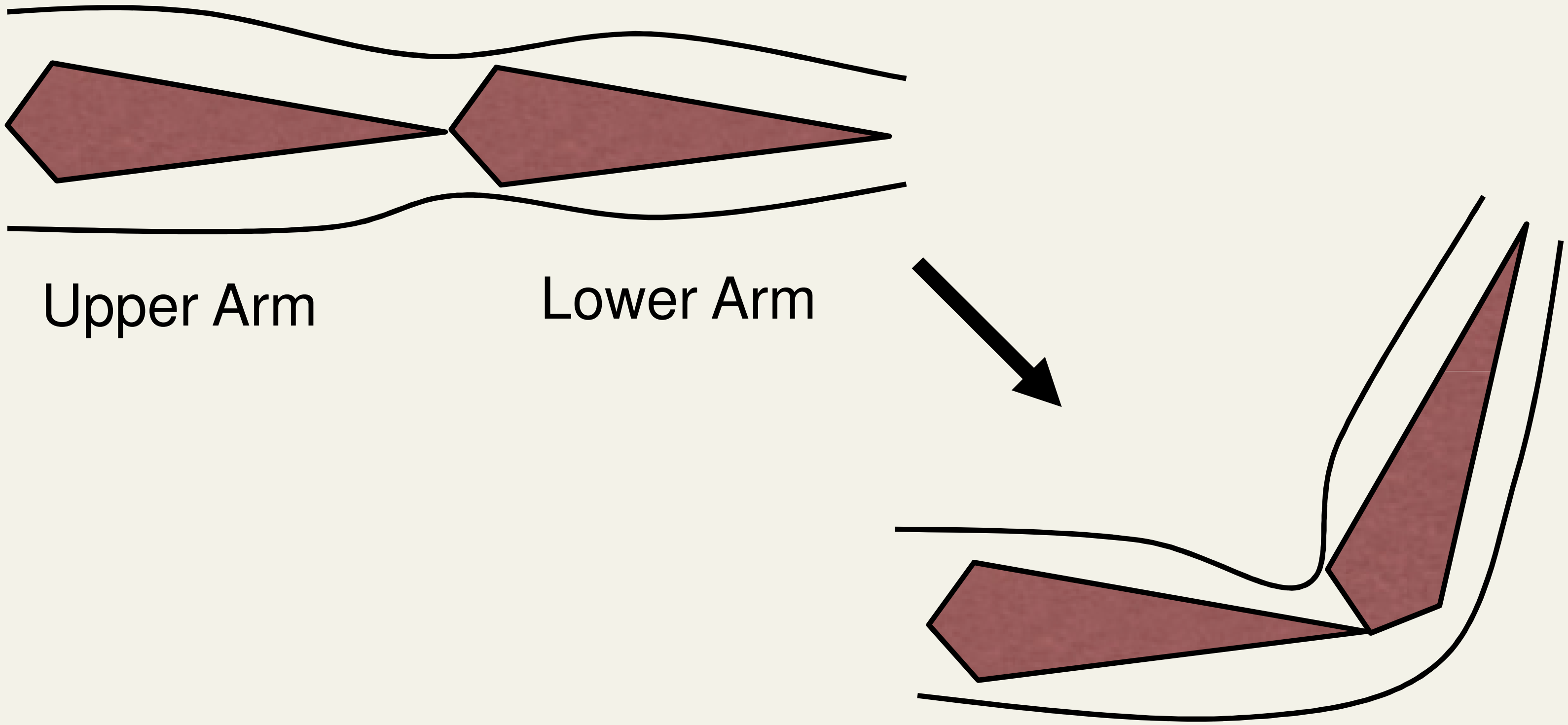
- Key Frame or Motion Capture
- Usually skeletal animation based
- Two Components:
  - Skeleton motion
  - Skin move

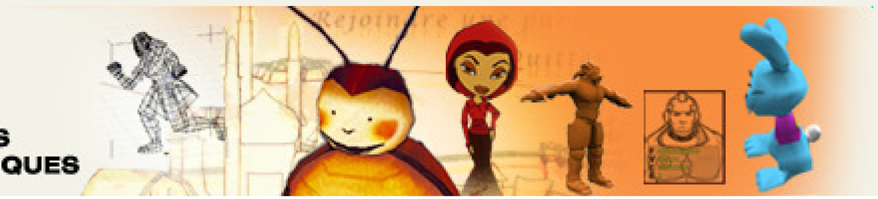


Jeff Lew

ENJMIN

Character Animation

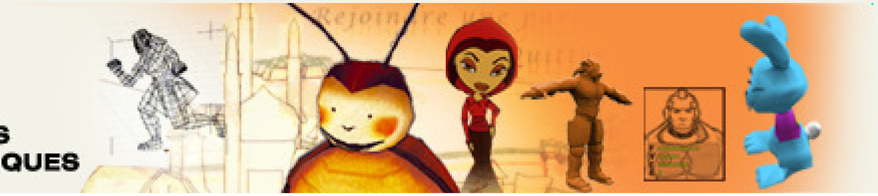




Character Animation

- Skin motion
  - Represent vertices in bone coordinate systems
  - Move bone coordinate systems
- Skeleton motion
  - Parameterized by the joint angles

Jeff Lew

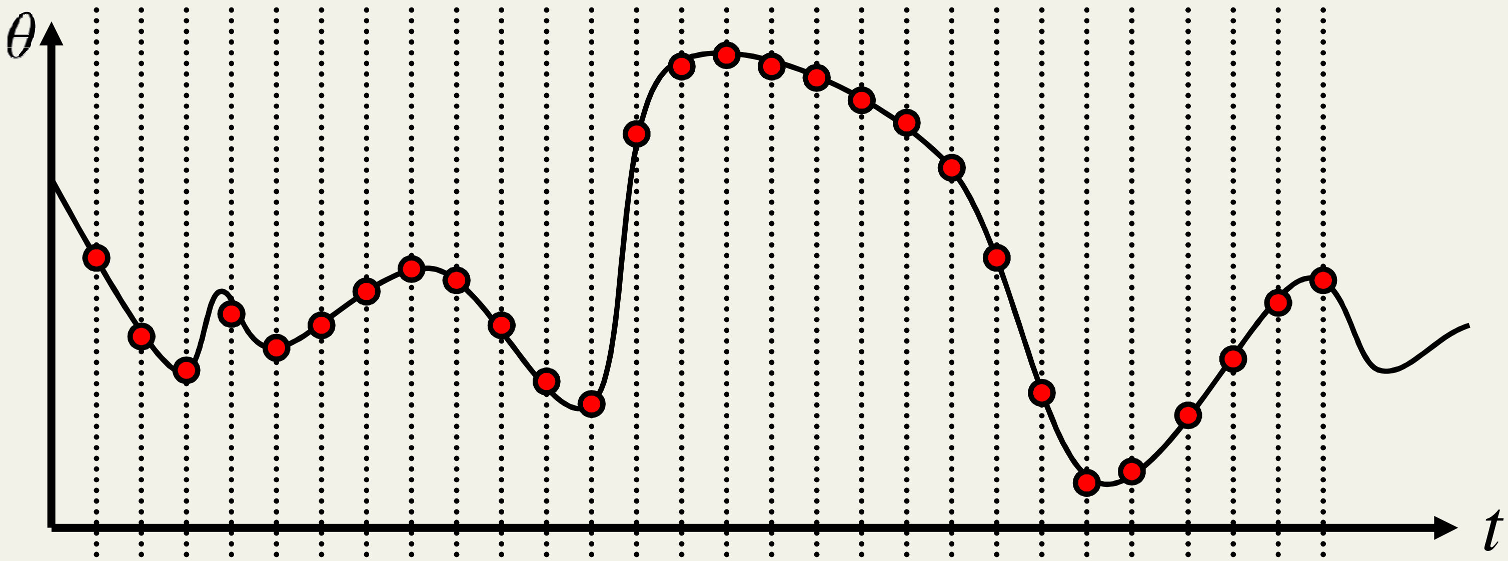


Character Animation

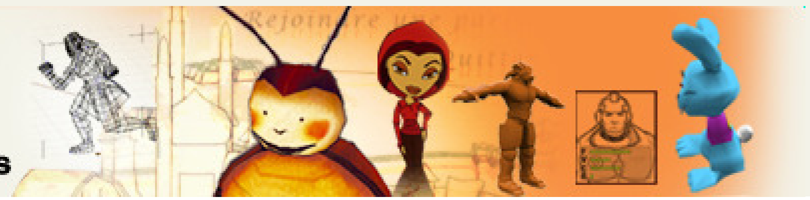
- Skeletal motion is a function of time

$$\theta = f(t)$$

- Representing this function

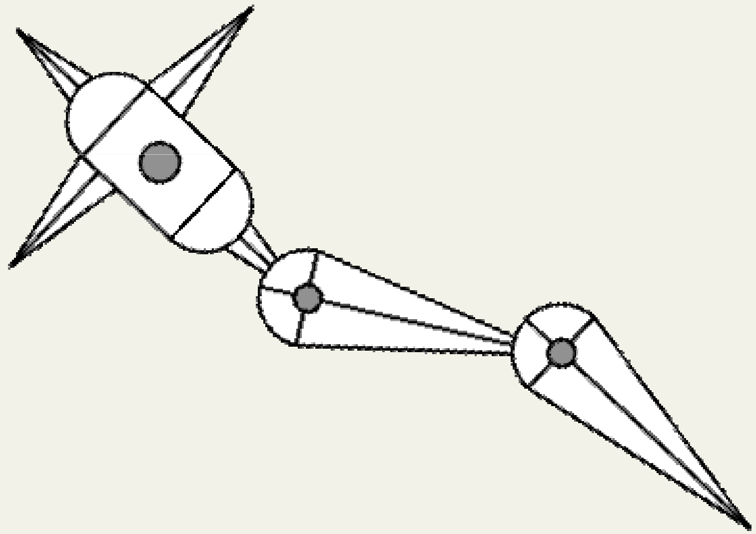


$$\theta_1 = f(t_1)$$
$$\theta_2 = f(t_2)$$
$$\theta_3 = f(t_3)$$
$$\dots$$
$$\theta_n = f(t_n)$$



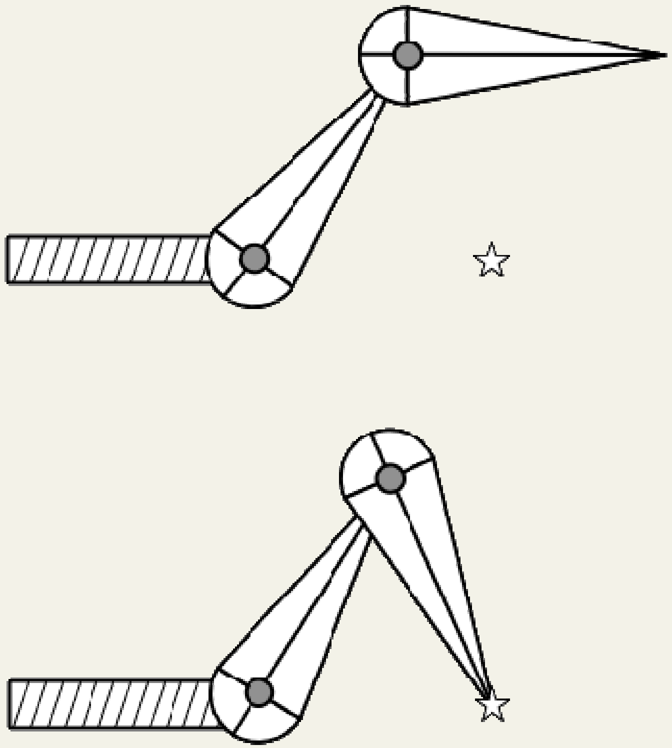
Forward Kinematics

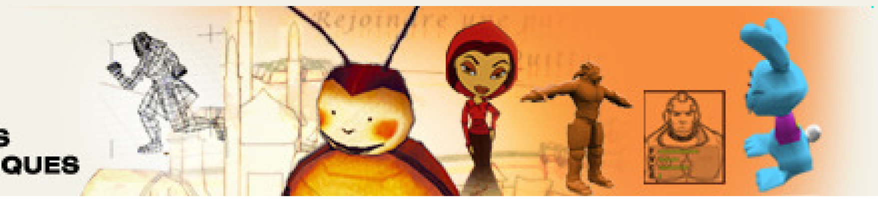
- Composite transformations down the hierarchy



Inverse Kinematics

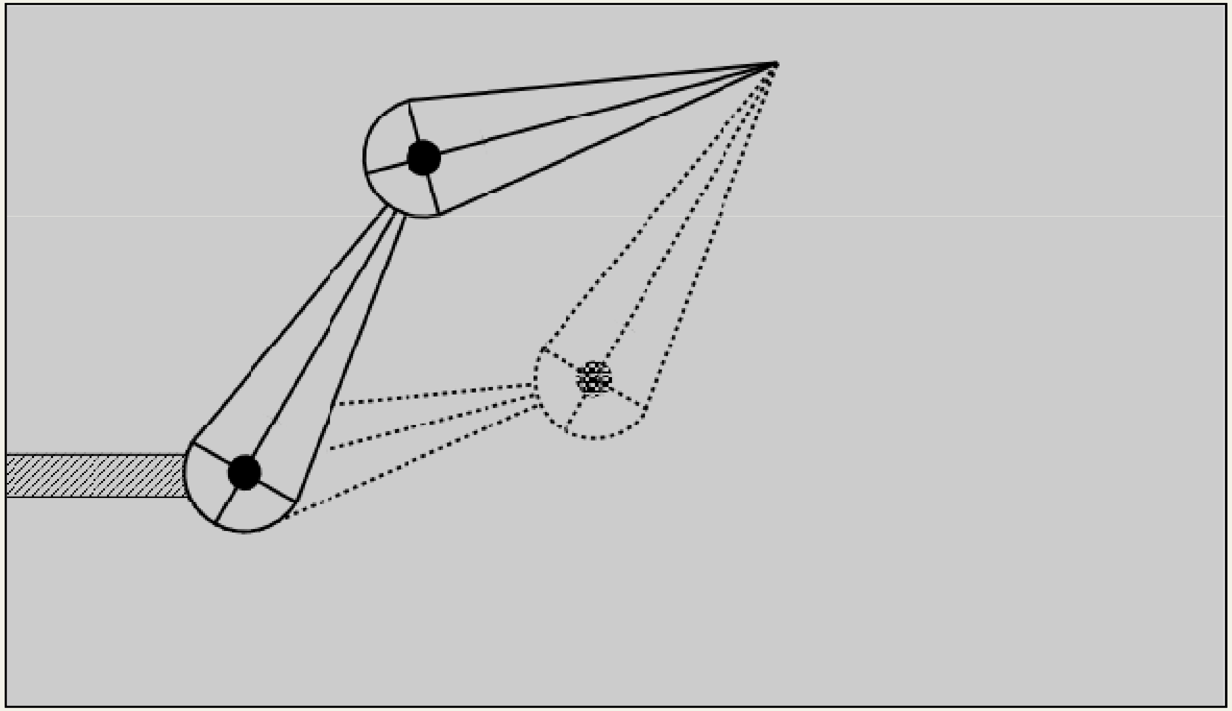
- Given
  - Root transformation
  - Initial configuration
  - Desired end point location
- Find
  - Interior parameter settings





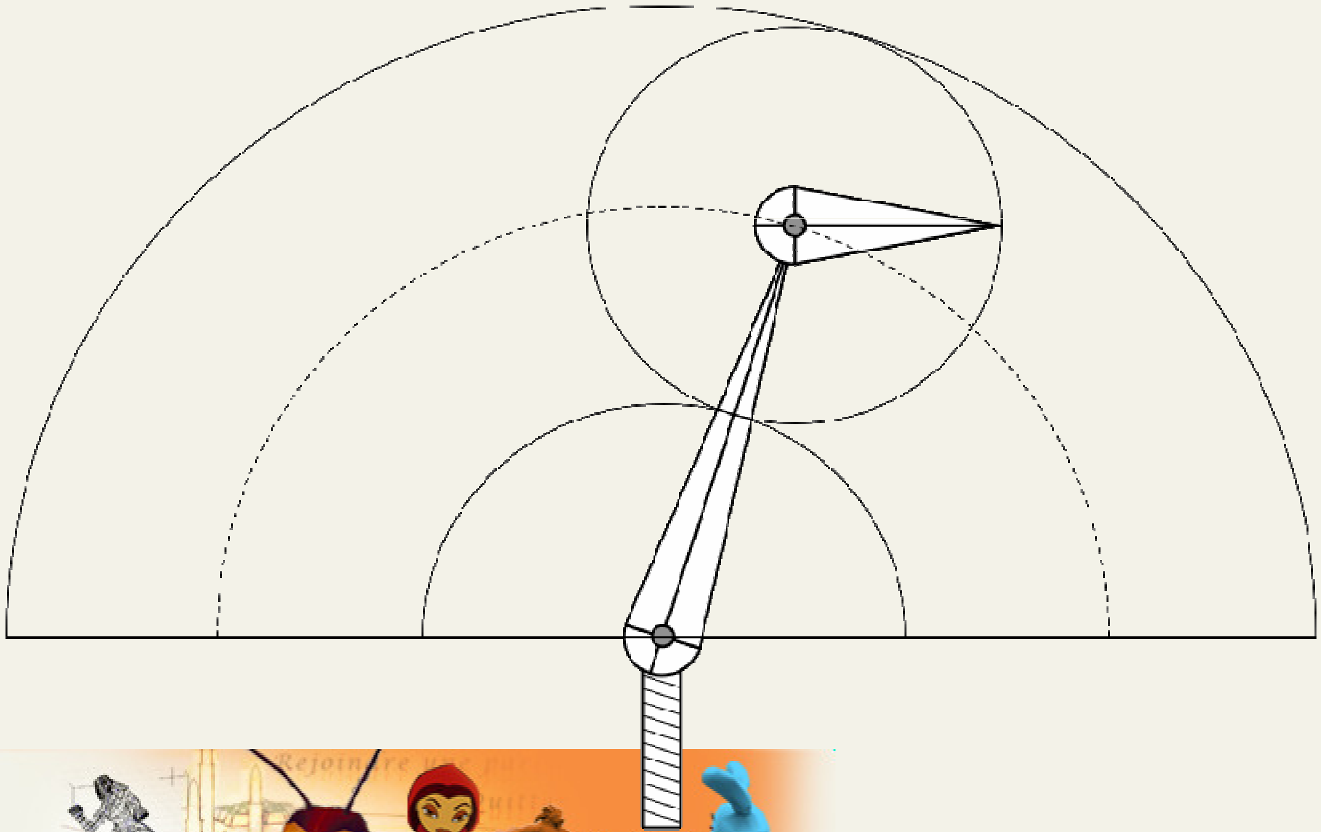
Inverse Kinematics

- Why is the problem hard?
  - Multiple solutions separated in configuration space

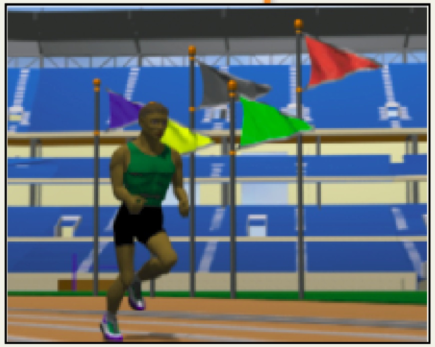


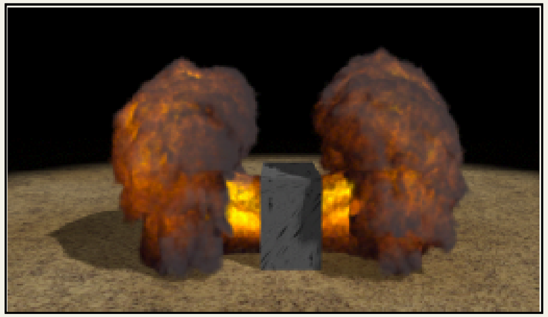
Inverse Kinematics

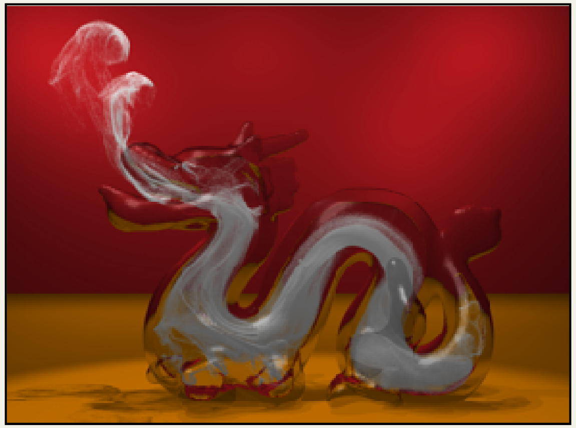
- Why is the problem hard?
  - Solutions may not always exist

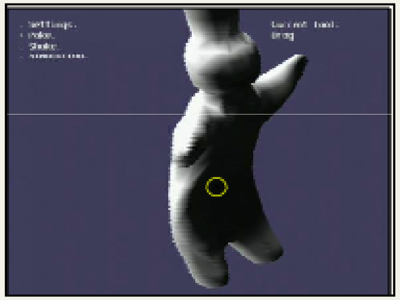


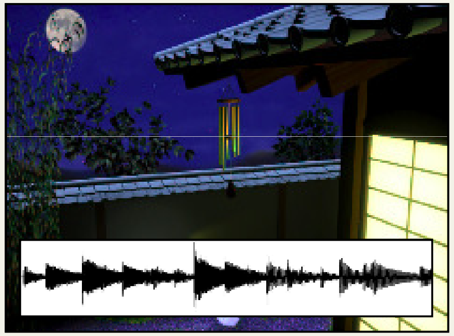
Physically Based Animation

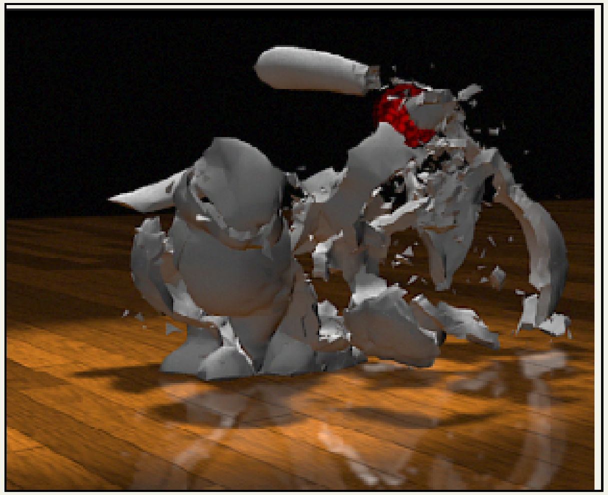


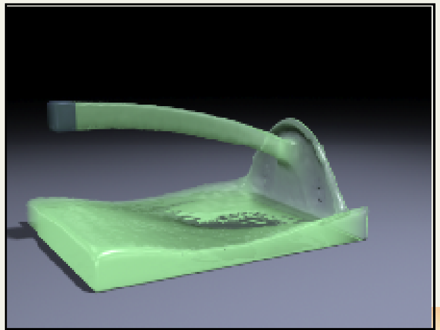


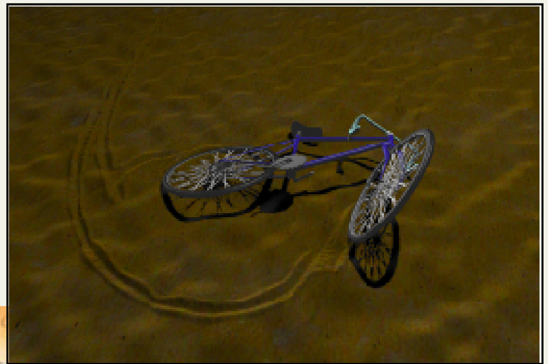


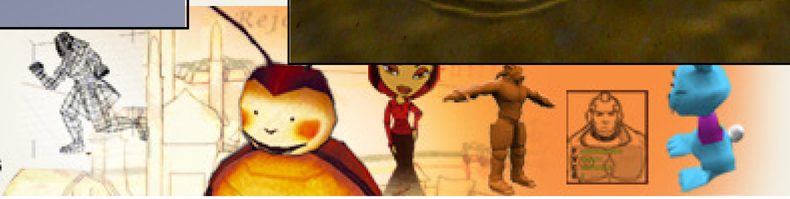












Physically Based Animation in Games

Half Life 2

Max Payne 2

Fuel

Black

Particle Systems

- Single particles are very simple
- Large groups can produce interesting effects
- Supplement basic ballistic rules
  - Collisions
  - Interactions
  - Force fields
  - Springs
  - Others...

Feldman, Klingner, O'Brien
SIGGRAPH 2005

Karl Sims
SIGGRAPH 1990

Basic Particles

$$\ddot{x} = \frac{1}{m}f$$

- Basic governing equation
  - is a sum of a number of things
    - Gravity: constant downward force proportional to mass
    - Simple drag: force proportional to negative velocity
    - Particle interactions: particles mutually attract and/or repel
      - Beware $O(n^2)$ complexity!
    - Force fields
    - Wind forces
    - User interaction

Spring Mass Systems

- Masses connected by springs

✸Can be used to model

⇨Deformable objects

⇨Cloth

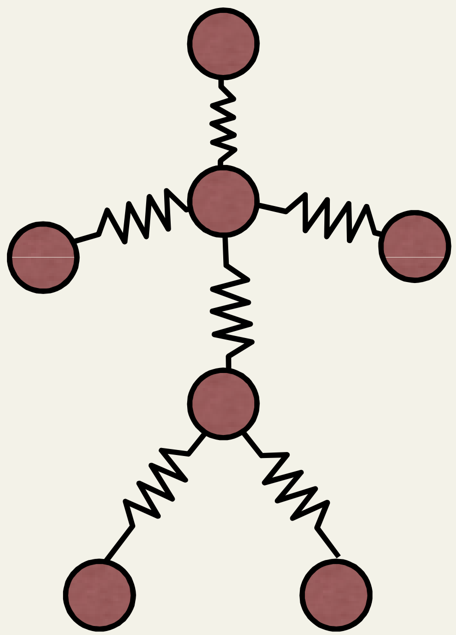
⇨Hair

⇨Rigid bodies

Hitman

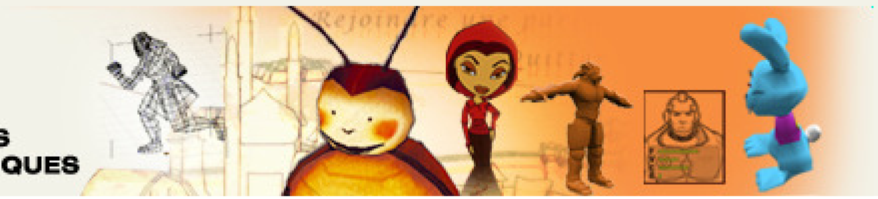
Spring Mass Systems

- Concrete example
  - The state
    - Position, velocity
  - The forces
    - Gravity, springs
  - The integration



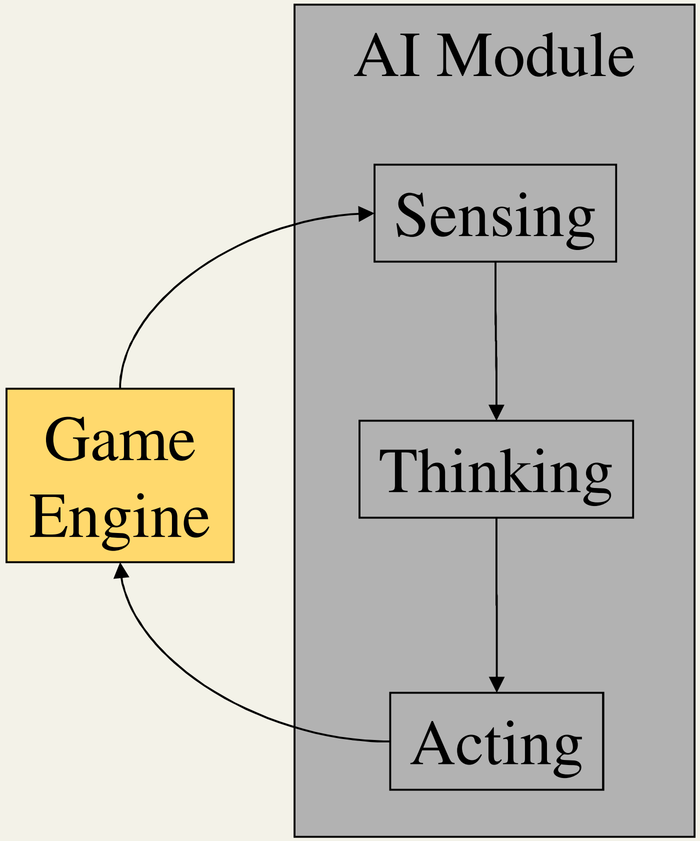
AI in the Game Loop

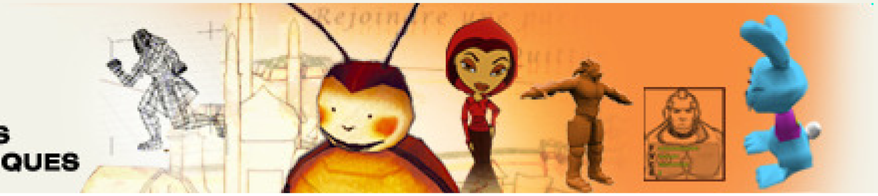
- AI is updated as part of the game loop, after user input, and before rendering
- There are issues here:
  - Which AI goes first?
  - Does the AI run on every frame?
  - Is the AI synchronized?



AI Update Step

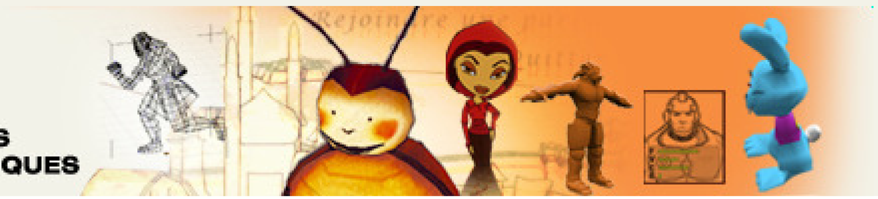
- The sensing phase determines the state of the world
  - May be very simple - state changes all come by message
  - Or complex - figure out what is visible, where your team is, etc
- The thinking phase decides what to do given the world
  - The core of AI
- The acting phase tells the animation what to do
  - Generally not interesting





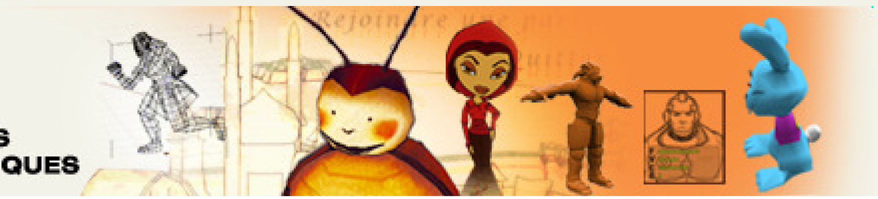
AI by Polling

- The AI gets called at a fixed rate
- Senses: It looks to see what has changed in the world. For instance:
  - Queries what it can see
  - Checks to see if its animation has finished running
- And then acts on it
- Why is this generally inefficient?



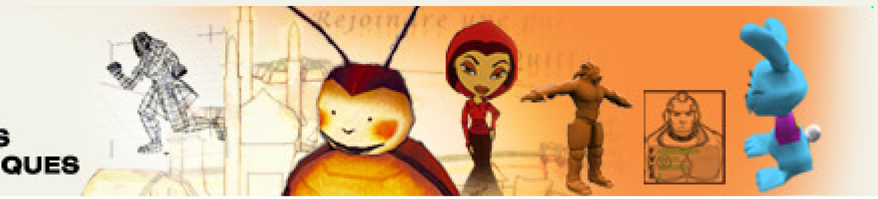
Event Driven AI

- Event driven AI does everything in response to events in the world
  - Events sent by message (basically, a function gets called when a message arrives, just like a user interface)
- Example messages:
  - A certain amount of time has passed, so update yourself
  - You have heard a sound
  - Someone has entered your field of view
- Note that messages can completely replace sensing, but typically do not. Why not?
  - Real system are a mix - something changes, so you do some sensing



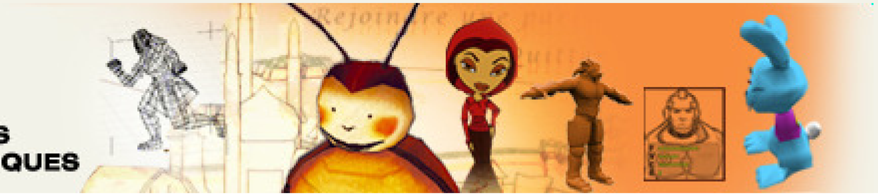
AI Techniques in Games

- Basic problem: Given the state of the world, what should I do?
- A wide range of solutions in games:
  - Finite state machines, Decision trees, Rule based systems, Neural networks, Fuzzy logic, …
- Even a wider range of solutions in the academic world:
  - Complex planning systems, logic programming, genetic algorithms, Bayes-nets, …
  - Typically, too slow for games



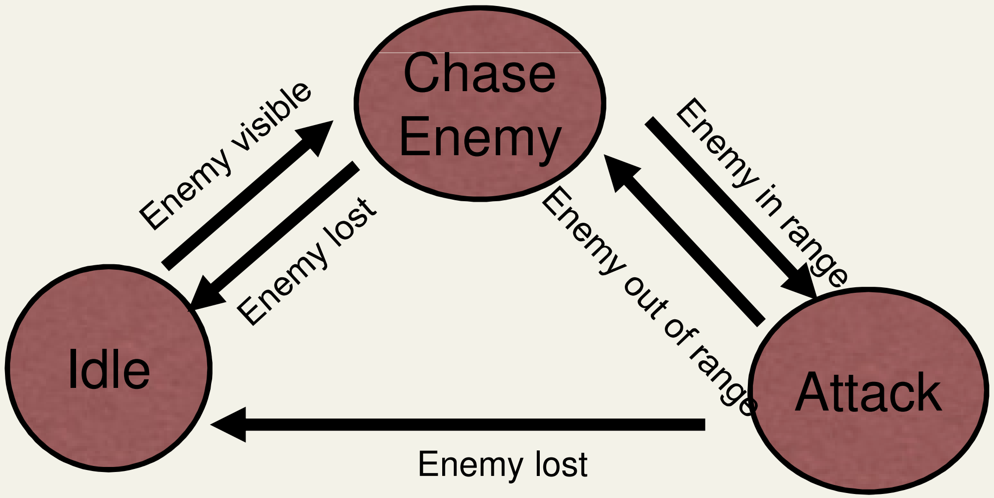
Goals of Game AI

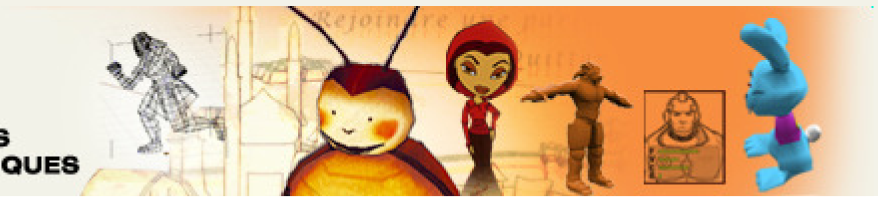
- Several goals:
  - Goal driven - the AI decides what it should do, and then figures out how to do it
  - Reactive - the AI responds immediately to changes in the world
  - Knowledge intensive - the AI knows a lot about the world and how it behaves, and embodies knowledge in its own behavior
  - Characteristic - Embodies a believable, consistent character
  - Fast and easy development
  - Low CPU and memory usage
- These conflict in almost every way



Finite State Machines (FSMs)

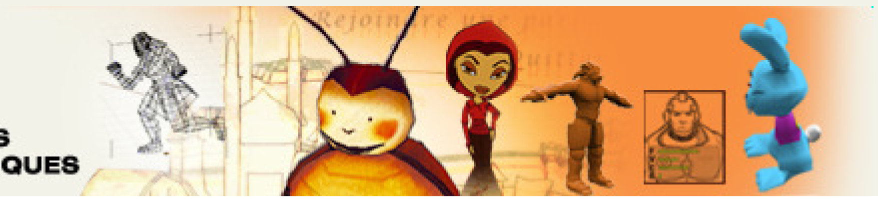
- A set of *states* that the agent can be in
- Connected by *transitions* that are triggered by a change in the world
- Normally represented as a directed graph, with the edges labeled with the transition event
- Ubiquitous in computer game AI





Quake Bot Example

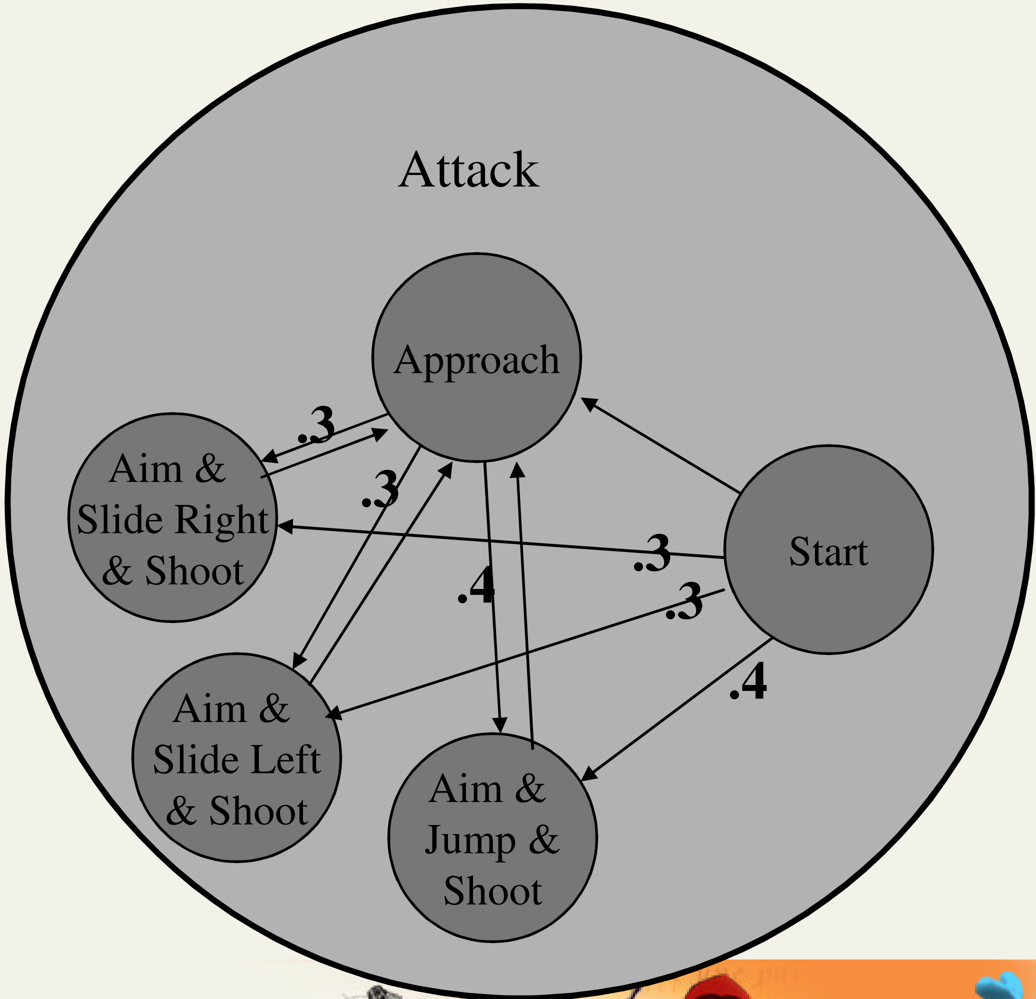
- Types of behavior to capture:
  - Wander randomly if don't see or hear an enemy
  - When see enemy, attack
  - When hear an enemy, chase enemy
  - When die, respawn
  - When health is low and see an enemy, retreat
- Extensions:
  - When see power-ups during wandering, collect them
- Borrowed from John Laird and Mike van Lent's GDC tutorial



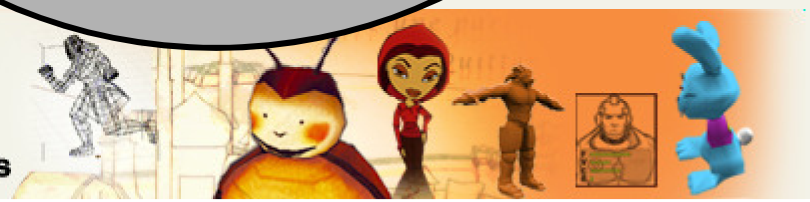
Hierarchical FSMs

- What if there is no simple action for a state?
- Expand a state into its own FSM, which explains what to do if in that state
- Some events move you around the same level in the hierarchy, some move you up a level
- When entering a state, have to choose a state for it’s child in the hierarchy
  - Set a default, and always go to that
  - Or, random choice
  - Depends on the nature of the behavior

Non-Deterministic Hierarchical FSM (Markov Model)

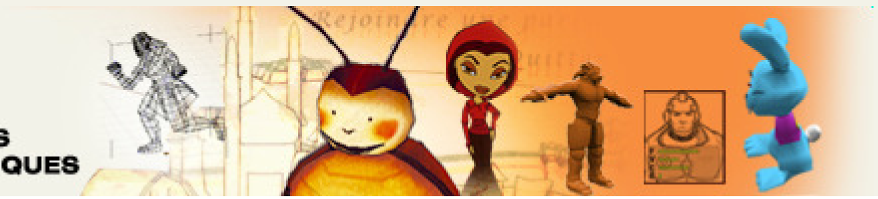


- Adds variety to actions
- Have multiple transitions for the same event
- Label each with a probability that it will be taken
- Randomly choose a transition at run-time
- Markov Model: New state only depends on the previous state



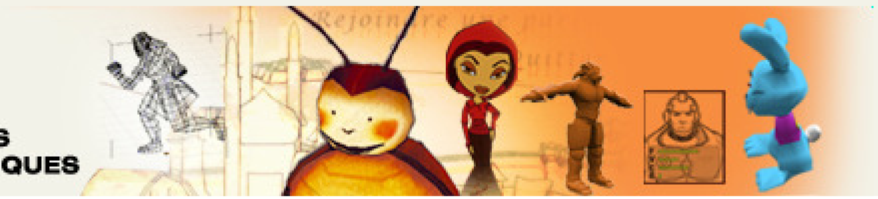
FSM Advantages

- Very fast – one array access
- Expressive enough for simple behaviors or characters that are intended to be “dumb”
- Can be compiled into compact data structure
  - Dynamic memory: current state
  - Static memory: state diagram – array implementation
- Can create tools so non-programmer can build behavior
- Non-deterministic FSM can make behavior unpredictable



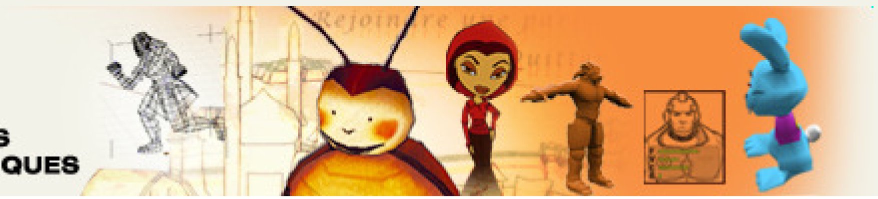
FSM Disadvantages

- Number of states can grow very fast
  - Exponentially with number of events: $s=2^e$
- Number of arcs can grow even faster: $a=s^2$
- Propositional representation
  - Difficult to put in “pick up the better powerup”, “attack the closest enemy”
  - Expensive to count: Wait until the third time I see enemy, then attack
    - Need extra events: First time seen, second time seen, and extra states to take care of counting



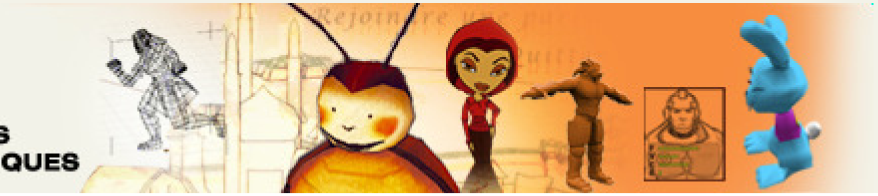
Path Finding

- Very common problem in games:
  - In FPS: How does the AI get from room to room?
  - In RTS: User clicks on units, tells them to go somewhere. How do they get there? How do they avoid each other?
  - Chase games, sports games, …
- Very expensive part of games
  - Lots of techniques that offer quality, robustness, speed trade-offs
- A* usual solution



Path Finding Problem

- Problem Statement (Academic): Given a start point, A, and a goal point, B, find a path from A to B that is clear
  - Generally want to minimize a cost: distance, travel time, …
    - Travel time depends on terrain, for instance
  - May be complicated by dynamic changes: paths being blocked or removed
  - May be complicated by unknowns – don't have complete information
- Problem Statement (Games): Find a reasonable path that gets the object from A to B
  - Reasonable may not be optimal – not shortest, for instance
  - It may be OK to pass through things sometimes
  - It may be OK to make mistakes and have to backtrack



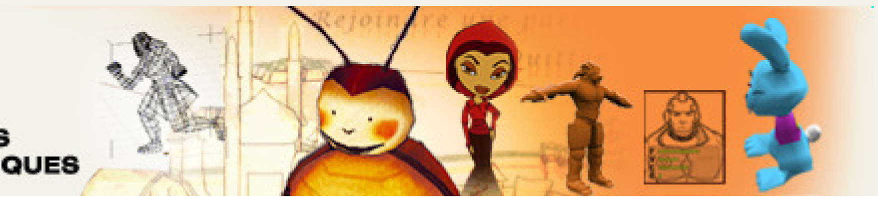
A* Problems

- Discrete Search:
  - Must have simple paths to connect waypoints
    - Typically use straight segments
    - Have to be able to compute cost
    - Must know that the object will not hit obstacles
  - Leads unnatural paths
    - Infinitely sharp corners
    - Funny paths across grids
- Efficiency is not great

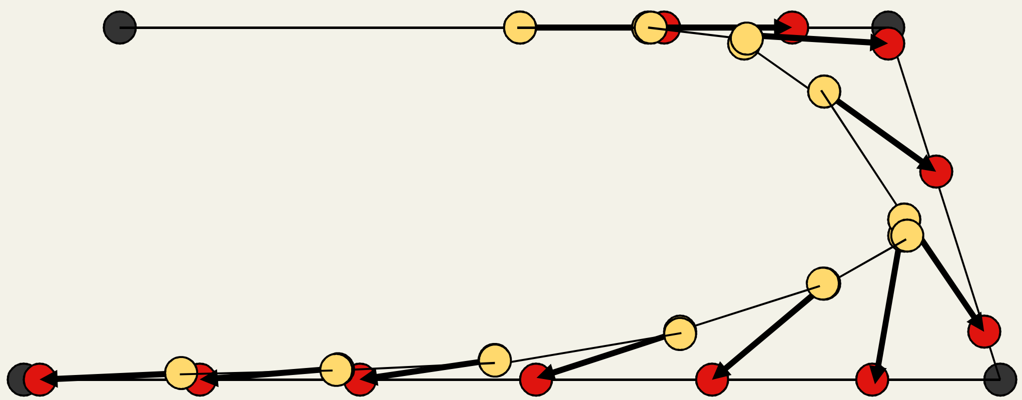
Starcraft

Chase the Point

- Instead of tracking along the path, the agent chases a target point that is moving along the path
- Start with the target on the path ahead of the agent
- At each step:
  - Move the target along the path using linear interpolation
  - Move the agent toward the point location, keeping it a constant distance away or moving the agent at the same speed
- Works best for driving or flying games

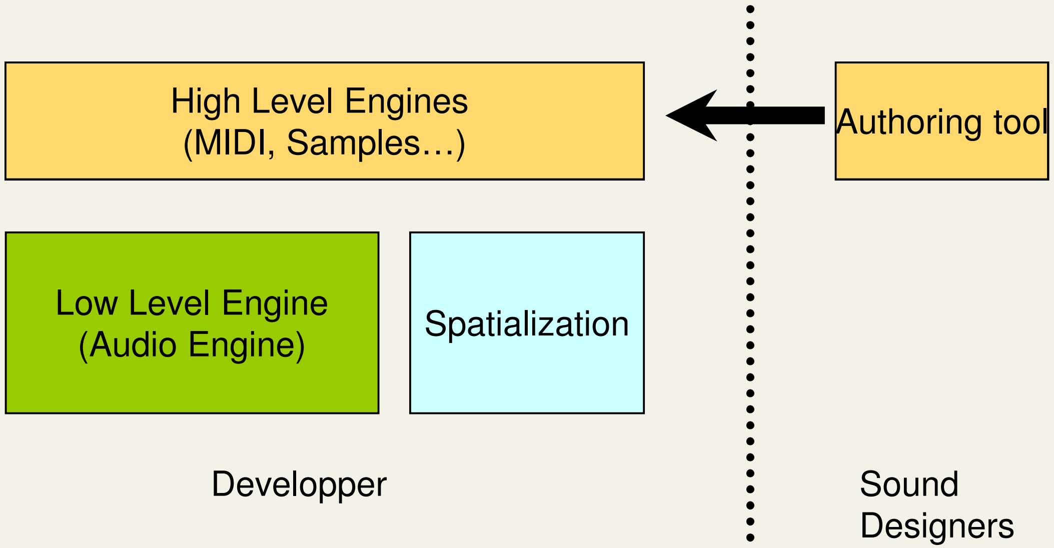


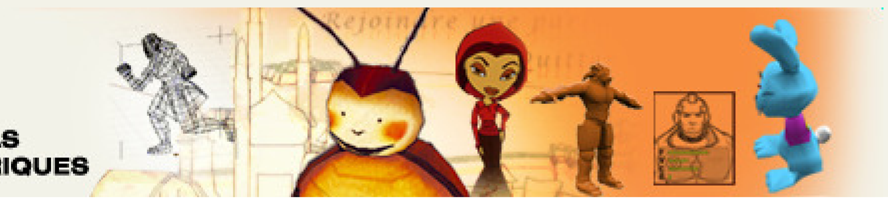
Chase the Point Demo



Audio

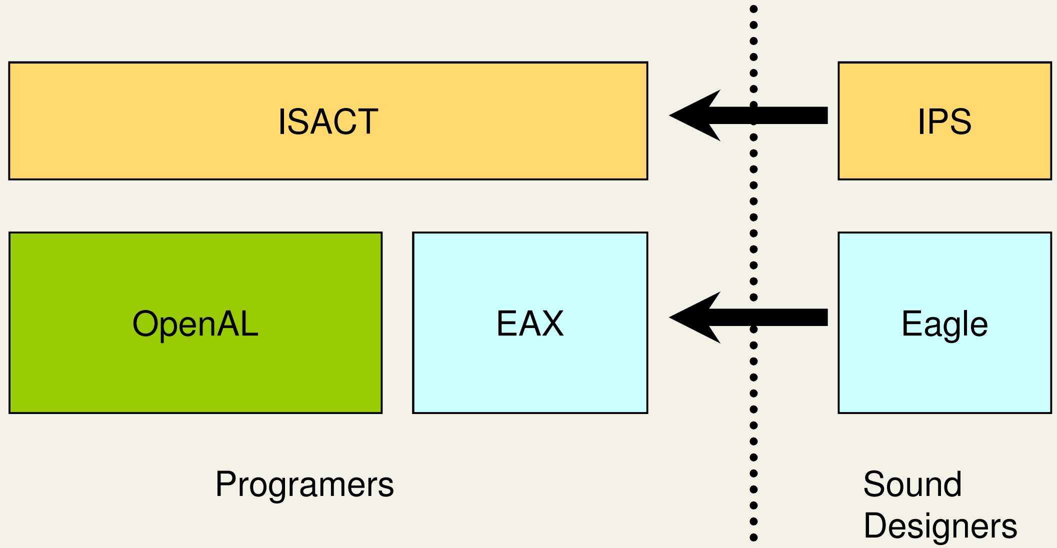
- Generic Engine

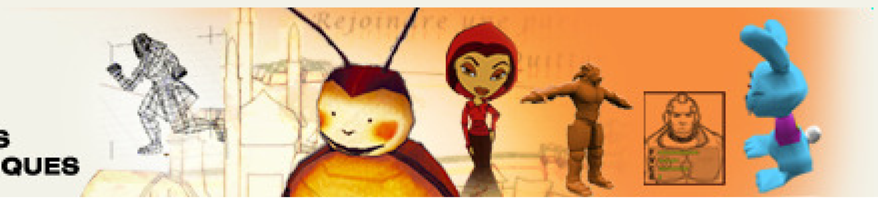




Audio

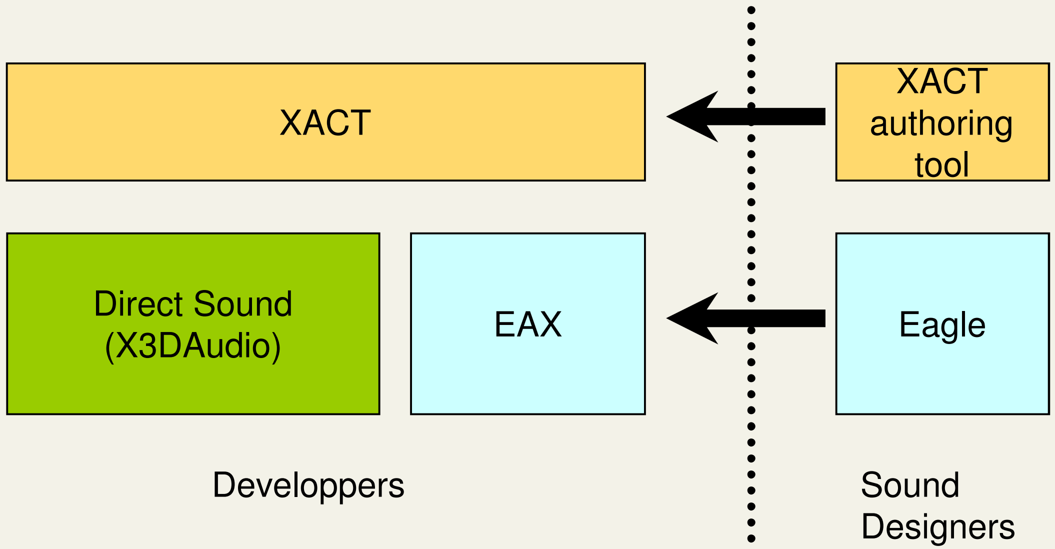
- Creative Labs





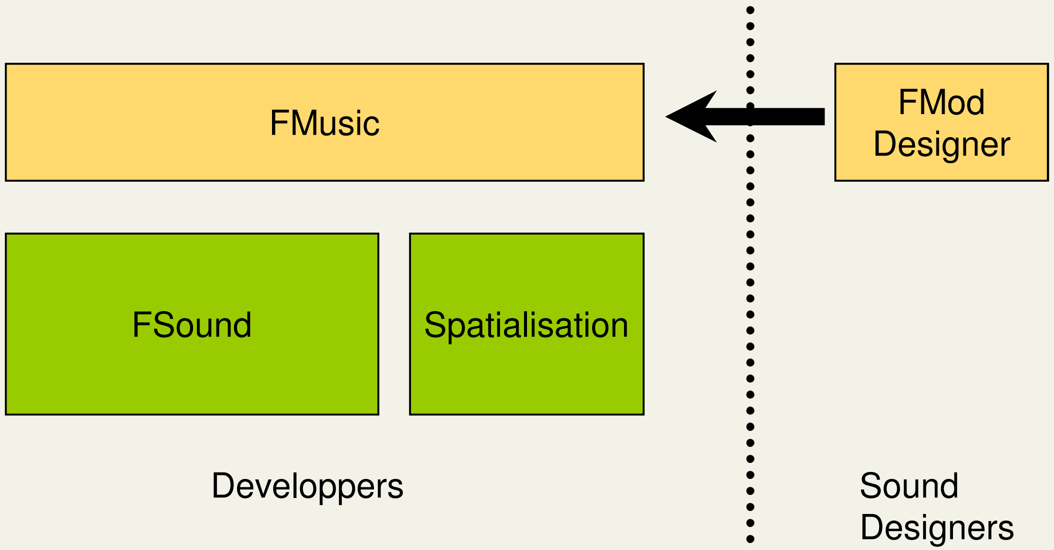
Audio

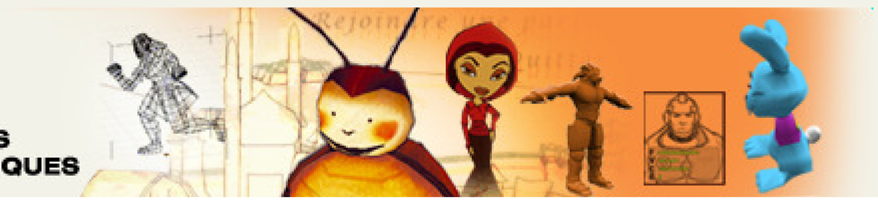
- Microsoft



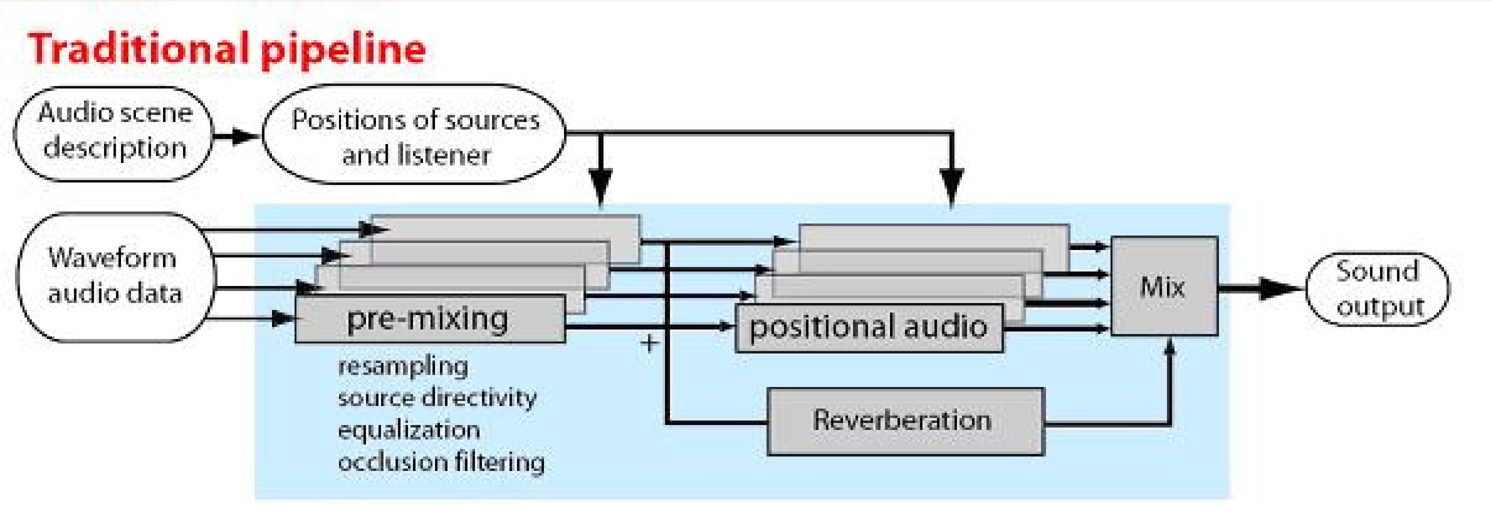
Audio

- Firelight



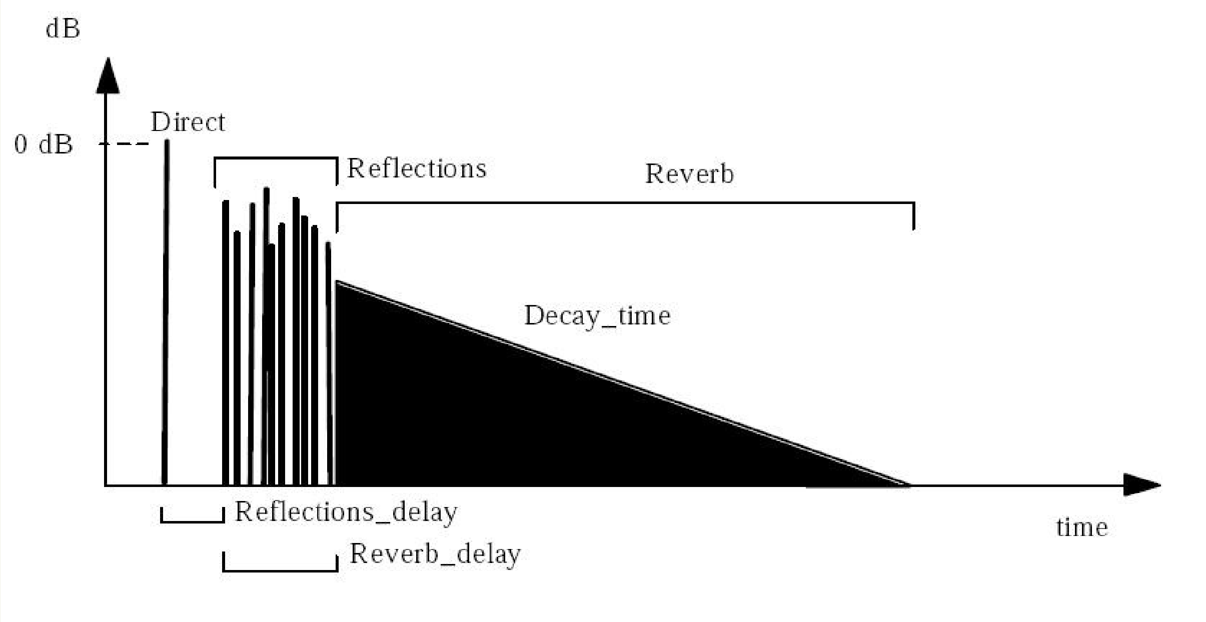


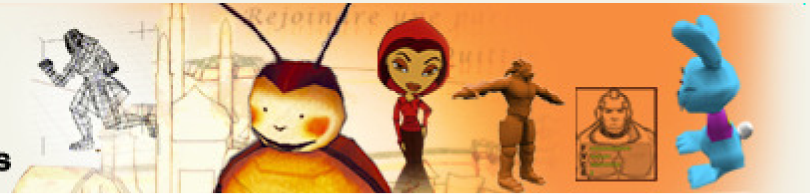
Audio


Traditional pipeline
Audio scene description
Positions of sources and listener
Waveform audio data
pre-mixing
resampling
source directivity
equalization
occlusion filtering
positional audio
Reverberation
Mix
Sound output

Audio

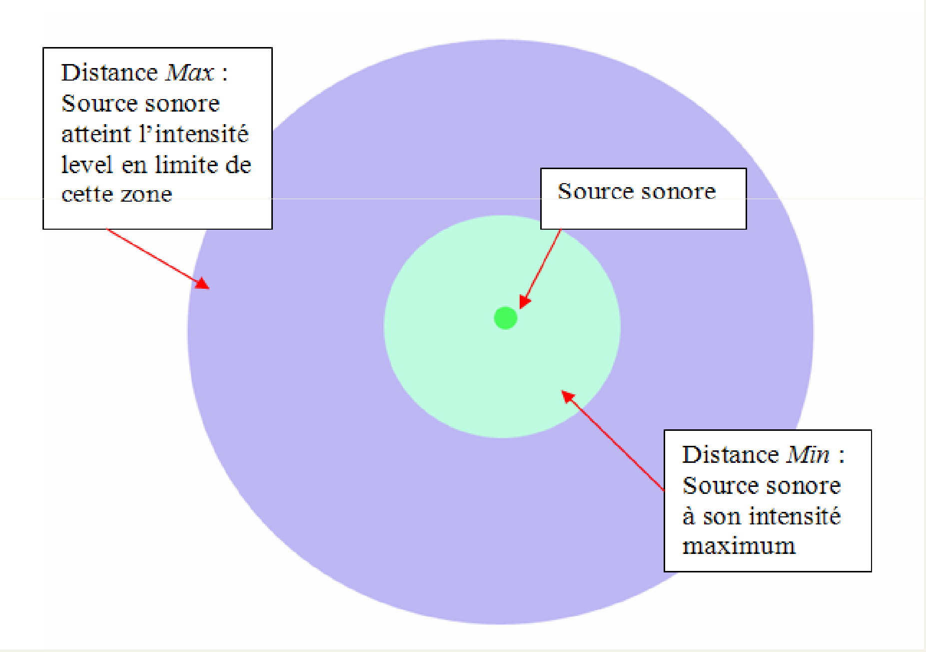
- Reverbation





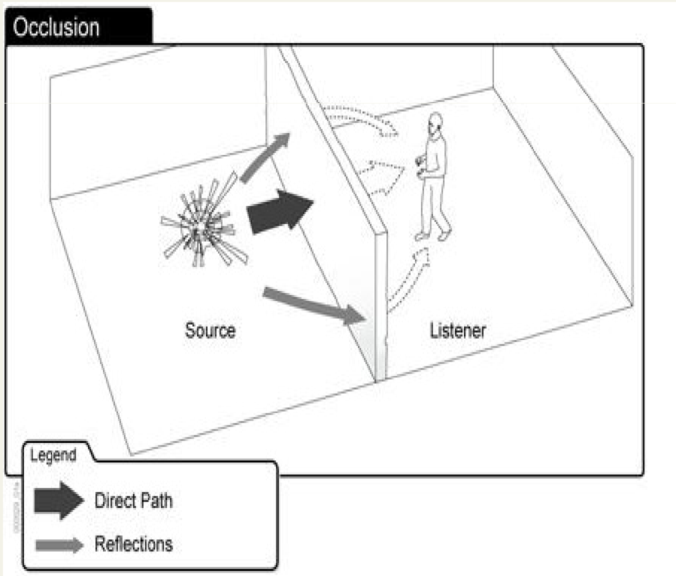
Audio

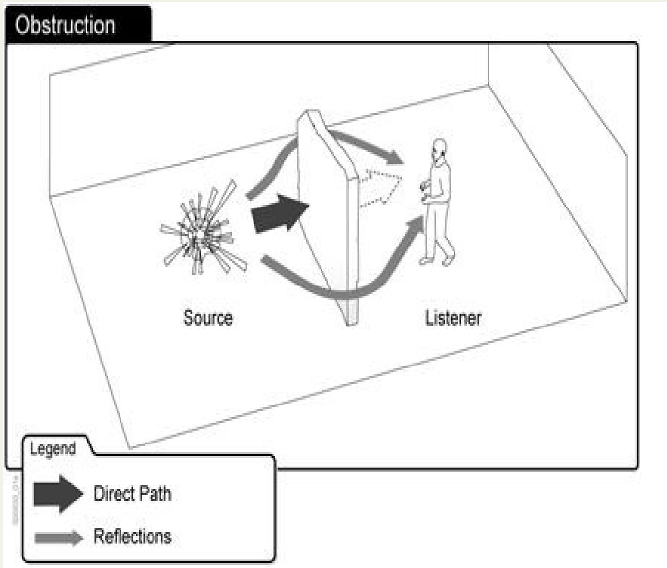
- Audible sphere

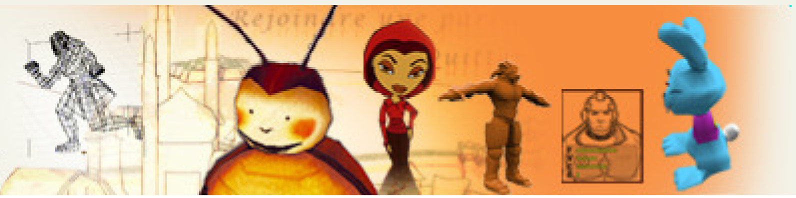


Audio

- Occlusions and obstructions







ENJMIN

Network problematics

- Lag/response time
  - f° of distance ? Sure ! But not only
  - f° of ressources & capacity (bandwidth, CPU, memory, …)
- Unreliable transmission
  - Packet loss, packet reordering
  - TCP stream Vs UDP datagram
- Network topologies
- Time sync problem without a global clock
  - Global Vs
- How to minimize the number of messages ?
  - Prediction techniques : Dead reckoning
  - Range of interest (RoI) : similar to culling and collision detection

ECOLE NATIONALE
DU JEU ET DES MEDIAS
INTERACTIFS NUMERIQUES