Video Game Engines

Chris Pollett
San Jose State University

Dec. 1, 2005.

Outline

- Introduction
- Managing Game Resources
- Game Physics
- Game AI

Introduction

- A Game Engine provides the core functionalities of a game:
  - It manages the objects, game levels, and other resources in the game world
  - It renders the viewable portion of the world to the user
  - It handles updating of the positions of objects
  - It also manages the behavior of non-player characters
- The same game engine can be used for many different games. For example, the Quake engine, Torque engine, etc.

Why study how to make game engines?

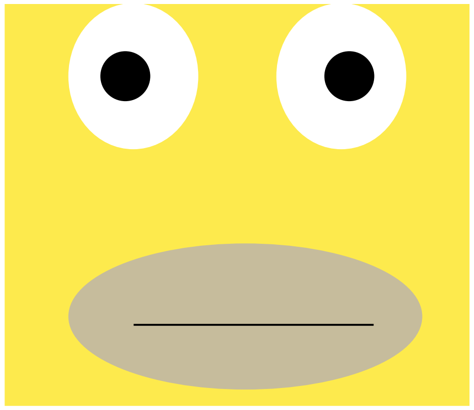
- Building a usable game engine involves skills from many different branches of computer science.
- This is because a game engine is like a highly multimedia operating system.
- Or to put it another way, the goal of a game engine is to provide a flexible way to simulate a fantasy reality quickly.

- For the rest of this talk I will briefly go over some of the things involved in making a game engine.

Managing Game Resources

- There are two aspects to this:
  - Loading in resources from secondary media such as models, textures, sounds files
  - Keeping track of resources once they loaded as well as keep tracking of objects that use those resources.

3D Models



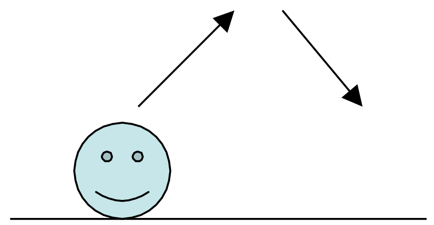
- One particular kind of resource we'd like to be able to store and load from disk are the games 3D models.
- To store such a model on disk we usually store:
  - header information (such as its name),
  - a skeleton
  - a sequence of frames
    - need a list vertices (points (x,y,z) in 3D) together with which skeleton joint associated with
    - need to say which texture to use and a sequence of 2D texture points associated with each vertex
    - need to list indexes of three vertices to make up the triangles of the figure
  - filenames of textures to use.

Keeping Track of Our Objects

- A game engine will typically load in the necessary resources when it starts running.
- These will be encapsulated into objects or structures. For instance, we might have a class or struct called Model3D.
- Instances of Model3D would be used to hold particular models we read from disk.
- All the models we'll use might then be stored in a data structure such as a hash table.
- We would use other data structures to store loaded sound files, etc.
- Finally, we would have a struct's or class's like Creature for the actual characters in the game and these would also be stored in a data structure like a list or array.
- A Creature might have a reference to which Model3D object it uses.

Rendering a Scene

- To keep the number of frames we can draw a second high, we want to draw as few objects as possible.
- We want to determine which objects are viewable from the player viewpoint and draw only those.
- Further, we might use different models for an object depending on how far away it is.
- The far away model typically has fewer triangles.
- Lastly, we might organize our triangles into arrangements like strips so the number of calls to the graphics layer is kept small

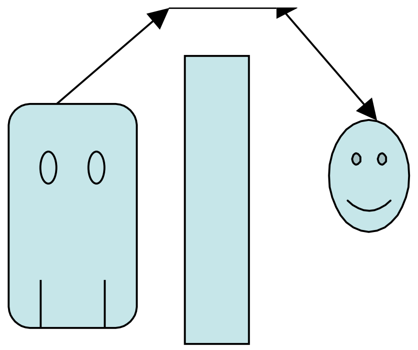


Game Physics

- Although games are a fantasy world, we expect game objects to behave roughly like similar objects in the real world.
- For instance, if a character jumps, we generally expect it to fall back down.
- It is useful for the Creature objects in our game engine thus to have a list of force objects such as gravity, friction, etc. which apply to them.

Collisions

- A game engine needs to be able also to detect when two objects collide and if so handle what to do next.
- If you have n objects any pair of which might collide this means we need at each step to do n(n-1)/2 checks.
- So for example, if there are a 100 objects in our scene we need to 100(99)/2 = 4950 checks.
- So we need to try to keep n small. To do this we could break scene into regions and only check collisions within those regions.
- We also want to keep the collision check fast. We might use bounding spheres or boxes and check for collisions on among these first.



Game AI

- We would like our non-playing characters to behave in an intelligent manner.
  - For instance, monsters should be able to find a short path around objects to get to you.
  - AI techniques like the A* algorithm can be used for this.
- We'd also like non-player characters to have goals and maybe even emotions. (Tamagotchi pets)
  - To some degree this can be faked with rule based AI. Or using finite automata. i.e., state(hungry) <-- last_ate(T), T> 5 min.
  - Some engines have mini-compilers to parse such rules.
- We'd like our non-player-characters to learn.
  - For instance, if a player tends to kick high and then block, the non-playing character should learn this and come up with an appropriate response.
  - One way to do this is to use a technique called n-grams.

Conclusion

- Today, we’ve talked about some of the major components of a game engine:
  - resource management
  - game physics and collision handling
  - game AI