CSCI 8220 Parallel & Distributed Simulation

Distributed Virtual Environments

Introduction

Outline

General Principles of Distributed Virtual Environments:

- What are they?
- Distributed Virtual Environments (DVE) versus Analytical Simulations
- Distributed Interactive Simulation (DIS)

DVE Techniques:

- Dead Reckoning

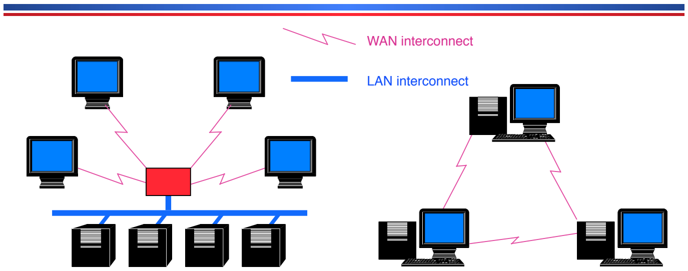
Distributed Virtual Environments (DVE)

- A synthetic world into which humans and/or physical devices are embedded
  - » Interaction between embedded and simulated elements
- Geographically distributed: Involves humans, devices and computations at different locations
- Examples
  - » Military training (SIMNET, Distributed Interactive Simulation, HLA)
  - » Multiplayer video games

DVE: Goals

- *Sufficiently* Realistic Representation
  - » 'Realistic' application dependent (e.g., training)
- Consistent views
  - » Each participant have consistent views of the DVE
  - » Consistent in time and space
- Fair fight:
  - » Outcome depends on the skill of the player rather than on artifacts in the environment
- Latency & limited communication bandwidth

DVE Architectures



Server Architecture

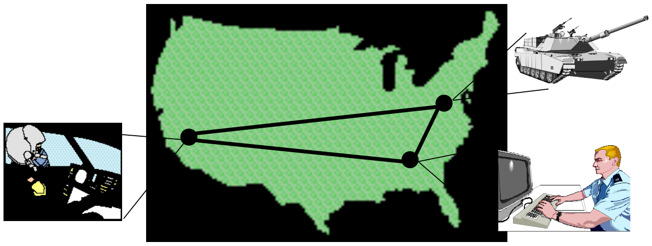
Distributed Architecture

Review: Analytic vs. DVE (Training)

	Analytical	DVE
Simulation Model	May be non-interactive	Interactive
Performance	As-fast-as-possible Speedup	Real-time Realism
Communication	Often point to point Reliable Multiprocessor/LAN OK w/arbitrary latencies	Broad/Multicast Best effort LAN/WAN Latency bounds,low jitter
Time Management	Time stamp order Synchronization Protocols	Receive Order No Synchronization Protocols
Issues	Efficient execution Easy of use	Training, Scalable execution
Typical Applications	Design Analysis	Training, entertainment

Distributed Interactive Simulation (DIS)

"The primary mission of DIS is to define an infrastructure for linking simulations of various types at multiple locations to create realistic, complex, virtual 'worlds' for the simulation of highly interactive activities" [DIS Vision, 1994].



- Developed in U.S. Department of Defense, initially for training
- DVEs widely used in DoD; growing use in other areas (entertainment, emergency planning, air traffic control)

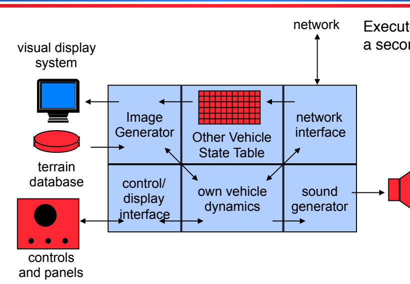
DIS Design Principles

- **Autonomy** of simulation nodes
  - » simulations broadcast events of interest to other simulations; need not determine which others need information
  - » receivers determine if information is relevant to it, and model local effects of new information
  - » simulations may join or leave exercises in progress
- Transmission of "ground truth" information
  - » each simulation transmits absolute truth about state of its objects
  - » receiver is responsible for appropriately "degrading" information (e.g., due to environment, sensor characteristics)

DIS Design Principles

- Transmission of state change information only
  - » if behavior "stays the same" (e.g., straight and level flight), state updates drop to a predetermined rate (e.g., every five seconds)
- "Dead Reckoning" algorithms
  - » extrapolate current position of moving objects based on last reported position
- Simulation time constraints
  - » many simulations are human-in-the-loop
  - » humans cannot distinguish temporal difference < 100 milliseconds (denotation and explosion)
  - » places constraints on communication latency of simulation platform

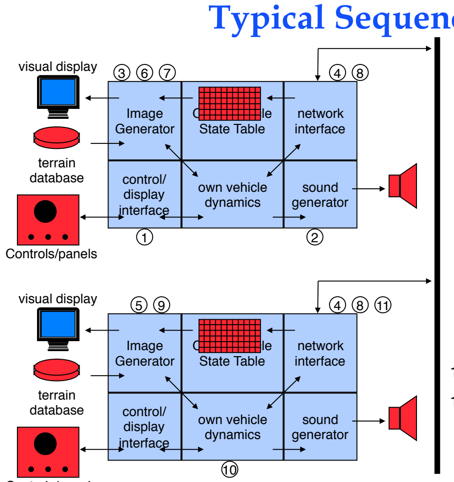
A Typical DVE Node Simulator



Execute every 1/30th of a second:

- receive incoming messages & user inputs update state of remote vehicles
- update local display
- for each local vehicle
  - » compute (integrate) new state over current time period
  - » send messages (e.g., broadcast) indicating new state

Typical Sequence



1. Detect trigger press
2. Audio "fire" sound
3. Display muzzle flash
4. Send fire PDU
5. Display muzzle flash
6. Compute trajectory, display tracer
7. Display shell impact
8. Send detonation PDU
9. Display shell impact
10. Compute damage
11. Send Entity state PDU indicating damage

Summary

- Distributed Virtual Environments have different requirements compared to analytic simulations, leading to different solution approaches
  - » May be acceptable to sacrifice accuracy to achieve better visual realism
  - » Limits of human perception can often be exploited
- Distributed Interactive Simulation (DIS) representative of approach used in building DVEs

PDES: Distributed Virtual Environments

Dead Reckoning

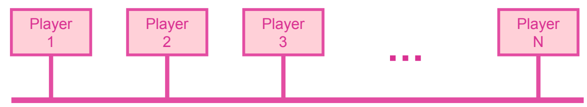
Outline

- Basic Dead Reckoning Model (DRM)
  - » Generating state updates
  - » Position extrapolation
- Refinements
  - » Time compensation
  - » Smoothing

Distributed Simulation Example

- Virtual environment simulation containing two moving vehicles
- One vehicle per federate (simulator)
- Each vehicle simulator must track location of other vehicle and produce local display (as seen from the local vehicle)
- Approach 1: Every $1/30^{th}$ of a second:
  - » Each vehicle sends a message to other vehicle indicating its current position
  - » Each vehicle receives message from other vehicle, updates its local display

Communication Requirements



- Multiple players on 10 Mbits/sec Ethernet LAN
- DIS: PDU contains 144 bytes (1152 bits)
- Each vehicle generates position update every $1/30^{th}$ second (33msec)
  - » 34,560 bits per second
- Upper bound: support 289 entities ($10 \times 10^6/34,560$)
- Above is *very* optimistic
  - » Cannot utilize all of the Ethernet's bandwidth
  - » Entities generate other PDUs (e.g., weapon fires)
  - » Multiple entities per human player (synthetic forces)
- 56 Kbits/sec modem: at best, only one vehicle!

- http://www.worldwidewords.org/qa/qa-dea7.htm

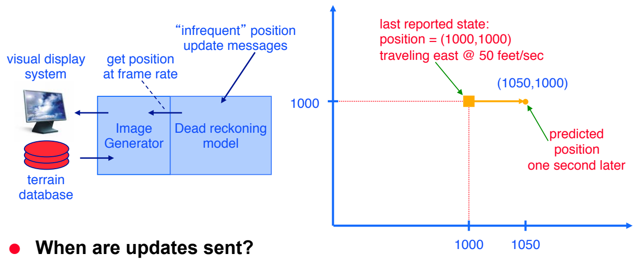
Issues

- Requires *generating many messages* if there are many vehicles; we need ways to economize on communication bandwidth
- Position information corresponds to location when the message was sent; doesn't take into account delays in sending message over the network

Dead reckoning is one technique that attempts to address each of these issues

Dead Reckoning

- Send position update messages less frequently
- Local dead reckoning model predicts the position of remote entities between updates

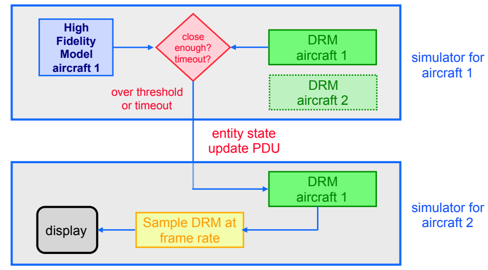


- When are updates sent?
- How does the DRM predict vehicle position?

Re-synchronizing the DRM

When are position update messages generated?

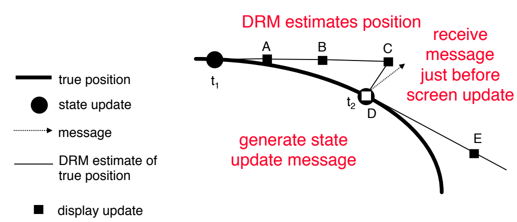
- Compare DRM position with *exact* position, and generate an update message if error is too large
- Generate updates at some minimum rate, e.g., 5 seconds (heart beats)



Dead Reckoning Models

- $P(t)$ = precise position of entity at time t
- Position update messages: $P(t_1)$, $P(t_2)$, $P(t_3)$ …
- $v(t_i)$, $a(t_i)$ = i^{th} velocity, acceleration update
- DRM: estimate $D(t)$, position at time t
  - » t_i = time of last update preceding t
  - » $\Delta t = t_i - t$
- Zeroth order DRM:
  - » $D(t) = P(t_i)$
- First order DRM:
  - » $D(t) = P(t_i) + v(t_i)*\Delta t$
- Second order DRM:
  - » $D(t) = P(t_i) + v(t_i)*\Delta t + 0.5*a(t_i)*(\Delta t)^2$

DRM Example



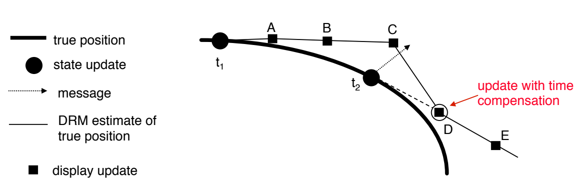
Potential problems:

- Discontinuity may occur when position update arrives; may produce "jumps" in display
- Does not take into account message latency
  - » Update position is already 'out of date'

Time Compensation

Taking into account message latency

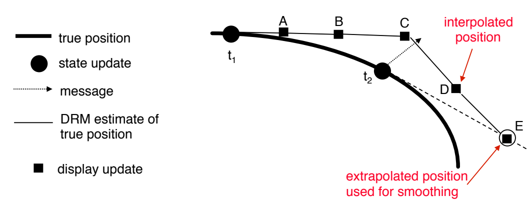
- Add time stamp to message when update is generated (sender time stamp)
- Dead reckon based on message time stamp



Smoothing

Reduce discontinuities after updates occur

- "phase in" position updates
- After update arrives
  - » Use DRM to project next k positions
  - » Interpolate position of next update



Accuracy is reduced to create a more natural display

Summary

- Managing communications is a major issue in implementing distributed simulations
- Dead reckoning model (DRM)
  - Extrapolate current position based on past updates
  - Send update messages when DRM error becoming too large
  - Reduces interprocessor communication
- DRM based on equations of motion
- Time compensation to account for message latency
- Smoothing to avoid “jumps” in display