Interactive Graphical Systems
HT2004

Networked Virtual Environments
This Lecture

- Background and history
- Fundamentals (TCP/IP, etc)
- Research in related topics
- Demo of a Networked VE used in education
Background and history

Multiplayer games

1961 Spacewar, MIT
http://lcs.www.media.mit.edu/groups/el/projects/spacewar

1979 MUD, Essex University

1993 Doom, Id Software

1994 Warcraft, Blizzard

1996 Quake, Id Software
Counterstrike, Halflife

2004 Doom III, Id Software
Background and history

Classification of multiplayer games

• Multiplayer
• Massively multipler (MMORPG)

• Peer-to-peer
• Client/server
Background and history

Military simulations

- 1983 - 1990 SIMNET (SIMulator NETworking)
- 1989 - … DIS (Distributed Interactive Simulation)
Background and history

Communities, chats, concept visualization, e-Learning

1997-2001 ActiveWorlds

2001 VASE, Uppsala University
http://www.it.uu.se/research/project/vase

2004 http://www.croquetproject.org
Fundamentals

Host A \[\rightarrow\] Host B

Message: \ldots01101001\ldots

- Message may or may not arrive
- Message may arrive as it was sent or as something else (it might be garbled)
- Message may arrive 0, 1 or more times
- If there are several messages, they may arrive in a different order than in which they were sent

How can we do anything useful under these premises?
Fundamentals

We only need to worry about TCP and UDP!

*how to use them, not how they work*
Fundamentals

**Access layer**

*ethernet, PPP etc*

Responsible for getting one:s and zero:s into the cable and out at the other end between hosts on the same network.

**Network layer**

*IP*

Responsible for routing data between hosts on different networks. This is a *best-effort* service.

Hosts are identified using *IP-numbers*, e.g. 130.238.147.242. DNS translates between names and numbers, e.g. *anafi.hci.uu.se* ←→ 130.238.147.242.
Fundamentals

**TCP - Connection oriented, stream oriented, reliable service**

Two hosts can create a virtual stream and send data to each other. The data is guaranteed to arrive, and to arrive in the correct order.

**UDP - Connectionless, datagram oriented, unreliable service**

The data is divided into discrete *datagrams*. A datagram might be lost or arrive out of sequence in relation to other datagrams (we have to deal with this).

**Multicast**

A method for delivering the same data to several hosts at the same time. Works only for UDP. Not widely implemented across the Internet, but might be in the future...

Ports are used to identify different applications on a given host, e.g. port 80 are used for http, port 21 for ftp.
Fundamentals

Should we use TCP or UDP?

• UDP is slimmer and more efficient

• UDP doesn’t have a built-in delay like TCP (Nagle algorithm)

• UDP can do multicast

• However, don’t use UDP unless you can live with it’s limitations! Do not re-invent the wheel, use TCP instead!

In order to cope with UDP’s limitations, send states, not state-changes.
Fundamentals

Some parameters to take into consideration

ISDN line 64kbps -> Approx 6400 bytes / sec (due to overhead)

30 fps -> 6400/30 = 213 bytes each frame

10 fps -> 6400/10 = 640 bytes each frame
Toolkits for Networked virtual environments

Which issues should a Toolkit deal with?

- State sharing (shared memory)
- Mutual exclusion of resources
- Data serialization
- Dead reckoning
Networking paradigms

Different ways to abstract out the actual sending and receiving of bits

Stream-based, like a ‘pipe’
- Stream is contiguous or discrete
- Both TCP and UDP fits here

Request-Response
- Like making a function call
- RPC, CORBA, COM, HTTP fits here

Shared data
- Sharing a common set of data
- SQL databases

The shared data paradigm is used in VRT
Networking paradigms

Network applications can be classified as *peer to peer* or *client-server*.

Both have advantages and disadvantages. Distributed systems research community favor the peer to peer model.
Achieving mutual exclusion

Mutual exclusion - a centralized approach

Potential problem: The coordinator (server) crashes
Achieving mutual exclusion

What if the coordinator crashes?
Other methods have been proposed ...

*Distributed*
Ask everyone if it’s OK.

*Token ring*
A token is passed around a ring.
Only the host holding the token may do critical updates.

Ops… these algorithms fail if *any* participant crashes, however this is detectable using timeouts.
Achieving mutual exclusion

Mutual exclusion - comparison of methods

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Messages per entry / exit</th>
<th>Delay before entry (in message times)</th>
<th>Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized</td>
<td>3</td>
<td>2</td>
<td>Coordinator crashes</td>
</tr>
<tr>
<td>Distributed</td>
<td>2 * (n –1)</td>
<td>2 * (n – 1)</td>
<td>Any participant crashes</td>
</tr>
<tr>
<td>Token ring</td>
<td>0 to n-1</td>
<td>0 to n-1</td>
<td>Any participant crashes</td>
</tr>
</tbody>
</table>
Distributed shared memory

• All participating hosts share a single “memory”
• Transparency important
• Ideally no more difficult to program than a single computer
• Can also be called
  • shared state repository
  • shared variables
  • shared database
Distributed shared memory

Distributed shared memory - consistency models

• **Strict consistency**
  • The distribution is totally transparent.
  • The shared memory acts exactly like local memory.

• **Weak consistency**
  • Explicitly tell the system when we want to use the memory, and when we are done.

• **Entry consistency**
  • When we want to use the memory, request contents from other hosts.
  • No need to send updates to others unless they request it.

• **Release consistency**
  • Send updates to others when we are done using the memory.
  • No need to send requests.
Data Serialization

Data serialization

Different architectures use different byte ordering, such as Intel and Motorola.

This should be handled as transparently as possible by the system.

Different approaches

Explicitly pack each primitive (float, integer, etc)

Provide structure description at runtime

Use precompiler to create special serialization code
Networked virtual environments - dead reckoning

A method for approximating an object’s position
Requires less frequent state updates

Simplest approach:

Time 3:
Position (4,3)
Velocity (3,2)

Time 4:
Position (7,5)

Predict new position using

\[ p_{pred} = p_0 + (t-t_0) \cdot v_0 \]
Networked virtual environments - dead reckoning

Avoiding jumpy movements (convergence)

Combine

\[ p_{\text{pred}} = p_0 + (t-t_0) * v_0 \]

and

\[ p_{\text{pred}} = p_1 + (t-t_1) * v_1 \]

using a blending-factor \( b \) and a convergence-period \( c \)

\[ p_{\text{pred}} = (1-b) * (p_0 + (t-t_0) * v_0) + b * (p_1 + (t-t_1) * v_1) \]

\[ b = \frac{(t-t_1)}{c} \text{ if } t < t_1 + c \]

\[ b = 1 \text{ if } t \geq t_1 + c \]

The object will converge from the earlier path into the new one in time \( c \).

Drawback: introduces extra latency! *(by time \( c \))"
Networked virtual environments - dead reckoning

Better prediction
Use higher order polynomials

Predict new position using

\[ p_{\text{pred}} = p_0 + (t-t_0)v_0 + (t-t_0)^2a_0 \]

Higher order polynomials can also be used for convergence

As a rule of thumb 10 updates/sec is enough for 60-70 fps
Networked virtual environments - application design

Should the logic reside in the clients or in the server? Should we use a reliable or an unreliable protocol? What data should we share?

How are these problems solved by others?
Networked virtual environments - application design

Should the logic reside in the clients or in the server?

Server
• The clients can be made more general
• Easier to guarantee consistency

Clients
• The server can be more general
• Communicates at a higher abstraction level (usually less traffic)
Networked virtual environments - application design

Should we use a reliable or an unreliable protocol?

Unreliable protocol requires frequent updates
Reliable protocol propagates slower and is less efficient

When should data be sent?

Everything always ← Only when something changes

Hybrid approaches:

Use UDP, but send the whole state using TCP once a second.
Use UDP for things that move, but TCP when it stops.
Networked virtual environments - application design

What data should be shared?

• Transformations
• Geometries
• Textures
• Materials
Networked virtual environments - application design

<table>
<thead>
<tr>
<th></th>
<th>General / specialized clients</th>
<th>Reliable / unreliable protocol</th>
<th>Peer to peer / Client - server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quake</td>
<td>specialized</td>
<td>unreliable</td>
<td>client - server</td>
</tr>
<tr>
<td>DIS /SimNET</td>
<td>specialized</td>
<td>unreliable</td>
<td>Peer to peer</td>
</tr>
<tr>
<td>DIVE</td>
<td>general</td>
<td>reliable</td>
<td>Peer to peer</td>
</tr>
<tr>
<td>ActiveWorlds</td>
<td>general</td>
<td>both</td>
<td>client - server</td>
</tr>
</tbody>
</table>
Toolkits for Networked virtual environments

Java Shared Data Toolkit

Uses three abstractions for data communication

ByteArrays
   Allows the programmer to create shared memory areas that are kept consistent

Tokens
   Can be grabbed and released, for mutual exclusion of resources

Channels
   For stream-based communication

Available at: http://java.sun.com/products/java-media/jsdt/
Toolkits for Networked virtual environments

CavernSOFT

Communication mainly streambased
Uses concept of ‘reflectors’ for group communication
TCP and UDP reflectors available

Provides higher-level
database classes for
state-sharing based on the
reflector concept.

Available at: http://www.openchannelsoftware.org/projects/CAVERNsoft_G2/
Toolkits for Networked virtual environments

Streep (*State Repository*)

- Based on shared semi-transparent memory
- Memory is allocated using functions resembling standard C functions.
- Updates are explicitly sent using ‘channels’
- Provides function for locking memory blocks
- Can automatically serialize memory given a description string
- Uses TCP, UDP or multicast

```c
SR *sr;
int *i;

sr=srCreateServer(1200, "udp tcp");
i=srAllocMem(sr,"test", sizeof(int), SR_NONE);
while (1)
{
    *i=*i+1;
    srNotify(sr, i, 0);
    srProcess(sr, 100, 100);
}

SR *sr;
int *i;

sr=srCreateClient("localhost", 1200);
i=srGetMem(sr,"test", SR_NONE);
srChannelSubscribe(sr,0);
while (1)
{
    printf("%d
",*i);
    srProcess(sr, 100, 100);
}
```
Toolkits for Networked virtual environments

VRT

Functions for sharing node transformations and on/off state exist in VRT

```c
extern int VRT_NetConnect(int port);
extern int VRT_NetDisconnect();
extern VRT_Node *VRT_SharedNodeNew(VRT_Node *parent, char *name);
```

VRT_NetConnect
- Searches to LAN for a server running the specified port. If no server exists one will be created. The first one to start will be the server.

VRT_SharedNodeNew
- All nodes with the same name on the network will share their transformations and their switch state. Functions such as VRT_NodeSetTranslation, VRT_NodeSetRotation, VRT_SwitchOn will send information to others.
Results from the experiment at SNDC / FHS
Results from the experiment at SNDC / FHS

Avg. milliseconds per presentation

0 – frontoparallell  1 – flat
2 – flat equal area to frontoparallell