Multiplayer Computer Games

Jouni Smed

Department of Information Technology,
University of Turku

2005

Course Syllabus

- credits: 4 cp (2 cu)
- prerequisites:
  - Algorithms for Computer Games
  - knowledge on the basic concepts of computer networks
- teaching methods: lectures
  - Tuesdays 14–16 and Thursdays 12–14, Auditorium
  - from November 1 to December 15
- assessment: examination only
- course web page:
  `http://staff.cs.utu.fi/staff/jouni.smed/mcg/`

Examinations 1 (2)

- examination dates
  1. January 16, 2006
  2. February 13, 2006
  3. March 2, 2006
- check the exact times and places at
  `http://www.it.utu.fi/opetus/tentit/`
- if you are *not* a student of University of Turku, you must register to receive the credits
  - further instructions are available at
    `http://http://www.tucs.fi/education/courses/participating_courses.php`

Examinations 2 (2)

- questions
  - based on both lectures and lecture notes
  - two questions, à 5 points
  - to pass the examination, at least 5 points (50%) are required
  - grade: $g = \lceil p - 5 \rceil$
  - questions are in English, but you can answer in English or in Finnish
- remember to enrol in time!

Outline of the Course

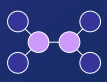
8. Communication layers
   - physical platform
   - logical platform
   - networked application
9. Compensating resourse limitations
   - aspects of compensation
   - protocol optimization
   - dead reckoning
   - local perception filters
   - synchronized simulation
   - area-of-interest filtering
10. Cheating prevention
    - attacking the hosts
    - tampering with network traffic
    - look-ahead cheating
    - collusion
    - offending other players

Guest Lecture

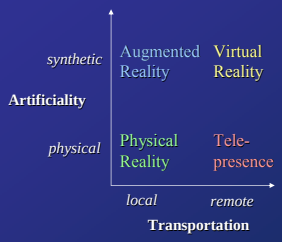
- Assoc. prof. Tomas Akenine-Möller (Dept. of CS, Lund University, Sweden): “Precomputed Local Radiance Transfer”
- Auditorium, Thursday, November 3, 1 p.m.

§8 Communication Layers

- physical platform
- logical platform
- networked application



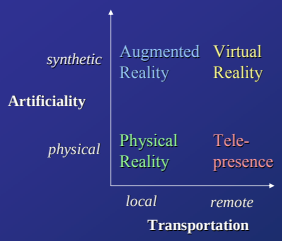
Classification of Shared-Space Technologies 1 (2)



- Physical reality
  - resides in the local, physical world
  - here and now
- Telepresence
  - a real world location remote from the participant's physical location
  - a remote-controlled robot

Benford et al., 1998

Classification of Shared-Space Technologies 2 (2)



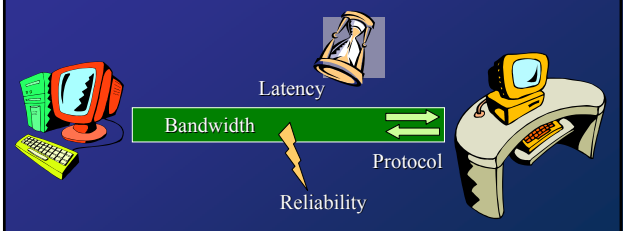
- Augmented reality
  - synthetic objects are overlaid on the local environment
  - a head-up display (HUD)
- Virtual reality
  - the participants are immersed in a remote, synthetic world
  - multiplayer computer game

Benford et al., 1998

§8.1 Physical Platform

- resource limitations
  - bandwidth
  - latency
  - processing power for handling the network traffic
- transmission techniques and protocols
  - unicasting, multicasting, broadcasting
  - Internet Protocol, TCP/IP, UDP/IP

Network Communication



Fundamentals of Data Transfer 1 (3)

- Network latency
  - network delay
  - the amount of time required to transfer a bit of data from one point to another
  - one of the biggest challenges:
    - impacts directly the realism of the game experience
    - we cannot much to reduce it
  - origins
    - speed-of-light delay
    - endpoint computers, network hardware, operating systems
    - the network itself, routers

Fundamentals of Data Transfer 2 (3)

- Network bandwidth
  - the rate at which the network can deliver data to the destination host (bits per second, bps)
- Network reliability
  - a measure of how much data is lost by the network during the journey from source to destination host
  - types of data loss:
    - dropping: the data does not arrive
    - corruption: the content has been changed

Fundamentals of Data Transfer 3 (3)

- Network protocol
  - a set of rules that two applications use to communicate with each other
  - packet formats
    - understanding what the other endpoint is saying
  - packet semantics
    - what the recipient can assume when it receives a packet
  - error behaviour
    - what to do if (when) something goes wrong

Internet Protocol (IP)

- Low-level protocols used by hosts and routers
- Guides the packets from source to destination host
- Hides the transmission path
  - phone lines, LANs, WANs, wireless radios, satellite links, carrier pigeons,…
- Applications rarely use the IP directly but the protocols that are written on top of IP
  - Transmission Control Protocol (TCP/IP)
  - User Datagram Protocol (UDP/IP)

TCP versus UDP

Transmission Control Protocol (TCP/IP)

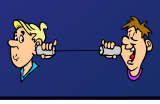
- Point-to-point connection
- Reliable transmission using acknowledgement and retransmission
- Stream-based data semantics
- Big overhead
  - data checksums
- Hard to 'skip ahead'

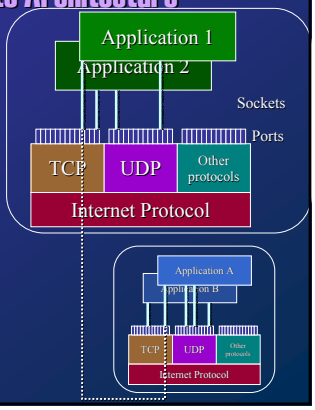
User Datagram Protocol (UDP/IP)

- Lightweight data transmission
- Differs from TCP
  - connectionless transmission
  - 'best-efforts' delivery
  - packet-based data semantics
- Packets are easy to process
- Transmission and receiving immediate
- No connection information for each host in the operating system
- Packet loss can be handled

The BSD Sockets Architecture

- A socket is a software representation of the endpoint to a communication channel
- Reliable/unreliable communication, single/multiple destinations, etc.
- Includes several pieces of information, such as
  - protocol
  - destination host and port
  - source host and port

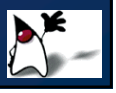




BSD = Berkeley Software Distribution

Sockets in Java

- Networking related classes are in the package `java.net`
- IP addresses are handled with the `InetAddress` class
  - Creation (note: no constructor):
    `InetAddress address = InetAddress.getByName(`*address*`);`
  - Parameter *address*
    - in DNS format (`"staff.cs.utu.fi"`)
    - as an IP number (`"139.232.75.8"`)
    - `null` (= `"localhost"` = `"127.0.0.1"`)
- Port numbers 1–1024 are reserved
- Socket types:
  - `ServerSocket`: handles connection requests directed to a given port
  - `Socket`: actual socket which handles the communication



Socket Example: The Code

Server

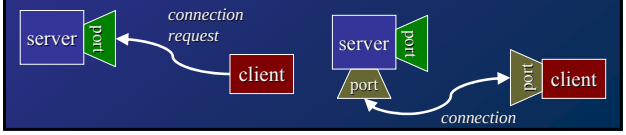
```
ServerSocket s =
    new ServerSocket(PORT);
try {
    Socket socket = s.accept();
    try {
        // Use the socket.
    } catch (IOException e) {
        // Transfer failed.
    } finally {
        socket.close();
    }
} catch (IOException e) {
    // Connection failed.
} finally {
    s.close();
}
```

Client

```
try {
    Socket socket = new
        Socket(address, PORT);
    try {
        // Use the socket.
    } catch (IOException e) {
        // Transfer failed.
    } finally {
        socket.close();
    }
} catch (IOException e) {
    // Connection failed.
}
```

Socket Example: What Happens

- Server creates `ServerSocket` which begins to listen to the given port
- The execution halts in the `accept` method call, until there is a connection request
- Client creates `Socket` with the server's address and the port number of the server socket
- Client's socket sends a connection request
- Server socket answers the reqest by creating `Socket` to *any* available port
- Server socket sends the port number of the new socket to the client
- Client's socket connects to the new socket
- In the client, the socket's constructor finishes
- In the server, the `accept` method returns the new socket.



Using Streams with Sockets

- Input stream:

```
BufferedReader in =
  new BufferedReader(
    new InputStreamReader(
      socket.getInputStream()));
```

- Output stream:

```
PrintWriter out =
  new PrintWriter(new BufferedWriter(
    new OutputStreamWriter(
      socket.getOutputStream())), true);
```

- Reading and writing as normal:
  - `out.println("foo");`
  - `String s = in.readLine();`
- Streams use TCP, which is reliable but slow

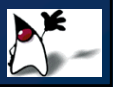
UDP and Datagrams in Java

- `DatagramSocket` can both send and receive packets
  - no server sockets because there is no need to establish a connection
- `DatagramPacket` includes all the data to be sent/received
  - maximum size 64 kB
- Constructing a receiving packet:

```
byte[] buffer = new byte[CAPACITY];
DatagramPacket dp1 =
          new DatagramPacket(buffer, CAPACITY);
```

- Constructing a packet to send:

```
byte[] message; // The bytes to send.
DatagramPacket dp2 =
          new DatagramPacket(message, message.length,
                             address, port);
```



Datagram Example

```
try {
    socket = new DatagramSocket(PORT);
    socket.receive(dp1);
    socket.send(dp2);
} catch (SocketException e) {
    // Could not open the socket.
} catch (IOException e) {
    // Problems with communication.
} finally {
    socket.close();
}
```

Datagram Contents

- Sender's address:

```
InetAddress addr = dp.getAddress();
```

- Sender's port:

```
int port = dp.getPort();
```

- Packet payload size:

```
int size = dp.getLength();
```

- Packet payload data:

```
byte[] data = dp.getData();
```