Virtual Reality & Physically-Based Simulation

Interaction Metaphors

G. Zachmann

University of Bremen, Germany

cgvr.cs.uni-bremen.de

History

- First computer game (probably):
  - Spacewars, 1961, MIT
  - Two players, two spaceships ("wedge" and "needle"), each can fire torpedos
  - With it came the first real interaction devices and metaphors

A Classification of Interaction Tasks

- *Universal Interaction Tasks (UITs) in VEs* [Bowman]:
  1. Navigation = change viewpoint
  2. Selection = define object or place for next task
  3. Manipulation = grasp, move, manipulate object
  4. System control = menus, widgets sliders, number entry, etc.
  - Model and modify geometry (very rare; not in Bowman's UITs)
- *Basic interaction tasks (BITs) in 2D GUIs* [Foley / vanDam]:
  - Selection (objects menus, ..)
  - Positioning (incl. orientation) or manipulation
  - Entering quantities (e.g., numbers)
  - Text input (via keyboard or speech input)

More Interaction Tasks

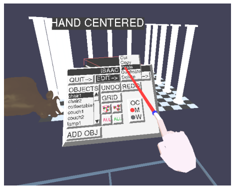
- Search (e.g., searching a scene for a specific object)
- Ambient, implicit, playful, non-purposeful interaction
  - E.g., playing around with a virtual spraying can
- Sculpting / modeling surfaces
- Making an avatar dance by whole body interaction

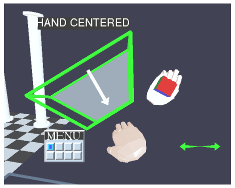
Digression: Classification of Widgets for 3D UIs

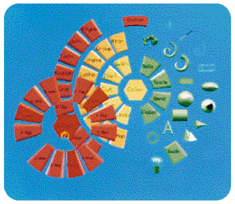
- **Direct 3D Object Interaction**
  - **Object Selection**
  - **Geometric Manipulation**
- **3D-Scene Manipulation**
  - **Orientation and Navigation**
  - **Scene Presentation Control**
- **Exploration and Visualization**
  - **Geometric Exploration**
  - **Hierarchy Visualization**
  - **3D Graph Visualization**
  - **2D-Data and Document Visualization**
  - **Scientific Visualization**
- **System / Application Control**
  - **State Control / Discrete Valuators**
  - **Continuous Valuators**
  - **Special Value Input**
  - **Menu Selection**
  - **Containers**

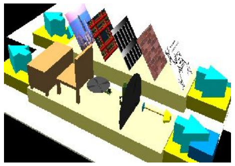
Menu Selection

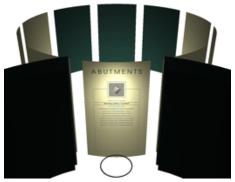
- Temporary Option Menus
  - *Rotary Tool Chooser*
  - *Menu Ball*
  - *Command & Control Cube*
  - *Popup Menu*
  - *Tool Finger*
  - *TULIP*
- Single Menus
  - *Ring menu*
  - *Floating Menu*
  - *Drop-Down-Menu*
  - *Revolving Stage*
  - *Chooser Widget*
  - *3D-Palette, Primitive Box etc.*
- Menu Hierarchies
  - *Hands-off Menu*
  - *Hierarchical Pop-Up Menus*
  - *Tool Rack*
  - *3D Pie Menu*
  - → Hierarchy Visualizations

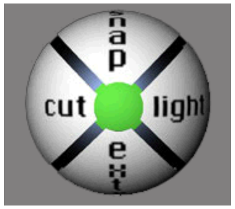












The Design of User Interfaces

- There are two main approaches:
  - Natural interaction:
    - Try to resemble reality and the interaction with it as closely as possible
  - "Magic" interaction
    - Give the user new possibilities beyond reality
    - Challenge: keep the cognitive overhead as low as possible, so that users don't get distracted from their task!
- Tools:
  - Direct *user action* (e.g., motion of the body, gesture, head turning, ...)
    - Pro: well suited if intuitive; con: possibilities are somewhat limited
  - Physical Devices (e.g., steering wheel, button, ...)
    - Pro: haptic feedback affords precise control
    - Con: not easy to find/devise novel & useful devices
  - Virtual devices (e.g., menus, virtual sliders, ...)
    - Pro: very flexible, reconfigurable, "anything goes"
    - Con: can be difficult to use because of lack of force feedback

- Goals (in particular in VR):
  1. Intuitive / natural interaction (usability)
     - By definition: easy to learn
     - Adjust to the users expertise (expert vs. novice)
  2. Efficient interaction (user performance)
     - Precision, speed, productivity of the users
- Problems (especially in VR):
  - No physical constraints (interaction in mid-air)
  - In particular: no haptic feedback
  - Efficient interaction with objects outside of the user's reach
  - Noise / jitter / imprecision in tracking data
  - Fatigue
  - No standards

> *There has never been a high performance task done in the history of this planet, to the best of my knowledge, that has ever been done well with an intuitive interface.*
> [Brian Ferran]

Gesture Recognition

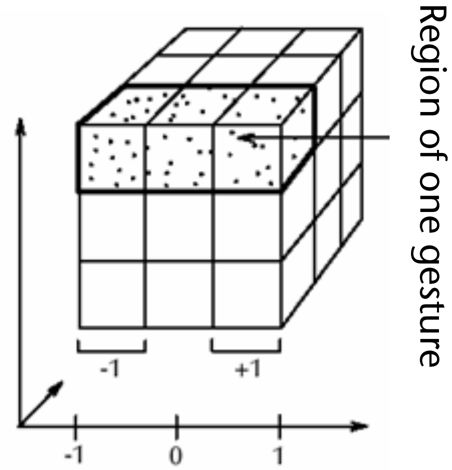
- Is basically a simple classification problem:
  - Given: a flex vector $x \in \mathbb{R}^n, \; n \approx 20$ = joint angles
  - Wanted: gesture $G(x) \in \{\text{“Fist“}, \text{“Hitch-hike“}, \ldots\}$
- Wanted: an algorithm that is ...
  - .. user independent
  - .. robust (> 99%)
  - .. Fast

An Extremely Simple Gesture Recognition Algorithm

- Neural network is fine, if lots of gestures, or some of them are inside the parameter space
  - However, experience show: users can remember only a small set (e.g. 5)
- In the following: only few gestures at the border of parameter space
  - Discretize flex vector

$$f \in [0,1]^d \rightarrow f' \in \{-1, 0, +1\}^d$$

0 = flex is "somewhere in the middle"



  - Gesture = region of d-dimensional parameter cube
  - Represent each region/gesture by a discrete vector:

$$g \in \{-1, 0, +1\}^d \qquad 0 = \textit{don't care}$$

  - Gesture i is recognized iff

$$f' \cdot g_i = |g_i|$$

  - Condition for this to work: regions of different gestures must not overlap

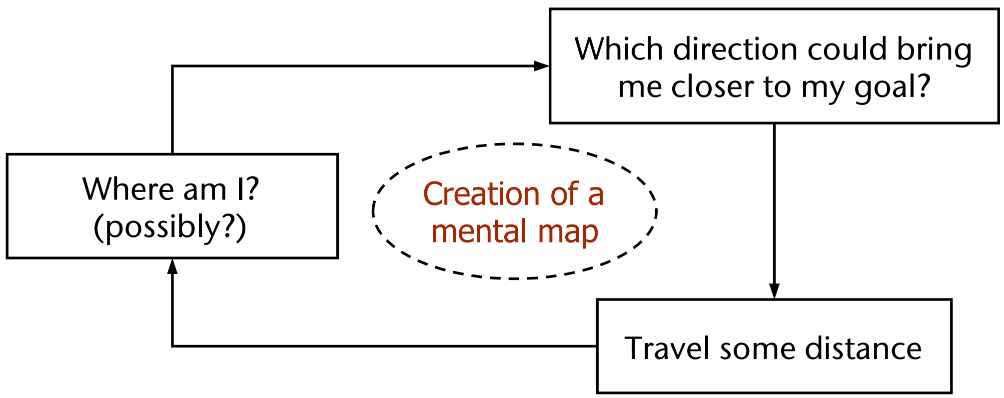
- Implementation details:
  - Do automatic calibration at runtime to fill the range [0,1]:
    - Maintain a running min/max and map it to [0,1]
    - Over time, shrink min/max gradually (for robustness against outliers)
  - Ignore transitory gestures
- Dynamic gestures =
  1. Sequence of static gestures (e.g., sign language)
  2. Path of a finger / hand
  - Utility for VR?

Navigation

- Comprises: *Wayfinding* & *Locomotion*
- Locomotion / Travel =
  - Cover a distance (in RL or in VR)
  - Maneuvering (= place viewpoint / viewing direction exactly)
- Wayfinding =
  - Strategy to find a specific place (in an unknown building / terrain)
  - Comprises: experience, cognitive skills, ...

How do People Solve a Wayfinding Task

- How do people find their *way*:
  - Natural hints/clues
  - Signs (man-made)
- A simple user model for way finding:



- In VEs, there can be 2 kinds of navigation [sic] aids:
  - Aids for improving the user's performance in the virtual environment
  - Aids that help increase the user's performance later in the real world (i.e., that increase the training effect)

- Question: do humans create a mental map of their environment in order to solve wayfinding tasks?
- Answer: probably yes, but not like a printed street map; rather like a non-planar graph that stores edge lengths

http://www.spiegel.de/wissenschaft/technik/0,1518,739416,00.html

Kerstin Schill, Uni Bremen

Techniques for Navigation in VEs

- Real user navigation, e.g., walking, turning head, …
- *Point-and-fly* (especially in Caves and HMDs)
- *Scene-in-hand*
- *World-in-Miniature*
- Orbital mode
- And some more …

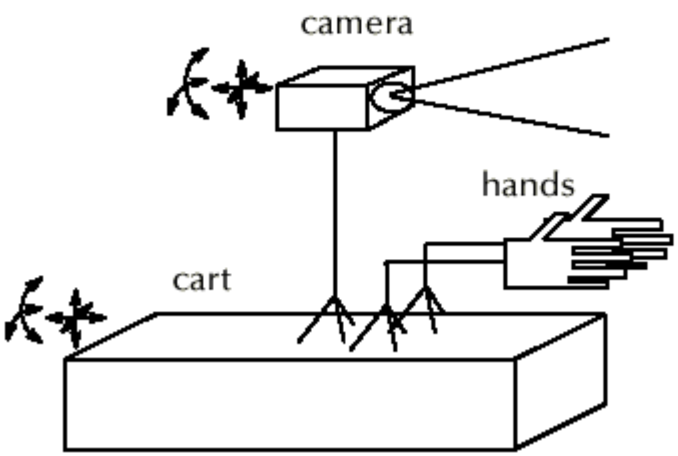
A Taxonomy for this Interaction Task

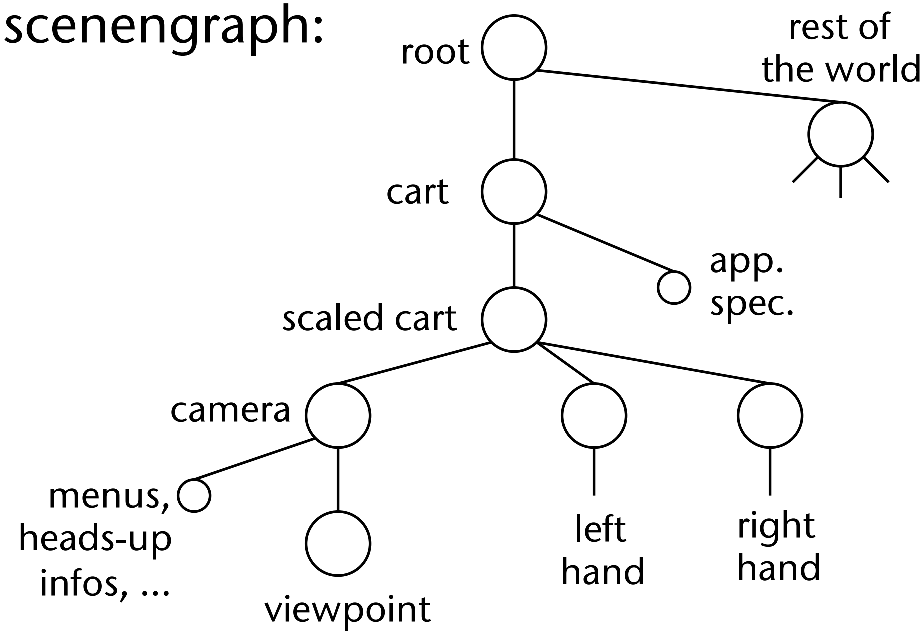
- Navigation
  - Specify direction/target
    - Viewing direction
    - Pointing direction
      - Hand
      - Other object
    - Pointing in 2D
    - Discretely
      - Lists (Menus)
      - Objects in VE
  - Specify speed/accel.
    - Constant
    - Gesture based
      - Flex value
      - Hand position
    - Explicit
    - Automatic
    - Incremental
      - Speech
      - Gesture
  - Condition that elicits navigation
    - Continuous mode
      - Bicycle
    - Start/stop
    - Automatischer Start/Stop

- **Taxonomies are a way to explore (exhaustively, if possible) the *design space* of an interaction task!**

An Abstract Representation of the User

- User = Head, Hand,
  perhaps whole body (avatar)
- The "flying carpet" metaphor :
  - User = camera
  - Camera is placed on a carpet / cart / wagon
- Representation as (part of) a scenengraph:

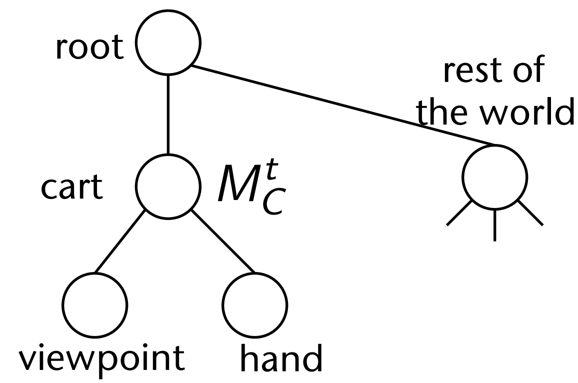




The Point-and-Fly Metaphor

- Controlling sensors:
  - Head sensor → viewpoint
  - Hand sensor → moves cart:

$$M_C^t = M_C^{t-1} \cdot \mathrm{Transl}(s \cdot \mathbf{t})$$

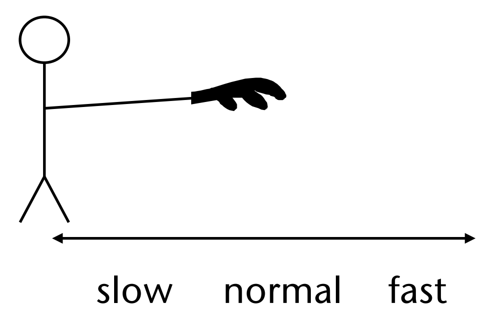


s = speed,

$\mathbf{t}$ = translation vector = 3rd column of hand tracking sesnor

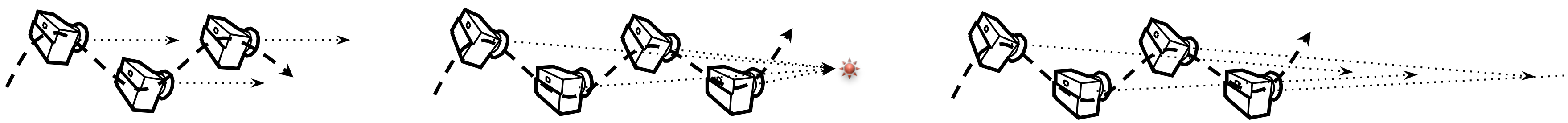
- Generalization: use graphical objects instead of sensor to derive translation direction

- Specification of the speed:
  - Constant (e.g. with Boom)
  - Flexion of the thumb
  - Depending on distance |hand – body|
  - Make it independent of framerate



Perception of the Distance Walked in VR [2009]

- Question: how can the sense of presence be increased while navigating in a VE? (using point-and-fly)
- Idea:
  - Make the viewpoint oscillate like in reality
  - (First-person-shooter games invented this earlier ;-))

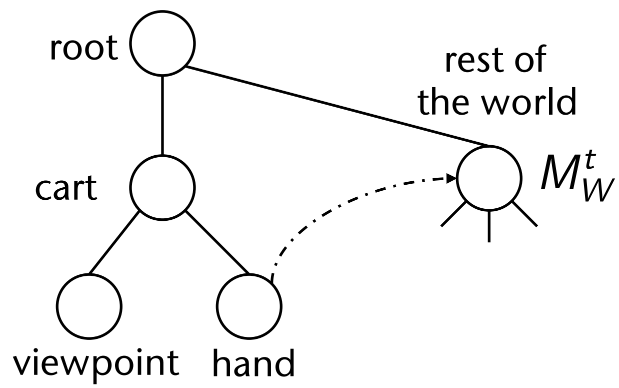


- Results:
  - Only vertical oscillation helps increase presence
  - Users prefer slight oscillation over no oscillation
  - Short "travel distances" can be estimated more precisely (~ factor 2)

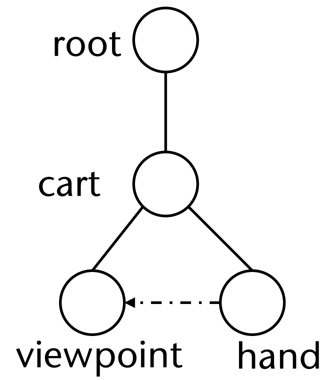
The Scene-in-Hand / Eyeball-in-Hand Metaphor

- *Scene-in-hand*:
  - *"Grabbing the air"* technique
  - *Cart* remains stationary, scene gets rotated by hand sensor about a specific point in space
  - The transformation:

$$M_W^t = M_H^t \cdot (M_H^{t_0})^{-1} \cdot M_W^{t_0}$$

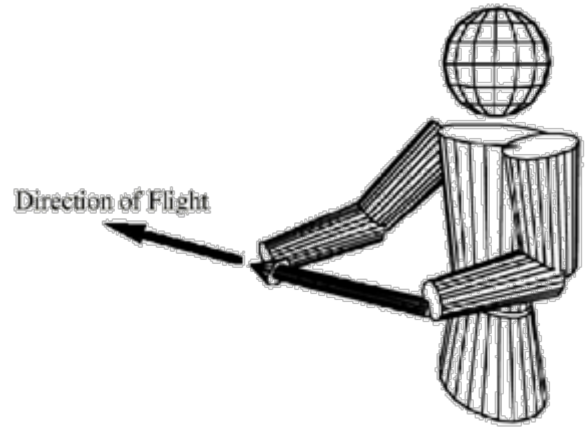


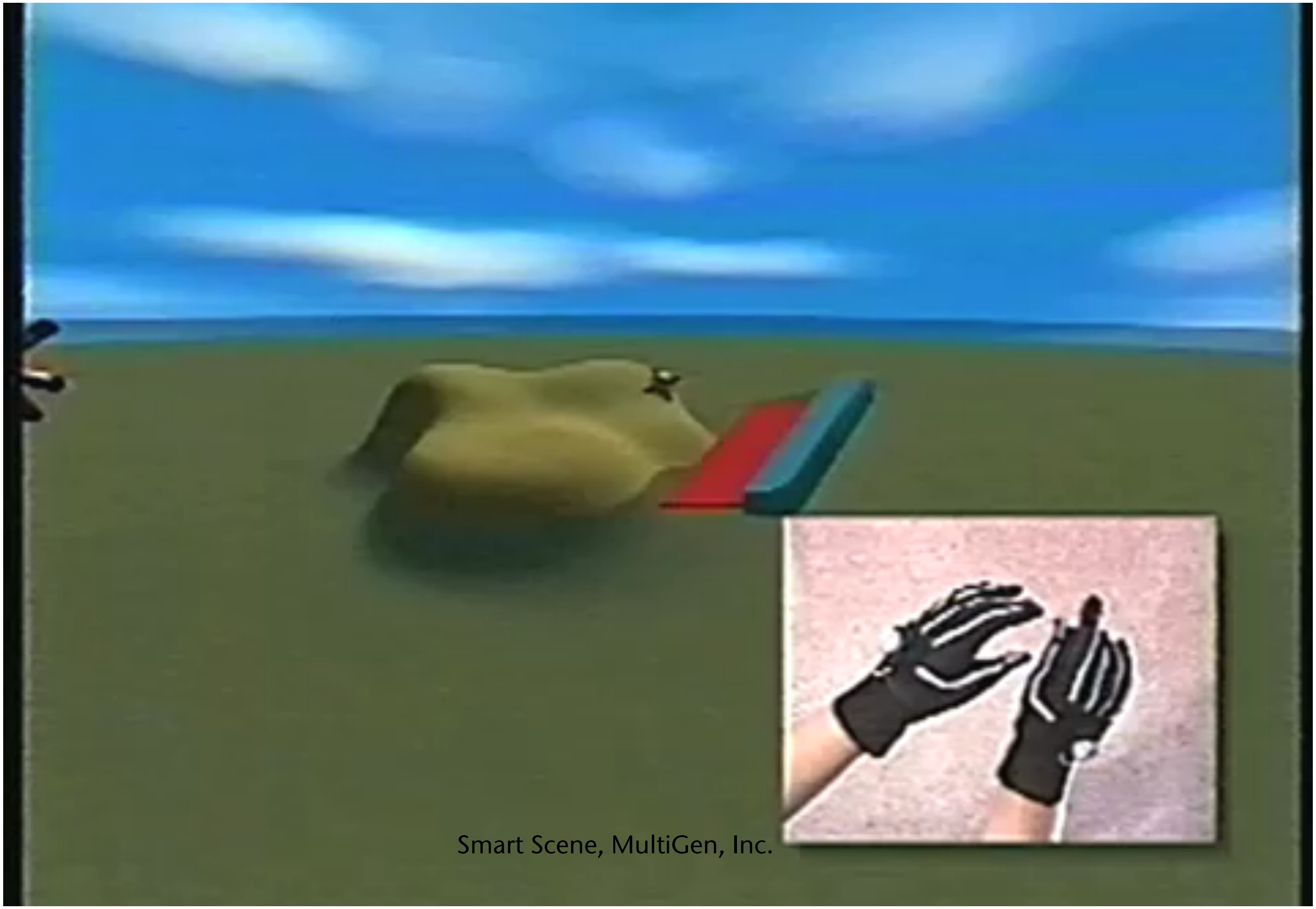
- *Eyeball-in-hand*:
  - Viewpoint is controlled directly by hand
  - Can be absolute or relative (accumulating) mode



Two-Handed Navigation (with Pinch Gloves)

- Question: how to navigate with both hands?
  (increase input bandwidth)
- Idea: only use 2 points and 1-2 triggers (→ pinch gloves)
- Idea: use "*scene-in-hand*"
  - 1 trigger, 1 moving point → translate the scene
  - 2 trigger, 1 fixed point , 1 moving point → rotate the scene
  - 2 trigger, 2 Punkte bewegt → scale the scene
- Not well-established in VR (probably because pinch gloves have not prevailed)
  - But: is the standard today on handhelds! ;-)
- Variation:
  - Direction = vector between both hands
  - Speed = length of vector





Smart Scene, MultiGen, Inc.

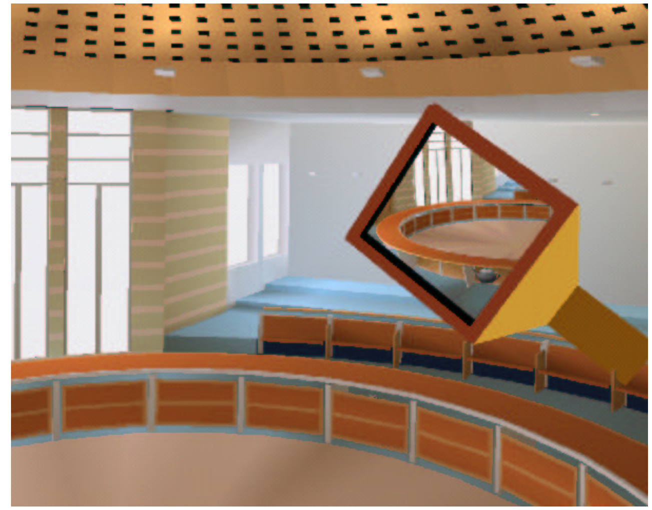
Navigation Without Hands

- Idea: project a scaled down version of the VE on the floor (map) and use feet
- Coarse navigation: teleportation → user walks to the new place/viewpoint on the map and triggers teleportation
- System commands involved:
  1. Bring up map = look at floor + trigger
  2. Teleportation = look at floor + trigger
  3. Dismiss map = look up + trigger
     - Trigger = speech command or "foot gesture"
- Accurate navigation: "lean" towards desired direction; speed = e.g., leaning angle

Exploration of VEs using a Magic Mirror

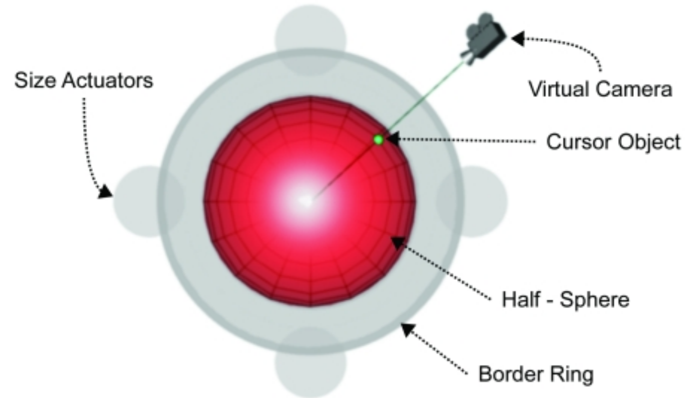
- Task/goal: present a second viewpoint (like inset in an image) intuitively in a VE, and allow for its manipulation
- Idea: use the mirror as a metaphor → "magic mirror"
  - One object serves as hand mirror (could even look like it)
  - Keeps a fixed position relative to camera (follows head motions)
  - Can be manipulated like any other object in the VE
- Additional features (not possible with real mirrors):
  - Zooming
  - Magnification / scaling down of image in mirror
  - Clipping of objects in front of mirror (which occlude mirror)
  - "*Un-mirror*" scene visible in mirror ("Richtig-herum-Drehen")
  - Switch between main viewpoint and mirror viewpoint

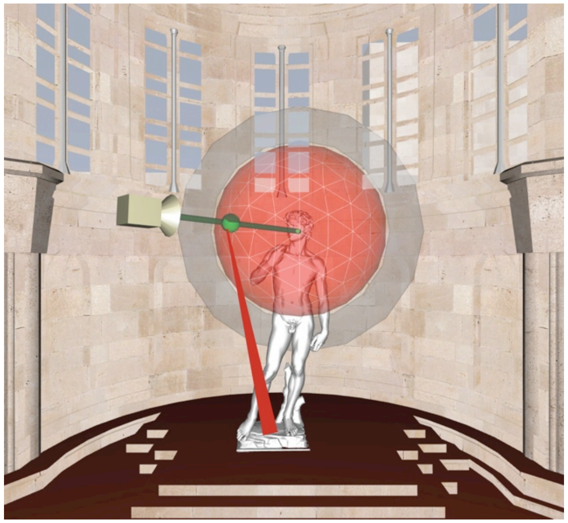
- Examples:
- Implementation:
  - Render 2x
  - First, render only into a small viewport (in the shape of the mirror) with mirrored viewpoint
  - Save as texture
  - Second, render into complete viewport from main viewpoint
  - Third, render texture on top of mirror object (no z test)
- Or, use method presented in Computer Graphics class



The Immersive Navidget [2008]

- Metaphor for defining the viewpoint directly
- Input device: *wand* with wheels and buttons
- Decomposition of the task:
  1. Define center of the sphere
     - Will be the new center of interest (COI)
     - E.g. by ray casting: shoot ray into scene, intersection point = new COI
  2. Define radius of sphere = distance of new viewpoint from COI
     - Here: specified using wheel on wand
  3. Define viewpoint on sphere (using ray)
  4. Animate viewpoint on path towards new viewpoint (= smooth teleportation)
  5. Switch to next phase using a button

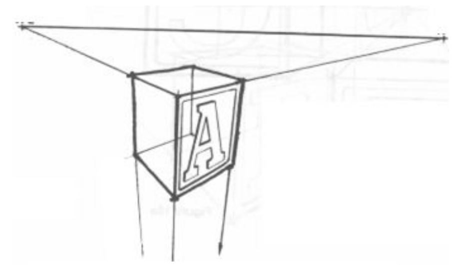


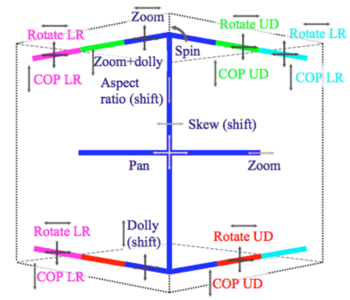


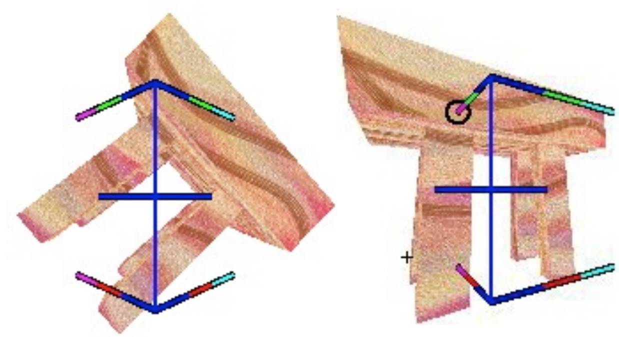

Navidget for Immersive Virtual Environments
Sebastian Knödel, Martin Hachet
iparla.labri.fr

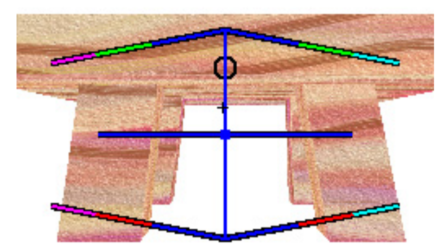
Digression: the IBar — an Intuitive 2D Metaphor

- Goal: an intuitive metaphor for manipulating the parameters of perspective projections
- Observation: drawing experts construct perspective drawings by way of vanishing points
- Idea:
  - Manipulate the vanishing points
  - As metaphor use the edges of a (projected) cube
- By manipulating the "handles" afforded by the cube, we can modify parameters:
  - Orientation, zoom, pan, proj. center









Interlude: User Models

- Idea: if we had a model of how users "work", then we could predict how they will interact with a specific UI and what their user performance will be
- Advantage (theoretically): no user studies and no UI mock-ups necesary any more
- Related fields: psychophysics, user interface design, usability

The Power Law of Practice

- Describes, what time is needed to perform an activity after the *n*-th repetition:

$$T_n = \frac{T_1}{n^a}$$

  T_1 = time needed for first performance of the activity,
  T_n = time for n-th repetition,
  $a \approx 0.2 \ldots 0.6$

- Warning:
  - Applies only to mechnical activities, e.g. :
    - Using the mouse, typing on the keyboard
  - Does not apply to cognitive activities, e.g., learning for exams! ;-)
- This effect must be kept in mind when designing experiments!

Hick's Law

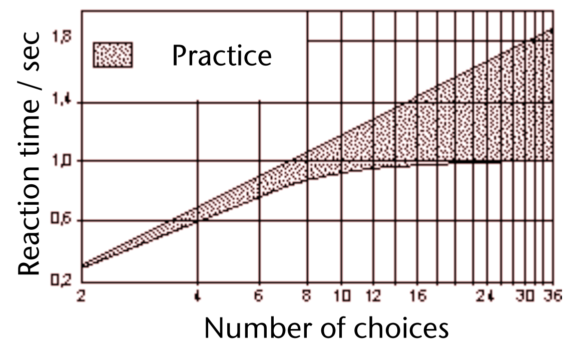
- Describes the time needed to make a 1-out-of-*n* selection, but there cannot be any cognitive workload involved:

$$T = I_c \log_2(n+1) \quad , \quad I_c \approx 150 \text{ msec}$$

  where n = number of choices

  - Example: *n* buttons + *n* lights, one is lighted up randomly, user has to press corresponding button
  - Assumption: the distribution of the choices is uniform!

- Warning: don't apply this law too blindly!
  - E.g., practice has a big influence on reaction time



- Sometimes, Hick's law is taken as proof that one large menu is more time-efficient than several small submenus ("rule of large menus") … I argue this is — mathematically — correct only because of the "+1", for which there is no clear experimental evidence! Besides, there are many other factors involved in large menus (clarity, Fitts' law, …)

Fitts's Law

- Describes the time needed to reach a target
- Task: reach and hit a specific target as quickly and as precisely as possible with your hand / pencil / mouse /etc., from a resting position → "target acquisition"
- The law:

$$T = b \log_2\left(\frac{D}{W} + 1\right) + a$$

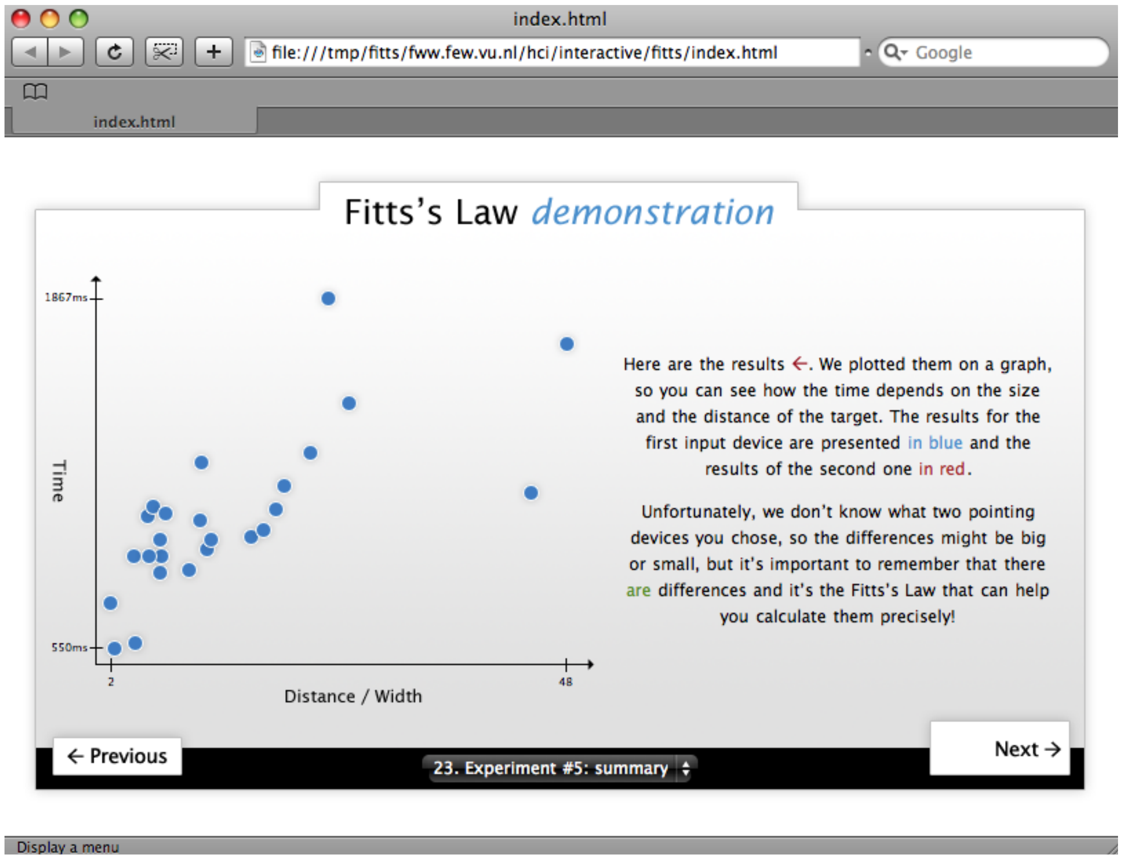
where D = distance between resting position and target,
W = diameter of the target

- The "index of difficulty" (ID) =

$$\log_2\left(\frac{D}{W} + 1\right)$$

Demo / Experiment

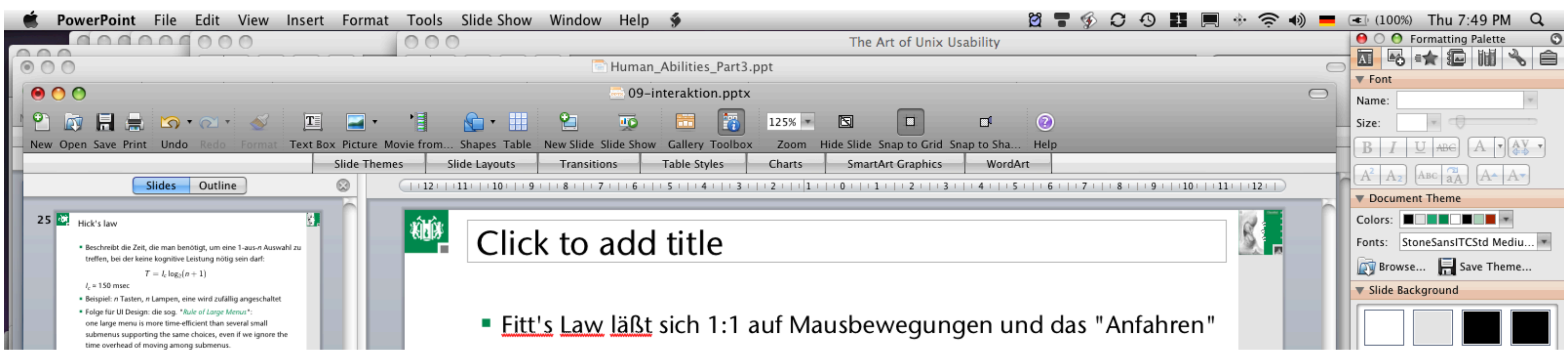
- Fitt's Law does apply directly to mouse movements needed to hit icons and buttons



Marcin Wichary , Vrije Universiteit: http://fww.few.vu.nl/hci/interactive/fitts/

Applications of Fitts' Law

- "Rule of Target Size": The size of a button should be proportional to its expected frequency of use
- Other consequences:
  "Macintosh fans like to point out that Fitts's Law implies a very large advantage for Mac-style edge-of-screen menus with no borders, because they effectively extend the depth of the target area off-screen. This prediction is verified by experiment."
  [Raymond & Landley: "The Art of Unix Usability", 2004]

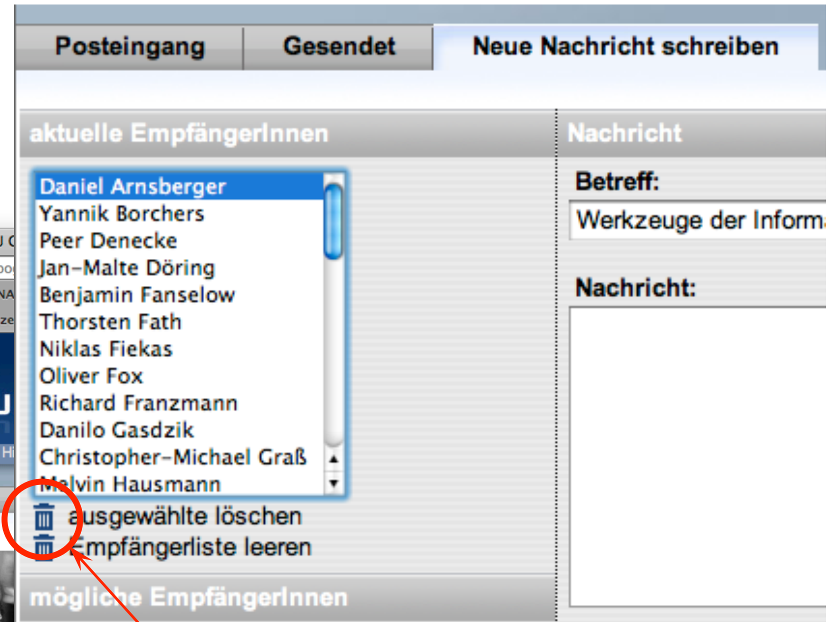


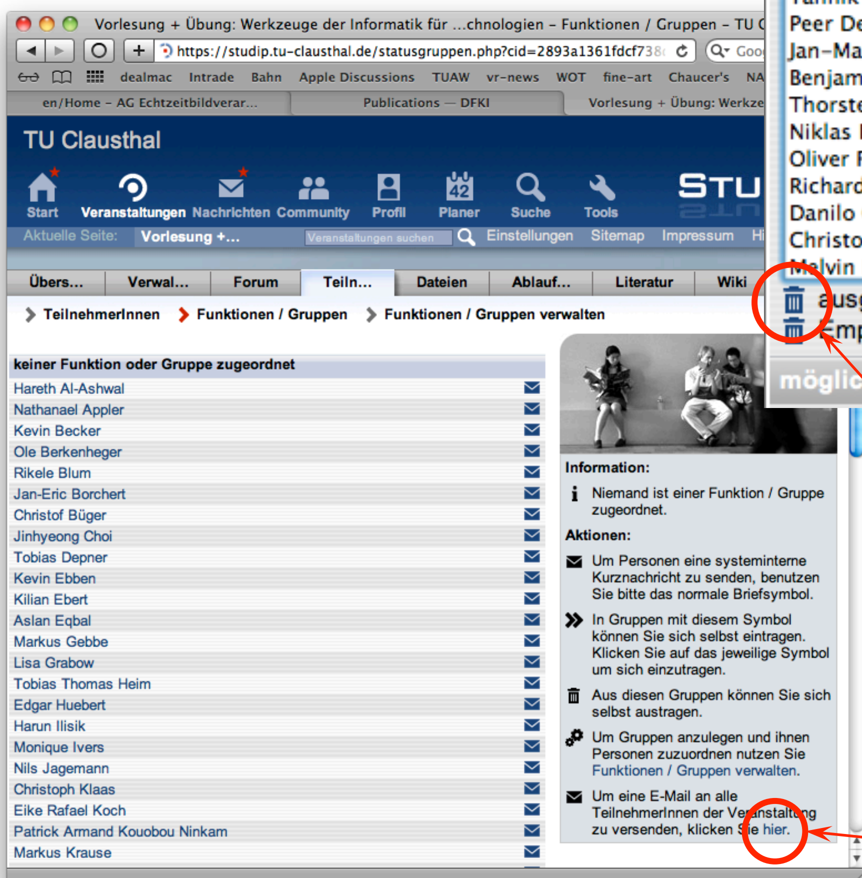
- *Tear-off menus* and *context menus*: they decrease the average travel distance *D*
- Apple's "Dock": the size of the icons gets adjusted dynamically

- Obvious limitations of Fitts's Law:
  - Fitts's Law cannot capture all aspects/widgets of a GUI
    - E.g. moving target (like scrollable lists)
  - There are many other decisions with regards to the design of a UI that are contrary to an application of Fitts's law

Fun and intructive quiz: go to the homepage of this VR course → scroll down to section "Online Literatur und Resources im Internet" → find "A Quiz Designed to Give You Fitts"

- Screenshots of Studip:





This small
symbol is a button!

This little word is a link!
(and hard to distinguish from the rest
of the text/background!)

Digression: the 80/20 Rule

- 80% all the total usage time of a product, we utilize only 20% of its features
  - Applies to menus, software as a whole, "consumer electronics", cars, ...
- 80% of the malfunctions of a product have their cause in only 20% of its components
- 80% all the old box in the software are caused by only 20% of its programmers and designers
- 80% all the revenue of the company is generated by only20% of their products
- ...