Universal design is about designing systems that are accessible by all users in all circumstances, taking account of human diversity in disabilities, age and culture.
Universal design helps everyone – for example, designing a system so that it can be used by someone who is deaf or hard of hearing will benefit other people working in noisy environments or without audio facilities. Designing to be accessible to screenreading systems will make websites better for mobile users and older browsers.
Multi-modal systems provide access to system information and functionality through a range of different input and output channels, exploiting redundancy.
Such systems will enable users with sensory, physical or cognitive impairments to make use of the channels that they can use most effectively. But all users benefit from multi-modal systems that utilize more of our senses in an involving interactive experience.
For any design choice we should ask ourselves whether our decision is excluding someone and whether there are any potential confusions or misunderstandings in our choice.

UNIT V

COGNITIVE MODELS:

The techniques and models in this chapter all claim to have some representation of users as they interact with an interface; that is, they model some aspect of the user understands, knowledge, intentions or processing. The level of representation differs from technique to technique – from models of high-level goals and the results of problem-solving activities, to descriptions of motor-level activity, such as keystrokes and mouse clicks. The

formalisms have largely been developed by psychologists, or computer scientists, whose interest is in understanding user behavior.

Competence models, therefore, represent the kinds of behavior expected of a user, but they provide little help in analyzing that behavior to determine its demands on the user. Performance models provide analytical power mainly by focussing on routine behavior in very limited applications.
Another useful distinction between these models is whether they address the acquisition or formulation of a plan of activity or the execution of that plan. Referring back to the interaction framework presented in Chapter 3, this classification would mean that some models are concerned with understanding the User and his associated task language while others are concerned with the articulation translation between that task language and the Input language. The presentation of the cognitive models in this chapter follows this classification scheme, divided into the following categories:

- hierarchical representation of the user's task and goal structure
- linguistic and grammatical models
- physical and device-level models.

GOAL AND TASK HIERARCHIES:

Many models make use of a model of mental processing in which the user achieves goals by solving subgoals in a divide-and-conquer fashion. We will consider two models, GOMS and CCT, where this is a central feature. Imagine we want to produce a report on sales of introductory HCI textbooks. To achieve this goal we divide it into several subgoals, say gathering the data together, producing the tables and histograms, and writing the descriptive material.
Concentrating on the data gathering, we decide to split this into further subgoals: find the names of all introductory HCI textbooks and then search the book sales database for these books. Similarly, each of the other subgoals is divided up into further subgoals, until some level of detail is found at which we decide to stop. We thus end up with a hierarchy of goals and subgoals. The example can be laid out to expose this structure:
These two questions are issues of granularity, and both of the methods described below leave this to some extent in the hands of the designer. Different design issues demand different levels of analysis. However, both methods operate at a relatively low level; neither would attempt to start with such an abstract goal as 'produce a report' which will involve real creativity and difficult problem solving. Instead they confine themselves to more routine learned behavior. This most abstract task is referred to as the unit task. The unit task does not require any problem-solving skills on the part of the user, though it frequently demands quite sophisticated problem-solving skills on the part of the designer to determine them.

GOMS:

The GOMS model of Card, Moran and Newell is an acronym for Goals, Operators, Methods and Selection [56]. A GOMS description consists of these four elements: Goals These are the user's goals, describing what the user wants to achieve.

Further, in GOMS the goals are taken to represent a 'memory point' for the user, from which he can evaluate what should be done and to which he may return should any errors occur.

Operators These are the lowest level of analysis. They are the basic actions that the user must perform in order to use the system. They may affect the system (for example, press the 'X' key) or only the user's mental state (for example, read the dialog box). There is still a degree of flexibility about the granularity of operators; we may take the command level 'issue the SELECT command' or be more primitive: 'move mouse to menu bar, press center mouse button . . .'.

Methods As we have already noted, there are typically several ways in which a goal can be split into subgoals. For instance, in a certain window manager a currently selected window can be closed to an icon either by selecting the 'CLOSE' option from a pop-up menu, or by hitting the 'L7' function key. In GOMS these two goal decompositions are referred to as methods, so we have the CLOSE-METHOD and the L7-METHOD:

DESIGN FOCUS:

GOMS saves money

Some years ago the US telephone company NYNEX were intending to install a new computer system to support their operators. Before installation a detailed GOMS analysis was performed taking into account the cognitive and physical processes involved in dealing with a call. The particular technique was rather different from the original GOMS notation as described here. Because an operator performs several activities in parallel a PERT-style GOMS description was constructed [192, 154]. The PERT analysis was used to determine the critical path, and hence the time to complete a typical task. It was discovered that rather than speeding up operations, the new system would take longer to process each call. The new system was abandoned before installation, leading to a saving of many millions of dollars.

Cognitive complexity theory

Cognitive complexity theory, introduced by Kieras and Polson [199], begins with the basic premises of goal decomposition from GOMS and enriches the model to provide more predictive power. CCT has two parallel descriptions: one of the user's goals and the other of the computer system (called the device in CCT). The description of the user's goals is based on a GOMS-like goal hierarchy, but is expressed primarily using production rules. We introduced production rules in Chapter 1 and we further describe their use in CCT below. For the system grammar, CCT uses generalized transition networks, a form of state transition network. This will not be described here, but state transition networks will be discussed in detail in Chapter 16.

The production rules are a sequence of rules: if condition then action where condition is a statement about the contents of working memory. If the condition is true then the production rule is said to fire. An action may consist of one or more elementary actions, which may be either changes to the working memory, or external actions such as keystrokes. The production rule 'program' is written in a LISP-like language.

As an example, we consider an editing task using the UNIX vi text editor. The task is to insert a space where one has been missed out in the text, for instance if we noticed that in the above paragraph we had written

'cognitivecomplexity theory'. This is a reasonably frequent typing error and so we assume that we have developed good procedures to perform the task. We consider a fragment of the associated CCT production rules.

```
(SELECT-INSERT-SPACE
IF (AND (TEST-GOAL perform unit task)
(TEST-TEXT task is insert space)
(NOT (TEST-GOAL insert space))
(NOT (TEST-NOTE executing insert space)) )
THEN ( (ADD-GOAL insert space)
(ADD-NOTE executing insert space)
(LOOK-TEXT task is at %LINE %COL) ))
(INSERT-SPACE-DONE
IF (AND (TEST-GOAL perform unit task)
(TEST-NOTE executing insert space)
(NOT (TEST-GOAL insert space)) )
THEN ( (DELETE-NOTE executing insert space)
(DELETE-GOAL perform unit task)
(UNBIND %LINE %COL) ))
(INSERT-SPACE-1
IF (AND (TEST-GOAL insert space)
(NOT (TEST-GOAL move cursor))
(NOT (TEST-CURSOR %LINE %COL)) )
THEN ( (ADD-GOAL move cursor to %LINE %COL) ))
(INSERT-SPACE-2
IF (AND (TEST-GOAL insert space)
(TEST-CURSOR %LINE %COL) )
THEN ( (DO-KEYSTROKE 'I')
(DO-KEYSTROKE SPACE)
(DO-KEYSTROKE ESC)
(DELETE-GOAL insert space) ))
```

To see how these rules work, imagine that the user has just seen the typing mistake and thus the contents of working memory (w.m.) are

```
(GOAL perform unit task)
(TEXT task is insert space)
(TEXT task is at 5 23)
(CURSOR 8 7)
```

Problems and extensions of goal hierarchies:

The formation of a goal hierarchy is largely a post hoc technique and runs a very real risk of being defined by the computer dialog rather than the user. One way to rectify this is to produce a goal structure based on pre-existing manual procedures and thus obtain a natural hierarchy [201]. To be fair, GOMS defines its domain to be that of expert use, and thus the goal structures that are important are those which users develop out of their use of the system. However, such a natural hierarchy may be particularly useful as part of a CCT analysis, representing a very early state of knowledge.

LINGUISTIC MODELS:

BNF

Representative of the linguistic approach is Reisner's use of Backus–Naur Form (BNF) rules to describe the dialog grammar [301]. This views the dialog at a purely syntactic level, ignoring the semantics of the language. BNF has been used widely to specify the syntax of computer programming languages, and many system dialogs can be described easily using BNF rules. For example, imagine a graphics system that has a line-drawing function. To select the function the user must select the 'line' menu option. The line-drawing function allows the user to draw a polyline, that is a sequence of line arcs between points. The user selects the points by clicking the mouse button in the drawing area. The user double clicks to indicate the last point of the polyline.

draw-line ::= select-line + choose-points
+ last-point
select-line ::= position-mouse + CLICK-MOUSE
choose-points ::= choose-one
| choose-one + choose-points
choose-one ::= position-mouse + CLICK-MOUSE
last-point ::= position-mouse + DOUBLE-CLICK-MOUSE
position-mouse ::= empty | MOVE-MOUSE + position-mouse

The names in the description are of two types: non-terminals, shown in lower case, and terminals, shown in upper case. Terminals represent the lowest level of user behavior, such as pressing a key, clicking a mouse button or moving the mouse.

Non-terminals are higher-level abstractions. The non-terminals are defined in terms of other non-terminals and terminals by a definition of the form

name ::= expression

The '::=' symbol is read as 'is defined as'. Only non-terminals may appear on the left of a definition. The right-hand side is built up using two operators '+' (sequence) and '|' (choice). For example, the first rule says that the non-terminal draw-line is defined to be select-line followed by choose-points followed by lastpoint.

All of these are non-terminals, that is they do not tell us what the basic user actions are. The second rule says that select-line is defined to be position mouse (intended to be over the 'line' menu entry) followed by CLICK-MOUSE. This is our first terminal and represents the actual clicking of a mouse.

Task–action grammar

Measures based upon BNF have been criticized as not 'cognitive' enough. They ignore the advantages of consistency both in the language's structure and in its use of command names and letters. Task–action grammar (TAG) [284] attempts to deal with some of these problems by including elements such as parametrized grammar rules to emphasize consistency and encoding the user's world knowledge (for example, up is the opposite of down).

To illustrate consistency, we consider the three UNIX commands: cp (for copying files), mv (for moving files) and ln (for linking files). Each of these has two possible forms. They either have two arguments, a source and destination filename, or have any number of source filenames followed by a destination directory:

The Challenge of Display-Based Systems:

Both goal hierarchical and grammar-based techniques were initially developed when most interactive systems were command line, or at most, keyboard and cursor based.

There are significant worries, therefore, about how well these approaches can generalize to deal with more modern windowed and mouse-driven interfaces.

Both families of techniques largely ignore system output – what the user sees.

The implicit assumption is that the users know exactly what they want to do and execute the appropriate command sequences blindly. There are exceptions to this.

We have already mentioned how Reisner's BNF has been extended to include assertions about output. In addition, TAG has been extended to include information about how the display can affect the grammar rules [180].

Another problem for grammars is the lowest-level lexical structure. Pressing a cursor key is a reasonable lexeme, but moving a mouse one pixel is less sensible. In addition, pointer-based dialogs are more display oriented. Clicking a cursor at a particular point on the screen has a meaning dependent on the current screen contents.

This problem can be partially resolved by regarding operations such as 'select region of text' or 'click on quit button' as the terminals of the grammar. If this approach is taken, the detailed mouse movements and parsing of mouse events in the context of display information (menus, etc.) are abstracted away.

Physical and Device Models:

Keystroke-level model

Compared with the deep cognitive understanding required to describe problem solving activities, the human motor system is well understood. KLM (Keystroke-Level Model [55]) uses this understanding as a basis for detailed predictions about user performance. It is aimed at unit tasks within interaction – the execution of simple command sequences, typically taking no more than 20 seconds. Examples of this would be using a search and replace feature, or changing the font of a word. It does not extend to complex actions such as

producing a diagram. The assumption is that these more complex tasks would be split into subtasks (as in GOMS) before the user attempts to map them into physical actions. The task is split into two phases:
acquisition of the task, when the user builds a mental representation of the task;
execution of the task using the system's facilities.
KLM only gives predictions for the latter stage of activity. During the acquisition phase, the user will have decided how to accomplish the task using the primitives of the system, and thus, during the execution phase, there is no high-level mental activity – the user is effectively expert. KLM is related to the GOMS model, and can be thought of as a very low-level GOMS model where the method is given.
The model decomposes the execution phase into five different physical motor operators, a mental operator and a system response operator:
K Keystroking, actually striking keys, including shifts and other modifier keys.
B Pressing a mouse button.
P Pointing, moving the mouse (or similar device) at a target.
H Homing, switching the hand between mouse and keyboard.
D Drawing lines using the mouse.
M Mentally preparing for a physical action.
R System response which may be ignored if the user does not have to wait for it, as in copy typing.
The execution of a task will involve interleaved occurrences of the various operators.
For instance, imagine we are using a mouse-based editor. If we notice a single character error we will point at the error, delete the character and retype it, and then return to our previous typing point. This is decomposed as follows:
1. Move hand to mouse H[mouse]
2. Position mouse after bad character PB[LEFT]
3. Return to keyboard H[keyboard]
4. Delete character MK[DELETE]
5. Type correction K[char]
6. Reposition insertion point H[mouse]MPB[LEFT]

Notice that some operators have descriptions added to them, representing which device the hand homes to (for example, [mouse]) and what keys are hit (for example, LEFT – the left mouse button).
The model predicts the total time taken during the execution phase by adding the component times for each of the above activities. For example, if the time taken for one keystroke is tK, then the total time doing keystrokes is
TK = 2tK
Similar calculations for the rest of the operators give a total time of
Texecute = TK + TB + TP + TH + TD + TM + TR

= 2tK + 2tB + tP + 3tH + 0 + 2tM + 0

In this example, the system response time was zero. However, if the user had to wait for the system then the appropriate time would be added. In many typing tasks, the user can type ahead anyway and thus there is no need to add response times. Where needed, the response time can be measured by observing the system.

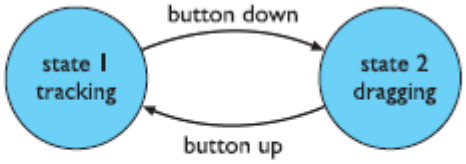


Figure 12.1 Mouse transitions: states 1 and 2

Figure 12.2 Light pen transitions: three states

COGNITIVE ARCHITECTURES:

The formalisms we have seen so far have some implicit or explicit model of how the user performs the cognitive processing involved in carrying out a task. For instance, the concept of taking a problem and solving it by divide and conquer using subgoals is central to GOMS. CCT assumes the distinction between long- and short-term memory, with production rules being stored in long-term memory and 'matched' against the contents of short-term (or working) memory to determine which 'fire'. The values for various motor and mental operators in KLM were based on the Model Human Processor (MHP) architecture of Card, Moran and Newell [56]. Another common assumption, which we have not discussed in this chapter, is the distinction between linguistic levels – semantic, syntactic and lexical – as an architectural model of the user's understanding.

The problem space model

Rational behavior is characterized as behavior that is intended to achieve a specific goal. This element of rationality is often used to distinguish between intelligent and machine-like behavior. In the field of artificial intelligence (AI), a system exhibiting rational behavior is referred to as a knowledge-level system. A knowledge-level system contains an agent behaving in an environment. The agent has knowledge about itself and its environment, including its own goals. It can perform certain actions and sense information about its changing environment. As the agent behaves in its environment, it changes the environment and its own knowledge. We can view the overall behavior of the knowledge-level system as a sequence of environment and

agent states as they progress in time. The goal of the agent is characterized as a preference over all possible sequences of agent/environment states.

Interacting cognitive subsystems

Barnard has proposed a very different cognitive architecture, called interacting cognitive subsystems (ICS) [24, 25, 27]. ICS provides a model of perception, cognition and action, but unlike other cognitive architectures, it is not intended to produce a description of the user in terms of sequences of actions that he performs. ICS provides a more holistic view of the user as an information-processing machine. The emphasis is on determining how easy particular procedures of action sequences become as they are made more automatic within the user.

ICS attempts to incorporate two separate psychological traditions within one cognitive architecture. On the one hand is the architectural and general-purpose information-processing approach of short-term memory research. On the other hand is the computational and representational approach characteristic of psycholinguistic research and AI problem-solving literature.

Ubiquitous Computing and Augmented Realities:

There are several ways in which the earliest assumptions of HCI are challenged. For example, we no longer assume there is a single user; rather, we consider groups and larger organizational concerns when discussing interactions. In this chapter, we challenge another assumption concerning the form factor of the computing device. Traditionally, we think of computers as a glass box, a workstation with keyboard, mouse and monitor sitting on a desk that we seek out when we want to do some work. Since the late 1980s, this traditional form factor has been expanded to include a variety of more mobile devices and computing services that are distributed throughout the physical world and more tightly integrated with it. The trend is towards a ubiquitous or pervasive computing experience, in which computing devices become so commonplace that we do not distinguish them from the 'normal' physical surroundings. Improved display technologies give us the ability to augment the physical world with electronic information through head-mounted displays or steerable projection surfaces. We have also seen display technologies that provide complete virtual replacements of the physical world, creating a so-called virtual reality.

Ubiquitous Computing Applications Research:

We first introduced the notion of ubiquitous computing (or pervasive computing) in Chapter 4. The interest in ubiquitous computing has surged over the past few years, thanks to some influential writings and plenty of experimental work. The defining characteristic of ubiquitous computing is the attempt to break away from the traditional desktop interaction paradigm and move computational power into the environment that surrounds the user. Rather than force the user to search out and find the computer's interface, ubiquitous computing suggests that the interface itself can take on the responsibility of locating and serving the user. Mark Weiser is credited with coining the phrase ubiquitous computing (or ubicomp) when he put forth a vision of people and environments augmented with computational resources that provide information and services when and where desired [369]. Though his vision has excited many technologists, it is important to realize that the main motivation behind Weiser's vision was centered on the impact ubicomp would have on the human experience:

Defining the appropriate physical interaction experience

Ubiquitous computing inspires application development that is 'off the desktop'. In addition to suggesting a freedom from a small number of well-defined interaction locales (the desktop), this vision assumes that physical interaction between humans and computation will be less like the current desktop keyboard/mouse/display paradigm and more like the way humans interact with the physical world. Humans speak, gesture and use writing implements to communicate with other humans and alter physical artifacts. The drive for a ubiquitous computing experience has resulted in a variety of important changes to the input, output and interactions that define the human experience with computing. We describe three of those changes in this section.

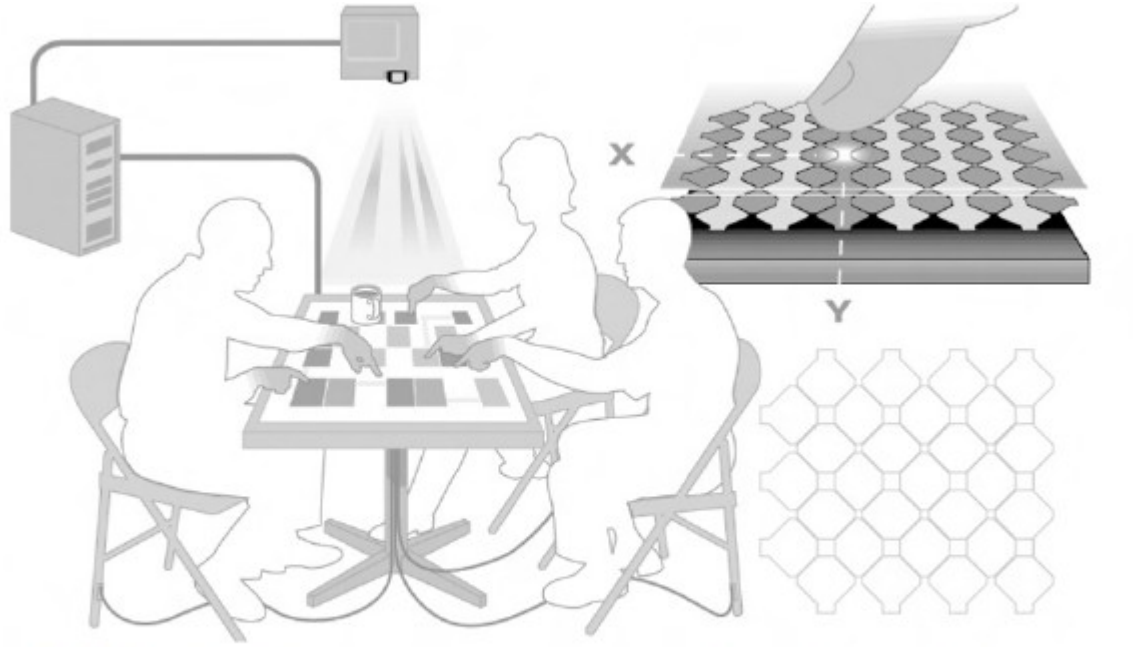


Figure 20.1 The DiamondTouch input technology from Mitsubishi Electric Research Lab (MERL) uses capacitive coupling through humans to provide a large-scale input surface for multiple simultaneous users. See www.merl.com/projects/DiamondTouch/ for more details. Source: Courtesy of Mitsubishi Electric Research Laboratories, Inc.

These recognition technologies are some examples of interpreting meaning from sensed signals of human activity. There are many other ways to infer information about people and environments by sensing a variety of other physical signals. There have been many recent advances in sensing the physical world; the significance here is that sensing and interpretation of human activity provides a more implicit notion of input to an interactive system. For example, many researchers have investigated how simple sensors such as RFID (the technology behind the security tags used on library books, clothes in shops, etc.), accelerometers, tilt sensors, capacitive coupling,infrared range finders and others can be incorporated into artifacts to increase the language of input from the user to control that artefact.

Seamless integration of physical and virtual worlds:

An important feature of ubicomp technology is that it attempts to merge computational artifacts smoothly with the world of physical artifacts. There have been plenty of examples demonstrating how electronic information can be overlaid upon the real world, thus producing an augmented reality [130].

Application themes for ubicomp:

Many applications-focussed researchers in HCI seek the holy grail of ubicomp, the killer app that will cause significant investment in the infrastructure that will then enable a wide variety of ubicomp applications to flourish. It could be argued that person– person communication is such a killer app for ubicomp, as it has caused a large investment in environmental and personal infrastructure that has moved us close (though not entirely) to a completely connected existence. Whether or not personal communication is the killer app, the vision of ubicomp from the human perspective is much more holistic.

Context-aware computing:

Two compelling early demonstrations of ubicomp were the Olivetti Research Lab's Active Badge [361] and the Xerox PARCTab [362], both location-aware appliances.

These devices leverage a simple piece of context, user location, and provide valuable services (automatic call forwarding for a phone system, automatically updated maps of user locations in an office). This technology was also the basis for the Pepys automatic diary system described in Chapter 18 (Section 18.4.1). These simple locationaware appliances are perhaps the first demonstration of linking implicit human activity with computational services that serve to augment general human activity.

Location of identifiable entities (usually people) is a very common piece of context used in ubicomp application development. The most widespread applications have been GPS-based car navigation systems and handheld 'tour guide' systems that vary the content displayed (video or audio) by a handheld unit given the user's physical location in an exhibit area [6, 68]. For example, The Sentient Computing Project uses a 3-D ultrasonic indoor location system to track each worker in a building and so maintain a map of office worker locations that helps coworkers find each other and talk by phone.

Virtual and Augmented Reality:

Virtual reality (VR) refers to the computer-generated simulation of a world, or a subset of it, in which the user is immersed. It represents the state of the art in multimedia systems, but concentrates on the visual senses. VR allows the user to experience situations that are too dangerous or expensive to enter 'in the flesh'. Users may explore the real world at a different scale and with hidden features made visible.

Alternatively, the virtual worlds that are generated may be entirely synthesized: realistic within themselves, but purely a manifestation of electronic structures. The term 'virtual reality' conjures up an image of a user weighed down with a helmet or goggles, grasping, apparently blindly, into empty space. The user, isolated within his virtual environment, moves through a simulated landscape, picking up objects on the way. This is fully immersive VR. However, it is only one part of the spectrum of VR, which also includes desktop VR, command and control situations, and augmented reality, where virtuality and reality meet.

VR technology

The technology involved in VR is quite elaborate. The individual devices have been discussed in Chapter 2, but now we shall see how they work together.

Since the user has to 'see' a new environment, a headset is usually used in a VR setup. With independent screens for each eye, in order to give a 3D image, the headset is often a large, relatively cumbersome piece of head-mounted gear. However, smaller, lighter VR goggles are now available and may soon become only slightly oversized spectacles.

Immersive VR:

Virtual reality can be used to produce environments that mimic our everyday world. Architects have always used models and sketches to show clients how a building will appear. Now they can use VR to take clients through a virtual tour of the building, fly over it from above, look at it from the streets outside, enter the doors and walk through the corridors. Similar techniques are used to plan kitchens and even gardens.

However, there are also many things that we cannot see, either because they are invisible to the naked eye (heat, magnetism, gravity) or because they are too small or too large. Scientific and data visualization systems make use of VR technology to expose and explore these features. In Section 20.4 we will see one example of this in the virtual wind tunnel, which makes the airflows around an aircraft wing visible using virtual bubble trails (see Figure 20.6). Another example is in the field of protein chemistry, where individual molecules are, of course, too small to see except by electron microscope.

VR on the desktop and in the home:

Virtual reality has been made possible by the advent of very fast high-performance computers. Despite the exponential rise in processor speeds, high-resolution immersive VR is still not available for mass-market applications, and many systems are primarily research projects. Desktop VR is a lower-cost alternative. In desktop VR, 3D images are presented on a normal computer screen and manipulated using mouse and keyboard, rather than using goggles and datagloves. Many readers may have used such systems on personal computers or games consoles: flight simulators, or interactive games such as DOOM or MYST.

Command and control:

In many command and control situations, real users are at one remove from the physical world, seeing it through windows, or cameras. The windows or video screens can be replaced by synthesized pictures. The user operates within an immediate physical environment, with real controls and instruments, but the world outside is virtual.

One such interactive VR application in widespread use is the flight simulator. A full cockpit system is placed in a hydraulically supported container, with large screens replacing the cockpit windows. Images are generated and projected onto the screens, whilst the box can be moved rapidly in any direction by the hydraulic rams.

The visual information and physical motion simulate accurately the conditions encountered by aircraft. Flight simulators are used extensively in pilot training programs.

Landings can be practiced, with the system responding to the commands of the pilot; descending too fast and off to one side, the pilot will have to correct the situation if she wishes to avoid a crash. Emergency situations can also be created, in which aircraft system malfunctions can be artificially created in order to train the pilot

to take the correct course of corrective or life-preserving action. With VR, entertainment is never far behind and this kind of system can also be found in many fun fairs!.

AUGMENTED REALITY:

In augmented reality systems electronic images are projected over the real world – virtuality and reality meet. The head-up displays in many aircraft and even some automobiles can be regarded as an example of this, but the data in such displays are not typically connected to the objects seen through them and, hence, the blend between virtuality and reality is quite weak.

A stronger sense of connection can be obtained using semi-transparent goggles. Users can move around in the real world and see real objects, but computer images are reflected off the inside of the glass and overlay the physical objects. Again, this can be used to show unrelated information; for example, some wearable computers allow users to read their email whilst walking around. However, the real sense of two worlds meeting comes when the projected image in some way links or refers to the object it overlays. For example, one experimental system has virtual balls, which can be picked up and thrown by the user [11]. When the virtual ball 'hits' the real wall it bounces off. The balls can even bounce down a real staircase.

INFORMATION AND DATA VISUALIZATION:

Virtual reality and 3D displays can be used to visualize scientific data and other complex information. Whether or not 3D representations are used, animation techniques, especially when under interactive user control, can give a sense of engagement with data, and encourage discovery and pattern formation.

Scientific and technical data:

Three-dimensional representations of scientific and technical data can be classified by the number of dimensions in the virtual world that correspond to physical spatial dimensions, as opposed to those that correspond to more abstract parameters. Perhaps the most engaging images are where all three dimensions have some physical validity. An example of this is the virtual wind tunnel [50]. In a physical wind tunnel, an accurate model of an aircraft is constructed and then subjected to winds that, when appropriately scaled, correspond to realistic situations. The intention is to investigate patterns of air movement and pressure, for example to discover those places where turbulence forms. Of course, air is invisible, so small pieces of ribbon may be attached to the aircraft surface, small bubbles released into the chamber or polarized light used to expose the hidden airflows. In the virtual wind tunnel, air movements are calculated using the equations of fluid dynamics. An engineer can then see the simulated aircraft using VR goggles and can move around a (virtual) baton from which stream (virtual) bubbles (Figure 20.6). By moving the baton to different parts of the aircraft, areas of interest can be investigated.

Structured information:

Scientific data are typically numeric, so can easily be mapped onto a dimension in virtual space. In contrast, the data sets that arise in information systems typically have many discrete attributes and structures: hierarchies, networks and, most complex of all, free text. Examples of hierarchies include file trees and organization charts. Examples of networks include program flow charts and hypertext structures.

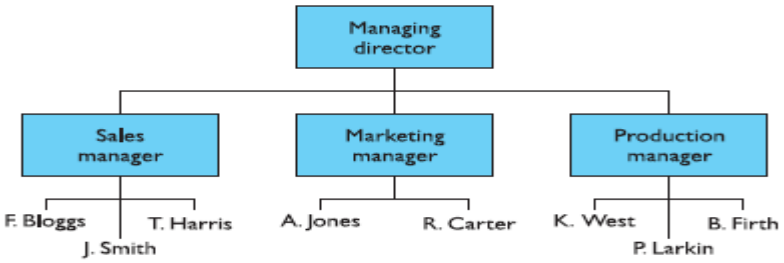


Figure 20.7 Two-dimensional organization chart

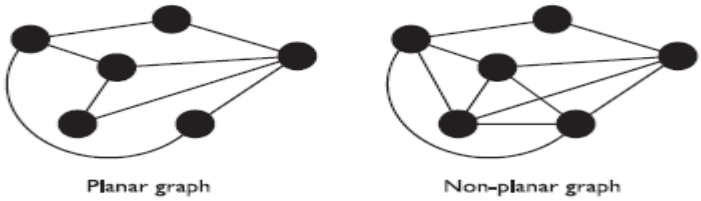


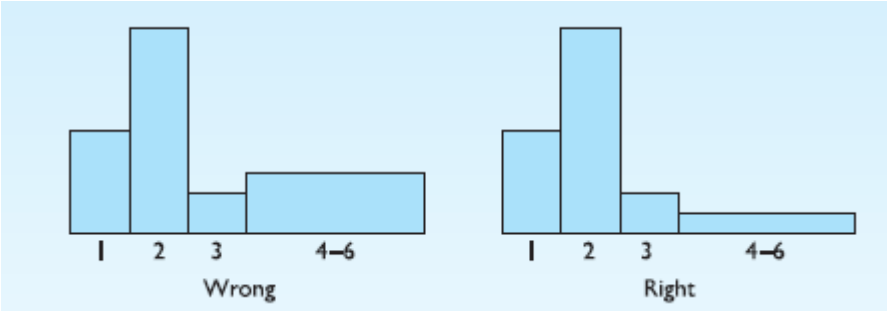
Figure 20.8 Two-dimensional network layout

Time and interactivity:

We can consider time and visualization from two sides. On the one hand, many data sets include temporal values (dates, periods, etc.) that we wish to visualize. On the other hand, the passage of time can itself be used in order to visualize other types of data. In 2D graphs, time is often mapped onto one spatial dimension; for example, showing the monthly sales figures of a company. Where the time-varying data are themselves 2D images, multiple snapshots can be used. Both comic books and technical manuals use successive images to show movement and changes, often augmented by arrows, streamlines or blurring to give an impression of direction and speed. Another type of temporal data is where events occur at irregular intervals. Timelines are often used for this sort of data, where one dimension is used to represent time and the second axis is used to represent the type of activity.

Getting the size right

When faced with a 2D histogram it is the area that we perceive as being the size of a bar, not the height.
Survey data of family groups at a fun fair have shown that 25% had only one child, 50% had two children, 10% had three children and the remaining 15% of families had between four and six children.



DESIGN FOCUS:

Getting the size right

If we display the height of the 4–6 group proportional to the percentage of families it makes it look as if this is much more than 15% of the data. This is because the column is three times wider than the rest. We actually perceive the data to be in the ratio 25:50:10:45. The right thing to do is to draw the area of the columns proportional to the number of families, which makes the data look right.

Similar problems arise in 3D representations. If we draw a map with 3D columns proportional to the population rising from each country, we will probably not be able to see anything except a single enormous block for China! We should display the heights of the blocks proportional to the population density. Of course, if we start with density data, we can simply use height to start with.

If data ranges are extremely large, we may use non-linear (for example, logarithmic) scales. Users clearly need to be aware of this, but at least if density data are used, bigger areas/volumes represent bigger data.