

Consiglio Nazionale delle Ricerche

**Design of a virtual world
for space planning**

Patrizia Palamidese

CNUCE C97-013

CNUCE

| | |
|--|-----------|
| 1. INTRODUCTION | 4 |
| 2. HUMAN-COMPUTER DIALOGUE | 6 |
| 2.1. The spaceball | 8 |
| 2.2. Interface states | 9 |
| 2.3. The <i>virtual hand</i> model | 10 |
| 2.4. The <i>eye on the transparent cap</i> model | 11 |
| 2.5. User's bearings | 11 |
| 3. STEREO VIEW | 12 |
| 3.1. The stereo pair | 13 |
| 3.2. Comments | 16 |
| 4. WORLD BEHAVIOUR | 17 |
| 4.1. Performance | 18 |
| 4.1.1. Time step | 18 |
| 4.1.2. Number of tests | 19 |
| 4.1.3. Accuracy | 19 |
| 4.2. Algorithms | 20 |
| 5. TESTS | 21 |
| 6. CONCLUSIONS AND FUTURE DEVELOPMENTS | 21 |
| 7. BIBLIOGRAPHY | 24 |

Abstract

We describe a low cost virtual reality interface to execute space planning in a 3D space by direct manipulations of 3D objects. Operators can work in the virtual world in a similar way to the real world: they perceive the position of objects through the depth cue of stereo view, and can grab and push them in any direction until they reach the destination. Objects can be put on top of each other or lined up. To help users reach their objectives, the virtual world is modeled to be a *job oriented* world, thus it is governed by a few simple rules which facilitate positioning. We describe the design and implementation strategies to obtain a real time performance on a low level workstation.

KEYWORDS

virtual reality, human-computer interaction, stereo view, 3D graphics, space planning

1. Introduction

There are many situations where one needs to compose 3D objects (like a 3D puzzle) and check several configurations before deciding on the right one. For example, in architecture and archaeology there are interesting applications where it may be necessary to restore a destroyed monument on the basis of a set of 3D pieces of evidence (e.g. heavy marble pieces) whose original position is unknown. In these situations it is more convenient to evaluate the feasibility of a reconstruction on a virtual model before deciding the real work.

Moving objects and finding their correct position is an obvious operation for a person who works in the real world with the help of human senses (mainly sight and touch) and physical laws (e.g. gravity and friction). In our virtual system the user is guided by the depth cue provided by a stereo view on the workstation screen. This technology doesn't have the same characteristics as a view in the real world, but nevertheless it offers an interesting environment for a number of applications. It has one main limitation: the view field is smaller and the user is not surrounded by objects, hence is not fully immersed in the world.

How immersive a virtual system is depends on the I/O devices it includes (Bryson [1]): usually the system is configured according to the costs permitted and the application requirements. We have chosen a partially immersive configuration to test the capability of a low cost environment to accomplish serious rather than pure entertainment tasks. Our configuration presents three considerable advantages: a) users can see the objects coming out of the screen in front of them (or moving inwards inside the computer) with a resolution superior than the head-mounted displays; b) users are not subject to any ill effects caused by head-mounted displays; c) this system is low cost since it is based on a Silicon Indigo workstation equipped with a stereo screen and a spaceball.

The approach we describe in this paper is not a CAD approach, rather it enables objects to be positioned in space with reasonable accuracy without resorting to precise measurements. In other words operators can execute the operation *manually* and decides that the task is finished on the basis of visual inspection. *What they see is what they get*: two objects might visually

seem to be satisfactorily aligned but they are not perfectly aligned from the geometric point of view.

On the other hand, since users cannot position an object where they want in space by sight only, we have associated some properties with objects to provide a response to user actions. Modeling virtual worlds is a research area which tries to balance the maximum realism to the maximum performance under direct manipulation. There are examples of virtual objects which satisfy either physical or geometric laws or both.

Papper [2] proposes a general solution to the virtual world modeling problem: he describes a world containing simplified physical constraints in order to provide a natural and intuitive working environment for users who want to position objects in space. He defines very intuitive objects properties so that users can find them by practice without specific explanations. These properties include gravity, friction, anchor, and push. This system, though running on a good graphics machine, (i.e. an Iris 4D/VGX equipped with a head-mounted display), has nevertheless some drawbacks: slow update rate and lack of precision when positioning objects, and low display resolution. It is suitable for making approximate arrangements of pieces of furniture in a room, such as putting objects on top of each other (e.g. put an object on the table) and moving them together.

Also Shinya [3] proposes a world containing physical constraints to automatically calculate object layouts. For example, the user drops an object on another and the falling object reaches the equilibrium point after a series of intermediate positions; or the user pushes an object which slides onto a surface below; or the user rotates an object around some constrained points (e.g. an edge). In this system the user doesn't guide the object's motion interactively but specifies the initial conditions on a 2D window and the system automatically calculates all the intermediate positions until it reaches the equilibrium. On average a calculation may take around 0.4 sec/frame on an IRIS 4D/310 VGX, which is not fast enough to build an interactive system requiring at least 10 frames/sec to guarantee a real time feedback.

In addition, although a world governed by physical laws is suitable for carrying out very general actions, it is not good enough to carry out a specific work. To perform different jobs

different environments are needed each providing the right support for a set of specific operations.

We propose a virtual space for space planning tasks which is modeled on the *building-yard* metaphor: it is a space where the user can move the objects and anchor them anywhere as if there were a supporting framework. Objects follow very simple rules: they do not occupy a space which is occupied by another object and an object interfering with another withdraws until it reaches a free zone (*no penetration*); objects at rest are subject to a *force field* which lines their local axes up to the world axes (not necessarily the homologous axes); objects can be stuck together to form a group (*sticking*) which is moved as a whole by the user. These simple rules are enough to ensure directional alignment of objects in space, relative alignment of objects with each other, and to draw one object close to another.

To always guarantee the real-time response of the system objects have a double representation: the geometric description is used by the rendering process to visualize their surface using a standard shading algorithm (e.g. Phong); a simplified representation based on cell decomposition is used by the interference control process when needed.

To improve the feeling of spatial immersion all user interactions depend on the spaceball: we have avoided mixing 2D and higher degrees of freedom techniques such as putting menus or 2D widgets into a virtual environment. Using the spaceball users can either push an object in any direction or move the point of view.

This approach guarantees a real time response even on a moderate graphics performance workstation such as a Silicon Indigo. On the one hand visualized objects are shaded with an acceptable quality, on the other hand the user doesn't notice any delay when the system controls object interference.

2. Human-computer dialogue

Like any other 3D interactive application [4] the very general objectives of this system are *object positioning* and *navigation*, but, because this system aims at being a working environment for a particular job, we have mapped these general concepts to this domain.

We consider two real world metaphors: the *hand* and the *eye on the transparent cap* (Table 1).

Table 1. Description of object positioning and navigation objectives.

| Objectives | Metaphor | Actions |
|---------------------------|--------------------------------|---|
| Object positioning | Hand | grab push release attach detach |
| Navigation | Eye_on_the_transparent _cap | move the eye on the cap |
| | | |

The metaphor approach is a top-down method which starts from the synthetic description of a behaviour in the real world and arrives at the definition of the elementary actions in the virtual world. The *hand* metaphor means that users move a virtual hand in the computer space like a real hand and make actions similar to those they do in the real world. Similarly, navigation is explained through the *eye on the transparent cap* metaphor which means that observers move their eyes on the surface of a sphere that surrounds the virtual world and watch below from a certain distance.

Metaphors are very useful [5] for at least two reasons: first of all they provide programmers with a conceptual framework which helps them to design a consistent, complete and simple human-computer dialogue; secondly users can easily understand how to use the system when it is described in terms of real world actions.

It should be noted that it is very difficult to implement a metaphor as an exact copy of real actions and that the way it is mapped to the virtual world depends on a number of causes. When a real world metaphor is applied to a target domain it is necessary to make some schematizations and simplifications on the real actions: in our case we do not need to reproduce all the capabilities of a real hand, only the ones needed to reach our objectives. There are also performance constraints to consider: in a virtual system time consuming controls which limit real time are not acceptable. Another limitation is due to the physical devices used for the interaction. We are aware that traditional 2D applications using the mouse cannot reproduce

immersive environments. But also sophisticated VR applications which use devices with many degrees of freedom (e.g. the 6 DOFs Polhemus 3Space or the 5DOFs Desktop Bat) have constraints. Some are due to the intrinsic limitations of virtual devices with respect to human senses, others derive from the erroneous choice of the device with respect to the task. This point is widely discussed by Ware [5] and Steed [4] who have made various experiments on users accomplishing particular tasks with different devices.

For these reasons the two main metaphors we have implemented very much depend on the physical device used: the spaceball.

2.1. The spaceball

We used a 6 DOFs device, a spaceball (Figure 1), equipped with 8 Function Buttons and a Pick Button. We built a pure 3D interface based exclusively on the spaceball, without keys, menus, buttons nor the widgets commonly used in 2D interfaces. However spaceball buttons are necessary for switching from one status to another during the interaction.

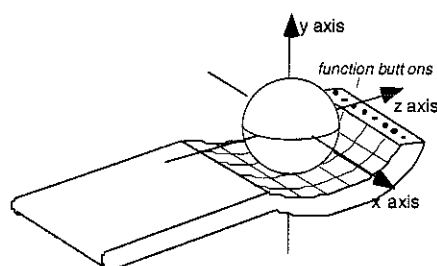


Figure 1. The spaceball

Users move the sphere in any direction and rotate it, the speed depends on the pressure of their hand.

To use the spaceball correctly, some precautions must be taken: the sphere must not be grabbed continuously with the palm of the hand because it is very sensitive to pressure from any direction and it easily generates unwanted roto-translations. To move an object the fingers' tips have to exert a brief pressure onto the sphere; and rotating an object requires a brief torsion to the sphere.

The positioning metaphor has been widely influenced by these constraints: to move an object from one position to another we say we *push* it instead of *carrying* or *dragging* it. Carrying is a natural action in reality, but would require a data glove to implement it with some degree of realism. Instead, pushing is a very natural action for spaceball users once they've realized that continuous grabbing leads to uncontrollable displacements.

2.2. Interface states

The user interface is always in one of the following states (Figure 2): *hand*, *camera*, *object*, *group*, depending on which objects the spaceball is linked to. The user presses one of the spaceball buttons to change state.

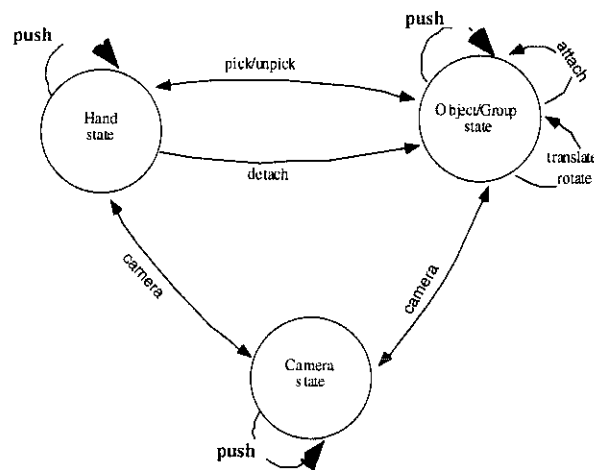


Figure 2. The state diagram of the interaction.

The kernel of the interface is an event manager which reads the event queue, recognizes permitted strings of events, and translates them into actions on the current object (Figure 3). The spaceball always transmits a sequence of events which includes these elements: tx, ty, tz, rotx, roty, rotz, period, and button. The first six elements are proportional to pressures along the orthogonal axes and to the angular momentum, respectively. It is common use to consider them as components of the velocity vector \mathbf{v} and of the angular velocity \mathbf{w} in the reference system of the spaceball. Because *period* represents the time elapsed since the last transmission, we can calculate the space (or the angle) covered by the object in the world coordinate system.

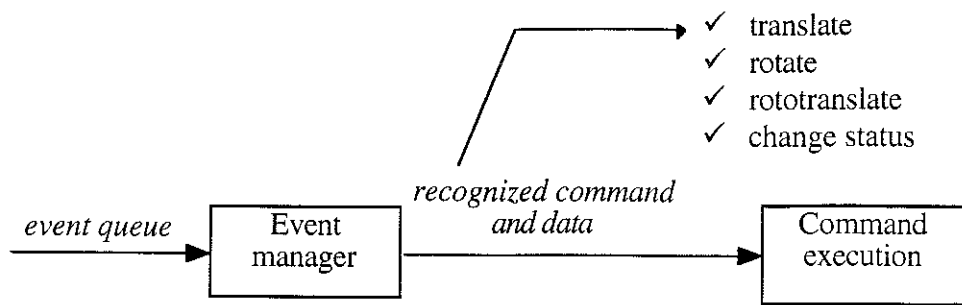


Figure 3. Interface components.

The event manager contains a programmable finite states automa which can be customized for new and different applications simply by defining other states and transition functions. With this mechanism the tool we are describing can be used to define new user-computer dialogues and test other physical devices.

The automa changes status according to its internal rules and at the end delivers a command with its data properly filtered: for example translation values are discarded when the rotation-only command is given.

2.3. The *virtual hand* model

The *virtual hand* model is a simplified version of the real hand metaphor. Users see their virtual hand in the virtual world as a *small cube* (Picture 1). They move the virtual hand freely in space and search for an object to grab. The virtual hand can always grab the object *at hand* that is the object which is closest to it and which is marked by a particular color (red in the picture). To grab the object the user must press the *grab button* of the spaceball.

The object may be glued to other objects and form a group with them: by pressing the *grab button* the whole group is grabbed, whereas the *detach button* separates the object from the rest of the group.

When an object is *in hand* it can be moved (rotated, translated) in space. Pressing the *translation/rotation* button switches from one mode to the other. Users can also decide to attach the object to the *object within easy reach* (or group) by pressing the *attach button*.

To create an effective positioning mechanism, we have separated rotations from translations. Free rototranslations in fact satisfy the visual inspection needs, not the positioning requirements: by separating these two tasks a higher degree of accuracy is obtained.

2.4. The eye on the transparent cap model

To navigate in a virtual world researchers have defined and tried out different metaphors depending on different navigation objectives. Ware and Osborne [5] provide an overview of cases and explain their experiments to measure the effectiveness of a number of metaphors with different tasks in a virtual world. They find that the *flying vehicle control* metaphor is the best for exploring virtual worlds. Unfortunately, this method doesn't offer satisfactory results with stereo on a workstation screen: objects exit from the field of view when they are still in front of the observer, an obvious consequence of the fact that this stereo configuration is not really immersive. We therefore decided not to allow scenes to be crossed but to explore the scene from outside by means of a constrained flight over it (*eye on the transparent cap* model).

The spherical cap is a transparent cap that surrounds the world: the eye moves on this surface looking at the scene below. The sphere is centered in the center of the world space and its radius is the distance of the observer from the screen. The point of view is linked to the spaceball, and when the user moves it the rotations around the X_C and Y_C axes are taken as rotations on parallels and meridians. We decided to inhibit a small zone around the north pole so that users cannot cross it, and to limit motion downward as far as the tropic.

2.5. User's bearings

Since users don't get any tactile feedback from the spaceball, they may quickly lose awareness of what action they are carrying out and which object they are handling. Although they can quickly verify this by simply moving the spaceball and looking at what happens in front of them, they feel more comfortable if they can orient themselves immediately when looking at the

world. We decided to code this information in a *status color* applied to a convenient object as described in this table:

Table 2: Visual feedback.

| status | status color applied to |
|---------------------|--|
| hand | the virtual hand |
| object/group | the current object/(object in a group) |
| camera | the floor |

3. Stereo view

The technologies to generate stereo views on the screen of a workstation are nowadays low cost technologies which can be used for a wide range of applications. There are two possibilities [6]: either a LCSS (Liquid Crystal Stereoscopic Shutter) with a pair of passive glasses, polarized and light, or a LED (Light Emitting Diode) which is a synchronization unit coupled with LCSS active glasses. We used the latter on a Silicon Indigo workstation.

To create a stereo visualization we use the *stereo pairs* method which consists in creating two images of the scene viewed from two slightly different points of view, corresponding to the two eyes of a virtual observer. The two images are rendered separately and projected onto the screen during the refresh cycle which is doubled (120 Hz) with respect to the normal (60Hz) on a Silicon Indigo.

The depth cue is determined by the horizontal parallax which is the distance between homologous points of a stereo pair, measured on the screen (Figure 4). Conventionally, objects behind the stereo plane have a positive parallax, objects between the observer and the screen have a negative parallax, and objects on the screen have zero parallax and give a small stereoscopic effect.

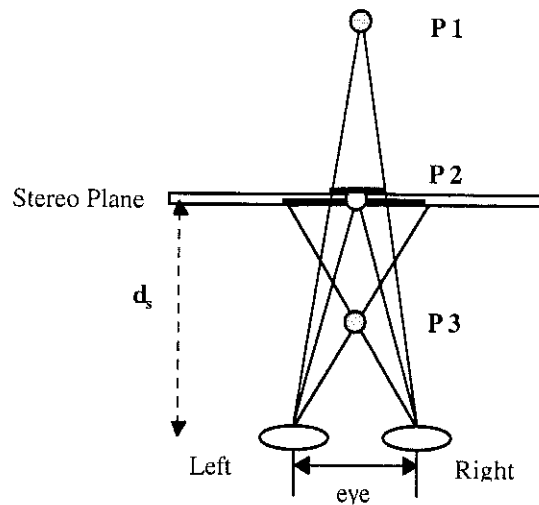


Figure 4. The stereo plane and the horizontal parallax. The stereo plane coincides with the screen. Point P1 has a positive parallax, point P2 has a negative parallax, and point P3 has zero parallax.

To produce an effective stereogram programmers have to assign correct values to the parameters of the stereo pairs, otherwise the result is wrong or unrealistic. The number of elements which contribute to create a good stereogram is high and includes both psychological parameters (e.g. image size, perspective, illumination) and physiological parameters (e.g. accommodation, convergence, retinal disparity, stereopsis). It is easy to build stereo pairs where all these parameters have contradictory values: in such case the human mind is forced to give preference to one parameter rather than another, and to give an arbitrary interpretation of the image.

3.1. The stereo pair

The calculation of a correct stereo pair depends on the geometric model adopted and the geometric constraints which are applied by the programmer to the model. We tested two geometric models, the *crossed axis projection* and the *parallel axis projection* [6] [7]. We adopted the last one (Figure 5) because it introduces less artefacts. There is only one problem which is image distortion at the borders (see the zones with no superimposed images in Figure 5) which is not relevant for the application we are discussing here.

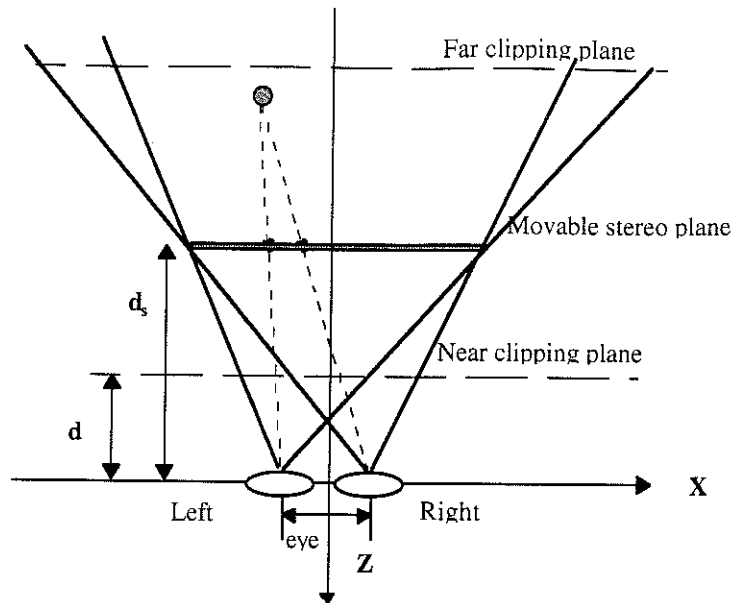


Figure 5. Parallel axes projection (or two centers of projection). The optical axes are parallel. The eyes have an interocular distance *eye*.

The parallel axes projection method is also called *two centers of projection* because it consists in taking two centers of projections at the distance *interocular distance* and making two projections with the same field of view on the same plane. This calculation can be implemented in a number of ways, some of which are illustrated by Hodges and Southard [7]. We adopted the approach which best fits the GL (Silicon Graphic Library) capability: to make a perspective projection, GL positions the camera in the origin looking at the z direction and projects the objects to the near plane. Therefore the origin is assumed as the center of projection. But we do not want the near plane to coincide with the stereo plane because we want to see objects on both sides of the stereo plane: to do this we move the stereo plane to a position between the near and the far clipping planes.

For each stereo pair we execute these calculations:

1. Translate the world and the window to the left along the x axis of a quantity $\frac{eye}{2}$ to build the righthand image and to the right to build the lefthand image.

2. Change the position of the stereo plane by changing the separation of the two views, that is scaling the parallax: decreasing the separation between the views moves the stereo plane backward and vice versa.

We define a scale factor μ for the parallax corresponding to a stereo plane at a position $d_s = \frac{d}{\mu}$. Instead of applying the scale factor to the coordinates of the two images (these values are not accessible because are handled directly by GL), we obtain the same result by adjusting the translation of the window at point 1 above.

To configure the stereo model, first of all we decide the distance d_s of the observer from the stereo plane (i.e. the screen) and then we fix the *near* plane at a distance $d = \frac{d_s}{2}$ from the observer. In fact the observer cannot see objects closer than half the distance to the stereo plane: the parallax, which is negative in this case, would be greater than the *interocular distance* and would produce an unnatural view. Instead the farthest object can be, in principle, at infinity, where the parallax assumes the maximum positive value, equal to the *interocular distance*. But if we consider the accommodation/convergence [6] conflict we see there is a constraint on the maximum parallax and, hence, on the depth of the stereogram.

The accommodation/convergence conflict affects the eyes when looking at virtual scenes. This inconvenience doesn't exist when looking at a real scene. When we look at a real object our eyes focus on the object (i.e. accommodate themselves by changing the lens) and, at the same time, converge on the object (i.e. rotate toward each other). This double process is called fusion and is controlled by two different muscles which work together. When we look at a virtual scene, we accommodate our eyes on the screen but converge to another point according to the parallax. Then these two muscles have to work separately and this causes discomfort.

To reduce this problem the convergence angle should not vary by more than a *delta* value [6] when moving the eyes from the closest object to the farthest. Using the value 1.6 deg found experimentally by Valyus on a sample of people, we calculated the stereogram depths corresponding to different observer distances from the screen:

Table 3. The stereogram depth depends on the observer distance.

| | | | |
|--------------------------|-------|--------|----------|
| distance from the screen | 50 cm | 100 cm | 230 cm |
| stereogram depth | 24 cm | 71 cm | infinite |

We see that an observer must have a distance of 2.30 m from the screen to look at distant objects. This distance makes sense with a large screen but is too far for a small workstation screen where the dimensions of objects must be compatible with the horizontal dimension of the screen to be visible. Therefore in a reasonable configuration the user sits at a distance of 50/100 cm from the screen. We have also noted experimentally that the Valius angle doesn't have an absolute meaning but is just indicative: the view is comfortable even if the stereogram depth is greater than the one calculated with this formula.

Nevertheless to prevent any discomfort caused by the accommodation/convergence conflict, we decided to concentrate the working area near the screen. When the working session starts, users can see the objects concentrated near the screen so that their eyes get used to the stereo environment; then they can move the objects backward and forward from the screen. A comfortable configuration has these dimensions:

d (distance of the observer from the screen): 50 cm

e (interocular distance): 5.5 cm

near (distance of the near plane from the screen): +25 cm

far (distance of the far plane from the screen): 17 cm

Objects near the screen have a small parallax and a reduced stereoscopic cue but it has been proved [8] that other parameters affect stereoscopy as well. One such parameter is perspective: by increasing the perspective, i.e. using a wide angle, the stereoscopic cue increases as well.

3.2. Comments

To create a good stereogram we considered other aspects as well:

- The sharpness of the stereo view is affected by the colors of the objects: too clear and too bright colors have a negative effect on the fusion of the two views and the two views

appear to be slight split. Dark and full colors, on the contrary, and a low illumination, favour views fusion and focus.

- Given a certain distance between the observer and the screen, *large size* objects are perceived more sharply than *small size* objects.
- Given a certain distance between the observer and the screen, the stereo effect increases if we increase the perspective.

The above experimental observations are supported by studies in MacAdam [9] which explain the existence of a synergetic relationship between 1) the familiarity with the object, 2) the perspective, and 3) the retinal disparity. The retinal disparity and the perspective have no absolute meaning but are interpreted by the human mind in relation to each other and also with respect to the familiarity the user has with the object: we expect to see an object of a certain size at a certain distance which is consistent with our real experience.

4. World behaviour

The virtual environment responds to user actions according to its properties and rules. The system's response is necessary to complete a task.

We have defined a virtual world with these characteristics:

1. Each object has a *spatial enumeration* representation [12] made of cubic cells with the same size as the virtual space cells. An object is decomposed into the smallest number of cells which include its surface. If the cells of an object penetrate the space occupied by another object, the object itself is pushed back along the motion direction of the quantity needed to exit from the occupied cells.
2. The virtual space has a *spatial-partitioning* representation made of equal size cubic cells. The system maintains an *octree* structure which describes the cell occupancy by objects [10] [11]. The octree is not updated continuously while the user moves the object but only when s/he leaves the spaceball. The virtual space applies a *force field* to the object *at rest* and attracts it to a discrete position.

The application of these properties is described in detail in the following sections. These rules are enough to ensure the following functionalities:

- *directional alignment*: two or more objects can be aligned in space along a direction. The alignment along the orthogonal world axes is guaranteed while the accuracy of diagonal alignments depends on the resolution of the space decomposition.
- *relative alignment*: an object can be positioned close to another object at one side or another. If the resolution is high and the cell decomposition representation doesn't differ too much from the object shape, there can be a face to face contact between two objects. Alternatively two objects may come close without penetration and minimum visible empty space in between but without geometric precision.

4.1. Performance

In a realistic world objects do not interfere with each other, that is different objects do not occupy the same region of space. This problem is part of the collision detection problem which is essential to apply a behaviour to objects based on physical laws. Since detecting the geometric interference between moving objects is time consuming most research has been done in the field of animation.

In virtual reality, motion is not generated by the object itself but is guided interactively by the user and is completely undetermined a priori. Algorithms that give a real time response to user interaction are needed. Our strategy is applicable to a moderate performance workstation, and applies some simplifications to the factors which are most time consuming: time step, number of tests, and geometric accuracy.

4.1.1. Time step

In animation the objects move and may interfere with each other at any one time. It is thus important to foresee collisions: the time step between successive controls is kept variable and heavier calculations only start when the object is in the collision zone. This *adaptive timestep* approach [13] consists in applying a variable timestep to collision detection, inversely proportional to collision probability.

We use a similar philosophy in our system to reduce the control frequency. The probability of collision when the user pushes an object is very low due to the way the user has to touch the spaceball: short pressures. Inevitably users release the spaceball continuously, especially when the object moved is close to another object. Thus we only make a collision control when the user has left the spaceball, i.e. only when needed (OWN). Our experiments demonstrate that this strategy is totally reasonable and that this behaviour is really natural and instinctive to a user.

4.1.2. Number of tests

In animation where all objects move, the number of tests is high because all the objects must be checked against all the others, and each face of an object must be checked against the faces of all the other objects. To reduce the number of geometric calculations, detection algorithms provide approximate methods thus leaving geometric calculations to the minimum number of probable cases.

In our interactive system where objects are not animated and only one move under direct user control the number of controls is drastically reduced: we can control only the current object against the others. To reduce this time even more and thus avoid the *all-pairs weakness* described by Hubbard [13] we use an *occupancy detection test* based on space and object subdivision. This test is independent of the number of objects, but depends exclusively on the number of cells of the object.

4.1.3. Accuracy

Another element is the geometric accuracy required by interference detection. Realistic animations aim to calculate the precise contact points to apply correct response transformations. Our system aims to position the cells of an object with respect to the cells of the world. With this approximation we avoid heavy geometric calculations which could not be handled in real time by our stereo system. In fact only very poor geometric calculations guarantee a real time feedback on a low performance workstation. Otherwise we soon reach a frame rate which is insufficient, such as the ones reported by [14], of 5 frames/sec on a Silicon 4D/240 GTX (see also [15] and [13]).

4.2. Algorithms

When the user releases an object, the system checks and corrects its position before visualizing it. This computation is a two step loop on the cells of the object: the object is repeatedly withdrawn as long as one of its cells occupies a cell taken by another object. When a good position is found, the world is visualized and the world octree is updated. Each step includes a control (C_1 and C_2 respectively) and a response (R_1 and R_2 respectively):

step 1: discrete test

C_1 - check if the object cells completely fill the world cells;

Condition C_1 is hardly ever verified because the user is free to move the object everywhere and the stopping position is unpredictable.

R_1 - find the closest discrete position and determine the corresponding modeling transformation M_d in the *discrete* world, which includes either a translation or a rotation, depending on the case.

To adjust the position after a translation, the algorithm looks for the grid node closest to the baricenter of the object and calculates the correct translation vector. In the case of rotation, if the relative rotation from the beginning of the motion is not a multiple of 90 deg, the system calculates the smallest angle, either positive or negative, to a discrete position around the rotation axis. The drawbacks derived from arbitrary rotations of cell decomposition structures are described in many papers and include approximate representations, the loss of the original shape after multiple rotations, and the cumulative distortion of inverse rotations [16] [17].

step 2: occupancy test

C_1 - the cells of the object are checked against the octree to find out if they would occupy cells already taken by another object;

R_2 - if so, the object is withdrawn by a small step from its initial position and the calculation starts again from point C_1 .

step 3: visualization

Once the alignment to the discrete space has been found, the system checks for cell occupancy before visualization. If penetration occurs into another object, the system withdraws the current object by a small step along the motion direction (e.g. half the cell size) and goes back to point C_1 .

After steps 1 and 2 have been satisfactorily completed, that is the object has reached a good position, the system runs the visualization algorithm and updates the octree structure with the new occupancy situation.

5. Tests

Picture 1 shows the application SomaKit which allows a user to assemble the pieces of Soma in various ways. The pieces of Soma are obtained by dividing into 7 non equal parts a cube formed by 27 small cubes, that is a $3 \times 3 \times 3$ cube. The play consists in searching for the maximum number of shapes which can be obtained by assembling all 7 pieces. To run this application, a user can choose a cell dimension for the virtual world which is the dimension of the Soma cube unit, and calibrate the resolution of the world to the performance of the workstation. This system has an immediate application to position and align objects whose faces are parallel to the faces of the cells.

Architectural applications where elements have sizes that are multiples of a common unit, are also good candidates for this system. Picture 2 shows an example of a simple temple composed of elementary blocks.

6. Conclusions and future developments

Manipulating and assembling elements in a 3D space is a task which interests a huge number of potential applications whether they deal with real or abstract objects. We have presented and tested our prototype with professionals from the field of cultural heritage preservation and theatrical/television production, as they have strict and innumerable requirements for space planning activities. People in the cultural heritage preservation sector often need to study

ancient ruins and present hypothetical reconstruction models designed quickly and approximately. On the other hand theatrical/television directors need to make approximate staging plans during the pre-production phase of a performance [18]. In particular, they need to design sets for planning real constructions. Anyone who has tried to manipulate 3D objects on a flat screen with a 2D input device (a mouse) knows how hard it is and how inappropriate 2D windows are to perform spatial tasks. For this reason, the users we contacted have shown great interest in our approach and the prospect of having operations in an affordable virtual reality environment.

The problem of developing natural interfaces for non-immersive working environment is still an open issue which is subject to research and experiments by computer scientists and psychologists. The simplicity of our interaction proposal relies on the fact that the user's entire workload is reduced to spatial manipulation and the user is not confused by a hybrid interface which allows both immersive and logical operations with the same input device. However we are aware that a realistic working world has a higher number of requirements which cannot all be satisfied by the spaceball functional buttons. The development of widgets that allow intuitive and easy interactions in a virtual world is still a subject of international research and our future work will investigate this area.

Again we have seen that when the main operation is direct manipulation of objects, it is natural to use both hands to bring two objects near to each other, hence two input devices would be a better solution.

Our prototype has a limitation with respect to its applicability to real cases due to the few objects it can handle in real time. This problem doesn't invalidate our approach as it depends only on the really moderate capacities of our workstation. It is obvious that real applications contain many objects but it is also evident that powerful low cost virtual platforms, at least for non photorealistic applications, will soon be available.

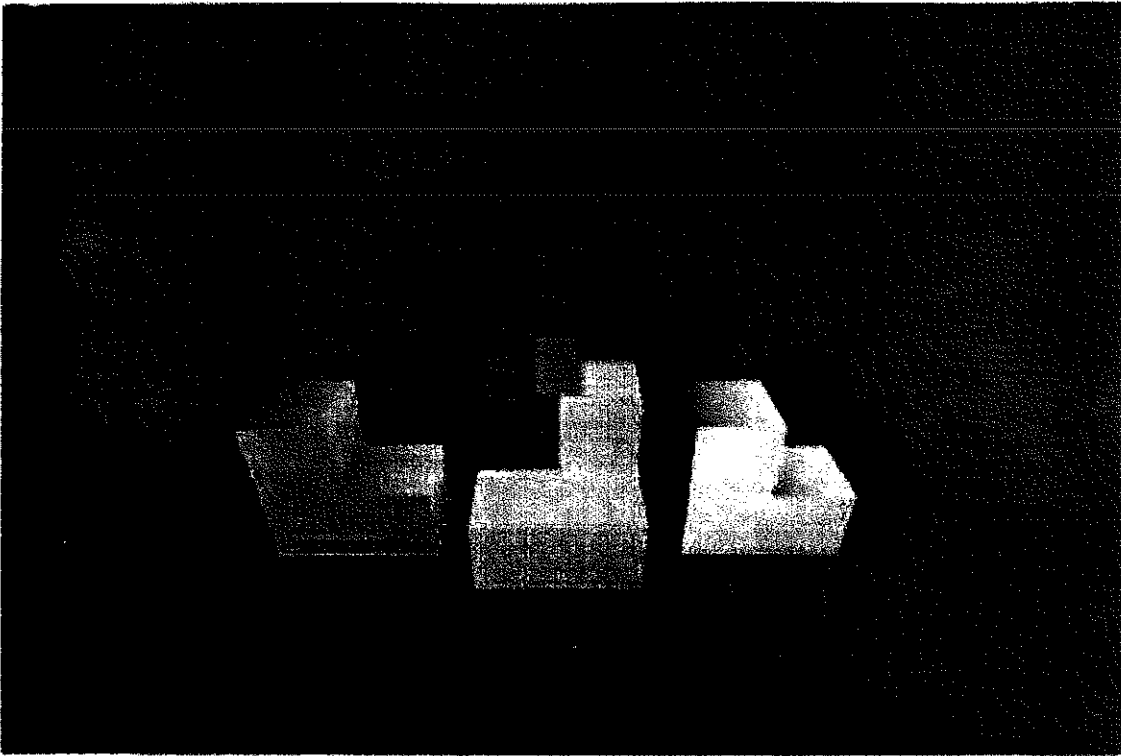
Since a stereo monitor screen is characterized by a narrow field of view, it is suitable for visualizing and rotating a single object; in fact it has been applied so far to a restricted number of applications the most well known of which include molecular modeling. The most recent

achievement in virtual working systems is the Responsive Workbench [19] whose main characteristic is the computer screen changed to a horizontal, enlarged worktop in place of the traditional two-dimensional vertical screen. This new environment has been tested so far principally for medical, chemical, and fluid dynamic visualization. Our practical investigations have confirmed that people consider the traditional vertical monitor screen as being insufficient to simulate their working worlds, though they have appreciated the simplicity of the proposed interaction and the reliability of the spaceball as control device. Our future work will include testing the concepts and algorithms described in this paper in a workbench environment and extending our functionalities for more realistic space planning applications.

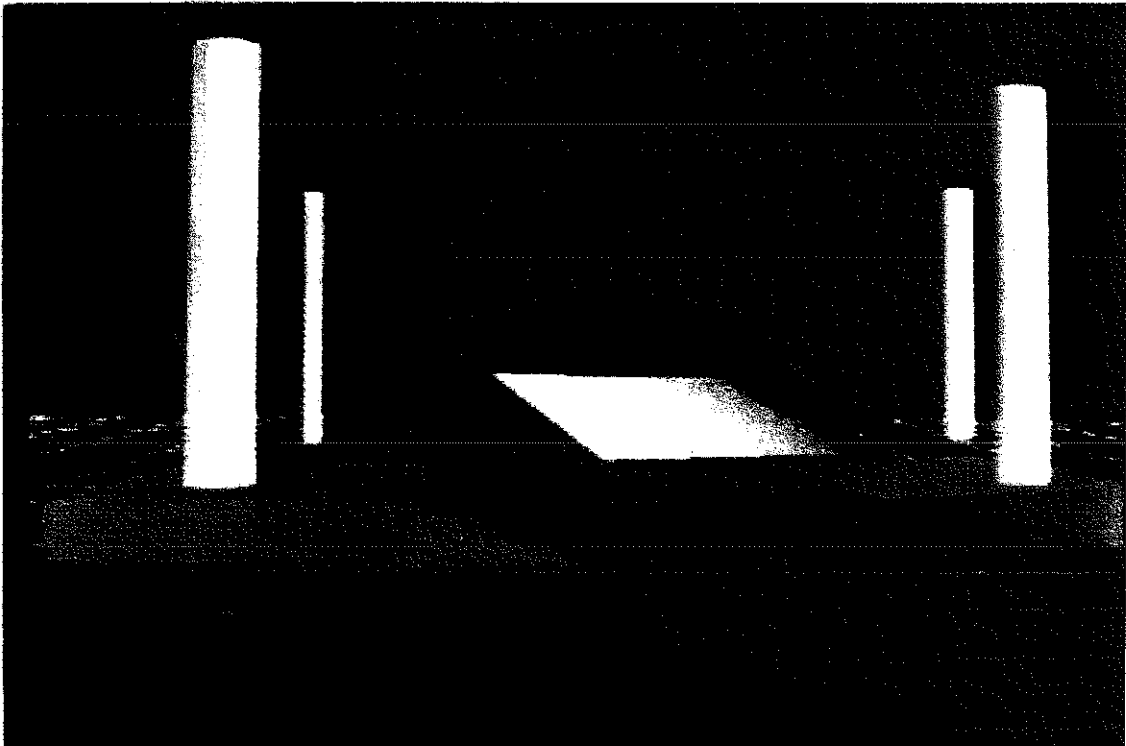
7. Bibliography

1. Bryson, S. (1993) Virtual Environments in Scientific Visualization. In: Virtual Reality for Visualization, IEEE, pp. 87-127.
2. Papper, M.J. and M.A. Gigante (1993) Using Physical Constraints in a Virtual Environment. In: Virtual Reality Systems, Academic Press, pp. 107-118.
3. Shinya, M. and M.V. Fogue (1995) Laying out objects with geometric and physical constraints. *The Visual Computer* (11), pp. 188-201.
4. Steed, A. and M. Slater (1995) 3D Interaction with Desktop Bat. *Computer graphics Forum* 14(2), pp. 97-104.
5. Ware, C. and S. Osborn (1990) Exploration and Virtual Camera Control in Virtual Three Dimensional Environments. In Symposium on Interactive 3D Graphics, Snowbird, Utah, March 25-28. ACM, pp. 175-184.
6. Southard, D.A. (1992) Transformations for stereoscopic visual simulation. *Computer & Graphics* 16(4), pp. 401-410.
7. Hodges, L.F. (1992) Tutorial: Time-Multiplexed Stereoscopic Computer graphics. *IEEE Computer Graphics & Applications* (March 1992), pp. 20-30.
8. Lipton, L. (1990) Some aspects of stereoscopic composition. In: Course Notes 6 Stereographics, SIGGRAPH 1990,
9. MacAdam, D.L. (1954) Stereoscopic perceptions of size, shape, distance and direction. *J. Soc. Motion Picture and Television Engineers* (62), pp. 271-293.
10. Foley, J.D., A.v. Dam, and al. (1990) *Computer Graphics - Principles and Practice*. Addison-Wesley, Reading MA.
11. Samet, H. and R.E. Webber (1988) Hierarchical Data Structures and Algorithms for Computer Graphics - Part II: Applications. *IEEE CG&A* (July), pp. 59-75.
12. Meagher, D. (1982) Geometric Modeling Using Octree Encoding. *Computer Graphics and Image Processing* 19, pp. 129-147.
13. Hubbard, P.H. (1993) Interactive Collision detection. In *IEEE Reserach Frontiers in Virtual Reality*, San Jose, CA. pp. 24-31.
14. Garcia-Alonso, A., N. Serrano, and J. Flaquer (1994) Solving the Collision Detection Problem. *IEEE Computer Graphics and Applications* (May), pp. 36-43.
15. Palmer, I.J. and R.L. Grimsdale (1995) Collision Detection for Animation using Sphere-Trees. *Computer Graphics Forum* 14(2), pp. 105-116.
16. Ahuja, N. and C. Nash (1987) Octrees Representations of Moving Objects. *Computer Vision, Graphics, and Image Processing* 26, pp. 207-216.
17. Weng, J. and N. Ahuja (1987) Octrees of Objects in Arbitrary Motion: Representation and Efficiency. *Computer Vision, Graphics, and Image Processing* 39, pp. 167-185.
18. Palamidese, P. M. Betro', and G. Muccioli (1993) The Virtual Restoration of the Visir Tomb. In Visualization '93, San Jose', California, October 25-29. IEEE Computer Society Press, pp. 420-424.
19. P. Palamidese. CATS: TV, Cinema, Theatre, Drama Schools Requirements. CNUCE Internal Report C96-25.
20. Krueger, W. and B. Froehlich, The Responsive Workbench (virtual work environment). *IEEE Computer Graphics and Applications*, 1994. 14(3): p. 12-15.

10/8/97



Picture 1. Assembling pieces of Soma (right and left image together).



Picture 2. Composing simple architectural elements (right and left image together).