pipeline 3D + traversal

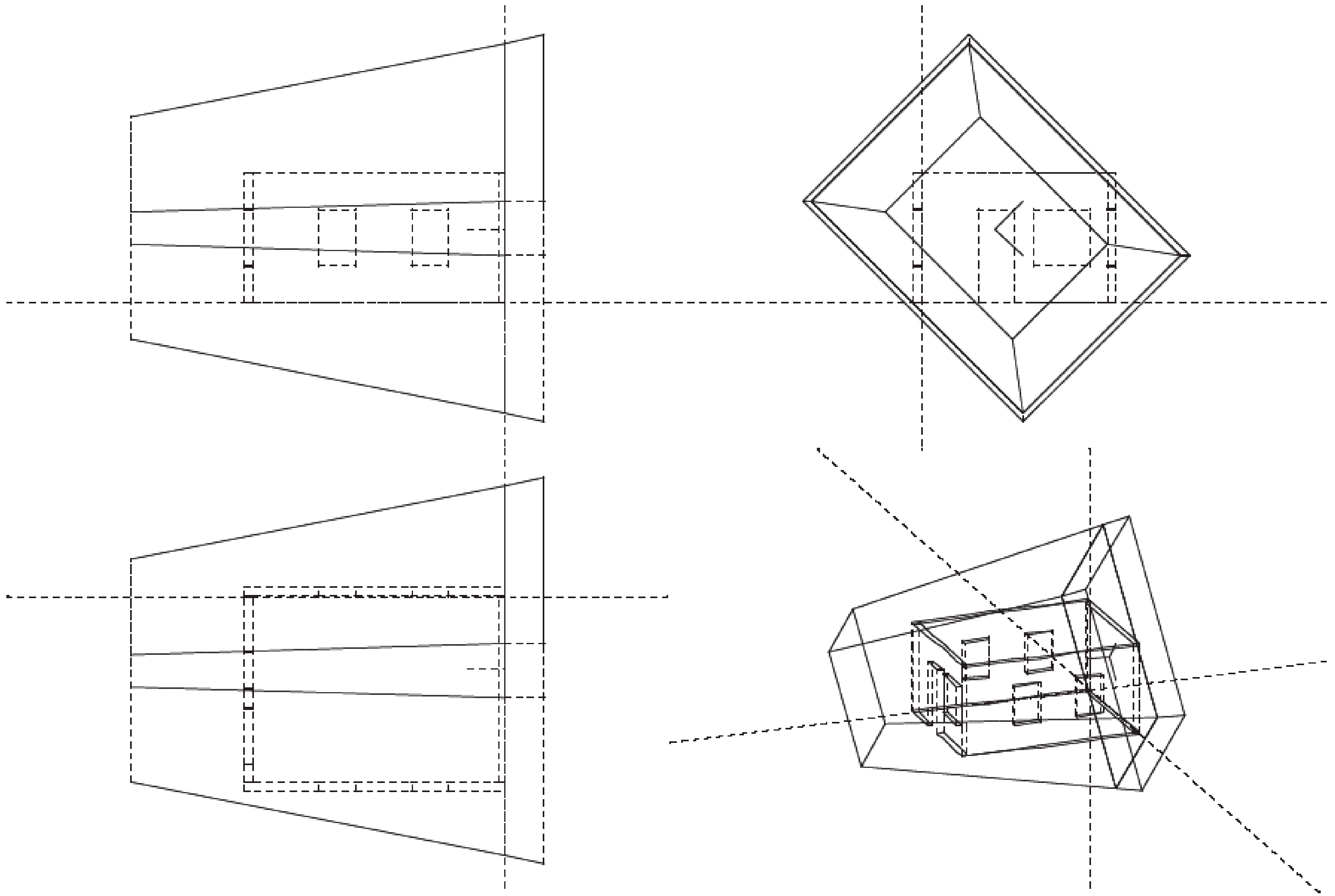

Figure 9.5 House model and perspective view volume in world coordinates

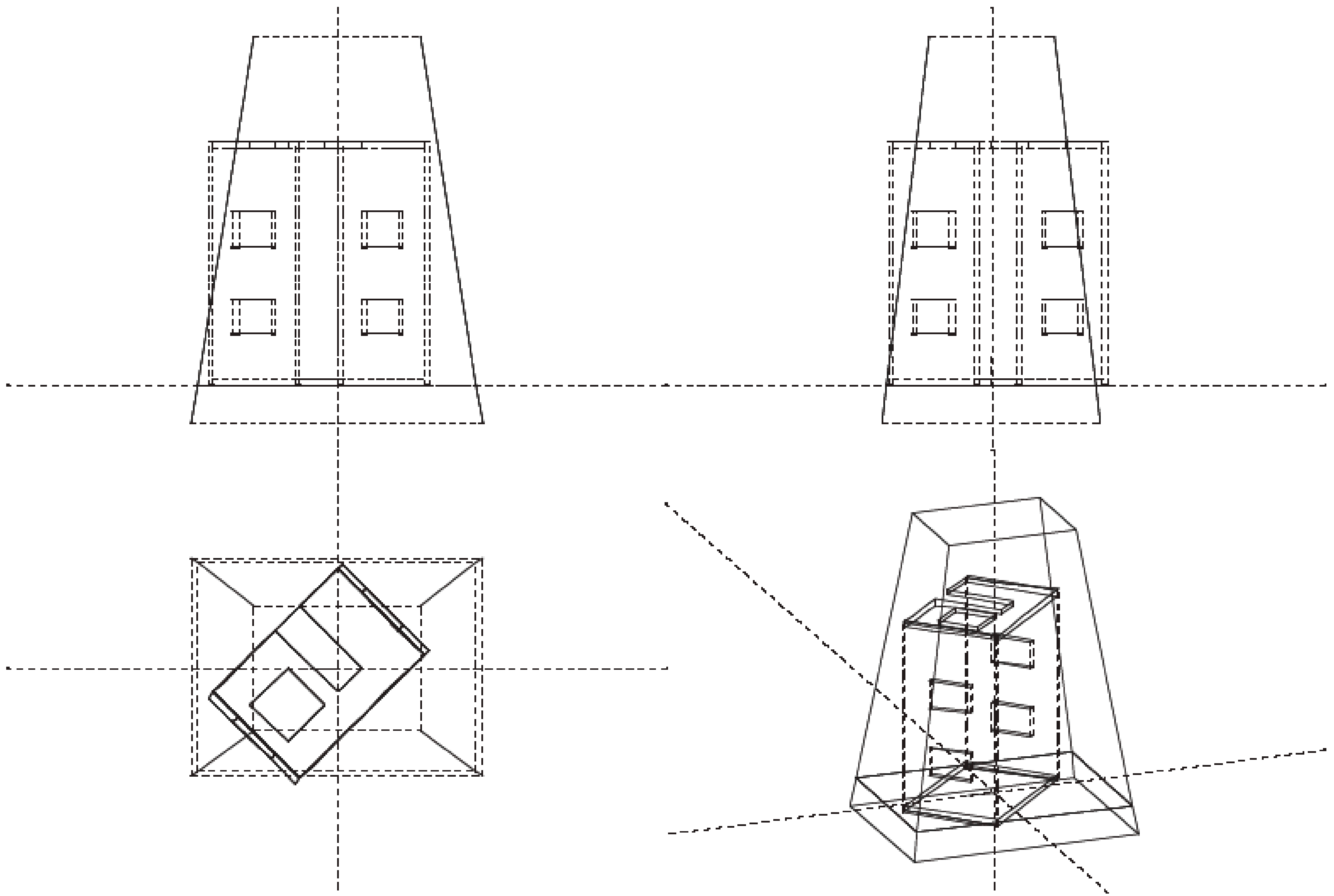

Figure 9.6 House model in VRC according to the view orientation transformation

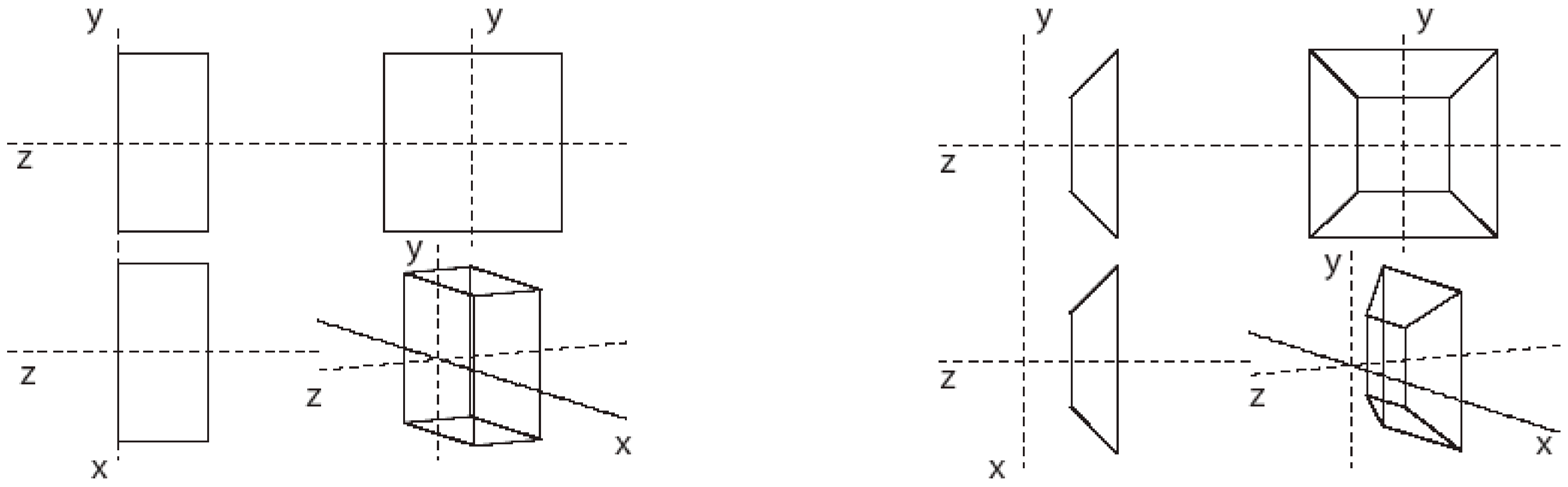

Figure 9.7 Canonical view volume: (a) parallel case (b) perspective case

Parallel case The view mapping tensor $\boldsymbol{VM}_{par}$ is the composition of a shearing, a scaling and a translation:

$$\boldsymbol{VM}_{par} = \boldsymbol{T}_{par} \circ \boldsymbol{S}_{par} \circ \boldsymbol{H}_z .$$

The shearing $\boldsymbol{H}_z$ must shear the Direction Of Projection (DOP) vector in $(0, 0, dop_z, 1)^T$, thus shearing the possibly oblique view volume into a straight one. The DOP vector in VRC is defined in PHIGS as difference between the Center of Window (CW) and the Projection Reference Point (PRP):

$$DOP = CW - PRP = \begin{pmatrix} (u_{max} + u_{min})/2 \\ (v_{max} + v_{min})/2 \\ 0 \\ 1 \end{pmatrix} - \begin{pmatrix} prp_u \\ prp_v \\ prp_n \\ 1 \end{pmatrix}$$

So, it must be

$$\begin{pmatrix} 0 \\ 0 \\ dop_z \\ 1 \end{pmatrix} = \boldsymbol{H}_z \begin{pmatrix} dop_x \\ dop_y \\ dop_z \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & sh_x & 0 \\ 0 & 1 & sh_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} dop_x \\ dop_y \\ dop_z \\ 1 \end{pmatrix},$$

and hence

$$sh_x = -\frac{dop_x}{dop_z}, \quad sh_y = -\frac{dop_y}{dop_z}.$$

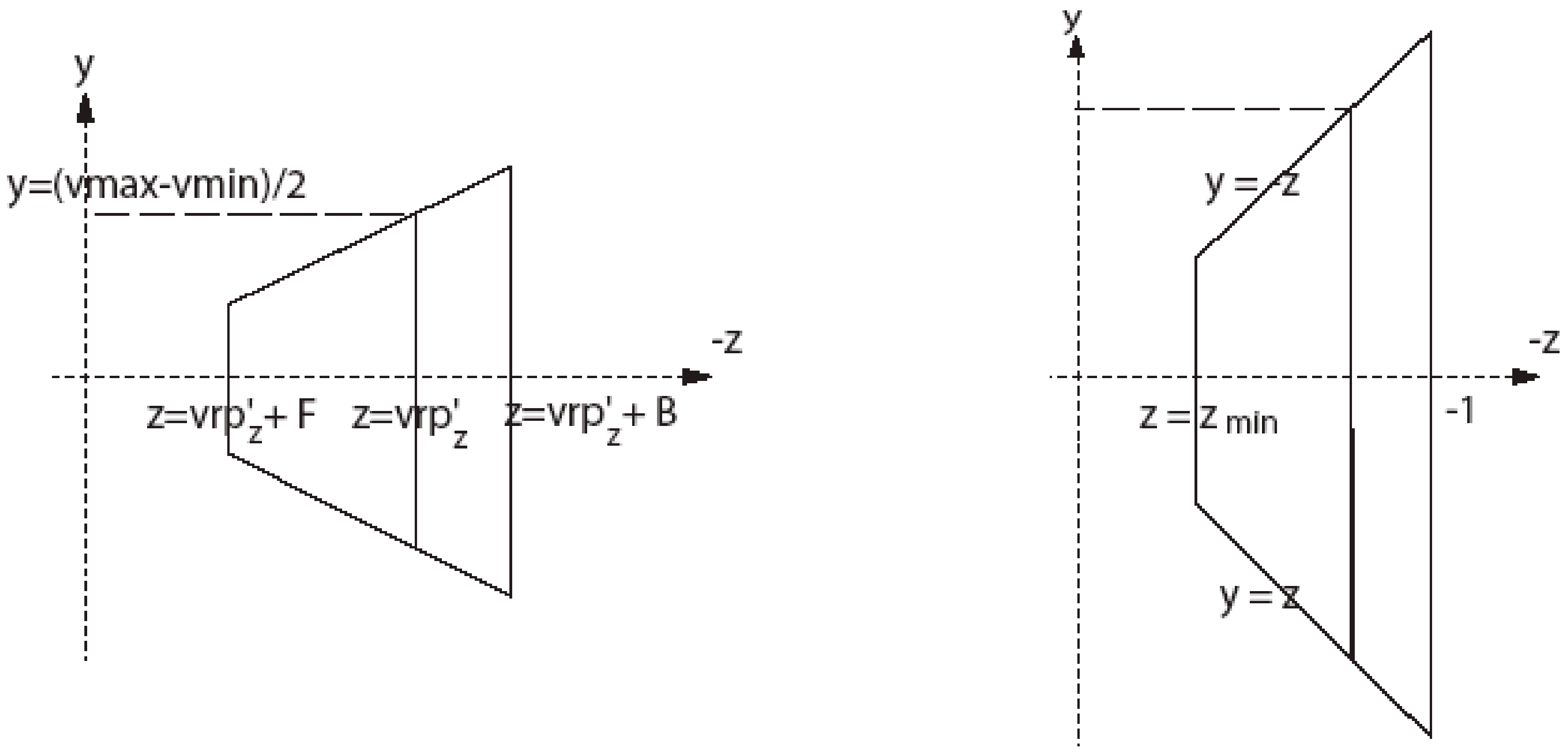

Figure 9.8 Scaling to canonical view volume of parallel case

After the action of such tensor, the bounds of view volume are

$$u_{min} \le x \le u_{max}, \quad v_{min} \le y \le v_{max}, \quad B \le z \le F$$

to be scaled and translated to the canonical volume

$$-1 \le x \le 1, \quad -1 \le y \le 1, \quad -1 \le z \le 0,$$

so that

$$\boldsymbol{T}_{par} = \boldsymbol{T}\left(-\frac{u_{min}+u_{max}}{2}, -\frac{v_{min}+v_{max}}{2}, -F\right),$$

$$\boldsymbol{S}_{par} = \boldsymbol{T}\left(\frac{2}{u_{max}-u_{min}}, \frac{2}{v_{max}-v_{min}}, \frac{1}{F-B}\right).$$

Perspective case The view mapping tensor $\boldsymbol{VM}_{per}$ is the composition of a translation of PRP, that coincides with COP in this case, to the origin, followed by a shearing to make straight the view pyramid, and by a composite scaling to map the result into the canonical volume:

$$\boldsymbol{VM}_{per} = \boldsymbol{S}_{per} \circ \boldsymbol{H}_z \circ \boldsymbol{T}(-PRP),$$

where:

1. $\boldsymbol{T}(-PRP)$ moves the center of projection to the origin;
2. the shearing $\boldsymbol{H}_z$ tensor coincides with the one of parallel case;
3. the scaling tensor can be decomposed as: $\boldsymbol{S}_{per} = \boldsymbol{S}_2 \circ \boldsymbol{S}_1$.

where $\boldsymbol{S}_1$ maps the straight view pyramid onto a unit slope pyramid:

$$\boldsymbol{S}_1 = \boldsymbol{S}(\frac{-2\ vrp'_z}{u_{max} - u_{min}}, \frac{-2\ vrp'_z}{v_{max} - v_{min}}, 1)$$

and where $\boldsymbol{S}_2$ uniformly scales the three-space to move the $z = B$ plane (the `Back plane`) to the $z = -1$ plane:

$$\boldsymbol{S}_2 = \boldsymbol{S}(\frac{-1}{vrp'_z + B}, \frac{-1}{vrp'_z + B}, \frac{-1}{vrp'_z + B}^T)$$

Notice that vrp'_z is obtained by mapping the VRC origin by the translation to PRP and by the subsequent shearing:

$$VRP' = (\boldsymbol{H}_z \circ \boldsymbol{T}(-PRP))(0, 0, 0, 1)^T$$

```
DEF ViewMapping = S_per ~ SH_per ~ T_per;
DEF T_per = T:<1,2,3>:(AA:-:prp);
DEF SH_per = MAT:
   << 1, 0, 0, 0 >,
    < 0, 1, 0, dopx / dopz >,
    < 0, 0, 1, dopy / dopz >,
    < 0, 0, 0, 1 >>
WHERE
   dopx = (umin + umax)/2 - s1:prp,
   dopy = (vmin + vmax)/2 - s2:prp,
   dopz = 0 - s3:prp
END;
DEF S_per = S:<1,2,3>:<sx,sy,sz>
WHERE
   sx = (2 * vrp_z)/((umax - umin)*(vrp_z + back)),
   sy = (2 * vrp_z)/((vmax - vmin)*(vrp_z + back)),
   sz = -1/(vrp_z + back)
END;


DEF umin = S1:window; DEF vmin = S2:window;
DEF umax = S3:window; DEF vmax = S4:window;
DEF vrp_z = (s3 ~ s1 ~ s1 ~ UKPOL ~ SH_per ~ T_per ~ MK): <0,0,0>;
```

9.2.6 Perspective transformation

The so-called *perspective transformation* [FvDFH90], mathematically an affine homology, maps the canonical pyramid volume of central projections onto the canonical parallelepiped volume of parallel projections.

Such a transformation moves the origin to the improper point of z axis and the front plane to the plane $z = 0$, while keeping invariant the back plane, of current equation $z = -1$. Such a perspective tensor is associated with a matrix

$$\boldsymbol{P} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{1+z_{min}} & \frac{-z_{min}}{1+z_{min}} \\ 0 & 0 & -1 & 0 \end{pmatrix}, \quad z_{min} \neq -1 \qquad (9.1)$$

Example 9.2.2 (Perspective transformation)
Let us consider the vertices

$$\boldsymbol{r} = \begin{pmatrix} -z_{min} \\ -z_{min} \\ z_{min} \\ 1 \end{pmatrix}^T \quad \text{and} \quad \boldsymbol{s} = \begin{pmatrix} 1 \\ 1 \\ -1 \\ 1 \end{pmatrix}^T$$

of canical view volume of Figure 9.7b, where $\boldsymbol{r}$ is at the intersection of planes $x = -z$, $y = -z$ and $z = z_{min}$, and $\boldsymbol{s}$ is at the intersection of planes $x = -z$, $y = -z$ and $z = -1$. They are respectively mapped to

$$\boldsymbol{P}(\boldsymbol{r}) = \begin{pmatrix} -z_{min} \\ -z_{min} \\ 0 \\ -z_{min} \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \end{pmatrix} \quad \text{and to} \quad \boldsymbol{P}(\boldsymbol{s}) = \begin{pmatrix} 1 \\ 1 \\ -1 \\ 1 \end{pmatrix}$$

Script 9.2.4

```
DEF perspTransf = (MAT ~ INV):<
  <      0,             0, 0,        -1>,
  <      0,             1, 0,         0>,
  <      0,             0, 1,         0>,
  <-:z_min/(1+z_min),  0, 0, 1/(1+z_min)> >
WHERE
  z_min = -:(vrp_z + front)/(vrp_z + back)
END;
```

The result of application of the `perspTransf` tensor to the model generated at the previous step is shown in Figure 9.10. The PLaSM expression which generates the model represented in such figure is

```
(perspTransf ~ ViewMapping ~ ViewOrientation): WCscene;
```

9.2.7 Workstation transformation

The workstation transformation maps a 3D workstation window in NPC onto a 3D workstation viewport in DC3. This mapping is similar to the 2D one defined by GKS and discussed in Section 9.1.2. It is composed of a translation that moves the NPC point of minimum coordinates to the origin, then by a scaling of the 3D extent to the size of the viewport and by a final translation of the origin to the DC3 viewport point of minimum coordinates. The device transformation is applied to the geometric data of primitives, including the control points of curves and surfaces (see Section 11.2). Such primitives are then rasterized in 3D, often using some variation of the z-buffer approximated algorithm for removing the hidden parts. Exact algorithms for hidden-surface removal would have already been applied in NPC coordinates.

So, if $W = [w_1, w_4] \times [w_2, w_5] \times [w_3, w_6] \subset NPC$, and $V = [v_1, v_4] \times [v_2, v_5] \times [v_3, v_6] \subset DC3$, then we have

$$\boldsymbol{T}_D : NPC \to DC3$$

such that

$$\boldsymbol{T}_D = \boldsymbol{T}(v_1, v_2, v_3) \circ \boldsymbol{S}(\frac{v_4 - v_1}{w_4 - w_1}, \frac{v_5 - v_2}{w_5 - w_2}, \frac{v_6 - v_3}{w_6 - w_3}) \circ \boldsymbol{T}(-w_1, -w_2, -w_3)$$

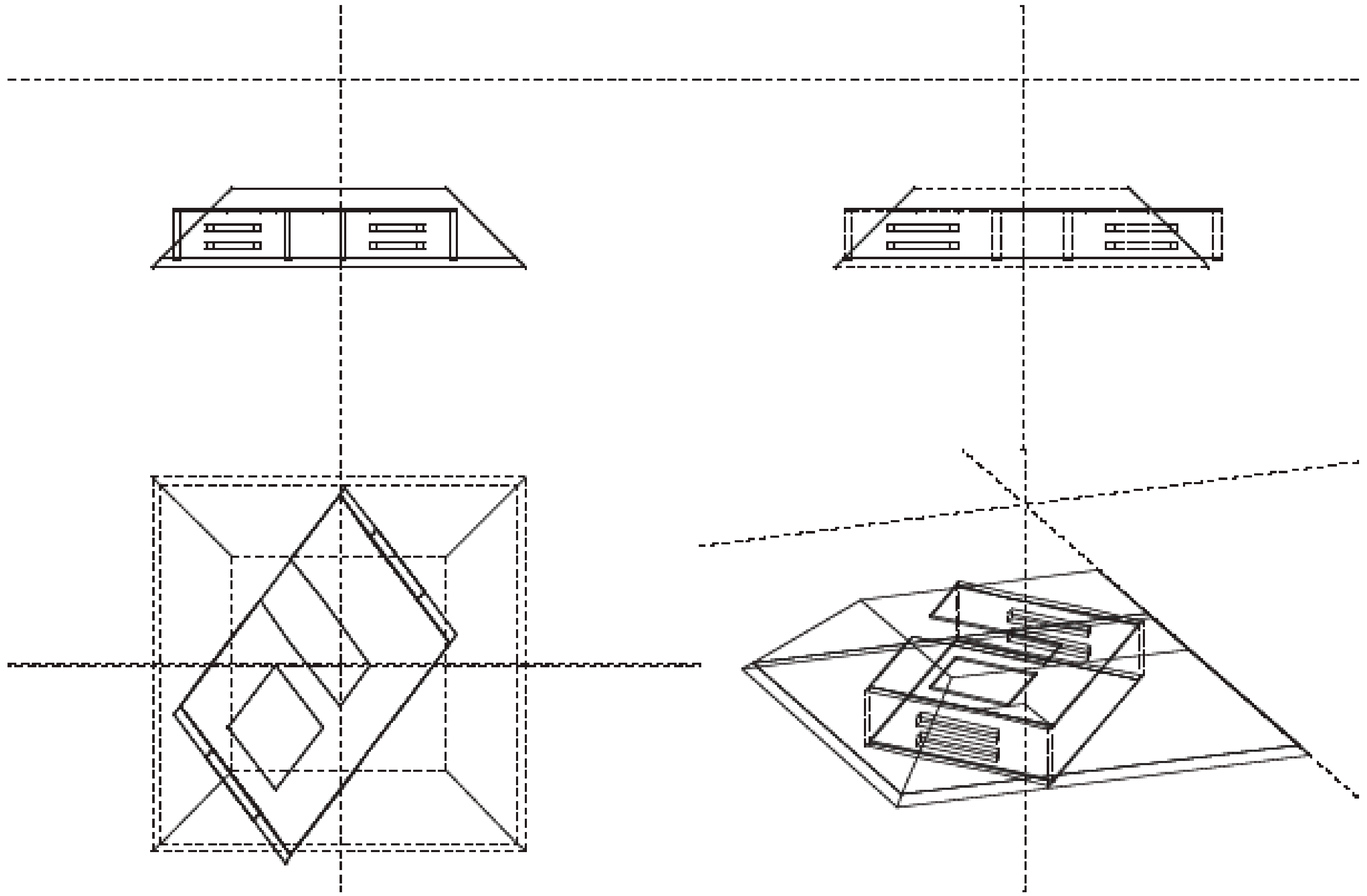

Figure 9.9 House model (in NPC) after the orientation and the view mapping transformation

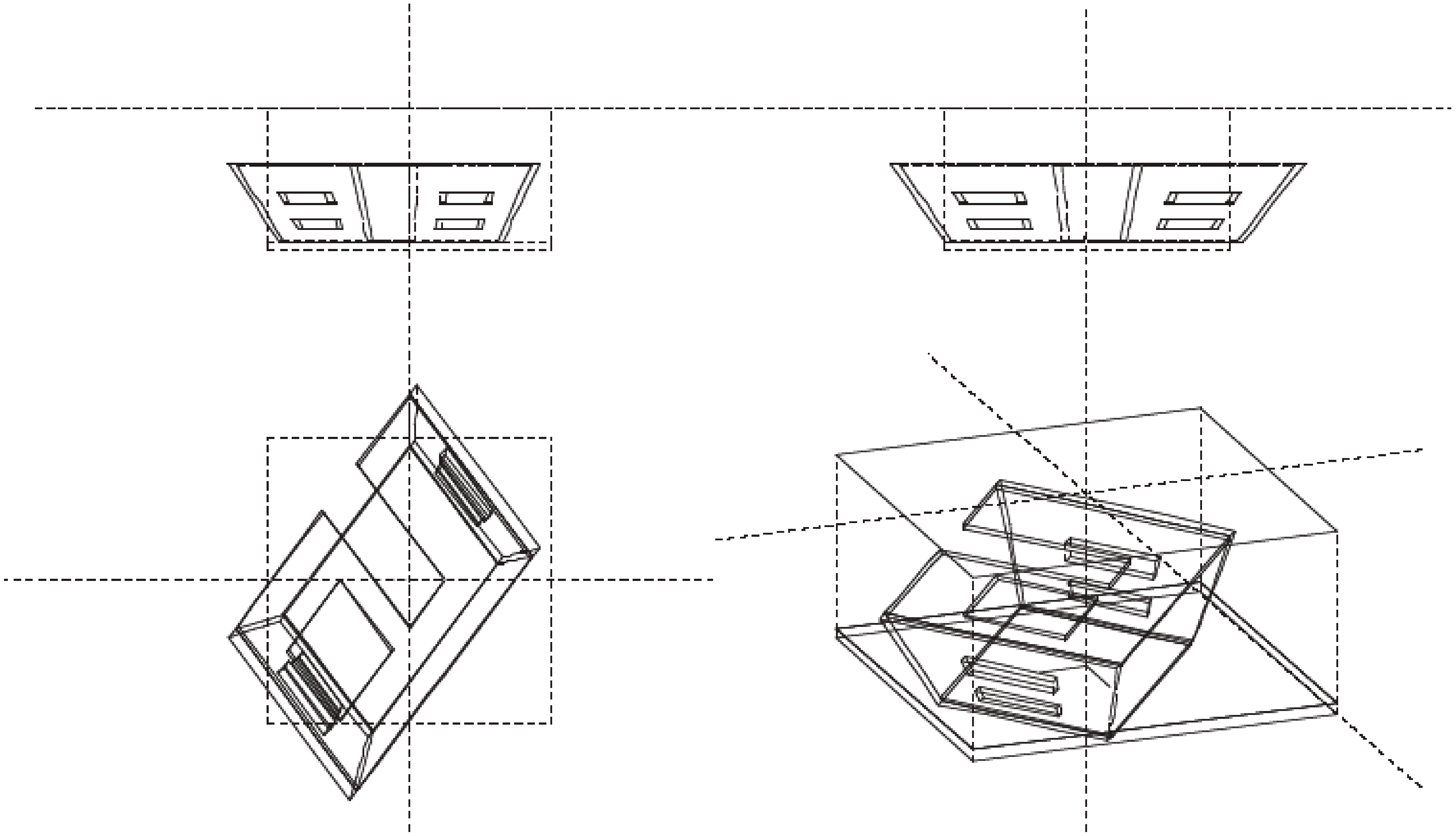

Figure 9.10 Canonical volume after perspective transformation

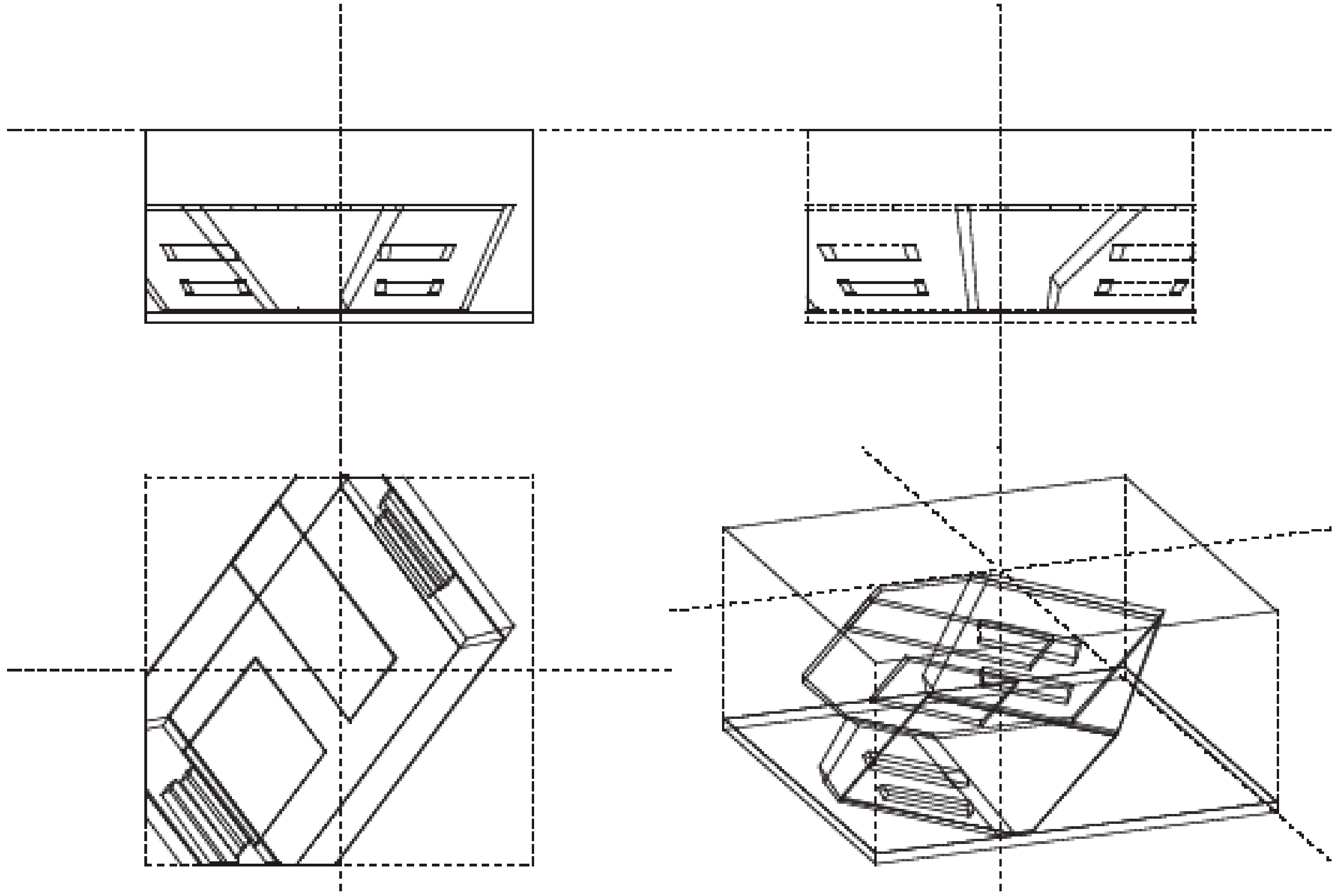

Figure 9.11 Canonical volume after perspective transformation and clipping

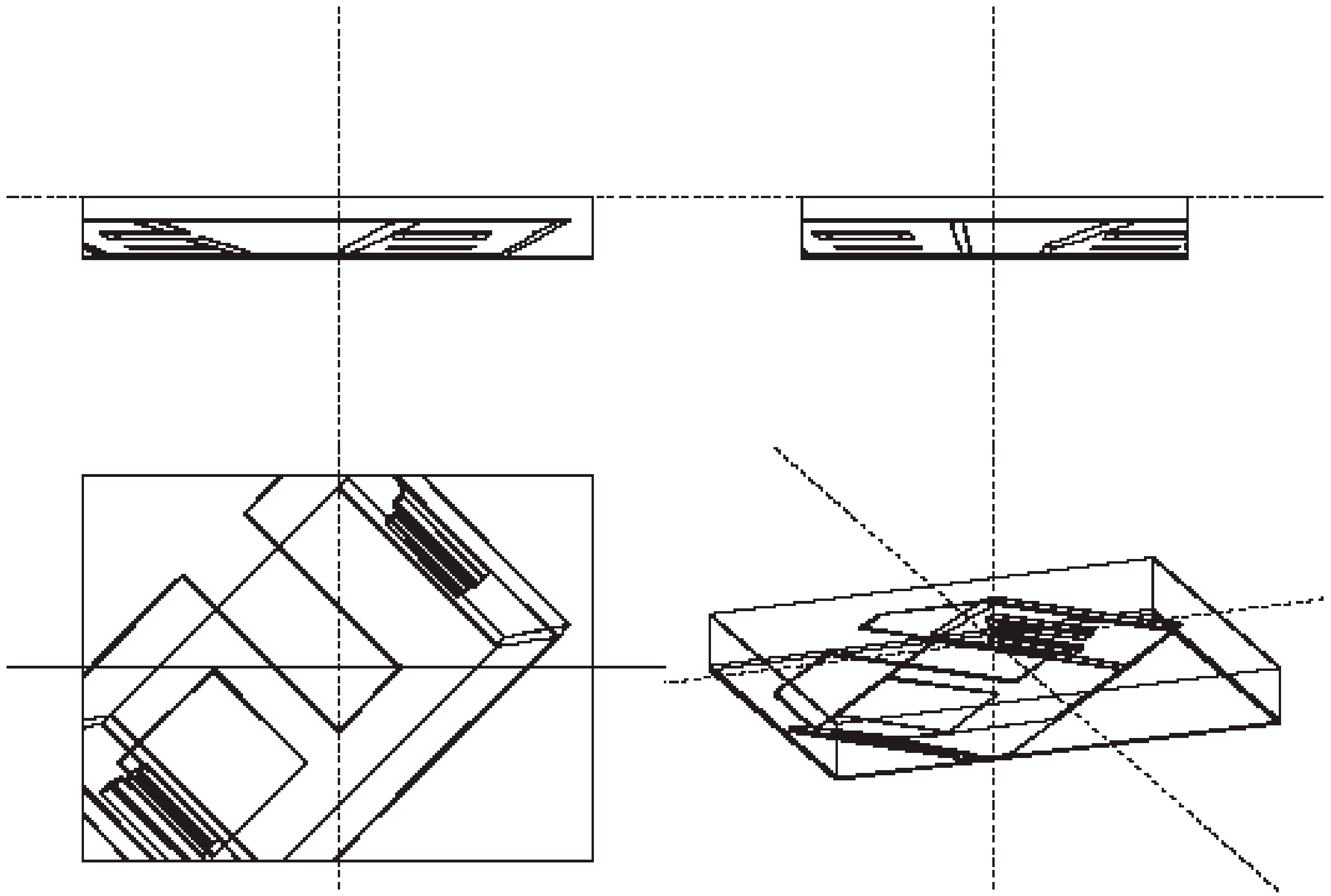

Figure 9.12 Canonical volume after perspective transformation, clipping and scaling to the window 2D

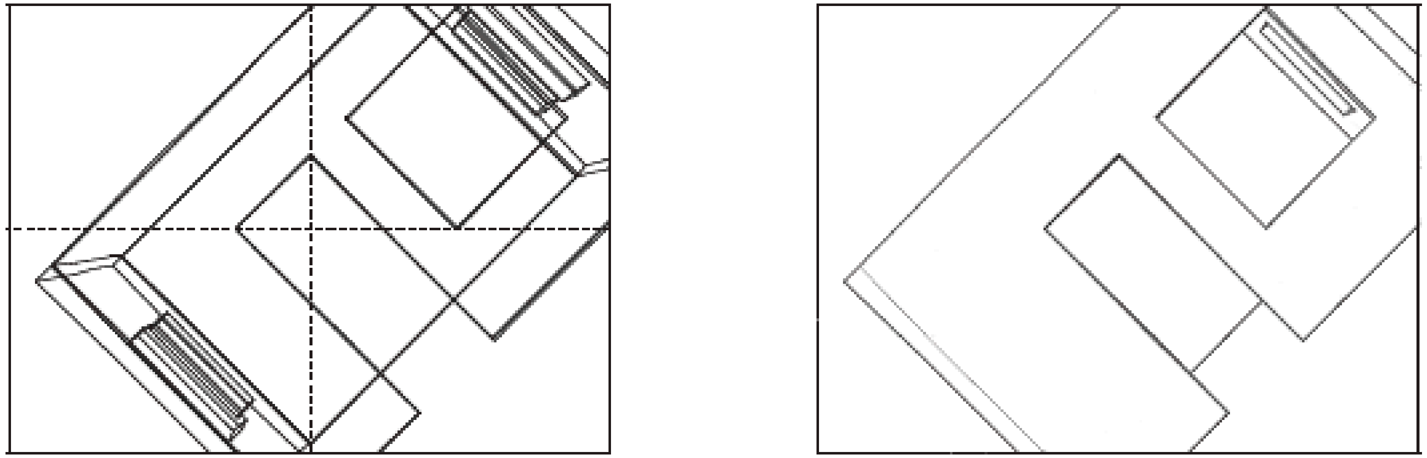

Figure 9.13 Final projected image: with (b) and without (a) removal of hidden lines

8.1 Hierarchical graphs

A hierarchical model, defined inductively as an assembly of component parts [LG85], is easily described by an *acyclic directed multigraph*, often called a *scene graph* or *hierarchical structure* in computer graphics. The main algorithm with hierarchical assemblies is the *traversal* algorithm, which transforms every component from *local coordinates* to global coordinates, called *world coordinates*.

In this case the notion of *multigraph* is introduced. A *directed multigraph* is a triplet $G := (N, A, f)$ where N and A are sets of nodes and arcs, respectively, and $f : A \rightarrow N^2$ is a mapping from arcs to node pairs. In other words, in a multigraph, the same pair of nodes can be connected by multiple arcs.

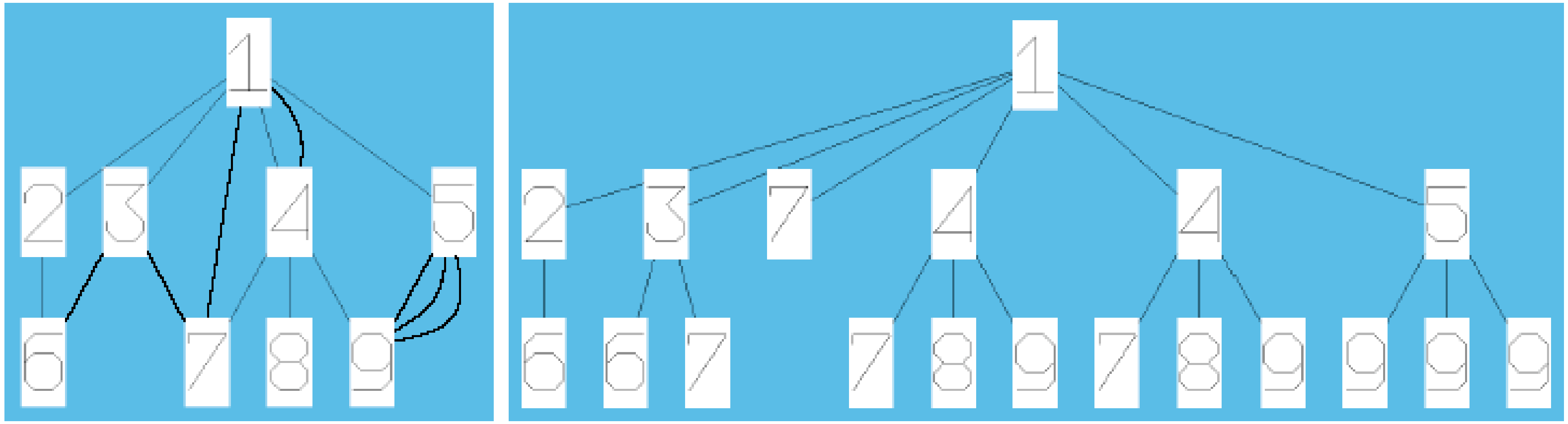

Figure 8.1 (a) Directed acyclic multigraph of an assembly (b) Tree of sub-assembly instances generated when traversing the multigraph

.1.1 *Local coordinates and modeling transformation*

 hierarchical multigraph is used to model a *scene database* in the sense describe
elow. In particular, each *node* may be considered a *container* of geometrical objects
here:

. The geometrical objects contained in a node a are defined using a system of
coordinates which is *local* to a.
. Each arc (a, b) is associated with an affine transformation of coordinates. In
simplest cases the identity transformation is used.
. The affine mapping of the arc (a, b) is used to transform the objects contained
within the b node to the coordinate system of the a node.

The previous properties are extended inductively to the subgraphs rooted in eac
ode. In particular:

1. the subgraphs rooted in the b_i sons of a, i.e., the geometrical data contained in such subgraphs, may be affinely mapped to the coordinates of a. The affine maps associated to (a, b_i) arcs are used at this purpose;
2. a subgraph may be instanced in a node (i.e., in its coordinate space) more than once. As shown in Figure 8.1 and discussed in Example 8.3.1, the number of instances of a subgraph in a node equates the number of different paths that connect the subgraph to the node.

Summary The main ideas concerning *scene graphs* can be summarized as follows. Nodes are *containers* of geometrical data stored in *local* coordinates. They are also used as root of subgraphs, whose data are transformed to the node coordinates by a traversal algorithm. Arcs (a, b) are associated with affine transformations, which map the data contained in b from their local coordinates to the coordinates of a. More than one arc may exist between the same node pair. This allows storage in memory only of *one copy* of each container. The composite transformations of coordinates applied to the linearized graph generated at traversal time are collectively known as the *modeling transformation.*

GKS segments In GKS (Graphical Kernel System) [EKP87, ISO85] the storage of graphical *segments* was introduced, which defines a two-level hierarchical system. More specifically: graphical *primitives* like polylines, polygons and text strings can be grouped into named collections, called *segments*. Segments are stored in a "normalized" coordinate space, and cannot be nested. Geometric transformations, including composite translation, rotation and scaling, can be applied to segments. Also, segments can be made visible/invisible, *picked* interactively and highlighted.

PHIGS structure network In PHIGS (Programmer's Hierarchical Interactive Graphics System) [HHHW91, ANSI87] *structure networks* are used, which can be visualized as acyclic graphs, where *structures* give the nodes of the graph, and *references* between structures give the arcs between nodes. Hierarchical assemblies of any depth can be modeled by such acyclic graphs. Structures are stored in a *centralized structure store* (CSS) independent of workstations, were structures can be *posted*. Structures can be interactively edited, by inserting, replacing and deleting *structure elements*.

Inventor′s scene graphs In Open Inventor [WO94] scene *databases* are defined as collections of *scene graphs*. A scene graph is an ordered collection of *nodes*, which are basic building blocks holding shape descriptions, geometric transformations, light sources or cameras. In other words, each node represents a geometry, property, or grouping object. Hierarchical scenes are created by adding nodes as children of grouping nodes. This approach clearly results in building scene graphs as acyclic directed graphs. No properties or transformations are attached to graph arcs, which just represent the containment relation between nodes. Node kits are provided as C++ classes with a predefined behavior, which can be customized by the application programmer by subclassing.

VRML scene graphs in VRML (Virtual Reality Modeling Language) [ANM97, ISO97] the same idea of scene graphs as ordered collections of nodes is used. The reader should notice that VRML originates from the File Format of Open Inventor. Such VRML files, written either in ASCII or gzipped binary format, can be used to import scene graphs into a scene database or even as an alternative to creating scene graphs programmatically. For example, scene graphs can be imported in Java 3D [SRD00] using VRML files. Some non-trivial differences exist between the semantics of scene graphs with versions 1.0 and 2.0 of VRML.

8.3 Traversal

The *traversal* of a hierarchical structure consists of a modified *Depth First Search* (DFS) of its acyclic multigraph,[3] where each arc — and not each node — is traversed only once. In particular, each node is traversed a number of times equal to the number of different paths that reach it from the root node.

The aim of the traversal algorithm is to “linearize” a structure network, by transforming all its substructures (i.e. all the subgraphs) from their *local coordinates* to the coordinates of the root node, assumed as *world coordinates*.

[3] Notice that the standard *dfs* graph traversal (see e.g. [AHU83]) visits all the nodes once, since it works by recursively visiting those sons of each node that it has not already visited.

For this purpose, a matrix denoted as the *current transformation matrix* (CTM) is maintained. Such a CTM is equal to the product of matrices associated with the arcs of the current path from the root to the current node. For the sake of efficiency, the traversal algorithm is implemented by using a stack of CTMs. When a new arc is traversed, the old CTM is pushed on the stack, and a new CTM is computed by (right) multiplication of the old one times the matrix of the arc. When unfolding from the recursive visit of the subgraph appended to the arc,[4] the CTM is substituted by the one popped from the stack. The TRAVERSAL algorithm is specified by some pseudo-language in Script 8.3.1.

Script 8.3.1 (Traversal of a multigraph)

```
algorithm TRAVERSAL ($(N, A, f)$ : $multigraph$) {
    $CTM$ := identity matrix;
    TraverseNode ($root$)
}

proc TRAVERSENODE ($n$ : $node$) {
    foreach $a \in A$ outgoing from $n$ do TraverseArc ($a$);
    ProcessNode ($n$)
}

proc TRAVERSEARC ($a = (n, m)$ : $arc$) {
    Stack.push ($CTM$);
    $CTM$ := $CTM$ * $a$.mat;
    TraverseNode ($m$);
    $CTM$ := Stack.pop()
}

proc PROCESSNODE ($n$ : $node$) {
    foreach object $\in n$ do Process( $CTM$ * object )
}
```