1 Introduction

The design of geometric shapes requires integration of a conceptual, top-down design process with an iterative bottom-up adjustment of design details and parameters. Traditional CAD systems support some aspects of this process by providing the designers with geometric construction techniques (primitive instantiation, rigid-body transformations, extrusions, Booleans, and blending) accessible through interactive front-end computers. The diversity of dialog scenarios available for specifying these construction or editing operations are largely responsible for the user-interface complexity and inefficiency in many CAD systems. Whereas CAD systems are widely used to represent, store, analyze, and process complex geometric models, they often lack facilities for interactive design. Interactive design requires special emphasis on the following aspects:

Top-down design

To cope with complexity, a designer generally uses abstraction and hierarchical decomposition. Initially, the major components are represented as abstract blocks with a vague, non-exact geometry. After evaluating alternative configurations, the individual components are designed in further detail, or alternatively, can be decomposed into other components. A design system should support facilities for fast specification of global shapes and incremental top-down refinement.

Bottom-up design

With bottom-up design, lower-level primitives, such as surfaces or solids, are combined into standardized parameterized parts or features that can be stored in a library, and reused in further designs. Techniques for storing, retrieval, and quick substitution of assembly components by standardized and optimized parts are required.

Hierarchical relations

3D representations involve various entities and hierarchical relations. On a functional level, the model may be represented as a part-of hierarchy. Geometric relations between components may be expressed via a tree of local coordinate systems or by a more complicated network of geometric
constraints. In addition, there may be a separate structure for combining objects with constructive solid geometry (CSG). Intuitive techniques for both representing and editing various hierarchical relations between components in a model are essential.

Parameter editing

Simple and effective techniques for selecting objects and changing parameters should be provided. A simple menu-interface may suffice for entering a known shape, but conceptual design requires smooth control over positions and dimensions with real-time graphical feedback.

The ABCSG system presented in this paper provides a paradigm for an easy-to-use, yet powerful front-end for designing CSG models of solids and assemblies. The paradigm is based on a unifying entity called cell, which captures the complete description of solid primitives (or of a regular pattern of these), with their dimensions orientation and relative positions. Hierarchical assemblies or CSG expressions of such cells may be specified, visualized, and easily edited through a hypertext spreadsheet-like interface. Incremental design is facilitated by the use of default instantiation parameters and by graphic tools for adjusting these parameters by direct manipulation.

The rest of the paper is organized as follows. Section 2 gives a brief introduction of related work on interfaces for modeling. Section 3 gives an overview of the ABCSG system, which encapsulates the concepts presented in this paper. Section 4 discusses the cell data structure. Section 5 discusses the hypertext facilities for editing the model. The integration of views is discussed in Sect. 6. Section 7 presents some general conclusions.

2 Related work

The efficiency of a CAD system depends both on the data structure for representing part-hierarchies and on the facilities for editing the data structure. Suitable data structures for representing hierarchical models include binary trees, n-trees, and directed acyclic graphs (DAG) (Sedgewick 1989, Braid 1978, Rappoport 1992). Interfaces for editing the data structure be roughly divided into two categories: language interfaces and interactive graphical interfaces.

The most elementary form of a language interface is a simple command interpreter that converts alphanumerical input into elementary operations on the data structure. Standard text-based techniques, such as macro substitution, learn, and redo, can be applied to facilitate model specification. Concise procedural model specifications can be obtained by using a modeling language (van Wijk 1986; Nackman et al. 1986; Rossignac et al. 1989). Standard control structures, such as variable assignment, loops, conditional statements, functions, and procedures, can be used to respectively define parameterization, repetition, constraints, and hierarchy.

A major disadvantage of language-type interfaces is the lack of interactivity and inherent incompatibility between the designer's three-dimensional mental representation and a one-dimensional sequence of textual commands.

To a large extent, these problems can be alleviated by applying an interactive graphical interface that enables the designer to edit a model by direct manipulation (Shneiderman 1983). Initially, direct-manipulation techniques were mostly restricted to manipulating objects in 2D orthogonal views. Recently, several techniques for manipulating models by direct manipulation on a 3D graphical representation have been presented (Bier 1986, 1990; Forrest 1986; Nelson and Olsen 1986; van Emmerik 1990). Graphical interfaces generally provide a more intuitive means for editing, but often lack the modeling functionality of procedural language.

Several approaches for the general concept of combining the user friendliness of a graphical interface with the modeling functionality of a procedural language have been presented. Most ideas evolve around the integration of two views, a textual view and a graphical view, which can both be used for editing. If the user changes something in the graphical view, the textual view is updated and vice versa.

The process of generating a textual description by editing a graphical representation is referred to as visual programming. Applications of visual programming have been presented for user interface design (Borning and Duesberg 1987; Myers 1988; Avrahami et al. 1989), picture editing (Nelson 1985), connecting graphics applications (Haebler 1988), and geometric modeling (Fuller and Prusinkiewicz 1988; van Emmerik 1991).

The system presented in this paper provides a three-view approach: dialog box view for editing a cell, spreadsheet-like textual view for editing the
Fig. 1a–f. By simply typing "flange B+C-5hole" the user creates seven instances of objects and specifies their Boolean combination. Although the parameters of the block "B" and cylinder "C" have not yet been specified, these primitives are displayed with default parameters to help the designer visualize the current stage in the design process. The "hole" has no definition and is represented by a coordinate system.

Using real-time visual feedback and direct-manipulation techniques, the designer can interactively adjust the parameters of the objects and the pattern. Subsequently, the object "hole" is defined as a cylinder and the CSG expression is displayed through direct CSG rendering or computed through boundary evaluation and can be stored for further graphic manipulation.

For top-down refinement of the holes, the user can select a hole by clicking either in the graphical view or in the textual view. The hole is displayed separately and its definition is refined by adding a cone "K." A separate view enables precise specification of parameters of the selected cell by typing values or dragging a slider. Finally, a new part is defined bottom-up as the union of four flanges and two blocks.
model structure, and 3D graphical view for direct manipulation on the 3D model. Hypertext (Nielsen 1990) techniques are used to selectively display parameters.

3 Overview of ABCSG

ABCSG provides an environment for interactive design of assemblies of primitive solids with constructive solid geometry. The focus is on top-down design, fast parameter adjustment, and ease of use. The resulting model is a parametric CSG model that can be processed by other applications. Our main contribution relies in the integration of three paradigms summarized below:

**Uniform cells**

There is only one modeling entity in the system, called cell. Complex models are be specified by grouping cells. Instead of having to deal with various structures and entities for CSG, geometry, and functional hierarchy, there is only one sequence of entities of the same type that integrates all of the above. This paradigm simplifies the interaction with the system and presents a clear conceptual model, because there are only two basic types of operations: combining cells and editing cells.

**Automatic defaults**

Everything is automatically defined with valid default values so the designer obtains immediate (visual) feedback after each incremental modification rather than having to wait until enough parameters for a unique solution have been specified. This not only applies to defaults for dimension and positions, but also for the definition of an object itself. An undefined new object is displayed with a default value (e.g., a coordinate system) and can be designed top down in a later stage of the design process.

**Hypertext interface**

Displaying both the structure of the model and all individual cell parameters is confusing. We propose a multiple-view user-interface which clearly separates the activities of editing cell parameters and combining cells, by offering a different view for the structure and the parameters of the selected cell. Hypertext techniques are applied to hide or show cell parameters and to traverse the object hierarchy during top-down design. A 3D graphical view gives real-time feedback during design and enables direct manipulation and graphical selection of cells. The paradigms are illustrated in Fig. 1 by an example that combines top-down and bottom-up design.

4 Cells

This section discusses the definition of a cell and describes how the designer can specify models by combining cells.

4.1 Graph semantics

Composite CSG objects are defined as a list of cells. A cell is either a primitive solid or a reference to another list of cells. Primitive solids are denoted as follows: "B" (block), "C" (cylinder), "K" (cone), "S" (sphere), and "T" (torus). Figure 2 shows the definition of object "flange" from Fig. 1. The flange is defined by a list comprising a block, a cylinder, and an object "hole." The object "hole" is defined by a cone and a cylinder. A cell comprises one of the set operators union ("+"), difference ("-"), or intersection ("*"). A set theoretical composition

---

Fig. 2. CSG objects are specified as a list of cells, which is evaluated from left to right. Cells are either primitives or point to another list of cells. The final model is represented as a list of lists forming a directed acyclic graph structure and unique instances are specified by paths.
is obtained by first evaluating the cells from left to right, and depth-first in case the cell points to another list of cells. The designer can incrementally refine an object by appending cells at the end of the list. The sequence of evaluation can be changed by switching the order of cells in a list.

The list of cells that form the definition of a composite object is referred to as a line and the set of lines that form the definition of an object is referred to as a sheet. The notion of lines and sheets are introduced to facilitate further discussion. A line can be regarded as a pointer to the first cell of an object definition and a sheet is a pointer to the first line in the definition of a model. Different cells may point to the same line or sheet so that the sheet can be regarded as a list of lists forming a directed graph structure (Sedgewick 1989). A simple form of recursion is provided by allowing a cell to point to the line in which it is instantiated (see Fig. 4).

4.2 Transformations

The transformation of a cell in the coordinate system of its line is specified by three transformations: a cell transformation, a repeat transformation, and definition transformation. The cell transformation specifies a transformation of the whole cell, including all instances specified by repetition. The repeat transformation specifies the incremental transformation between instances obtained via repetition. Finally, the definition transformation specifies the local coordinate system of a cell. As a result, the local transformation of instance $i$ of a cell is defined as:

$$T_{\text{local } i} = T_{\text{cell}} \cdot \prod_{i=1}^{i-1} T_{\text{rep}} \cdot T_{\text{def}}.$$  

Each transformation comprises three subtransformations for translation, scaling, and rotation, which are performed in the following order:

$$T = T_{\text{tra}} \cdot T_{\text{sc}} \cdot T_{\text{rotx}} \cdot T_{\text{roty}} \cdot T_{\text{rotz}}.$$  

Figure 3 shows the tree transformations that have been specified in order to obtain the pattern of holes in Fig. 1. The definition transformation moves the hole along the $x$-axis. A repeat transformation specifies the incremental rotation of 72 degrees along the $z$-axis. Because the definition transformation is performed after the repeat transformation, the $y$-translation is interpreted in the rotated coordinate system defined by the repeat transformation. Finally, the cell transformation moves the whole cell along the $x$-axis in order to position the pattern. The final transformation of an instance is obtained by multiplying the local transformations of the cells in its path. Most of the transformations inside a cell are for identification purposes and are not used. For cells that represent a single object, the designer can switch the standard order of rotation, translation, and scaling by using both the cell and definition transformation.

Another way of constructing patterns of objects is through recursion. A cell may point to the same line in which it is defined and controls the relative transformation between the instances by cell local transformation. Figure 4 shows two examples that combine repetition with recursion. The repetition number of a cell is displayed before the cell name and the recursion number is displayed after the name. The first example shows a line “ring” that is defined by the union of a torus “T” and an instance of the line “ring” itself. The instance is repeated three times and is provided with a repeat transformation that scales, moves, and rotates the instances “ring” so they are positioned around the

Fig. 3. Relative transformation is obtained by successively multiplying the cell repeat and definition transformation.
torus. Subsequently, the whole procedure is recursed two times. (Figure 13 shows several instances of the object in a bridge.) The tree in Figure 4 depicts a more complex non-symmetrical object defined by double recursion. Initially, a line “t” is defined as a cone (“K”) plus an instance of “t.” The instance is scaled, moved upwards, and rotated. A second instance of “t” is appended and transformed with slightly different parameters. Finally, both instances of “t” are provided with a recursion number.

5 Multiple-view interface

Three-dimensional design involves two relatively independent activities: positioning and dimensioning objects relative to each other, and top-down and bottom-up design of the hierarchical structure. The two types of activities are typically performed in a cyclic manner. First, a global simple structure is defined. Subsequently, the various components are positioned and dimensioned. Next, the structure of one or more components is refined, and so on.

ABCSCG provides separate user-interface styles for both activities. If the designer is concentrating on positioning objects, he or she should not be bothered with the general hierarchical structure of the model. Also, it is of no use to display all parameters and transformations of an object if the designer concentrates on changing the overall structure. We propose three different views for: editing the hierarchy (SheetView), editing parameters of a cell (CellView), obtaining 3D graphical feedback and direct manipulation (GraphicsView).

Implementation and integration of the three different views will be explained in the next sections.

5.1 SheetView

SheetView enables the designer to view and edit the hierarchical structure of the model, i.e., the graph structure discussed in Sect. 4. Displaying a graph as a 2D graphical diagram is a well-known problem. Difficulties arise when many nodes and links are involved and the graph should be displayed without overlapping nodes or crossing links. In our data structure, the display is even more complicated by the fact that links should be represented in an orderly way to represent the sequence of CSG evaluation. In addition to an understandable graph representation, an intuitive interface for creating deleting nodes and links is essential.

In our concept, we address both problems of presenting and editing a complex graph structure by providing a metaphor that resembles a spreadsheet. SheetView comprises a set of lines and each line defines a generic object (Fig. 5). The line starts with the name of the object, followed by an equal sign (“=”), and a Boolean expression of cells. Instances used in the definition of a generic cell may be defined by other lines.

For example, the object “prop” in Fig. 5 is defined as the union of a sphere (“S”), a cylinder (“S”), and four instances of a “blade.” The object “blade” is defined in a separate line as the union of a cone (“K”) and a sphere (“S”). Note that an object may be instantiated in several other objects. For example, the object “blade” is used both in the propeller and in the object “wing” to represent the tips of the wing. The concept of defaults and top-down design implies that objects can be instantiated before their definition is given. The user could have entered the top line first and defined the blade afterwards or vice versa.

For top-down design, it is convenient to temporarily display only a particular part of the model that should be refined, perhaps with a different viewpoint. SheetView enables the designer to show only parts (sub-solids) of the model by changing the “active” line of the sheet. The active line is indicated by an arrow (Fig. 5) and determines what is currently displayed (generally the top line). The user can move the arrow with the mouse. For example, to display only the propeller, the user can move the arrow to the third line.

The user can interact with SheetView via a keyboard as well as with a general purpose full-screen text editor. The major difference with an ordinary text editor is the fact that the editor is linked directly to underlying graph data structure. Each change (e.g., overwrite, delete) is immediately processed and checked for validity so that SheetView always reflects the current model. For example, if the user
deletes the "+" in a line "A = B + C," the system is set up to delete the whole cell "C," because the intermediate state "A = BC" would represent something completely different. In general, all keystrokes checked before SheetView is updated and illegal actions (e.g., press "+" on the "=" of a line) are discarded. Note that it would be impossible to use a separate general editor and language parser. For example, consider a sheet with two blocks represented as "A = B + B." If a general-purpose editor were used, it would never be clear whether the user had deleted the first or second block by parsing the new line "A = B."

The general idea of SheetView is to provide an easy-to-understand metaphor to represent and edit graph structures. A complex graph with say a 100 links can be represented by a few lines so the user can analyze the hierarchy without having to follow (crossing) links or use large scrollable windows. By presenting standard text-editing facilities, users intuitively know how to edit the graph, assuming they are familiar with the basics of text editing (insert, delete, arrows). A tight integration with the underlying data structure provides incremental validity check and prevents the user from creating invalid models.

It is clear that SheetView alone does not enable the designer to specify a model, because it does not provide a means for editing cell parameters. SheetView does enable the designer to select a particular cell via arrows or mouse. A selected cell can be subsequently edited in CellView or GraphicsView.

5.2 CellView

CellView enables the designer to view and edit all parameters of a cell after it has been selected by picking in SheetView or GraphicsView (see Fig. 13). CellView combines a set of 2D user-interface components for editing text (e.g., name), integers (e.g., repeat number), and floating-point values (e.g., transformation parameters).

In addition, there are three switches for selecting one of the three transformations and a button for obtaining additional user-interface components for editing material and surface properties (e.g., color, diffuse). A switch enables the designer to toggle between policies for inheriting material and surface properties (inherit/overwrite). The major function of CellView is the precise specification of all cell parameters after global dimensions and positions have been defined by direct manipulation. Special devices for editing integer and floating-point values are provided.

Figure 6a shows an integer editor based on a mileage-counter metaphor. The user can increment or decrement digits by clicking in area 2 and 3 and can add and remove digits by clicking in areas 1 and 0. A floating-point editor is designed to integrate smooth graphical control with precise numerical specification by combining a text editor with an infinite dial (Fig. 6b). The first value represents the actual floating-point value and the second value is used as a grid for truncating the first value. The user can increment or decrement the value graphically by dragging the dial-pointer in area 1 or can enter values directly by positioning the mouse in area 0. Areas 2 and 3 enable fast skipping of floating-point ranges by repetitive click.

CellView is implemented with the SigVig User Interface Toolkit (van Emmerik and Rappoport 1991), which is designed to provide user-interface tools across mixed graphics environments (X11/ GL) by providing an independent toolkit server.

5.3 GraphicsView

GraphicsView gives a three-dimensional representation of the model through shaded images, wireframes, or hidden-line renderings. Its purpose is three-fold: visual feedback during interaction in the other views, parameters editing via direct manipulation, and graphical selection. It is implemented in the graphics library GL.

Visual inspection

Interactive conceptual design requires immediate graphical feedback of user actions, especially dur-
Fig. 7. GraphicsView enables real-time visual inspection and graphical interaction via a wireframe images, b shaded images, c silhouette images, and d hidden-line images.
ing parameter manipulation via sliders or direct manipulation. Developments in commercially available graphics hardware enable real-time shaded display of moderately complex three-dimensional models. Although shaded images are generally preferable to wireframe images, they have limitations in engineering situations where visual inspection of interior or obscured parts is required.

For example, the shaded image in Fig. 7a does not show whether the cylinders touch or intersect the cubes. Wireframe images (Fig. 7b) enable inspection of the whole model, but do not show which objects are front and typically become difficult to interpret when many objects are displayed. The display mode depicted in Fig. 7c aims to alleviate this problem by displaying only silhouette edges and intersection edges. Note that obscured edges are displayed in a dashed-line style. Finally, a hidden-line display mode is provided to enable hard copies on monochrome output devices. The display styles shown in Fig. 7c, d are generated in real-time via a multiple-pass display using a hardware Z-buffer (Rossignac and van Emmerik 1991).

**Direct manipulation**

Direct manipulation provides the most intuitive way of specifying dimensions, position, and orientation of objects and is, therefore, very valuable for conceptual design. Clean integration of intuitive 3D-direct-manipulation techniques has been a central issue in designing the system. The approach followed is packaging direct-manipulation techniques into reusable objects called Direct Manipulation Devices (Dmd). Dmds can be regarded as a 3D analogue of reusable 2D user-interface components, such as X11 Widgets (Young 1989) and Vigs (van Emmerik 1991). Dmds are used for graphical representation and manipulation of values. Similar to 2D user-interface components, they have a current value (e.g., a floating point value), graphical appearance, and are provided with handles that enable the user to change a value with the mouse. Currently, the system provides Dmds for setting the viewpoint and dimensions, position, orientation and, scaling of cells.

Figure 8 shows two examples of Dmd: a Dmd for specifying dimensions (DmdDim) and a Dmd for specifying transformations (DmdLcs). DmdDim resembles a dimension-annotation symbol used in engineering drawings. The user can select the handle at the end of the arrow and drag it with the mouse to increase or decrease the current dimension. The input value of the Dmd is the type of object (e.g., block, cone) and its local coordinate system, so that Dmd knows where to display itself. The return value of Dmd is real value used to update the current cell parameter. Figure 9 shows three DmdLcs instantiated for manipulating the parameters of a cone.

DmdLcs enables the user to specify nine transformation parameters (translation, rotation, and scaling) with a 2D mouse. The concept of specifying 3D transformations with a 2D input device has been addressed earlier (Bier 1988; Nielsen and Olsen 1986; Chen et al. 1988). This implementation is based on a technique presented earlier (van Emmerik 1990). The input value of DmdLcs is a $3 \times 4$ transformation matrix and the output value is a floating-point value and an identifier that specifies the type of transformation: either translation (tx, ty, tz), rotation (rx, ry, rz), or scaling (sx, sy, sz). Its visual appearance resembles a local coordinate system with handles in the center and at the endpoints of the displayed axis.

Translations are specified by dragging the center (0) of the local coordinate coordinate system along one of the projected x-, y-, or z-axes. The system first samples a series of mouse events to detect along which axis the user wants to move and subsequently constrains the movement of the object along this axis. The 3D position is obtained by projecting a ray from the eyepoint through the cursor on the particular axis. Rotations along axis
9a

\[ \text{dim} = 3L + k \]

\[ \text{blade} = k \]

b

GraphicsView

\[ \text{stair} = C + 12 \text{step} \]
\[ \text{step} = B + \text{pole} \]
\[ \text{pole} = C + S + 2T \]

GraphicsView

\[ \text{stair} = C + 12 \text{step} \]
\[ \text{step} = \text{grate} + \text{pole} \]
\[ \text{pole} = C + S + 2T \]
\[ \text{grate} = 21B + 5B \]

10a

b

Edit

- Cut ALT-X
- Copy ALT-C
- Paste ALT-V
- Duplicate ALT-D
- Open ALT-O
- Close ALT-F
- Define ALT-R

249
of the coordinate system are specified by dragging axis end-points parallel to either one of the other axes. For example, to rotate around the z-axis, the user can drag the end-point of the positive x-axis (1) in the direction of the positive y-axis or the end-point of the negative y-axis in the direction of the positive x-axis. After sampling enough mouse events to detect the intention of the user (rotation around an axis), a ray from the eye through the cursor is intersected with the plane through the origin orthogonal to this axis. Subsequently, the object is rotated so that the axis that is manipulated by the mouse passes through the intersection point. Finally, scaling is specified by moving end-points of an axis along the same axis. Note there are several ways to obtain the same rotation or scaling and those indicated in Fig. 8 are just examples.

The visual design of Dmds is implemented using the sheet data structure (Fig. 9). For example, DmdDim comprises three lines ("L") and a cone ("K"). The system implementer (typically not the end-user) can use the system itself to design complex 3D Dmds. To link the visual appearance of the Dmd to the underlying implementation, the system designer can add labels (integers) to the cells. The general-selection mechanism of the application links the labels to callback functions in Dmd implementation. By using the system itself to specify the graphical design of Dmd, it is very easy to experiment with different alternatives and dynamically change the "look-and-feel" of the interface.

6 Integration of views

General issues in any multiple-view system are how to integrate the different views and how to enforce consistency among them. If the user changes something in one view, it may or may not be necessary to update other views. For example, during direct manipulation in GraphicsView, the current parameters in CellView are updated, but there is no need to change SheetView. Also, it is essential that all views be synchronized. They should not only represent the same model, but also the same state (e.g., selected cell).

Selection

GraphicsView is unique in the sense that it displays all instances derived from the graph and gives intuitive feedback about positions and dimensions of objects. Hence, it is the most natural interface for selecting a part of the model for further manipulation. Selection by graphics pick is complicated because there is no one-to-one relation between the cells in the sheet and the objects displayed on the screen. Picking does not select a particular cell, but a particular path in the graph. For example, Fig. 10 shows a staircase defined by three lines. After clicking at the block of the second stair, the system by default interprets the first cell (step) of the selected path (step B) as the current selected cell. The designer can then manipulate this cell or alternatively traverse the path down to the next cell "B." The current selected cell is indicated by showing the local coordinate system (DmdLocs). It is also possible that more than one path (instance) is selected by one pick. For example, a pick near the center cylinder in the stair will select both the path (C) and the path (step B). Different combinations of mouse buttons or click modes (single/double) can be used to both cycle through selected paths and traverse the current path up and down.

Fig. 11a. A user action in one of the views is passed as an update request to the ViewManager. b ViewManager updates the selections and changes the data structure if necessary. c If the request is valid, all views are updated.

Fig. 9a, b. Graphical design of DmdDim with the system (a) and its instantiation for manipulating dimensions of a cone (b). A two-pass display mode of DmdLocs draws obscured parts in dashed lines so that the user can manipulate objects inside other objects.

Fig. 10a, b. A path is selected by picking with the mouse pointer. The selected cell is indicated by displaying its local coordinate system. In addition, the whole path is displayed in the SheetView. After opening cell "step," block "B" is replaced by an instance of a new cell "grate"
\[ c = b + b \]
\[ b = 22a \]
\[ a = 57b \]

\[ t = 3a + 3a + B \]
\[ a = 17B + a5 \]

<table>
<thead>
<tr>
<th>Name</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup</td>
<td>-</td>
</tr>
<tr>
<td>Rep</td>
<td>5</td>
</tr>
<tr>
<td>Rec</td>
<td>1</td>
</tr>
<tr>
<td>l</td>
<td>0.100</td>
</tr>
<tr>
<td>w</td>
<td>1.100</td>
</tr>
<tr>
<td>h</td>
<td>0.050</td>
</tr>
<tr>
<td>Tra x</td>
<td>0.000</td>
</tr>
<tr>
<td>Tra y</td>
<td>0.000</td>
</tr>
<tr>
<td>Tra z</td>
<td>0.000</td>
</tr>
<tr>
<td>Rot x</td>
<td>0.000</td>
</tr>
<tr>
<td>Rot y</td>
<td>0.000</td>
</tr>
<tr>
<td>Rot z</td>
<td>0.000</td>
</tr>
<tr>
<td>Sca x</td>
<td>1.000</td>
</tr>
<tr>
<td>Sca y</td>
<td>1.000</td>
</tr>
<tr>
<td>Sca z</td>
<td>1.000</td>
</tr>
<tr>
<td>Scale</td>
<td>1.000</td>
</tr>
<tr>
<td>Cell</td>
<td>off</td>
</tr>
<tr>
<td>Repeat</td>
<td>off</td>
</tr>
<tr>
<td>Def</td>
<td>on</td>
</tr>
<tr>
<td>Material</td>
<td>-</td>
</tr>
</tbody>
</table>
6.1 Integration

Consistency and integration of views is enforced by a separate module, the ViewManager (Fig. 11). A primary task of ViewManager is to provide the interface to the sheet data structure. If the user has changed something in a particular view, a request for update of the models is sent to ViewManager. ViewManager checks if the request is valid and updates the sheet data structure. Subsequently, the other views are notified that the model has been changed and can re-display themselves if necessary.

A second task of ViewManager is selection management. As discussed in the previous section, the user can select paths, traverse paths, or directly click on lines and cells in SheetView. ViewManager keeps track of the various selections and enforces consistency between selections. For example, if the user clicks on an instance in GraphicsView, ViewManager stores all the paths and clears the previous selections. In addition, it stores the current selected path and the current selected level in the current path. All views have direct access to the selections and can display selections in their own way. SheetView displays the cells of the selected path and displays the cursor on the selected cell. CellView displays all parameters of the selected cell and GraphicsView displays a local coordinate system (Fig. 10). The general idea is that different views are allowed to read both the sheet and the selections, but are not allowed to set them directly. Instead, views send messages to ViewManager that are tagged with two labels: a view identifier and change-type identifier. The view identifier is necessary to prevent recursive re-display of the same view that is sent the request. The change type defines the type of request (e.g., translate cell, delete cell) so that each view can decide whether a (partial) re-display is necessary.

6.2 Hypertext

The multiple-view approach presented in this paper provides an interaction model with many hypertext-like features (Nielsen 1990). An integrated environment with textual, 2D graphical and real-time 3D-graphical-interaction techniques is used. SheetView provides a 2D non-sequential representation of the model that hides details (cell parameters). The user can click on a particular cell to display the contents in CellView. Navigation through the object hierarchy is enabled by an "open-close" mechanism. The user can select a cell and choose the option "open" from a menu to display the definition of the cell. If the cell is defined in the same sheet, the active line in SheetView is set to the line that defines the cell and the line is displayed in GraphicsView. The user can subsequently "open" other cells in the line or change the line itself. A cell can also be defined by another sheet (e.g., the cell "stair" in Fig. 12). In this case, the sheet that defines the cell is retrieved from file and presented to the user. At any time, the user can backtrack the model traversal by choosing the option "close."

The main hypertext features in ABCSG can be summarized by the concepts of non-sequential data, hiding data, linking objects, and following and backtracking links. SheetView provides a navigation utility that enables the user to understand the network structure and keep track of his current position.

7 Conclusions

This paper presents a new approach for interactive design of complex part-hierarchies. We have presented a data structure that simplifies the designer's conceptual model by integrating data structures and control structures into a list of uniform cells. The multiple-view interface provides an integrated environment for combining cells bottom-up and top-down and for editing cell parameters. User benefits can be summarized as follows:

- Integrated approach for hierarchy, repetition, and recursion.
- Easy to understand metaphor for editing hierarchy.
- Hypertext techniques for selective display of parameters and navigation.

We believe the concept provides a very easy-to-understand model that still enables a designer to specify fairly complex models (Fig. 12). The ver-

Fig. 12a, b. Repetition and recursion can be applied for interactive design of patterns and textures

Fig. 13. A bridge is represented by four lines and an instance of the "stair" from Fig. 10. The object hierarchy can be traversed by picking in SheetView or GraphicsView. CellView enables precise specification of all cell parameters
enity and general applicability can be illustrated by the fact that the system is used for designing the system's internal direct-manipulation devices and that it can be applied to design of textures and patterns as well (Fig. 13). However, we are fully aware that current implementation lacks some major functionality for a full-fledged standalone CAD system. Future developments will focus on the integration of free-form surfaces, deformations, and constraints.

The concept presented in this paper is implemented in C on an IBM Risc System 6000 with graphics accelerator. GraphicsView is implemented with the GL Graphics Library and the 2D user-interface components are implemented with the SigVig User Interface Toolkit (van Emmerik and Rappoport 1991).

Acknowledgements. This work has been carried out at the T.J. Watson IBM Research Centre. We are very grateful for the support and recommendations received from Paul Borrel, Bengt-Olof Schneider, and John Mastroguillio. We would especially like to thank Paul Borrel for his excellent job in designing the bi-directional interface with Catia.

References

van Emmerik MJGM, Rappoport A (1991) Sigvig: a concept for separation of application and user interface toolkit. IBM Research Report 17303, T.J. Watson Research Center, Yorktown Heights, NY

Rossignac JR, van Emmerik MJGM (1991) Hardware assisted determination of visible lines and silhouette edges in tessellated wireframes. IBM Research Report, T.J. Watson Research Center, Yorktown Heights, NY

Maarten van Emmerik received his MSc in Industrial Design Engineering and PhD in Computer-Aided Design, both from Delft University of Technology in the Netherlands. He visited the Interactive Geometric Modeling group at the IBM T.J. Watson Research Center, Yorktown Heights, New York, as a postdoctoral research fellow. Currently, he coordinates MicroStation third-party software development in Europe as the MicroStation Independent Software Developer (ISD) Program Manager at Intergraph European Headquarters.
ARI RAPPOPORT received his Ph.D. in computer graphics from the Institute of Computer Science, The Hebrew University of Jerusalem, Israel, in 1990. He afterwards visited the Interactive Geometric Modeling group at the IBM T.J. Watson Research Center, Yorktown Heights, New York. Currently, he heads the Geometric Modeling and Computer Graphics lab at the Institute of Computer Science, The Hebrew University. His research interests include interaction techniques for geometric applications, constraints and parameterizations in modeling, data structures and algorithms for geometric objects, scientific visualization, and applications of image processing and computer vision to computer graphics.

JAREK ROSSIGNAC joined the IBM T.J. Watson Research Center in 1985 with a PhD in EE from the University of Rochester, New York, for his work on offsetting and blending solids. He became the manager of Interactive Geometric Modeling in 1989, where he leads research projects in interactive design and animation, engineering graphics, and geometric modeling. He has published survey papers and research contributions on many aspects of solid modeling and graphics and has received best-paper awards from Computers and Graphics in 1990 for his work on feature-based design and from Eurographics '91 for joint work on 3D metamorphosis. Rossignac is associate editor of the ACM Transactions on Graphics, where he guest-edits a special issue on "Interactive sculpting." He also is on the board of Computer-Aided Design where he guest-edited the 1991 "Beyond Solid Modeling" special issue. He is the co-chairman of the bi-annual ACM Symposium on Solid Modeling for 1991 and 1993 and co-editor of special issues of IJCGA and CG&A devoted to selected papers from that conference.