Generating sample looks for geometric objects in a visual design language

Omid Banyasad
Philip T. Cox

Technical Report CS-2008-04

March 17, 2008

Faculty of Computer Science
6050 University Ave., Halifax, Nova Scotia, B3H 1W5, Canada

Generating sample looks for geometric objects in a visual design language

Omid Banyasad
banyasad@ca.ibm.com
IBM Software Group
Ottawa Lab
Ottawa, Ontario, Canada

Philip T. Cox
pcox@cs.dal.ca
Faculty of Computer Science,
Dalhousie University
Halifax, Nova Scotia, Canada

Abstract. A major application of visualisation is to the design of structured objects such as buildings, machinery and electronic circuits, as in Computer-Aided Design (CAD) systems. Complex designs are frequently parameterised so that they represent families of objects rather than single artifacts, and building them requires design environments that support both the concrete visualisation and manipulation of components, and the abstract specification of how they are related. CAD systems usually separate these two aspects, providing the abstract programming capability via a textual programming language grafted on to a 3D object editor and solid modeler.

A recently proposed design language merges these two activities by embedding representations of solid objects in a visual logic programming language. A practical issue that arises is how to automatically generate a "sample look", a reasonable representation for a parameterised object which can be displayed during execution (assembly) of a design. We present a solution to this problem based on "factoring", which separates the constraints on a solid object from its geometric properties.

1 Introduction

The work reported here is part of a project originally motivated by the hypothesis that since both geometric objects and some of the operations used for combining them can have concrete visual representations, a visual design environment that provides programming capabilities via manipulation of such representations would be a powerful tool for creating parameterised objects. Such a design environment would be an attractive alternative to conventional CAD systems, where the algorithms used for the parameterised design and the objects on which they operate have dissimilar representations, one textual, the other pictorial.

This observation led to Language for Structured Design (LSD), a programming language for parameterised design that provides a close mapping between the programming domain and the design space. LSD also allows designers to more easily solve certain problems within the design space, by employing its declarative problem solving capabilities [3].

The original LSD, as a simple exploration of the notion of design in a visual language, had just one operation on solids. However, the selection of operations required in a design language depends on the domain of application, so LSD clearly needed to be extensible. As a basis for extensibility, a very general notion of design space was required, providing just enough detail to capture the essence of solids in logic. To meet this need, a formal model for characterising objects in a design space was proposed in [6]. This formalisation also defines the concept of an operation on solids, and selective interfaces that allow solids to declare what operations can be applied to them. Based on this model, the required generalisation of LSD was made. It is important to note that this formalisation characterises solids and operations only to the extent required by LSD, so does not address any of the practical details of solid modeling. Hence, an implementation of LSD needs to utilise a capable solid modeling package in order to achieve its objectives. By a "solid modeling package" we mean a solid modeling kernel extended with additional capabilities to provide services suitable for LSD. For example, the same way that Autodesk Inventor [1] extends the ACIS kernel from Spatial Technologies [5,11], and SolidWorks [10] or Bentley Systems' MicroStation [4] extend the Parasolid kernel [9] from Unigraphics.

In another thread, it was noted in [3] that LSD, like any other Visual Programming Language (VPL) would substantially benefit from being implemented in a carefully designed environment. For example, an interactive debugger which allows a programmer to observe and analyse the behaviour of an LSD program at run-time through the use of features such as execution pause and dynamic execution rollback, would be a valuable part of an LSD implementation. Following this observation, a programming environment with certain debugging features including an animated execution of programs was discussed in [3].

The major motivation for providing the formal model in [6] was to generalise LSD with the notion of a general operation rather than extending LSD with an exhaustive list of operations. However, other requirements for the solid modeler, were not addressed. In this paper, we will extend the solid modeler to formally describe the notion of sample look for objects required during the animated execution of an LSD program.We also make some small but important adjustments to some of the definitions in [6] to better capture the notion of invalid objects.

Since our solutions to the aforementioned problems involves an extension to the definitions given in [6], in the next section, we will briefly sum-

marise the core concepts presented there, modifying some of the original definitions to remove an anomaly in the way they dealt with invalid objects. The reader is encouraged to consult [6] for a thorough discussion of the original formal model. First, however, we define some notation.

If x and y are sequences, we denote the concatenation of x and y by x•y. For example, if x is $x_1,x_2,\ldots,x_n$ and y is $y_1,y_2,\ldots,y_m$ then x•y means $x_1,x_2,\ldots,x_n,y_1,y_2,\ldots,y_m$. If x is a sequence and y is not a sequence, for consistency, by x•y we mean $x_1,x_2,\ldots,x_n,y$ and by y•x we mean $y,x_1,x_2,\ldots,x_n$. We use () to represent a sequence of length zero. If x is a sequence, we may also use x to denote the set of elements in x.

If **C** is a formula, by var(**C**) we mean the set of all free variables of **C**. If **C** is a formula, and a is a term, we use the standard notation $\mathbf{C}_x[a]$ for the formula obtained from **C** by substituting a for every occurrence of x. We extend this notation to sequences of variables and terms of the same length. That is, if $x_1,x_2,\ldots,x_n$ is a sequence of variables and $a_1,a_2,\ldots,a_n$ is a sequence of terms, then $\mathbf{C}_{x_1,x_2,\ldots,x_n}[a_1,a_2,\ldots,a_n]$ is the formula obtained from **C** by substituting a_i for every occurrence of x_i for each i ($1 \le i \le n$). We might also combine single variables and sequences of variables in this notation. For example, $\mathbf{C}_{x,y}[a,b]$ means that each of x and y is either a single variable or a sequence, and each of a and b is either a single term or sequence of terms, of the same length as x and y, respectively. Another example, is $\mathbf{C}_{x•y}[a]$ which means that a is a sequence of terms such that $|a| = |x•y|$. $\mathbf{C}_{x•y,z}[a,b]$ means that a is a sequence of terms such that $|a|=|x•y|$, and b is a sequence of terms iff z is a sequence of variables of the same length as b.

We use a set of formulae as an alternate notation for a formula which is the conjunction of the formulae in the set such that none of them are conjunctions. The conjunctions of two or more sets of formulae is understood to be the conjunction of the formulae in the sets. If **C** is a set of formulae and $x \subseteq$ var(**C**), by $\mathbf{C}^x$ we mean the set of formulae $\{ \mathbf{E} \mid \mathbf{E} \in \mathbf{C}, x \cap \text{var}(\mathbf{E}) = \varnothing \}$. If x is a variable, $\mathbf{C}^x$ has the obvious meaning $\mathbf{C}^{\{x\}}$.

If Φ is a partial function from A to B, by scope(Φ) we mean the set $\{x \mid x \in A$ and Φ is defined for $x\}$ and by the image of Φ we mean the set $\{\Phi(x) \mid x \in \text{scope}(\Phi)\}$.

If **C** is a formula, Γ is a partial function, and $x•y•u \subseteq$ var(**C**), $\Phi(x•y•z)=[\mathbf{C}]\ \Gamma(x•z•u)$ defines the partial function Φ such that $a•b•c \in$ scope(Φ) if there is a unique d such that $\mathbf{C}_{x,y,u}[a,b,d]$ holds and $a•c•d \in$ scope(Γ). If the partial

function Γ can be defined uniquely and unambiguously by an expression **E**, then we can substitute **E** for Γ in the definition of Φ.

2 Formal Model

Solids are modeled in a *design space*. The design space is a normal Euclidean space augmented with an arbitrary but fixed finite number of real-valued *properties*. A solid is a function that maps a list of *parameter values* to a set of points in the design space constituting the volume of the solid. Therefore, each solid in the design space represents a family of objects, each realised by a particular choice of parameter values. According to [6], if a particular vector of values assigned to the variables over which a solid is defined produce the empty set of points, this result is considered to be invalid: that is, to correspond to an impossible object. However, the empty set can also represent a valid solid, for example when a solid results from computing the intersection of two non-intersecting objects. In order to rectify this anomaly, we modify some of the definitions in [6]. Each definition in this section is either the same as the corresponding definition in [6] and is reproduced for completeness, possibly with notational changes, or has been modified to deal with the anomaly. In the latter case, the modification will be explained following the definition.

Definition 2.1: A *design space in m dimensions over r properties* for some integers $m > 0$ and $r \geq 0$ is the set of all subsets of $\mathbf{R}^m \times \mathbf{R}^r$.

Definition 2.2: A *solid* in a design space D in *n variables* for some $n \geq 0$ is a partial function Φ: $\mathbf{R}^n \rightarrow D$ such that, if (y,p) and $(y,q) \in \Phi(y)$ for some $y \in \mathbf{R}^n$, then $p=q$. A *variable* of Φ is an integer i such that $1 \leq i \leq n$. Symbolic names may also be used to refer to the variables of a solid.

The intuition behind this definition is that something we normally think of as a solid can be characterised by a set of points in space, where each point has a unique value for each property associated with it.

Definition 2.2 differs from that in [6] by defining a solid to be a partial function instead of a function. This effectively makes the interpretation of the empty set of points unambiguous, allowing us to consider every set of points produced by such a function to correspond to a realisable object, whether or not the set is empty.

This definition also provides parameterisation of objects. The image of a solid corresponds to a family of real-world objects, each created by the partial function from a vector of variable values.

We will continue to refer to a partial function that defines a solid simply as a "function" when the meaning is clear in context.

Definition 2.3: A solid Φ in D is *valid* iff the image of Φ is not empty.

Note that a solid can generate the empty set of points as one of the members of the family that it defines. The image of a solid may even contain just the empty set, although such solid would be of no practical significance.

Definition 2.4: If Φ and Ψ are solids in a design space D, Φ and Ψ are said to be *equivalent*, denoted $\Phi \equiv \Psi$, iff they have identical images.

This definition recognises that a family of objects in a design space may have more than one representation as a solid (partial function) and is simpler than its counterpart in [6], since there is no longer any need to eliminate invalid objects from the images of Φ and Ψ.

Consider a two dimensional design space over the three properties *temperature*, *colour*, and *material* where *temperature* is determined by *colour* and *material,* and *colour is* determined by *temperature* and *material.* A **Square** solid in this space can obviously be defined by a function Φ of variables (*b,c,l,*α*,temperature,colour*) or by another function Ψ of variables (*b,c,d,e,material,colour*), where *b*, *c*, *d*, *e*, *l* and α are the geometric variables shown in Figure 1

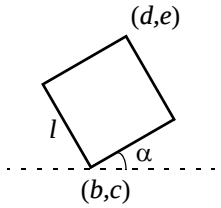

Figure 1: A **Square** in the design space.

Example 2.5: To further illustrate the definition, let D be a design space in 2 dimensions over zero properties: then the partial function **Disk** is a solid in three variables which characterises the family of disks with radius *r* and centre at (*b,c*) defined as

$$\textbf{Disk}(b,c,r) = [\, 0 \le r \,] \; \{ \; (x,y) \mid x, y \in \mathbf{R} \text{ and } (x-b)^2 + (y-c)^2 \le r^2 \; \}.$$

Definition 2.6: If D is a design space and n is a positive integer, an *n-ary operation in* D is a quintuple $((a_1,\ldots,a_n),(\vec{z}_1\bullet\ldots\bullet\vec{z}_n),F,L,\mathbf{C})$ where

- $a_1,\ldots,a_n$ and $\vec{z}_1\bullet\ldots\bullet\vec{z}_n$ are sequences of distinct variables such that for each i $(1\leq i\leq n)$, $\vec{z}_i$ is a sequence of variables and $a_1,\ldots,a_n$ and $\vec{z}_1\bullet\ldots\bullet\vec{z}_n$ are disjoint.
- F is a partial function from D^n to D such that $\forall x \in \text{scope}(F)$, if (e,g) and $(e,h) \in F(x)$, then $g=h$.
- $L=(\mathbf{L}_1,\ldots,\mathbf{L}_n)$ is a sequence of formulae, called *selectors,* such that $\text{var}(\mathbf{L}_i)=a_i\bullet\vec{z}_i$. The integer $|\vec{z}_i|$ is called the *size* of $\mathbf{L}_i$.
- $\mathbf{C}$ is an open formula, called the *constraint* of the operation, such that $\text{var}(\mathbf{C})=\vec{z}_1\bullet\ldots\bullet\vec{z}_n$.

This definition differs from that in [6] in that the function F is now defined as a partial function from D^n to D, allowing the scope of the solid resulting from the application of an operation to its operand solids to be expressed, among other things (see below), in terms of the scope of F.

Example 2.7: The **Bonding** operation in a 2D design space is defined by $((a_1,a_2),(p_1,p_2,p_3,p_4,p_5,p_6,p_7,p_8),\mathbf{fuse},\{\mathbf{edge}1,\mathbf{edge}2\},\mathbf{bond})$, where $\mathbf{fuse}(x,y)=[\,x\cap y=l\,](x\cup y)$, such that l is a line, $\mathbf{bond}=\{p_1=p_7\ ,\ p_2=p_8\ ,\ p_3=p_5\ ,\ p_4=p_6\}$, $\mathbf{edge}1=\mathbf{edge}_{a,b,c,d,e}[a_1,p_1,p_2,p_3,p_4]$, $\mathbf{edge}2=\mathbf{edge}_{a,b,c,d,e}[a_2,p_5,p_6,p_7,p_8]$, and $\mathbf{edge}(a,b,c,d,e)$ is true iff (b,c) and (d,e) are points in a, every point on the line between them is in a, every point to the right of this line is in a, and every point to the left of this line is not in a.

Definition 2.8: If Φ is a solid in n variables and $\mathbf{L}$ is a selector of size k of some operation, where $\text{var}(\mathbf{L})=a\bullet\vec{z}$ and $|\vec{z}|=k$, then *an* L*-interface to* Φ is a partial function $\phi:\mathbf{R}^n\rightarrow\mathbf{R}^k$ such that $\phi(u)=[\,u\in\text{scope}(\Phi)\ ,\ \mathbf{L}_{a,\vec{z}}[\Phi(u),y)]\,]\,y$, and $\forall u,v \in \text{scope}(\Phi)$, if $\Phi(u)=\Phi(v)$ then $\phi(u)=\phi(v)$.

This definition modifies its counterpart in [6] to account for our revised notion of validity of solids.

Example 2.9: Let **Disk** be as in Example 2.5. Since no two points in a disk satisfy the **edge** selector from Example 2.7, no **edge**-interface to **Disk** can be defined. Therefore, **Disk** cannot be an operand for the **Bonding** operation.

Definition 2.10: Let $\otimes=((a_1,\ldots,a_n),(\vec{z}_1\bullet\ldots\bullet\vec{z}_n),F,L,\mathbf{C})$ be an n-ary operation; where $L=(\mathbf{L}_1,\ldots,\mathbf{L}_n)$, and for each i $(1\leq i\leq n)$ let Φ_i be a solid in n_i vari-

ables, and ϕ_i be an $\mathbf{L}_i$-interface to Φ_i. A solid Ψ in $t = \sum_{i=1}^{n} n_i$ variables, called *the application of* $\otimes$ *to* $\Phi_1,\ldots,\Phi_n$ *via* $\phi_1,\ldots,\phi_n$ is defined as follows. If $y \in \mathbf{R}^t$ denote by y_1 the first n_1 elements of y, denote by y_2 the next n_2 elements of y and so forth, then $\Psi(y)$ is defined as

$$\Psi(y)=[\ \mathbf{C}_{\tilde{\chi}_1,\ldots,\tilde{\chi}_n}[\phi_1(y_1),\ldots,\phi_n(y_n)]\ ]\ F(\Phi_1(y_1),\ldots,\Phi_n(y_n)).$$

This definition modifies its counterpart in [6] by properly accounting for the validity of a generated solid. The solid generated by an operation is defined for a particular selection of values for its variables provided that the constraint of the operation is satisfied by those values, each of the operand solids is defined for that selection of values, and the list of subsets of $\boldsymbol{D}$ is in the scope of $\boldsymbol{F}$. This condition, implicit in the definition, is made explicit in following lemma.

Lemma 2.11: If Ψ is the solid defined in Definition 2.10 and $u \in$ scope(Ψ), then $u_i \in$ scope(Φ_i) for each i ($1 \le i \le n$), and $(\Phi_1(u_1),\ldots,\Phi_n(u_n)) \in$ scope($\boldsymbol{F}$).

Proof. Since $u \in$ scope(Ψ), it has to be in the scope of the partial function defined by the expression $\boldsymbol{F}(\Phi_1(u_1),\ldots,\Phi_n(u_n))$. Therefore, $u_i \in$ scope(Φ_i) for each i ($1 \le i \le n$), and $(\Phi_1(u_1),\ldots,\Phi_n(u_n)) \in$ scope($\boldsymbol{F}$). □

Example 2.12: Assume that we want to attach the two semi ring-shape objects in Figure 2 using the **Bonding** operation. Although the constraint of the **Bonding** operation is true, the two objects cannot simultaneously satisfy the **Bonding** condition and be in the scope of **fuse** as defined in Example 2.7, and therefore the application of the **Bonding** operation to these objects will always fail. By failing, we mean that the image of the solid resulting from the application of **Bonding** to these objects is empty.

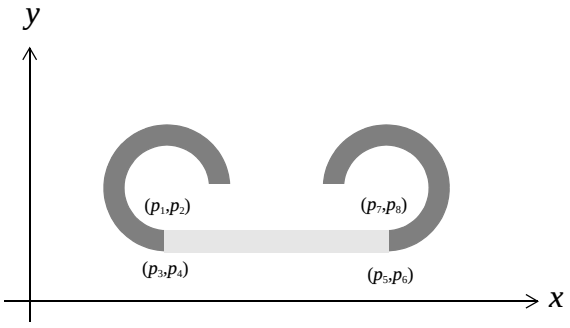

Figure 2: An unsuccessful application of **Bonding**.

In the next example, we show how the interfaces in the application of an operation are handled.

Example 2.13: Let $\mathbf{rectangle_1}(x_1,y_1,w_1,h_1)=\mathbf{Rectangle}(x_1,y_1,w_1,h_1)$ and $\mathbf{rectangle_2}(x_2,y_2,w_2,h_2) = \mathbf{Rectangle}(x_2,y_2,w_2,h_2)$ be two rectangle solids in a 2D design space over zero properties where

$$\mathbf{Rectangle}(x,y,w,h)=[\ 0\le w,\ 0\le h\ ]\ \{\ (a,b)\ |\ x\le a\le x+w\ ,\ y\le b\le y+h\ \}.$$

Also let $\mathbf{rightSide}(x,y,w,h)=(x+w,y+h,x+w,y)$ and $\mathbf{leftSide}(x,y,w,h)=(x,y,x,y+h)$ be **edge**-interfaces to $\mathbf{rectangle_1}$ and $\mathbf{rectangle_2}$, respectively. The application of **Bonding** to $\mathbf{rectangle_1}$ and $\mathbf{rectangle_2}$ via **rightSide** and **leftSide** is the solid

$$\begin{aligned}\Psi(x_1,y_1,w_1,h_1,x_2,y_2,w_2,h_2)&\\ &= [\ \mathbf{bond}_{p_1,\ldots,p_8}[\ x_1+w_1,\ y_1+h_1,\ x_1+w_1,\ y_1,\ x_2,\ y_2,\ x_2,\ y_2+h_2\ ]\ ]\\ &\qquad \mathbf{fuse}(\mathbf{Rectangle}(x_1,y_1,w_1,h_1),\mathbf{Rectangle}(x_2,y_2,w_2,h_2))\\ &= [\ x_1+w_1=x_2\ ,\ y_1+h_1=y_2+h_2\ ,\ y_1=y_2\ ]\\ &\qquad \mathbf{fuse}(\mathbf{Rectangle}(x_1,y_1,w_1,h_1),\mathbf{Rectangle}(x_2,y_2,w_2,h_2)).\end{aligned}$$

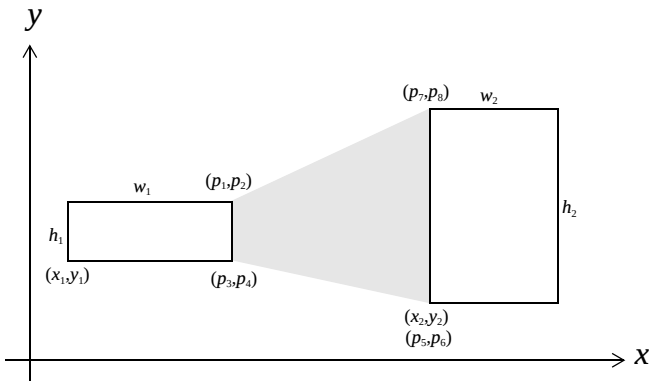

Figure 3: Application of **Bonding** to two rectangles

The values required for the constraint of an operation are computed from the operand solids by the interfaces. For instance, in Example 2.7, **edge**-interfaces **rightSide** and **leftSide** are used to compute the bonding edges of the two rectangles used by the operation for fusing the two rectangles together. However, it would be easier if the solids explicitly provided the information required by the constraint of an operation. That is, an inter-

face is a simple mapping of the vector values used for the creation of the solid on to a selection of those values. When the definition of a solid includes all the variables required by an interface of an operation in the list of its input values, the solid is said to expose that interface, captured by the following definition modified from its counterpart in [6] to account properly for validity.

Definition 2.14: If Φ is a solid in n variables, **L** is a selector of size k of some operation, and ϕ is an **L**-interface to Φ, Φ is said to *expose* ϕ iff there exists a sequence $p_1,\ldots,p_k$ of variables of Φ, such that for every $y \in$ scope(Φ), for all i ($1 \le i \le k$) $\phi(y)_i = y_{p_i}$. Each of the variables p_i is said to be *required* by ϕ. The variables $p_1,\ldots,p_k$ are not necessarily distinct, and the sequence $p_1,\ldots,p_k$ is not necessarily unique.

3 Sample Look

As we discussed earlier, one of the solid modeler's requirements arising from the LSD debugger is support for a sample look for a solid: that is, a visual representation of some member of the solid's image. Providing a sample look for a predefined component can be achieved by assigning sample values to the variables that define the geometry and appearance of the corresponding solid; for example, size, position, orientation, colour, and possibly material. These values can then be used by the solid modeler to instantiate the designated member of the family of objects corresponding to the solid, capturing the essential visual aspects of the family. The sample look for a family of objects is only a visual aid for tracing the execution of an LSD assembly when it is executed in LSD's debugger environment. Note that the sample values chosen are of no logical significance.

The sample values for the variables of a predefined solid could be assigned by the designer of the solid and later be overridden by the programmer in order to customise the look. This conveniently solves the sample look problem for predefined solids and also gives the programmer the option to change sample looks on a case-by-case basis. However, it is not clear how such sample values should be selected or refined once an operation is applied to operand solids. In other words, although the above mentioned approach is useful for preassembled solids, it will not serve for solids assembled during execution. For instance, when a variable is instantiated to a value different from its sample value, or when a variable is unified with or constrained to another variable, the predefined sample value must

be invalidated. Consider an example where two planar solids are to be attached with a bonding operation as illustrated in Figure 4(a). The first operand is a solid in five variables defining a square with fixed size and anchored in place defined as

$$\textbf{Base}(x_1,y_1,x_2,y_2,a) = [\, x_1 = .4 \,,\, y_1 = .2 \,,\, a = .2 \,,\, x_2 = x_1 \,,\, y_2 = y_1 + a \,]$$
$$\{\, (x,y) \mid x,y \in \mathbf{R} \,,\, x_1 - a \le x \le x_1 \,,\, y_1 \le y \le y_1 + a \,\}.$$

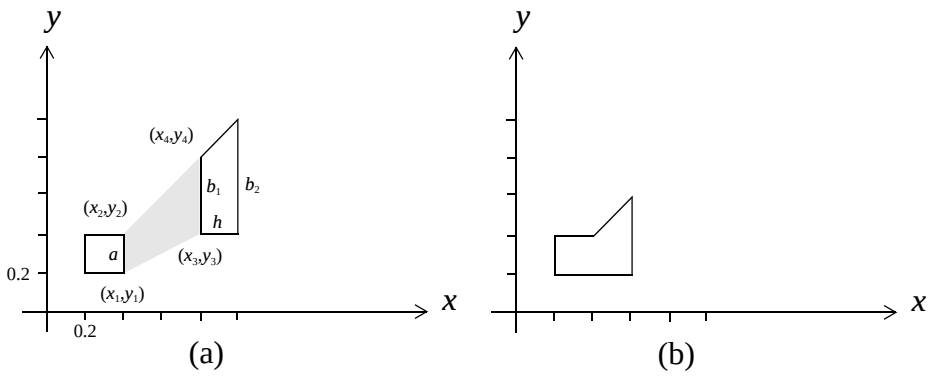

Figure 4: Sample looks of two trapezoids.

In this definition, the formula anchors the reference point of the square at (.4, .2) and fixes its size at .2 while the remainder defines a set of points comprising a square. Note that **Base** is a very restricted solid which could be expressed as a constant function. However, we find the above definition useful to describe the sample look computation process.

The second operand is a completely free right trapezoid defined by

$$\textbf{RightTrapezoid}(x_3,y_3,x_4,y_4,b_1,b_2,h) = [\, x_3 = x_4 \,,\, b_1 = y_4 - y_3 \,,\, b_1 < b_2 \le 2b_1 \,,\, 0 < h \,]$$
$$\{\, (x,y) \mid x,y \in \mathbf{R} \,,\, x_3 \le x \le x_3 + h \,,\, y_3 \le y \le \mathit{line} \,\}.$$

Note that the scope of **RightTrapezoid** is constrained such that the long base of **RightTrapezoid** cannot be greater than twice the length of the short base. In this definition, *line* is the expression representing the long leg of the trapezoid defined by the line that passes through the points (x_3,y_3+b_1) and (x_3+h,y_3+b_2), that is;

$$\mathit{line} = y_3 + b_1 + (x - x_3)(b_2 - b_1)/h$$

In Figure 4(a), **Base** is represented by its actual appearance, so its sample values are x_1= .4, y_1= .2, x_2= .4, y_2= .4, and a= .2, while **RightTrapezoid** is represented by a sample look defined by the sample values x_3= .8, y_3= .4, x_4= .8, y_4= .8, b_1= .4, b_2= .6, and h= .2.

Clearly, after the application of the bonding operation to **Base** and **RightTrapezoid**, the short base of **RightTrapezoid** will be constrained to the size of **Base** which is .2, different from the original sample value of b_1= .4. Consequently the sample value of the long base, b_2, of **RightTrapezoid** is invalidated.

In this section, we consider the problem of computing values with which to generate a sample look for an object resulting from the application of an operation to its operands.

One solution is to prohibit the variables of a solid from being dependant, thereby ensuring that changes in one variable will not affect the others. Since the lack of dependencies between the variables of a solid means that the scope of the solid is the cartesian product of subsets of **R**, one for each variable of the solid, the sample value for a particular variable can be chosen simply by selecting a value from the corresponding subset. However, this would put a major limitation on the solid modeler, severely limiting the set of representable objects. Instead, we introduce the notion of factoring to provide for the automatic computation of a sample look. As we will show later, factoring has other implications as well. For example, it can be used to restructure the representation of a solid in LSD so that it exposes different sets of interfaces.

4 Factoring

A geometric kernel that provides solid modeling for LSD, as part of its functionality, rejects any list of values that would result in creating an impossible object. Therefore, the definition of a solid as a partial function is a reasonable reflection of how geometric kernels operate. Although this responsibility for avoiding the creation of impossible objects could entirely be achieved by a chosen geometric kernel, there are other reasons for knowing the dependencies between the values passed to the kernel:

- computing values for a sample look, as discussed above
- finding parameter sets, and
- finding equivalent solids which expose required interfaces.

The example in the previous section required such information about the dependencies between variables, and expressing the solids using the notation used in Definition 2.10 made that information explicit.

In order to make the information about the dependencies between variables available, we extend the formal model of design space and solids so that the generative aspects of the employed kernel can be separated from the expression of the dependencies between values that are used by the kernel for creating instances of objects. This concept, called *factoring*, is used to express the dependencies between variables of a solid in a wrapper around a solid modeling kernel.

Definition 4.1: If Φ is a solid in n variables over D, a *factoring* of Φ is a quintuple $(x,y,z,\mathbf{C},\Psi)$ where

- x, y, and z are disjoint sequences of distinct variables of lengths n, m and k, respectively, called the *input*, *internal* and *geometric* variables, respectively. **(*a*)**
- $\mathbf{C}$ is a set of formulae, called the *constraint*, such that $\text{var}(\mathbf{C})=x \bullet y \bullet z$. **(*b*)**
- $\forall s \in z$, s has only one occurrence in $\mathbf{C}$ and there exists a unique equality $\mathbf{E} \in \mathbf{C}$ such that $\text{var}(\mathbf{E})=\{s\}$ or $\text{var}(\mathbf{E})=\{s,l\}$ and $l \in x \cup y$ **(*c*)**
- Ψ is a solid in k variables in D. **(*d*)**
- $\forall u \in \mathbf{R}^n$, if $u \in \text{scope}(\Phi)$, then $\exists\, v \in \mathbf{R}^m$ and $\exists\, w \in \mathbf{R}^k$ such that $\mathbf{C}_{x,y,z}[u,v,w]$ is true. **(*validity*)**
- $\forall u \in \mathbf{R}^n$, $\forall v \in \mathbf{R}^m$, $\forall w \in \mathbf{R}^k$, if $\mathbf{C}_{x,y,z}[u,v,w]$ is true, then $u \in \text{scope}(\Phi)$ iff $w \in \text{scope}(\Psi)$. **(*conformity*)**
- $\forall u \in \mathbf{R}^n$, $\forall v \in \mathbf{R}^m$, $\forall w \in \mathbf{R}^k$, if $u \in \text{scope}(\Phi)$ and $\mathbf{C}_{x,y,z}[u,v,w]$ is true, then $\Phi(u)=\Psi(w)$. **(*identity*)**

The factoring is said to be *perfect* if the following condition also holds:

- $\forall u \in \mathbf{R}^n$, $\forall v \in \mathbf{R}^m$, $\forall w \in \mathbf{R}^k$, if $\mathbf{C}_{x,y,z}[u,v,w]$ is true, then $u \in \text{scope}(\Phi)$. **(*perfection*)**

The intuition behind this definition is that values for the variables of a solid are checked for consistency by the constraint. Once consistency is ensured, Ψ is applied to the values computed by the constraint for its variables to obtain the desired object. An important property of a perfect fac-

toring is that if **C** approves of the values provided for the input variables, corresponding values for the geometric variables are guaranteed to be acceptable to the partial function Ψ.

The four conditions *validity*, *perfection*, *identity*, and *conformity* are separated in order to emphasise their roles in the definition of a factoring. The validity condition indicates that if a vector of values is in the scope of a solid Φ, then this vector can be extended by adding values for the internal and geometric variables of the factoring in such a way that the extended vector of values satisfies **C**. This condition ensures that a factoring does not reject a legitimate list of values. The perfection condition says that **C** can be used to determine if a list of values is indeed in the scope of Φ. This is the property of a factoring that will help us to compute a sample look for an assembled object. The identity condition ensures that a factoring of a solid corresponds to the same family of objects as Φ. An obvious consequence of the identity condition is that the image of Φ is a subset of the image of Ψ. The conformity condition indicates that once a vector of values satisfies **C**, the lists of input and geometric values both are in the scopes of Φ and Ψ, respectively, or both are not. The practical implication of this condition is that if a vector of values satisfies **C**, and Ψ successfully generates an object, the object is in fact in the image of Φ.

A solid can have more than one factoring.

Example 4.2: Consider the solid Ψ generated in Example 2.13, defined by

$$\Psi(x_1,y_1,w_1,h_1,x_2,y_2,w_2,h_2) = [\, x_1+w_1=x_2\ ,\ y_1+h_1=y_2+h_2\ ,\ y_1=y_2\ ]$$
$$\textbf{fuse}(\textbf{Rectangle}(x_1,y_1,w_1,h_1),\textbf{Rectangle}(x_2,y_2,w_2,h_2)).$$

Clearly, $((x_1,y_1,w_1,h_1,x_2,y_2,w_2,h_2),\ (),\ (z_1,\ldots,z_8),\ \mathbf{C},\ \textbf{fuse}(\textbf{Rectangle}(z_1,z_2,z_3,z_4),\ \textbf{Rectangle}(z_5,z_6,z_7,z_8)))$ is a factoring of Ψ where

$$\mathbf{C} = \{\ x_1+w_1=x_2\ ,\ y_1+h_1=y_2+h_2\ ,\ y_1=y_2\ ,$$
$$x_1=z_1\ ,\ y_1=z_2\ ,\ w_1=z_3\ ,\ h_1=z_4\ ,\ x_2=z_5\ ,\ y_2=z_6\ ,\ w_2=z_7\ ,\ h_2=z_8\ \}$$

and the last item in the tuple is an expression defining a partial function which is a solid.

Another factoring of Ψ is

$((x_1,y_1,w_1,h_1,x_2,y_2,w_2,h_2),(),(z_1,\ldots,z_8),\mathbf{X},\textbf{fuse}(\textbf{Rectangle}(z_1,z_2,z_3,z_4),\textbf{Rectangle}(z_5,z_6,z_7,z_8)))$ where

$$\mathbf{X} = \{\ x_1+w_1=x_2\ ,\ y_1+h_1=y_2+h_2\ ,\ y_1=y_2\ ,\ 0\le w_1\ ,\ 0\le h_1\ ,\ 0\le w_2\ ,\ 0\le h_2\ ,$$

$x_1=z_1$, $y_1=z_2$, $w_1=z_3$, $h_1=z_4$, $x_2=z_5$, $y_2=z_6$, $w_2=z_7$, $h_2=z_8$ }.

Note that the latter factoring includes some formulae that guarantee that (x_1,y_1,w_1,h_1), $(x_2,y_2,w_2,h_2) \in$ scope(**Rectangle**), making this a perfect factoring.

We now prove two useful results that relate factorings of solids and the application of operations.

Lemma 4.3: If D is a design space, $\otimes=((a_1,\ldots,a_n),(p_1\bullet\ldots\bullet p_n),F,L,\mathbf{C})$ is an n-ary operation in D, $\Phi_1,\ldots,\Phi_n$ are solids in D, Ψ is the solid resulting from the application of $\otimes$ to $\Phi_1,\ldots,\Phi_n$ via interfaces $\phi_1,\ldots,\phi_n$, and $(x_i,y_i,z_i,\mathbf{C}_i,\Psi_i)$ is a factoring of Φ_i for each i $(1\le i\le n)$, then $(x,y,z,\mathbf{X},\Delta)$ is a factoring of Ψ where x is $x_1\bullet\ldots\bullet x_n$, y is $y_1\bullet\ldots\bullet y_n$, z is $z_1\bullet\ldots\bullet z_n$, and

- $\mathbf{X} = \mathbf{C}_{p_1,\ldots,p_n}[\phi_1(x_1),\ldots,\phi_n(x_n)] \cup \mathbf{C}_1 \cup \ldots \cup \mathbf{C}_n$, and
- $\Delta(z) = F(\Psi_1(z_1),\ldots,\Psi_n(z_n))$

Proof. We assume that for all i and j, where $j \neq i$, var($\mathbf{C}_i$) and var($\mathbf{C}_j$) are disjoint.

Let $t=|x|$, $m=|y|$, and $k=|z|$. If $u \in \text{þ}\mathbf{R}^t$, then u can be written as $u_1\bullet\ldots\bullet u_n$ where $|u_i|=|x_i|$ for each i $(1 \le i \le n)$. Elements of $\mathbf{R}^m$ and $\mathbf{R}^k$ can be similarly decomposed.

(a): Since for each i $(1 \le i \le n)$, $(x_i,y_i,z_i,\mathbf{C}_i,\Psi_i)$ is a factoring of Φ_i, x_i, y_i, and z_i are disjoint sequences of distinct variables, therefore, because of the above assumption, x, y, and z are disjoint sequences of distinct variables.

(b): Clearly var($\mathbf{X}$)$=x\bullet y\bullet z$.

(c): Suppose $s \in z$, then $s \in z_i$ for some i $(1\le i\le n)$. Because of the definition of factoring $s \notin x_i$ and because of our assumption that var($\mathbf{C}_i$) and var($\mathbf{C}_j$) are disjoint for all $j \neq i$, $s \notin x_j$. Since the only variables that occur in $\mathbf{C}_{p_1,\ldots,p_n}[\phi_1(x_1),\ldots,\phi_n(x_n)]$ are those in $x_1,\ldots,x_n$, s does not occur in $\mathbf{C}_{p_1,\ldots,p_n}[\phi_1(x_1),\ldots,\phi_n(x_n)]$. Therefore, because of the definition of factoring, each $s \in z$ occurs in a unique equality $\mathbf{E} \in \mathbf{X}$ such that var($\mathbf{E}$)$=\{s\}$ or var($\mathbf{E}$)$=\{s,l\}$ and $l \in x \cup y$.

(d): Δ is a partial function from $\mathbf{R}^k$ to D. Suppose that for some w $\in$scope(Δ), $(e,g),(e,h) \in \Delta(w)$. Then $w_i \in$ scope(Ψ_i) for each i $(1\le i\le n)$, $(\Psi_1(w_1),\ldots,\Psi_n(w_n)) \in$scope($F$) and (e,g), $(e,h) \in F(\Psi_1(w_1),\ldots,\Psi_n(w_n))$ so by the condition on F imposed by the definition of operation, $g=h$. Hence, Δ is a solid in D.

It remains to show that the validity, conformity and identity conditions are satisfied.

(*Validity*): Suppose that $u \in \mathbf{R}^t$ and $u \in \text{scope}(\Psi)$, then according to Definition 2.10, $\mathbf{C}_{p_1,\ldots,p_n}[\phi_1(u_1),\ldots,\phi_n(u_n)]$ is true, and according to Lemma 2.11, $u_i \in \text{scope}(\Phi_i)$ for each i $(1 \le i \le n)$. Since $(x_i,y_i,z_i,\mathbf{C}_i,\Psi_i)$ is a factoring of Φ_i, and $u_i \in \text{scope}(\Phi_i)$, by the validity condition we can conclude that there exist v_i and w_i such that $(\mathbf{C}_i)_{x_i,y_i,z_i}[u_i,v_i,w_i]$ is true for each i $(1 \le i \le n)$. Therefore, $\mathbf{X}_{x,y,z}[u,v,w]$ is true where v is $v_1 \bullet \ldots \bullet v_n$, and w is $w_1 \bullet \ldots \bullet w_n$, proving that the validity condition holds.

(*Conformity*): Suppose that $u \in \mathbf{R}^t$, $v \in \mathbf{R}^m$, $w \in \mathbf{R}^k$, and $\mathbf{X}_{x,y,z}[u,v,w]$ is true.

If $u \in \text{scope}(\Psi)$, then according to Lemma 2.11, $u_i \in \text{scope}(\Phi_i)$ for each i $(1 \le i \le n)$, and $(\Phi_1(u_1),\ldots,\Phi_n(u_n)) \in \text{scope}(F)$. Since $\mathbf{X}_{x,y,z}[u,v,w]$ is true, then $(\mathbf{C}_i)_{x_i,y_i,z_i}[u_i,v_i,w_i]$ is true for each i $(1 \le i \le n)$, and since $u_i \in \text{scope}(\Phi_i)$, by the conformity condition, $w_i \in \text{scope}(\Psi_i)$ and by the identity condition $\Phi_i(u_i)=\Psi_i(w_i)$, for each i $(1 \le i \le n)$. Since $w_i \in \text{scope}(\Psi_i)$ for each i $(1 \le i \le n)$, and $(\Psi_1(w_1),\ldots,\Psi_n(w_n)) \in \text{scope}(F)$, $w \in \text{scope}(\Delta)$.

If $w \in \text{scope}(\Delta)$, then $w_i \in \text{scope}(\Psi_i)$ for each i $(1 \le i \le n)$, and $(\Psi_1(w_1),\ldots,\Psi_n(w_n)) \in \text{scope}(F)$. Since $(\mathbf{C}_i)_{x_i,y_i,z_i}[u_i,v_i,w_i]$ is true and $w_i \in \text{scope}(\Psi_i)$ for each i $(1 \le i \le n)$, by the conformity condition on the factoring $(x_i,y_i,z_i,\mathbf{C}_i,\Psi_i)$ of Φ_i, $u_i \in \text{scope}(\Phi_i)$, and by the identity condition $\Phi_i(u_i)=\Psi_i(w_i)$, for each i $(1 \le i \le n)$. Therefore, $(\Phi_1(u_1),\ldots,\Phi_n(u_n)) \in \text{scope}(F)$. Since $u_i \in \text{scope}(\Phi_i)$ for each i $(1 \le i \le n)$, and $(\Phi_1(u_1),\ldots,\Phi_n(u_n)) \in \text{scope}(F)$, u is in the scope of the partial function defined by the expression $F(\Phi_1(x_1),\ldots,\Phi_n(x_n))$. Since $\mathbf{C}_{p_1,\ldots,p_n}[\phi_1(u_1),\ldots,\phi_n(u_n)]$ is true, and u is in the scope of the partial function defined by the expression $F(\Phi_1(x_1),\ldots,\Phi_n(x_n))$, $u \in \text{scope}(\Psi)$, establishing the conformity condition.

(*Identity*): Suppose that $u \in \mathbf{R}^t$, $v \in \mathbf{R}^m$, $w \in \mathbf{R}^k$, $u \in \text{scope}(\Psi)$ and $\mathbf{X}_{x,y,z}[u,v,w]$ is true. Since $u \in \text{scope}(\Psi)$, then u is in the scope of the partial function defined by the expression $F(\Phi_1(u_1),\ldots,\Phi_n(u_n))$. Therefore, $u_i \in \text{scope}(\Phi_i)$ for each i $(1 \le i \le n)$. Since $\mathbf{X}_{x \bullet y \bullet z}[u \bullet v \bullet w]$ is true, $(\mathbf{C}_i)_{x_i,y_i,z_i}[u_i,v_i,w_i]$ is true for each i $(1 \le i \le n)$, and since $(x_i,y_i,z_i,\mathbf{C}_i,\Psi_i)$ is a factoring of Φ_i, and $u_i \in \text{scope}(\Phi_i)$, by the conformity condition $w_i \in \text{scope}(\Psi_i)$ and by the identity condition $\Phi_i(u_i)=\Psi_i(w_i)$ for each i $(1 \le i \le n)$. Therefore,

$$\begin{aligned}\Psi(u) &= F(\Phi_1(u_1),\ldots,\Phi_n(u_n)) \\ &= F(\Psi_1(w_1),\ldots,\Psi_n(w_n))\end{aligned}$$

$$= \Delta(w)$$

establishing the identity condition. □

Lemma 4.4: If in Lemma 4.3, $(x_i,y_i,z_i,\mathbf{C}_i,\Psi_i)$ is a perfect factoring of Φ_i for each i $(1 \le i \le n)$, and $\mathbf{C}_{p_1,\ldots,p_n}[\phi_1(u_1),\ldots,\phi_n(u_n)]$ implies that $(\Phi_1(u_1),\ldots,\Phi_n(u_n)) \in \text{scope}(F)$, then $(x,y,z,\mathbf{X},\Delta)$ is a perfect factoring of Ψ.

Proof. Since $\mathbf{X}_{x,y,z}[u,v,w]$ is true $(\mathbf{C}_i)_{x_i,y_i,z_i}[u_i,v_i,w_i]$ is true for each i $(1 \le i \le n)$, so for each i, $u_i \in \text{scope}(\Phi_i)$ since $(x_i,y_i,z_i,\mathbf{C}_i,\Psi_i)$ is a perfect factoring of Φ_i, and $\Phi_i(u_i)=\Psi_i(w_i)$ by the identity condition. Also, since $\mathbf{X}_{x,y,z}[u,v,w]$ is true, $\mathbf{C}_{p_1,\ldots,p_n}[\phi_1(u_1),\ldots,\phi_n(u_n)]$ is true. Therefore, $(\Phi_1(u_1),\ldots,\Phi_n(u_n)) \in \text{scope}(F)$, so that $(\Psi_1(w_1),\ldots,\Psi_n(w_n)) \in \text{scope}(F)$. Hence $w \in \text{scope}(\Delta)$ and since $\mathbf{X}_{x,y,z}[u,v,w]$ is true, $u \in \text{scope}(\Psi)$ by the conformity condition. □

In practice , the solid inside a factoring could be a function call to a geometric kernel for creating an object. The constraint in a factoring, realised as a wrapper around the chosen kernel, reveals some of the dependencies between the variables of the factored solid. If a factoring is perfect, the constraint of the factoring adds further restrictions to the scope of the solid inside the factoring such that once a vector of values is consistent in the constraint, then that vector is guaranteed to produce a valid object, facilitating the automatic generation of sample values for the sample look of a solid.

In Example 4.2, we showed how a factoring can be constructed for a solid defined by a partial function. Now we will show how, given a factoring for a solid, a partial function defining the factored solid can be constructed. Let $(x,y,z,\mathbf{C},\Psi)$ be a factoring of Φ, and $k=|z|$. Then according to the definition of a factoring, for each i $(1 \le i \le k)$, z_i occurs in a unique equality $\mathbf{E}_i$ in $\mathbf{C}$. If we rearrange $\mathbf{E}_i$ into the form $e_i=z_i$ where e_i is an expression, then Φ can be expressed as $\Phi(x)= [\mathbf{C}^{\tilde{z}}]\ \Psi(e_1,\ldots,e_k)$.

The sample values for the variables of predefined solids are chosen by the designer as discussed earlier, and would be a solution to the constraint of any perfect factoring of the corresponding solid. When an operation is applied to factorings of its operand solids, a new set of sample values must be computed for the sample look of the new solid, represented by the factoring computed as shown in the above lemmas, such that all the constraints in the factoring are satisfied. Note that the constraints may originate from the factorings of operand solids or from the constraint of the opera-

tion. Once a list of sample values are computed, they are passed to a renderer for drawing. The sample value generation process will have no effect on the internal state of the geometric kernel and does not force the kernel to actually instantiate an object. Sample values are used only by a renderer which could be completely independent from the kernel.

Clearly, an algorithm that determines new sample values for the variables will operate according to certain criteria. For example, one criterion could be to keep the new values for the free variables as close as possible to their original values. Hence, the problem of finding a sample look is reduced to a constraint satisfaction problem.

An interesting property of a factoring with respect to sample values is that only the geometric variables need be assigned sample values, as they are the only values used by geometric kernels.

Now, using the notion of factoring, we revisit the previous example of bonding **Base** and **RightTrapezoid**.

Base can be factored to $(u_1,v_1,w_1,\mathbf{C}_1,\Phi_1)$ where u_1 is (x_1,y_1,x_2,y_2,a), v_1 is $()$, and w_1 is (z_1,z_2,z_3) and

$\mathbf{C}_1 = \{\ x_2=x_1\ ,\ y_2=y_1+a\ ,\ x_1=.4\ ,\ y_1=.2\ ,\ a=.2\ ,\ z_1=x_1\ ,\ z_2=y_1\ ,\ z_3=a\ \}$ and

$\Phi_1(w_1)=[\ \mathbf{C}_1\ ]\ \{\ (x,y)\ |\ x,y\ \in \mathbf{R}\ ,\ z_1-z_3 \le x \le z_1\ ,\ z_2 \le y \le z_2+z_3\ \}$. The sample values for **Base** are given as $z_1=.4$, $z_2=.2$, and $z_3=.2$.

RightTrapezoid is factored to $(u_2,v_2,w_2,\mathbf{C}_2,\Phi_2)$ where u_2 is $(x_3,y_3,x_4,y_4,b_1,b_2,h)$, v_2 is $()$, and w_2 is (z_4,z_5,z_6,z_7,z_8) and

$\mathbf{C}_2 = \{\ x_3=x_4\ ,\ y_4-y_3=b_1\ ,\ b_1 < b_2 \le 2b_1\ ,\ 0 < h\ ,$

$z_4=x_3\ ,\ z_5=y_3\ ,\ z_6=b_1\ ,\ z_7=b_2\ ,\ z_8=h\ \}$

and

$\Phi_2(w_2) = [\ \mathbf{C}_2]\ \{\ (x,y)\ |\ x,y\ \in \mathbf{R}\ ,\ z_4 \le x \le z_4+z_8\ ,$

$z_5 \le y \le z_5+z_6+(x-z_4)(z_7-z_6)/z_8\ \}$

and the sample values are $z_4=.8$, $z_5=.4$, $z_6=.4$, $z_7=.6$, and $z_8=.2$.

Recall that the **Bonding** operation in a 2D design space is defined by $((a_1,a_2),(p_1,p_2,p_3,p_4,p_5,p_6,p_7,p_8),\mathbf{fuse},\{\mathbf{edge1},\mathbf{edge2}\},\mathbf{bond})$, where $\mathbf{fuse}(x,y)=[\ x \cap y=l\ ](x \cup y)$, such that l is a line, $\mathbf{bond}=\{p_1=p_7\ ,\ p_2=p_8\ ,\ p_3=p_5\ ,\ p_4=p_6\}$, $\mathbf{edge}1=\mathbf{edge}_{a,b,c,d,e}[a_1,p_1,p_2,p_3,p_4]$, $\mathbf{edge}2=\mathbf{edge}_{a,b,c,d,e}[a_2,p_5,p_6,p_7,p_8]$, and $\mathbf{edge}(a,b,c,d,e)$ is true iff (b,c) and (d,e) are points in a, every point on the line

between them is in a, every point to the right of this line is in a, and every point to the left of this line is not in a.

Also let **squareSide**$(x_1,y_1,x_2,y_2,a)=(x_2,y_2,x_1,y_1)$ and **trapSide**$(x_3,y_3,x_4,y_4,b_1,b_2,h)=(x_3,y_3,x_4,y_4)$ be **edge**-interfaces to **Base** and **RightTrapezoid**, respectively. Note that **Base** exposes **squareSide** and **RightTrapezoid** exposes **trapSide**. The solid resulting from execution of the **Bonding** operation can be factored to $(x,y,z,\mathbf{X},\Psi)$ where $x=u_1 \bullet u_2$, $y=v_1 \bullet v_2$, $z=w_1 \bullet w_2$ and

$$\begin{aligned}
\mathbf{X} &= \mathbf{C}_1 \cup \mathbf{C_2} \cup \mathbf{bond}_{p_1,\ldots,p_8}[x_2,y_2,x_1,y_1,x_3,y_3,x_4,y_4] \\
&= \{\, x_2=x_1\ ,\ y_2=y_1+a\ ,\ x_1=.4\ ,\ y_1=.2\ ,\ a=.2\ ,\ z_1=x_1\ ,\ z_2=y_1\ ,\ z_3=a, \\
&\qquad\qquad x_3=x_4\ ,\ y_4-y_3=b_1\ ,\ b_1 < b_2 \le 2b_1\ ,\ 0 < h\ , \\
&\qquad\qquad z_4=x_3\ ,\ z_5=y_3\ ,\ z_6=b_1\ ,\ z_7=b_2\ ,\ z_8=h, \\
&\qquad\qquad x_2=x_4\ ,\ y_2=y_4\ ,\ x_1=x_3\ ,\ y_1=y_3\ \} \\
&= \{\, x_1=x_2=x_3=x_4=.4\ ,\ y_4=y_2=y_1+a\ ,\ y_3=y_1=.2\ ,\ a=.2\ , \\
&\qquad\qquad y_4-y_3=b_1\ ,\ b_1 < b_2 \le 2b_1\ ,\ 0 < h\ , \\
&\qquad\qquad z_1=x_1\ ,\ z_2=y_1\ ,\ z_3=a\ ,\ z_4=x_3\ ,\ z_5=y_3\ , \\
&\qquad\qquad z_6=b_1\ ,\ z_7=b_2\ ,\ z_8=h\ \}
\end{aligned}$$

and

$$\Psi(z) = [\ \mathbf{X}\ ]\ \mathbf{fuse}(\Phi_1(z_1,z_2,z_3,z_4),\Phi_2(z_5,z_6,z_7,z_8)).$$

Sample values for some of the geometric variables are fixed, that is $z_1=.4$, $z_2=.2$, $z_3=.2$, $z_4=.4$, $z_5=.2$, and $z_6=.2$. The constraints involving the remaining geometric variables simplify to two equalities $z_7=b_2$, and $z_8=h$, and three inequalities $.2 < b_2 \le .4$, $0 < h$. A new sample value for z_7 can be assigned depending on how the constraint solver operates. For example, if the criterion for the assignment of new sample values is to keep them as close as possible to the sample values they replace, the new sample value for z_7 would be .4. However, in the absence of this criterion any value for z_7 in the solution of **X** would do. Lastly, since neither z_8 nor h depends on any other variable in **X**, the previous sample value of z_8 could be preserved. In summary, the sample values for the sample look of the new solid are $z_1=.4$, $z_2=.2$, $z_3=.2$, $z_4=.4$, $z_5=.2$, $z_6=.2$, $z_7=.4$, and $z_8=.2$, as illustrated in Figure 4 (b).

Example 4.5: To further illustrate the computation of sample values for a new solid, let $(u,v,w,\mathbf{C},\Phi)$ be a factoring of the solid **Disk** in Example 2.5 where u is (x_1,y_1,z_1), v is $()$, w is (b_1,c_1,r_1),

$$\mathbf{C} = \{\ 0 \le z_1\ ,\ b_1 = x_1\ ,\ c_1 = y_1\ ,\ r_1 = z_1\ \},\ \text{and}$$

$$\Phi(b_1,c_1,r_1) = [\ 0 \le r_1\ ]\ \{\ (x,y)\ |\ x,y \in \mathbf{R}\ \text{and}\ (x-b_1)^2+(y-c_1)^2 \le r_1^{\,2}\ \}.$$

Also let the sample values for the sample look of **Disk** be $b_1=0$, $c_1=0$, and $r_1=1$. An operation named **Punch** that punches a hole represented by one **Disk** at the centre of another **Disk** is defined by $((a_1,a_2),(x_1,y_1,z_1,x_2,y_2,z_2),p-q,\{\mathbf{centre1},\mathbf{centre2}\},\mathbf{align})$ where $\mathbf{align} = \{\ x_1 = x_2\ ,\ y_1 = y_2\ ,\ z_2 < z_1\ \}$ and

$$\mathbf{centre1}(a_1,x_1,y_1,z_1) = \mathbf{centre}_{a,b,c,r}[a_1,x_1,y_1,z_1],$$

$$\mathbf{centre}2(a_2,x_2,y_2,z_2) = \mathbf{centre}_{a,b,c,r}[a_2,x_2,y_2,z_2],$$

and $\mathbf{centre}(a,b,c,r)$ is true iff a is a **Disk** with centre (b,c) and radius r. The solid resulting from the application of **Punch** to two disks $\mathbf{Disk_1}$ and $\mathbf{Disk_2}$ with similar factorings $((x_1,y_1,z_1),(),(b_1,c_1,r_1),\mathbf{C},\Phi)$ and $((x_2,y_2,z_2),(),(b_2,c_2,r_2),\mathbf{C},\Phi)$ and sample values $b_1=0$, $c_1=0$, $r_1=1$ and $b_2=0$, $c_2=0$, $r_2=1$, respectively, called **Ring**, has a factoring $((x_1,y_1,z_1,x_2,y_2,z_2),(),(b_1,c_1,r_1,b_2,c_2,r_2),\mathbf{X},\Psi)$ where

$$\mathbf{X} = \{\ 0 \le z_1\ ,\ b_1 = x_1\ ,\ c_1 = y_1\ ,\ r_1 = z_1\ ,\ 0 \le z_2\ ,\ b_2 = x_2\ ,$$
$$c_2 = y_2\ ,\ r_2 = z_2\ ,\ x_1 = x_2\ ,\ y_1 = y_2\ ,\ z_2 < z_1\ \}$$

and

$$\Psi(b_1,c_1,r_1,b_2,c_2,r_2) = [\ \mathbf{X}\ ]\ \Phi(b_1,c_1,r_1) - \Phi(b_2,c_2,r_2).$$

Since none of the variables in **X** are bound to constants, no immediate conclusion about the new sample values for the variables can be made. Fortunately in this case, most of the original sample values for the variables are consistent with the constraints in **C** which yields the following new sample values $b_1=0$, $c_1=0$, $b_2=0$, and $c_2=0$. The original sample values for r_1 and r_2, however, are not consistent with the inequality $z_2 < z_1$. Again, the constraint solver's criteria will determine how two values for r_1 and r_2 are picked to satisfy the constraints. A possible choice is $r_1=1$ and $r_2= .5$.

Finding new sample values for the sample look of a solid resulting from the application of an operation is performed only when an LSD program executed in **Debug** mode requires rendering of partially assembled objects for the animation of an execution of a program, and for displaying solids

resulting from geometric computations. In **Run** mode, sample look computation is required only at the very last step of the execution when any resulting free assembled object need to be rendered.

Relaxed Factoring

The intuition underlying Definition 4.1 is that if all the conditions that constrain the variables of a solid are extracted from the geometric kernel, a list of sample values computed by a constraint solver working in conjunction with LSD is guaranteed to generate a member of the image of the solid, that is, an object that can be constructed in the design space. This is captured in Definition 4.1 by the perfection condition. However, in practice, the perfection condition is too strong to be easily attainable. For example, the bonding operation used for attaching right trapezoid and box in our earlier example, is generic enough to be used to attach any two solids if they expose the necessary interfaces. But the condition in the bonding operation does not check that operand solids do not overlap somewhere other than line along which they are bonded. Hence, although the two solids could be bonded, the geometric kernel may still reject the resulting object as illustrated in Example 2.12.

A factoring which is not perfect, called a *relaxed factoring*, is still useful in practice. For example, the solver and the geometric kernel might cooperate to find a list of sample values on which they both can agree. When a list of values consistent in **C** is not in the scope of Ψ, the geometric kernel could report a failure back to the solver and request a different list of values. Such a dialogue between the geometric kernel and the constraint solver would ensure that the selected sample values are both consistent in **C** and in the scope of Ψ. Several negotiations between the solver and the kernel might be required.

Sample Value Assignment Criteria

As we have mentioned, different criteria may be used for the assignment of sample values to the uninstantiated variables in the constraint store used for the computation of sample values. The aim of a sample value assignment criterion is to ensure that values selected for the uninstantiated variables in the constraint store are not only consistent, but also satisfy some additional conditions. This is to ensure that the sample value assignment process can account for certain cognitive factors that can be formulated in the assignment algorithm. For example, the sample value assignment algo-

rithm may introduce a new constraint for the assignment of values to a cylinder to make sure that the ratio of radius to height of a visual representation of a free cylinder are within some desired range. This may be achieved by a *weak constraint* defined by the designer of a predefined solid for the algorithm employed for the computation of its sample values. However, this should not in any way change the behaviour of an execution involving the solid. Such weak constraints are used only by the constraint solver computing sample values and do not imply any additional restrictions on the solid itself. This mechanism gives the designer of solids more influence over how the sample value assignment algorithm works.

In practice, the sample value assignment algorithm is more critical than it may appear at first. While the algorithm's aim should be to capture and reveal the essential and common visual aspects of a family of solids, this should not be done arbitrarily. The choice of sample values and the transition from one set of values to another as the result of the application of an operation should be made in such a way that the resulting visualisations do not imply any extra semantics beyond those inherent in the program. In other words, the algorithm should not mislead the user.

5 Summary

We have extended the formal model for solids proposed in [6]. We first identified an ambiguity in the interpretation of the empty set in the previous definitions noting that the empty set could signify either an invalid solid or one which happens to include no points in the space. To rectify the problem, we defined a solid as a partial function.

We raised and investigated the issue of sample looks for solids in LSD programs and defined the notion of factoring of a solid, applying it to the automatic computation of a sample look for a partially free solid. The intuition behind factoring is to extract the constraints that define how a solid can be configured so that we can manipulate them in various ways, for example to generate sample values for the parameters of a solid. We noted that although this may be an ideal situation, it is difficult to achieve in practice. To make the notion of factoring more practical we introduced relaxed factoring, in which some, but not necessarily all, constraints are identified. In this case, selecting sample values that are consistent with the constraint of the solid is likely to generate a valid sample look for the object, but this is not guaranteed. We discussed how, in an implementation, the geometric

kernel and the constraint solver might engage in a dialogue to iterate towards appropriate sample values.

Another important use of factoring is for computing reduced solids, a notion originally defined in [6] as a practical issue for simplifying solids. In [2] we have introduced the notion of reduced sets for sets of equalities and discussed some of their important properties. Then we applied this notion to solids and demonstrated how a reduced set could be computed from a factoring of that solid.

6 Acknowledgments

This work was partially supported by the Natural Sciences and Engineering Research Council of Canada via Discovery Grant OGP0000124.s

7 References

[1] Autodesk Inc., *Autodesk Inventor 5*, 2001.

[2] O. Banyasad, *Programming Parameterised Geometric Objects In a Visual Design Language*, Ph.D. dissertation, Dalhousie University, 2006.

[3] O. Banyasad, P. T. Cox, Integrating Design Synthesis and Assembly of Structured Objects in a Visual Design Language, *The Theory and Practice of Logic Programming,* 5(6), Cambridge University Press, pp. 601-621, 2005.

[4] Bentley Systems Inc., *MicroStation V8 2004 Edition User Guide*, 2004.

[5] J. Corney, *3D Modeling with the ACIS kernel and toolkit*, John Wiley & Sons Ltd., 1997.

[6] P.T. Cox, T. Smedley, A Formal Model for Parameterized Solids in a Visual Design Language, *Journal of Visual Languages and Computing*, Volume 11, Number 6, Academic Press, pp. 687-710, 2000.

[7] J. C. Nash, *Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation, 2nd ed.*, Bristol, England: Adam Hilger, 1990.

[8] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in FORTRAN: The Art of Scientific Computing, 2nd ed.*, Cambridge, England: Cambridge University Press, 1992.

[9] Shape Data Limited and Electronic Data Systems Corporation, *Parasolid v5.0 Programming Reference Manual*, Cambridge, England,1992.

[10] SolidWorks Corporation, *SolidWorks 2006 What's New*, 2006.

[11] Spatial Technology Inc., *ACIS Geometric Modeler*, Version 4.0, 1997.

[12] E. Süli, D. Mayers, *An Introduction to Numerical Analysis*, Cambridge University Press, 2003.