# Bag Context Shape Grammar Implementation: From Theory to Useable Software

Blessing Ogbuokiri[1] Mpho Raborife[2]

[1]School of Computer Science and Applied Mathematics University of the Witwatersrand, Johannesburg, ogbuokiriblessing@gmail.com
[2]Department of Applied Information Systems, University of Johannesburg, mraborife@uj.ac.za

Corresponding author: Blessing Ogbuokiri, ogbuokiriblessing@gmail.com

**Abstract.** Shape grammar implementation tools play an important role in the generation of designs. Most of the available tools were created to allow the application of shape grammar rules without restrictions. This is largely because shape grammars are context free, as such, the application of a rule is not controlled during derivation. This is also the reason some generate designs that have parts that are too small for the human eye to see. This work presents a tool that allows the addition of bag context to shape grammar rules such that it automatically allows when a rule should be applied based on a defined range. Bag context represents information that is not part of a developing design but instead evolves separately during a derivation. This helps to generate an infinite number of images that are similar but not identical. This tool could offer a wide range of application areas such as aiding the teaching of shape grammar or bag context shape grammar implementation in formal language classes or higher learning in general, a framework to support designers for the development of an improved bag context shape grammar interpreter tool, and others.

**Keywords:** Formal Language, Shape Grammar, Bag Context Grammar, Algorithm Design, Bag Context Shape Grammar Interpreter.
**DOI:** https://doi.org/10.14733/cadaps.2020.458-474

## 1 INTRODUCTION

A shape grammar is defined as a device that uses spatial rules to generate designs [26, 28]. Research into the implementation of shape grammars has been active since its development and spans across various disciplines, from architecture to engineering and art to product design [11, 3, 2]. However, despite some progress made, a number of these implementations have not satisfactorily reached their intended potential as compared to the theoretical developments of the shape grammar formalism [10]. This could be attributed to some challenges involved in the implementation of shape grammar systems [17].

A good deal of progress has been made by researchers in providing feasible solutions to the shape grammar implementation problem [27]. One such progress is the idea of implementing shape grammar systems in a software tool called shape grammar interpreter [28].

Shape grammar interpreter tools play an important role in the analysis, interpretation, and generation of designs, using formal rules [28, 12, 11]. A number of these tools have shown to be limited in their applications [15, 16, 18]. They do not have the power to generate regulated designs automatically [21]. Some do not allow for the control of the rules during the generation process [3]. The inability of a number of these tools to control the grammar rules could be the reason some generate images that have parts that are too small for the human eye to see [17, 21].

To date, there have been efforts to create a general shape grammar interpreter tool that can provide a lasting solution to the challenges of shape grammar implementation. Most of the available tools were either created for a specific purpose, have limited functionality or were created for a specific operating system [28, 2, 1, 5]. Some of these tools do not have the ability to control the generation process, such that the number of times a rule is applied can be controlled or regulated by the simple click of a button.

Therefore, we present a tool called bag context shape grammar interpreter (BCSGI). BCSGI allows users to generate designs in an interactive and regulated way using shape grammar rules. In summary, the contributions of this work are as follows:

- Bag context is added to shape grammar rules to automatically control when these rules can be applied based on a defined range.
- The improvement and implementation of the existing shape grammar interpretation algorithm [5].
- The generation of well-regulated designs or galleries using BCSGI.

BCSGI will be useful to lecturers and students for the teaching of shape grammars or bag context shape implementation in formal language classes. It will also be a framework to support designers for the development of an improved BCSGI tool, and many other application areas.

The remaining part of this paper is organized into relevant section. We begin by discussing background and related work in Section 2, followed by Section 3 which presents the definition of bag context shape grammars with an example. We present the BCSGI tool in Section 4. Finally, Section 5 is the conclusion.

## 2 Background and Related Work

In this section, we present an overview of context free grammars with emphasis on shape grammars. We also present bag context grammars and some works relevant to our type of study.

A context-free grammar is a system with which one can make an infinite set of objects or strings [14, 10]. One such grammar in this class is the shape grammar [10, 24]. A shape grammar is defined as a system that uses spatial rules to generate designs [5]. We give the formal definition of shape grammars in Section 2.1.

### 2.1 Shape Grammars

We define a new additive shape grammar class that can allow the use of any shape to be used to implement images during rendering after a derivation. By additive, we mean the application of a rule is done by building upon the axiom during the generative process. We give an example to demonstrate this notion. Firstly we present some notation and terms used in the rest of this paper.

#### 2.1.1 Terms and Notation

The symbol $\mathbb{N}$ represents the set of natural numbers $\{0, 1, 2, \ldots\}$. $\mathbb{N}_+$ represents the set $\{1, 2, \ldots\}$. For $k \in \mathbb{N}_+$, the set $\{1, 2, \ldots, k\}$ is denoted by $[k]$. $\mathbb{Z}$ represents the set of integers $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$, then $\mathbb{Z}^2 = \{(x, y) \mid x, y \in \mathbb{Z}\}$. $\mathbb{Z}_\infty$ represents $\mathbb{Z} \cup$ {-∞, ∞}. If $I = [k]$, then elements of $\mathbb{Z}^I_\infty$ are written as $k-$tuples. If we have a vector of the form ($k$, $k$, . . . , $k$), then we denote it by **k**. For example, the vector (3, 3, . . . , 3) will be denoted by **3**.

A point, $(x,\ y)$ in $\mathbb{Z}^2$ is a position determined by a pair $(x,\ y)$, where $x, y \in \mathbb{Z}$. We denote a point by $p$. A line, $l$, is the shortest distance between two points [**?** ]. When we draw lines in geometry, we use an arrow at each end to show that it extends infinitely. In this part, a line segment, $\bar{l}$, is determined by any pair of two distinct points ($p_1$, $p_2$) on a line together with all the points of the line between $p_1$ and $p_2$, where $p_1$ and $p_2$ are called the endpoints. A shape, $\sigma$, is defined by the area enclosed by a finite set of line segments. A label can be denoted by any of the lowercase or uppercase letters, $a$–$z$ and $A$–$Z$, respectively.

A labelled shape is a pair ($A$, $\sigma$), where $A$ is the label of the shape taken from the uppercase or lowercase letters and the symbol $\sigma$ is as defined above. A grid is a coordinate plane consisting of a space of small squares, with horizontal ($x$) axis and vertical ($y$) axis, see Figure 1. Every labelled shape is presented in a unit square on the grid. The initial shape is usually labelled $S$. The labelled shape ($A$, $\sigma$), is denoted by ($A$, ($x$, $y$)) in the rest of this work. Where $(x,\ y) \in \mathbb{Z}^2$ denotes the lower lefthand corner of the unit square the shape, $\sigma$, will be drawn.

An additive shape grammar rule is commonly expressed in the form in Rule 1,

$$(A,\ (x,\ y)) \longrightarrow \{(A_1,\ (x_1,\ y_1)), (A_2,\ (x_2,\ y_2)), \ldots, (A_i,\ (x_i,\ y_i))\} \tag{1}$$

where $A$ is a variable (uppercase label) and $(x,\ y) \in \mathbb{Z}^2$ is as defined above. The arrow "$\longrightarrow$" is interpreted as "transformed to", $A_1, A_2 \ldots A_i$ represent variable(s) and terminal(s) (lowercase labels), for $i \in \mathbb{N}_+$ and $(x_1,\ y_2)$, $(x_1,\ y_2)$, . . . , $(x_i,\ y_i) \in \mathbb{Z}^2$.

The interpretation is as follows: a rule as in Rule 1 can be applied to a developing image if the developing image contains the variable $A$. Then, we take the set difference of the developing image and the variable $A$ and take the set union of the developing image and $\{(A_1,\ (x_1,\ y_1)), (A_2,\ (x_2,\ y_2)), \ldots, (A_i,\ (x_i,\ y_i))\}$.

The number ($n$) of times a rule ($r$) is applied during the generative process is denoted by $r^{(n)}$. For instance, $2^{(5)}$ means that rule 2 is applied five times. The operations of shape union and difference treat shapes in the same basic way as the set theoretic operations of union and difference treat sets.

Next, we demonstrate how an additive shape grammar rule is rendered graphically. Let
$(S,\ (x,\ y)) \longrightarrow \{(d,\ (x,\ y)),\ (A,\ (x+1,\ y)),\ (B,\ (x,\ y-1))\}$ be a rule which is also the same as

$$(S,\ (0,\ 0)) \longrightarrow \{(d,\ (0,\ 0)),\ (A,\ (1,\ 0)),\ (B,\ (0,\ -1))\},$$

when $x$, $y = 0$. The rule above will produce Figure 1 when rendered using square shapes.

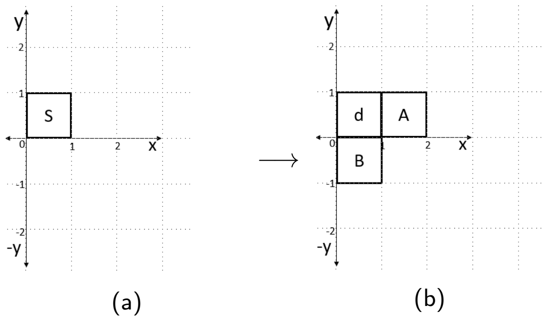


**Figure 1**: A simple shape grammar rule on the grid.

For visualisation purposes, every terminal is associated with a shape and the shape is filled with a chosen colour. The image is rendered to any size according to what is needed.

### 2.1.2 Formal Definition of Additive Shape Grammars

The definition of additive shape grammars is motivated by definition of shape grammars in [25], and has been modified where appropriate.

**Definition 1.** *An Additive Shape Grammar (ASG)* $G = (V_\mathrm{M},\ V_\mathrm{T},\ R,\ (S,\ (x,\ y)))$ *has a finite alphabet* $V$ *of labels, consisting of disjoint subsets* $V_\mathrm{M}$ *of variables and* $V_\mathrm{T}$ *of terminals.* $R$ *is the set of rules of the form* $(A,\ (x,\ y)) \longrightarrow \{(A_1,\ (x_1,\ y_1)),\ (A_2,\ (x_2,\ y_2)), \ldots,\ (A_i,\ (x_i,\ y_i))\}$ *where* $A \in V_\mathrm{M}$, $\{A_1, A_2, \ldots, A_i\} \subseteq V$ *and* $(x,\ y)$, $(x_1,\ y_1)$, $\ldots$, $(x_i,\ y_i) \in \mathbb{Z}^2\ \forall\ i \in \mathbb{N}_+$, $(S, (x,\ y))$ *is the initial labelled shape, with* $S \in V_\mathrm{M}$.

**Definition 2.** *A pictorial form or evolving image is any set (composition) of labelled shapes in the plane denoted by* $\Pi$*. If* $\Pi$ *is a pictorial form, we denote by* $l(\Pi)$ *the set of labels used in* $\Pi$*.*

**Definition 3.** *For an* $ASG$ $G$ *and pictorial forms* $\Pi$ *and* $\Gamma$*, there is a derivation step from* $\Pi$ *to* $\Gamma$*, if there is a rule* $(s,\ (x,\ y)) \longrightarrow \{(s_1,\ (x_1,\ y_1)), (s_2,\ (x_2,\ y_2)), \ldots, (s_i,\ (xi1,\ y_i))\}$ *in* $R$*, where* $\Pi$ *contains a labelled shape* $(s, (x,\ y))$*:* $s \in V_M$ *and* $\Gamma$ *is* $(\Pi \setminus \{(s, (x,\ y))\}) \cup \{(s_1,\ (x_1,\ y_1)), (s_2,\ (x_2,\ y_2)), \ldots, (s_i,\ (x_i,\ y_i))\}$*:* $\{s_1, s_2, \ldots, s_i\} \subseteq V$ *for* $i \in \mathbb{N}_+$ *as usual.*

*We denote the derivation step by* $\Pi \Longrightarrow \Gamma$*. This simply means that* $\Gamma$ *is directly derived from* $\Pi$*. If there is a sequence of zero or more derivation steps from* $\Pi$ *to* $\Gamma$*, then we denote that by* $\Pi \Longrightarrow^{*} \Gamma$*. We now say that* $\Gamma$ *is derived from* $\Pi$*.*

**Definition 4.** *An image is a pictorial form* $\Pi$ *with* $l(\Pi) \subseteq V_\mathrm{T}$*.*

**Definition 5.** *The gallery* $\mathcal{G}(G)$ *generated by an additive shape grammar* $G$ *is the set of images,* $\Pi$*, derivable from the initial shape* $(S,\ (x,\ y))$*, represented as:* $\mathcal{G}(G) = \{\Pi \colon (S,\ (x,\ y)) \Longrightarrow^{*} \Pi\}$*.*

Next, we demonstrate how additive shape grammars work in Example 1.

**Example 1.** *We want to generate the gallery of images that consist of the letter E (see Figure 2). Let* $G_\mathrm{E} = (V_M, V_T, R, (S,\ (x,\ y)))$*, where* $V_M = \{S,\ A,\ B,\ C,\ D,\ E\}$*,* $V_T = \{\ d\ \}$*,* $(S,\ (x,\ y)) = (S, (0, 0))$*, and* $R$ *is shown in Figure 3.*

Some of the images in $\mathcal{G}(G_\mathrm{E})$ when rendered using square shapes with the label $d$ associated with dark colour are shown in Figure 2. The images were scaled to the same size for presentation purposes as in Figure 2. The illustration of a derivation of $G_\mathrm{E}$ is given in Figure 4, having the lower lefthand corner of the initial shape $S$ start at $x$, $y = 0$.

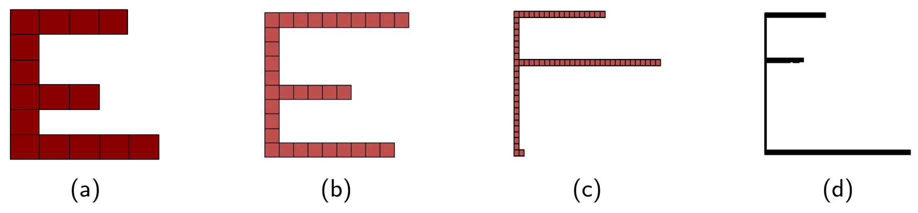


**Figure 2**: Some images in $\mathcal{G}(G_\mathrm{E})$.

The main purpose of our type additive shape grammars is to allow any shape to be used during image rendering after derivation. The gallery in Figure 5 shows that different shapes can be used to implement image rending when the derivation is complete. Some images obtained when the derivation in Figure 4 is rendered with different shapes are shown in Figure 5.

Observe that after Rule 2, any of the Rules 3–6 in Figure 3 are applicable and can be applied at any time. The same goes for Rules 7–10 after Rule 6, and Rules 11 and 12 after Rule 8. This can be seen as a weakness, as the images in the gallery may not only contain the letter $E$ (see Figures 2(b) and (c)). This means that the additive shape

$$R = \Bigg\{$$

$$
\begin{aligned}
(S,(x,y)) &\longrightarrow \{(d,(x,y)),(A,(x+1,y)),(B,(x,y-1))\} && (2)\\
(A,(x,y)) &\longrightarrow \{(d,(x,y)),(A,(x+1,y))\}\mid && (3)\\
&\phantom{\longrightarrow}\ \{(d,(x,y))\} && (4)\\
(B,(x,y)) &\longrightarrow \{(d,(x,y)),(B,(x,y-1))\}\mid && (5)\\
&\phantom{\longrightarrow}\ \{(d,(x,y)),(C,(x+1,y)),(D,(x,y-1))\} && (6)\\
(D,(x,y)) &\longrightarrow \{(d,(x,y)),(D,(x,y-1))\}\mid && (7)\\
&\phantom{\longrightarrow}\ \{(d,(x,y)),(E,(x+1,y))\} && (8)\\
(C,(x,y)) &\longrightarrow \{(d,(x,y)),(C,(x+1,y))\}\mid && (9)\\
&\phantom{\longrightarrow}\ \{(d,(x,y))\} && (10)\\
(E,(x,y)) &\longrightarrow \{(d,(x,y)),(E,(x+1,y))\}\mid && (11)\\
&\phantom{\longrightarrow}\ \{(d,(x,y))\} && (12)
\end{aligned}
$$

$$\Bigg\}$$

**Figure 3**: The set of rules for $G_{\mathrm{E}}$ in Example 1.

$$
\begin{aligned}
\{(S,(0,0))\} &\overset{2}{\Longrightarrow} \{(d,(0,0)),(A,(1,0)),(B,(0,-1))\}\\
&\overset{3^{(2)},4}{\Longrightarrow} \{(d,(0,0)),(d,(1,0)),(d,(2,0)),(d,(3,0)),(B,(0,-1))\}\\
&\overset{5^{(2)},6}{\Longrightarrow} \{(d,(0,0)),(d,(1,0)),(d,(2,0)),(d,(3,0)),(d,(0,-1)),(d,(0,-2)),\\
&\qquad (d,(0,-3)),(C,(1,-3)),(D,(0,-4))\}\\
&\overset{9,10}{\Longrightarrow} \{(d,(0,0)),(d,(1,0)),(d,(2,0)),(d,(3,0)),(d,(0,-1)),(d,(0,-2)),\\
&\qquad (d,(0,-3)),(d,(1,-3)),(d,(2,-3)),(D,(0,-4))\}\\
&\overset{7,8}{\Longrightarrow} \{(d,(0,0)),(d,(1,0)),(d,(2,0)),(d,(3,0)),(d,(0,-1)),(d,(0,-2)),\\
&\qquad (d,(0,-3)),(d,(1,-3)),(d,(2,-3)),(d,(0,-4)),(d,(0,-5)),\\
&\qquad (E,(1,-5))\}\\
&\overset{11^{(3)},12}{\Longrightarrow} \{(d,(0,0)),(d,(1,0)),(d,(2,0)),(d,(3,0)),(d,(0,-1)),(d,(0,-2)),\\
&\qquad (d,(0,-3)),(d,(1,-3)),(d,(2,-3)),(d,(0,-4)),(d,(0,-5)),\\
&\qquad (d,(1,-5)),(d,(2,-5)),(d,(3,-5)),(d,(4,-5))\}
\end{aligned}
$$

**Figure 4**: A derivation of the picture in Figure 2(a) for $G_{\mathrm{E}}$ in Example 1.

grammars, in as much as they can allow for any shape to be used to render images after derivation (see Figure 5), may not always generate images in a regulated manner using the same grammar. This simply means that additive shape grammars alone may not be able to generate images in a regulated manner at all times thus the need for the concept of bag context being added to the additive shape grammar rules to generate similar images of choice in a controlled

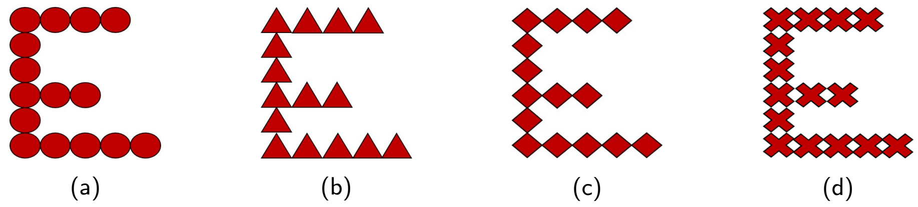


**Figure 5**: Image rendering of the derivation in Figure 4 with different shapes.

manner. This control to the additive shape grammar rules and thus to the derivation process with the help of bag context will enable us to generate a gallery of images that will contain only images of $E's$ whose legs are equal length with the upper and lower spines the same length.

Next, we present bag context grammars.

## 2.2 Bag Context Grammars

According to Drewes `et al` [7] bag context was introduced as a device for regulated rewriting in tree grammars. Bag context (BC) can be said to be a technique used to control when a context free grammar rule can be applied during a derivation. BC controls the application of a rule by a $k$-tuple of integers called the bag. The bag changes during the derivation of an image. A rule in the grammar can only be applied if the bag at that point is within the range defined by the lower and upper limits of the rule. When a rule is applied, it causes the bag values to change by adding the bag adjustment, which is part of the rule. This type of technique, when added to shape grammar rules, can also be used to produce or generate an infinite number of images in a regulated manner.

The formal definition of bag context grammars (BCG) and some examples are adapted from Drewes et al [7]. Next we present related work.

## 2.3 Related Work

Here we go straight to describe related work based on the works that are similar or relavant to shape grammar interpreter tools. These works are discussed under three categories; such as, arts, architecture and product design. Then, the reason for our type of bag context shape grammar interpreter. A shape grammar interpreter is a software application that uses shape grammar rules to generate designs [5, 28, 27, 17].

### 2.3.1 Arts

According to Crumley `et al` [5], shape grammars are specified and applied to produce a required set of shapes. In this work, the user provides a set of shape grammar rules and the axiom shape. A new shape or design is produced when a rule set is run on the axiom which modifies the existing shape in each iteration. Trescak `et al` [28] modified Krishnamurti's algorithm [17] for shape grammar interpretation. Their shape grammar interpreter allows users to interactively indicate the shape and rules, and additionally, have total control throughout the development process. One major challenge with the Trescak `et al` [28] work is that users select the rule to be applied during derivation; this becomes cumbersome when the rules are up to 1000 or more.

Meanwhile, Ertelt and Shea [8], used OpenCASCADE, an open source geometric kernel to implement the Shape Grammar for Machining Planning (SGMP) system. The SGMP uses shape grammars whose rules are encoded into lines, points, and volumes. In the workpiece (derivation), the system shows which volume can be removed from a workpiece or not. According to Ertelt and Shea [8], when a rule is applied, the control codes for the machine tool get instantiated. A rule is applied based on the constraints (context) regarding the machining process [8]. The vocabulary of the grammar represents the volumes—designs.

### 2.3.2 Architecture

According to Mhin et al [20], an interactive design of probability density functions for shape grammars was introduced as a framework that enables a user to interactively design a probability density function for a shape grammar and to generate models according to the designed pdf. They extended existing exploratory modeling tools that are suitable to select a single model from a shape space to modeling a distribution of shapes. They also proposed a user interface to display, sort, and sample models to enable a user to quickly assign preference scores. In Correia `et al` [4], a discursive interpreter called `MALAGA` was developed for the customization of Portuguese mass housing. It was composed of two modules, the `programa` and `designa`. In the `programa`, users provide the description of the type of house they want with the help of a programmed graphic user interface. The `designa` computes the design according to some architectural styles [4]. According to Li `et al` [18], a prototype system for two and three dimensional shape grammars was developed to enable users to develop grammars for the modelling of building plans. In this work, the system supports the user in editing grammars and switching between two types of activity. Jowers and Earl [15, 16], implemented shape grammars on curved shapes using algorithms for shape operation on shapes composed of parametric curves for the development of buidling plans.

Furthermore, Daniel et al [6], described an interactive system that enables both creating new buildings in the style of others and modifying existing buildings in a quick and intuitive manner. Santiago et al [22], focused on a specific editing application: copy and paste for procedural building models. They provided a simple to use, intuitive editing metaphor that allows non-experts to generate new content using pre-existing rulesets. In [23], Simon et al, proposed a real-time rendering approach for procedural cities. They implemented a new lightweight grammar representation that compactly encodes facade structures and allows fast per-pixel access.

### 2.3.3 Product design

Spatial grammars was implemented by McKay `et al` [19] to provide a framework for discussions around a next generation of design systems. They tried to study and implement the representation schemes of shape grammars with the associated computation systems, and the product development activities which include design synthesis and fabrication that such systems might be used within. In [12], Hoisl and Shea used a grammar based system that uses a set of parameterized functions to develop and implement an interactive three-dimensional grammar framework. The parameterized functions use these parameters and or primitives which includes the definition of non-parametric and parametric rules, and their automatic application. Here rules are matched from the left hand side to the working shape automatically. The parametric relations is also defined in the rules. One prototype example was the "kindergarten grammar"—the wheel rims.

Meanwhile, Jowers et al [13] provided a formal proof that adopting a finite fixed structure limits the capability of a shape grammar,and can make it impossible to fully implement shape computations. Here, full implementation is in the sense that all possible rule applications are supported. The shape grammars investigated have previously been used to support arguments that in shape computations, shape structures should not be fixed.

### 2.3.4 Why Bag Context Shape Grammar Interpreter?

These challenges and others contribute to the reason for the introduction of our type of `BCSGI` tool.

- The majority of the existing shape grammar interpreter tools do not automatically control when a rule should be applied and when not during a derivation.
- They do not control how many times a rule should be applied during a derivation.
- No work to the best of our knowledge has considered bag context added to shape grammars for the generation of images.

Next, we present bag context shape grammars in Section 3.

## 3 Bag Context Shape Grammars

The idea of Bag Context Shape Grammars (BCSGs) was first proposed by S. Ewert (sigrid.ewert@wits.ac.za, University of the Witwatersrand). In this part, we formally define bag context shape grammars and give some examples to illustrate how they work.

## 3.1 Formal Definition of Bag Context Shape Grammars

This section is motivated by the definition of bag context tree grammars in [7, 9] and the definition of additive shape grammars in Section 2.1.2 of this paper.

**Definition 6.** *A BCSG is a grammar with the form $G = (V_{\mathrm{M}}, V_{\mathrm{T}}, R, (S, (x, y)), I, \beta_0)$, where $V_{\mathrm{M}}$, $V_{\mathrm{T}}$, and $(S, (x, y))$ are as in Definition 1, $R$ is the set of shape grammar rules, where every rule is of the form $(A, (x, y)) \longrightarrow \{(A_1, (x_1, y_1)), (A_2, (x_2, y_2)), \dots, (A_i, (x_i, y_i))\}\ (L_{\mathrm{b}}, U_{\mathrm{b}}\ ;\delta)$, where $(A, (x, y)) \longrightarrow \{(A_1, (x_1, y_1)), (A_2, (x_2, y_2)), \dots, (A_i, (x_i, y_i))\}$ is as in Definition 1, $L_{\mathrm{b}}, U_{\mathrm{b}} \in \mathbb{Z}^I_\infty$, and $\delta$ is the bag adjustment, $\delta \in \mathbb{Z}^I$. $I$ is the finite bag index set of the form $[k]$ and $\beta_0$ is the initial bag, $\beta_0 \in \mathbb{Z}^I$.*

**Definition 7.** *Let a $configuration$ be a pair $(\Pi, \beta)$ where $\Pi$ is pictorial form and $\beta$ is the bag. For a BCSG $G$ and $configurations$ $(\Pi, \beta)$ and $(\Gamma, \beta')$, there is a derivation step from $(\Pi, \beta)$ to $(\Gamma, \beta')$, if there is a rule $(s, (x, y)) \longrightarrow \{(s_1, (x_1, y_1)), (s_2, (x_2, y_2)), \dots, (s_i, (x_i, y_i))\}\ (L_{\mathrm{b}}, U_{\mathrm{b}}\ ;\delta)$ in $R$, where $\Pi$ contains a labelled shape $(s, (x, y))$: $s \in V_M$ with $L_{\mathrm{b}} \leq \beta \leq U_{\mathrm{b}}$. $\Gamma = (\Pi \setminus \{(s, (x, y))\}) \cup \{(s_1, (x_1, y_1)), (s_2, (x_2, y_2)), \dots, (s_i, (x_i, y_i))\}$ and $\beta' = \beta + \delta$.*

*We denote the derivation step by $(\Pi, \beta) \Longrightarrow (\Gamma, \beta')$. This simply means that $(\Gamma, \beta')$ is directly derived from $(\Pi, \beta)$. If there is a sequence of zero or more derivation steps from $(\Pi, \beta)$ to $(\Gamma, \beta')$, then we denote that by $(\Pi, \beta) \Longrightarrow^* (\Gamma, \beta')$. We now say that $(\Gamma, \beta')$ is derived from $(\Pi, \beta)$.*

**Definition 8.** *An image is a pictorial form $\Pi$ with $l(\Pi) \subseteq V_T$ and the bag, $\beta$.*

**Definition 9.** *The gallery $\mathcal{G}(G)$ generated by a BCSG $G$ is the set of images, $\Pi$, derivable from the initial labelled shape $(S, (x, y))$ and the initial bag $\beta_0$, represented as: $\mathcal{G}(G) = \{\Pi\colon ((S, (x, y)), \beta_0) \Longrightarrow^* (\Pi, \beta)$, for some $\beta \in \mathbb{Z}^I_\infty$ and $l(\Pi) \subseteq V_T\}$. The class of all galleries generated by BCSGs is denoted by $BCSG_{\mathrm{G}}$.*

Next, we demonstrate BCSGs in Example 2. In Example 2, we generate images with the following: three legs of equal length and a spine. The second leg of the letter *E* to be generated divides the spine into an upper and a lower half which are the same length.

**Example 2.** *We extend Example 1 by adding bag context to generate the images in Figure 6. Let $G_{\mathrm{E-bag}} = (V_M, V_T, R, (S, (x, y)), \{1, 2, 3, 4\}, 0)$, where $V_M = \{S, A, B, C, D, E\}$, $V_T = \{d\}$, $(S, (x, y)) = (S, (0, 0))$, and $R$ is shown in Figure 7.*

Some of the images in the gallery produced by $G_{\mathrm{E-bag}}$ when rendered using square shapes with the label $d$ associated with dark colour are shown in Figure 6. The images were scaled to the size as in Figure 6 for presentation purposes.

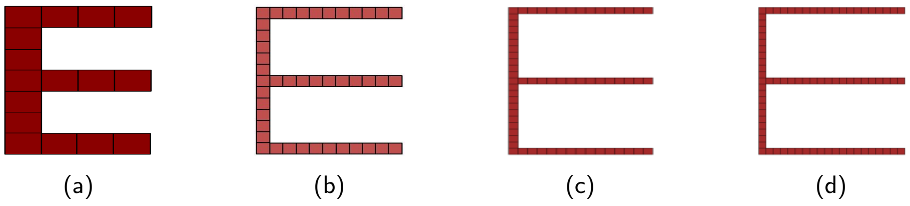


**Figure 6**: Some images in $\mathcal{G}(G_{\mathrm{E-bag}})$

The strategy here is that the first, third, and fourth bag positions are used to control how the first, second and third legs grow. The second bag position is used to control the upper and lower spines of the letter *E*. The first (top) leg of the letter $E$ is formed before any other part. The reason for this is that the number of times the rule that generates the top leg is applied is used to control the other legs. The upper spine is formed immediately after the top leg. The second leg is formed after the upper spine formation. Then the lower spine is formed followed by the third leg formation. The second leg must be exactly at half the spine which makes the upper and lower spines to be the same length. The three legs must also be the same length.

$$R = \Bigg\{$$

$$
\begin{aligned}
(S,(x,y)) &\longrightarrow \{(d,(x,y)),(A,(x+1,y)),(B,(x,y-1))\}(0,0;(1,1,0,0)) && (13)\\
(A,(x,y)) &\longrightarrow \{(d,(x,y)),(A,(x+1,y))\}((1,1,0,0),\infty;(1,0,0,0)) \mid && (14)\\
&\phantom{\longrightarrow} \{(d,(x,y))\}((2,1,0,0),(\infty,\infty,\infty,1);(-1,-1,0,1)) && (15)\\
(B,(x,y)) &\longrightarrow \{(d,(x,y)),(B,(x,y-1))\}((1,0,0,1),(\infty,\infty,\infty,1);(0,1,0,0)) \mid && (16)\\
&\phantom{\longrightarrow} \{(d,(x,y)),(C,(x+1,y)),(D,(x,y-1))\} && \\
&\phantom{\longrightarrow} \quad ((1,2,0,1),(\infty,\infty,0,1);(0,0,0,1)) && (17)\\
(D,(x,y)) &\longrightarrow \{(d,(x,y)),(D,(x,y-1))\}((0,1,1,3),(0,\infty,\infty,3);(0,-1,0,0)) \mid && (18)\\
&\phantom{\longrightarrow} \{(d,(x,y)),(E,(x+1,y))\}((0,0,1,3),(\infty,0,\infty,3);(0,0,0,1)) && (19)\\
(C,(x,y)) &\longrightarrow \{(d,(x,y)),(C,(x+1,y))\}((1,1,0,2),(\infty,\infty,\infty,2);(-1,0,1,0)) \mid && (20)\\
&\phantom{\longrightarrow} \{(d,(x,y))\}((0,1,2,2),(0,\infty,\infty,2);(0,0,0,1)) && (21)\\
(E,(x,y)) &\longrightarrow \{(d,(x,y)),(E,(x+1,y))\}((0,0,1,4),(\infty,\infty,\infty,4);(1,0,-1,0)) \mid && (22)\\
&\phantom{\longrightarrow} \{(d,(x,y))\}((1,0,0,4),(\infty,\infty,0,4);0) && (23)
\end{aligned}
$$

$$\Bigg\}$$

**Figure 7**: The set of rules for $G_{\mathrm{E-bag}}$ in Example 2.

Next, we explain how the rules in Figure 7 are applied to form the image in Figure 6(a).

To achieve the strategy, the derivation starts with the initial labelled shape in the pictorial form, $\Pi = \{(S,(x,y))\}$. The application of Rule 13 of Figure 7 replaces the initial labelled shape $(S,(x,y))$ with the shapes labelled $(d,(x,y))$, $(A,(x+1,y))$ and $(B,(x,y-1))$ in the set. The bag adjustment, (1, 1, 0, 0) is added to the bag, (0, 0, 0, 0) which is (0 + 1, 0 + 1, 0 + 0, 0 + 0) then the bag becomes (1, 1, 0, 0).

We can apply Rule 14 twice because the bag at each application is within the defined range. That is the bag is greater than or equal to the lower limit, (1, 1, 0, 0) and less than or equal to the upper limit, $\infty$. The bag adjustment, (1, 0, 0, 0) is added to the bag at each application of Rule 14. This is followed by the application of Rule 15 because it is also within the defined range.

At this point, the bag is (2, 0, 0, 1). We apply Rule 16 twice to form the upper spine; the bag is (2, 2, 0, 1). Rule 17 is then applied to begin the formation of the second leg and the lower spine; the bag adjustment, (0, 0, 0, 1) is added to the bag, (2, 2, 0, 1) then the bag becomes (2, 2, 0, 2).

We further apply Rule 20 twice and Rule 21 once to form the second leg. At this point, the bag adjustment, (0, 0, 0, 1) is added to the bag, (0, 2, 2, 2) the bag becomes (0, 2, 2, 3). We apply Rule 18 twice followed by Rule 19 because the bag at each application is within the defined range. At this point, the bag adjustment, (0, 0, 0, 1), is added to the bag, (0, 0, 2, 3) then the bag becomes (0, 0, 2, 4) and the lower spine is formed.

Finally, we apply Rule 22 twice and Rule 23 once to form the third leg. The bag adjustment, (0, 0, 0, 0) is added to the bag, (2, 0, 0, 4) the bag becomes (2, 0, 0, 4). At this point the picture formation is completed and the letter $E$ is generated (see Figure 6(a)).

We can also generate more images by repeating the process and applying Rules 14 and 16 as long as they are within the defined range at each application. This is because the number of times Rule 14 is applied determines the number of times Rules 20 and 22 will be applied and the number of times Rule 16 is applied determines the number of times Rule 18 will be applied with the help of the bag. Then, the process can be terminated with Rules 15, 21 and 23 respectively in order to form a complete picture (see Figures 6(b)–(d)).

An illustration of one of the derivations of $G_{\mathrm{E-bag}}$ using the picture in Figure 6(a) is given in Figure 8. The derivation has the lower lefthand corner of the initial shape $S$ start at $x$, $y = 0$. The derivation is summarised in Table 1 showing the operations in the bag when an adjustment is made.

$$
\begin{aligned}
\{(S,(0,0))\} &\xRightarrow{13} \{(d,(0,0)),(A,(1,0)),(B,(0,-1))\}\\
&\xRightarrow{14^{(2)},15} \{(d,(0,0)),(d,(1,0)),(d,(2,0)),(d,(3,0)),(B,(0,-1))\}\\
&\xRightarrow{16^{(2)},17} \{(d,(0,0)),(d,(1,0)),(d,(2,0)),(d,(3,0)),(d,(0,-1)),(d,(0,-2)),\\
&\qquad (d,(0,-3)),(C,(1,-3)),(D,(0,-4))\}\\
&\xRightarrow{20^{(2)},21} \{(d,(0,0)),(d,(1,0)),(d,(2,0)),(d,(3,0)),(d,(0,-1)),(d,(0,-2)),\\
&\qquad (d,(0,-3)),(d,(1,-3)),(d,(2,-3)),(d,(3,-3)),(D,(0,-4))\}\\
&\xRightarrow{18^{(2)},19} \{(d,(0,0)),(d,(1,0)),(d,(2,0)),(d,(3,0)),(d,(0,-1)),(d,(0,-2)),\\
&\qquad (d,(0,-3)),(d,(1,-3)),(d,(2,-3)),(d,(3,-3)),(d,(0,-4)),\\
&\qquad (d,(0,-5)),(d,(0,-6)),(E,(1,-6))\}\\
&\xRightarrow{22^{(2)},23} \{(d,(0,0)),(d,(1,0)),(d,(2,0)),(d,(3,0)),(d,(0,-1)),(d,(0,-2)),\\
&\qquad (d,(0,-3)),(d,(1,-3)),(d,(2,-3)),(d,(3,-3)),(d,(0,-4)),\\
&\qquad (d,(0,-5)),(d,(0,-6)),(d,(1,-6)),(d,(2,-6)),\\
&\qquad (d,(3,-6))\}
\end{aligned}
$$

**Figure 8**: A derivation of the picture in Figure 6(a) for $G_{\text{E}-\text{bag}}$.

Some images obtained when the derivation in Figure 8 is rendered with different shapes are shown in Figure 9.

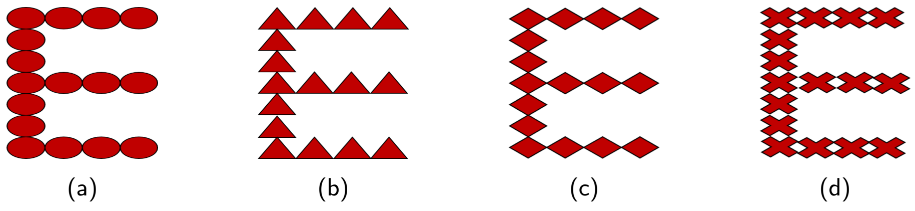


**Figure 9**: Image rendering of the derivation in Figure 8 with different shapes.

Next, we present the tool in Section 4.

## 4 BCSGI Tool Design

Here, we present the building blocks of the tool as follows:

### 4.1 Structure

The process of generating a gallery using the `BCSGI` tool consists of five stages. The five stages are:

1. **Initialization stage:** The rules are loaded into the rule set using the predefined format in Figure 1. The rule set is sorted into different lists according to the nonterminal on the left hand side of each rule. The initial shape is assigned to the pictorial form and the initial bag is declared.
2. **Outer collection stage:** The nonterminals are picked from the pictorial form and a nonterminal is randomly selected.

**Table 1**: The bag during the derivation in Figure 8.

| Rules | $\beta$ | $\delta$ |
|---:|---:|---:|
| 13 | 0 | (1, 1, 0, 0) |
| $14^{(2)}$ | (1, 1, 0, 0) | (1, 0, 0, 0) |
| 15 | (3, 1, 0, 0) | (-1, -1, 0, 1) |
| $16^{(2)}$ | (2, 0, 0, 1) | (0, 1, 0, 0) |
| 17 | (2, 2, 0, 1) | (0, 0, 0, 1) |
| $20^{(2)}$ | (2, 2, 0, 2) | (-1, 0, 1, 0) |
| 21 | (0, 2, 2, 2) | (0, 0, 0, 1) |
| $18^{(2)}$ | (0, 2, 2, 3) | (0, -1, 0, 0) |
| 19 | (0, 0, 2, 3) | (0, 0, 0, 1) |
| $22^{(2)}$ | (0, 0, 2, 4) | (1, 0, -1, 0) |
| 23 | (2, 0, 0, 4) | 0 |

3. **Inner collection stage:** The nonterminal selected in stage 2 is used to identify the sorted list that contains the rule(s) with the selected nonterminal on the left hand side of each rule. Then a rule is randomly picked from the identified sorted list.
4. **Derivation stage:** Checks if the bag of the selected rule in stage 3 is within the defined range and then performs rule derivation, automatically adjusting the bag. Execution is returned to stage 2 to randomly pick another nonterminal from the new pictorial form.
5. **Conversion stage:** Checks if the pictorial form consists completely of terminals. The pictorial form is then converted to a bitmap image.

Next, we present the architectural flow of the stages for the generation of a gallery in our tool in Section 4.2.

## 4.2 Architecture

Figure 10 shows the architectural flow of the BCSGI tool showing how the stages are linked and communicate with each other.

## 4.3 Interpretation

Algorithm 1 shows the implementation of the BCSGI tool. The first stage of the algorithm requires that a user defines and loads the rules. A simple rule with shape whose lower left corner units are in centimetres is loaded using the following format in Listing 1. An example of the rule format in Listing 1 is shown in Listing 2.

Listing 1: A simple bag context shape grammar rule format.

```
RuleNo ([label], [coordinate]): ([label(s)], [coordinate(s)]) [bag]
```

Listing 2: A bag context shape grammar rule.

```
("S", (0,0),=): ("d", (0,0)), ("A", (0,1)) (0, 0; (1, 0))

("A", (0,0)): ("d", (0,0)), ("A", (0,1)), ("B", (−1,0)) ((1, 0), (1, 2); (−1, 1))
```

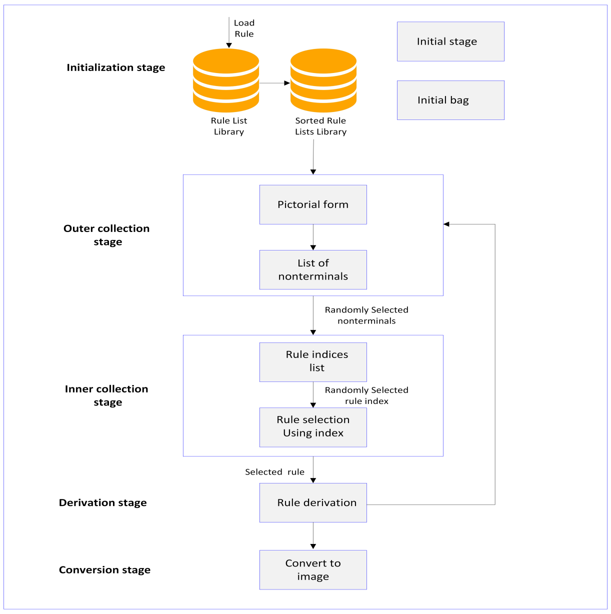


**Figure 10**: The architectural flow of BCSGI. The purpose of the randomization is to avoid giving any nonterminal or rule priority.

Next, we explain in detail how Algorithm 1 works.

The rules are loaded into the `ruleList` in Line 1. Line 2 sorts the `ruleList` into separate lists—`sortedruleLists`, according to the nonterminals in the left hand side of each rule in the `ruleList`. We initialise the `Bag` and the `pictorialForm` in Lines 3 and 4. Line 5 picks the `nonTerminal(s)` in the `pictorialForm` and stores the same in the `nonTerminalList()`. Line 6 randomly picks a number `i` from 1 to the length of the `nonTerminalList()`. The number is used as an index to pick a `nonTerminal` from the `nonterminalList(i)` in Line 7, then the `nonTerminal` with the index `i` is removed from the `nonTerminalList` in Line 8 in order not to pick it again because it has been picked already. Line 9 identifies and picks the list with the nonterminal which is picked in Line 8 from the `sortedruleLists()` and stores the same in the `sortedRule()` list.

Lines 10—12 pick the indices of all the rules in the `sortedRule` list and insert them in a list `ruleIndexes(ruleIndex)`.add, then Line 13 randomly picks a number `ruleNo` from 1 to the length of `ruleIndexes()`. Line 14 uses the `RuleNo` picked in Line 13 to select the rule with the index from the `sortedRule()` list, then, checks if the `Bag` is within the defined range. Line 15 does the derivation to the `pictorialForm`. The picked `nonTerminal` in the `pictorialForm` is replaced with its `Terminal` equivalent which also appeared on the right hand side (`RHS`) of the selected rule from the `sortedRule()` list; the remaining parts of the `RHS` of the selected rule maintain their positions. This automatically adjusts the bag in Line 16. The index is then removed from the `ruleIndexes()` list in Line 17 and Line 18 returns execution to Line 5 to pick a nonterminal from the new pictorial form.

If the `Bag` is not within the defined range as in Line 14 and the `ruleIndexes()` list is not empty as in Line 19, then, Line 20 removes the `RuleNo` from the `ruleIndexes` list because the rule with the index is not applicable at this time. Line 21 returns execution to Line 13 to pick another rule index from the `ruleIndexes` list. If the `Bag` is not within the defined range as in Line 14 and the `ruleIndexes()` list is empty, then, Line 23 returns execution to Line 5 to pick a `nonTerminal` from the `nonTerminalList()` which is extracted from the current `pictorialForm`.

A picture is formed by converting the `pictorialForm` to a bitmap in Line 26 when all the `nonTerminals` in the `pictorialForm` have been replaced with `Terminals` as in Line 25 and the terminal are associated with any shape. The process is terminated at this point and Line 27 returns the image formed.

**Algorithm 1:** Bag context shape grammar implementation

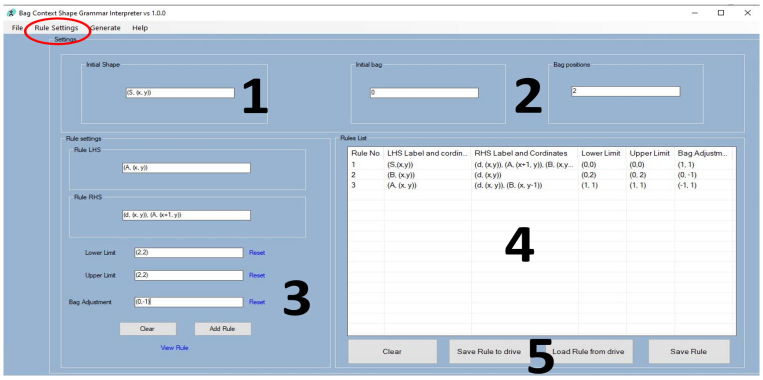
```
 1 ruleList $\leftarrow$ loadRules()
 2 sortedruleLists $\leftarrow$ Sort(ruleList)
 3 Bag $\leftarrow$ iniBag
 4 pictorialForm $\leftarrow$ iniShape
 5 nonTerminalList $\leftarrow$ pictorialForm().getNonterminal
 6 i $\leftarrow$ randomize(1, (nonTerminalList().max))
 7 nonTerminal $\leftarrow$ nonTerminalList(i)
 8 nonTerminalList(i).remove
 9 sortedRule() $\leftarrow$ sortedruleLists.getList(nonTerminal)
10 forall ruleIndex $\in$ sortedRule do
11     ruleIndexes(ruleIndex).add
12 end
13 RuleNo $\leftarrow$ randomize(1,(ruleIndexes().max)
14 if Bag $\geq$ sortedRule[RuleNo].lowerLimit and Bag $\leq$ sortedRule[RuleNo].upperLimit then
15     pictorialForm $\leftarrow$ (pictorialForm $\setminus$ LHS(sortedRule[RuleNo])) $\cup$
        RHS(sortedRule[RuleNo])
16     Bag.adjust
17     ruleIndexes(RuleNo).remove
18     pick another nonterminal from the new pictorial form at line 5
19 else if ruleIndexes().empty $\neq$ true then
20     ruleIndexes(RuleNo).remove
21     pick another rule number at line 13
22 else
23     pick all the nonterminals from the new pictorial form at line 5
24 end
25 if forall pictorialForm $\subseteq$ Terminals then
26     pictorialForm(AnyShape).convertToBitmap
27     return pictorialForm
28 end
```

## 4.4 Implementation and Result

The BCSGI tool was implemented using the .Net framework Class Library (FCL). It can be run on any computer with Windows 7 operating system or above, 2GB of *RAM* or above, 10GB of HDD or above. The framework can also run on any machine with CPU architecture of 32–bit or 64–bit. It is designed to provide an interactive and comfortable user experience, to produce images that are similar but not identical in a regulated manner. It performs in a way that a grammar can generate an infinite number of similar but not identical images in a regulated manner. The BCSGI project file, the executable file and some example files (rules) are available at (https://bit.ly/2QJ1PXQ) to support designers for the implementation of shape grammar and bag context shape grammar in the formal language communities. Figure 12 shows some images we generated using the BCSGI tool.

**GUI** Allows for the automatic response by the click of a mouse when a rule set is loaded. The GUI is organized into two major sections, Section One–Rule Settings and Section Two–Generate, each section having its different parts.

The Rule Settings section as in Figure 11 contains the rule settings panel displays. Parts 1 and 2 initialize the starting label and the coordinate, initial bag and the bag index. Part 3 adds the rules to the rule list. Part 4 displays the list of rules added in `Part 3`. `Part 5` contains `Save rule`—keeps the rule ready for the derivation, `Load rule from drive`—retrieves rule as text file into the system, `save rule to drive`—extracts rule as text file to an external drive, and `clear`—wipes all the rules from the rule list.



Initial Shape: (S, (x, y))

Initial bag: 0

Bag positions: 2

Rule settings

Rule LHS: (A, (x, y))

Rule RHS: (d, (x, y)), (A, (x+1, y))

Lower Limit: (2,2)

Upper Limit: (2,2)

Bag Adjustment: (0,-1)

Rules List

| Rule No | LHS Label and cordin... | RHS Label and Cordinates | Lower Limit | Upper Limit | Bag Adjustm... |
|---|---|---|---|---|---|
| 1 | (S,(x,y)) | (d, (x,y)), (A, (x+1, y)), (B, (x,y... | (0,0) | (0,0) | (1, 1) |
| 2 | (B, (x,y)) | (d, (x,y)) | (0,2) | (0, 2) | (0, -1) |
| 3 | (A, (x, y)) | (d, (x, y)), (B, (x, y-1)) | (1, 1) | (1, 1) | (-1, 1) |

**Figure 11**: The BCSGI rule settings.

The `Generate` section contains the parts for the image generation. `Part 1` does the rule derivation which generates an image by requesting the user to select any shape of choice. `Part 2` is the plain canvas that displays the image generated. `Part 3` saves the image generated to drive while `Part 4` clears the content of `Part 2`. Then, `Parts 5` and 6 activate and show the stepwise process of the image derivation graphically.

**Output** Some images generated using the `BCSGI` tool are shown in Figure 12.

## 4.5 Performance

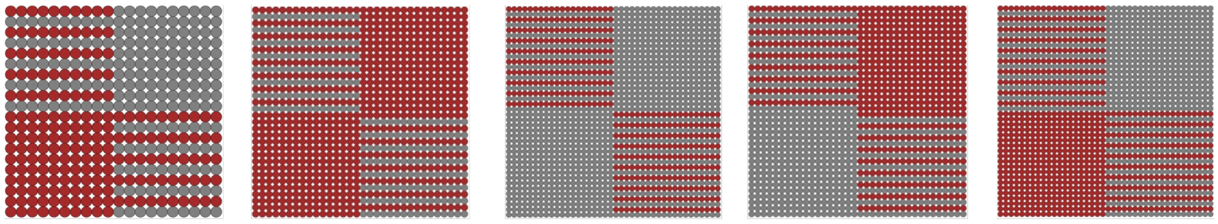
This `BCSGI` is strictly a proof-of-concept to show that bag context can be added to shape grammar rules in order to produce images in a regulated manner. There are a few lines in the algorithm which we believe should potentially reduce the running time. For instance, during the derivation, the rules are selected from the `sortedRule` list using the picked nonterminal without needing to check the entire rule list from beginning to end. The selected rule number is removed from the rule indexes list in order not to pick the rule number again. The selected rule number is used to pick a particular rule at a time without needing to check from the beginning of the sorted rule list to end. This clearly reduces the number of random selection of terminals and rules during the derivation (see Algorithm 1).

This `BCSGI` is designed to provide an interactive and comfortable user experience, to produce images that are similar in a regulated and timely manner. It performs in a way that a grammar can generate an infinite number of similar images in a regulated manner.

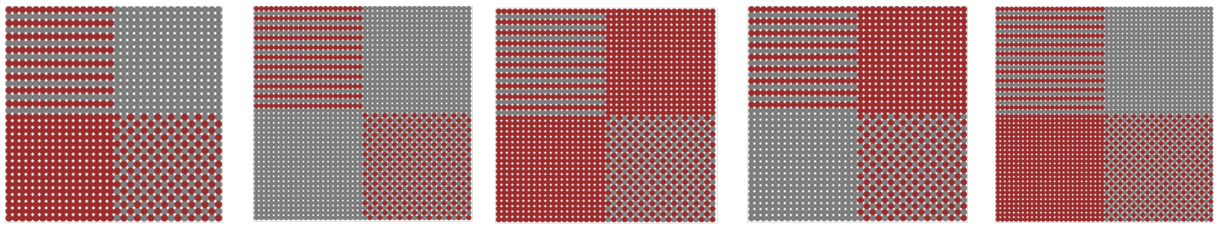
Next, we conclude this work in Section 5.

## 5 Conclusion and Future Work

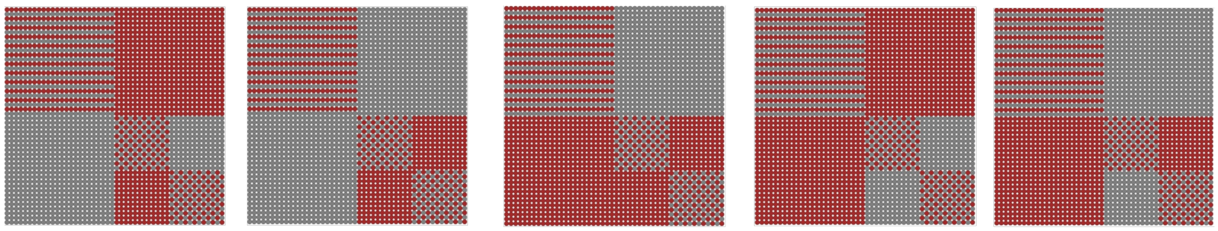
In this work, we presented `BCSGI`, a tool that generates images in a regulated manner using shape grammar rules. The `BCSGI` accepts shape grammar rules and automatically allows when a rule should be applied during a derivation. These rules are designed in a way their application is controlled during a derivation. We improved upon and implemented the existing shape grammar interpretation algorithm in [5]. We also tested the `BCSGI` with some bag context shape grammar rules and it generated the expected images. We argued that the `BCSGI` tool could offer a wide range of application areas such as aiding the teaching of shape grammar implementation in higher learning, a framework to support designers for the development and implementation of an improved `BCSGI` tool, etc.



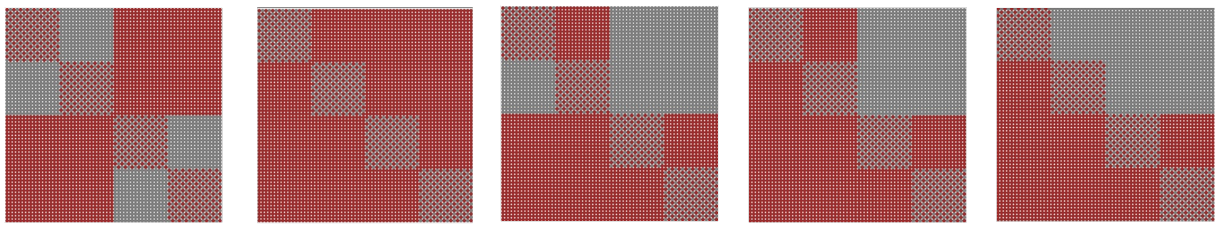
(a) Some pictures in Carpet–A.



(b) Some pictures in Carpet–B.



(c) Some pictures in Carpet–C.



(d) Some pictures in Carpet–D.

**Figure 12**: Some galleries generated by the BCSGI.

In the future, we will implement a more robust BCSGI tool that can run on different platforms with improved optimized performance.

## REFERENCES

[1] Ahmad, S.; Chase, S.: Transforming grammars for goal driven style innovation. In Predicting the future, Proceedings of the 25th Conference on Education in Computer Aided Architectural Design in Europe (eCAADe), Frankfurt am Main, Germany, 879–886, 2007.

[2] Al-Kazzaz, D.A.; Bridges, A.H.: A framework for adaptation in shape grammars. Design Studies, 33(4), 342–356, 2012.

[3] Chin, R.C.: Product grammar: Construction and exploring solution spaces. Ph.D. thesis, Massachusetts Institute of Technology, 2004.

[4] Correia, R.C.; Duarte, J.P.; Leitão, A.M.: Malag: A discursive grammar interpreter for the online generation of mass customized housing. In Proceedings of the 4th International Conference on Design Computing and Cognition, 2010.

[5] Crumley, Z.; Marais, P.; Gain, J.: Voxel-space shape grammars. In WSCG 2012: 20th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, Plzen, Czech Republic, 25 - 28 June, 1–10, 2012.

[6] Daniel, G.A.; Paul, A.R.; Daniel, R.B.: Style grammars for interactive visualization of architecture. IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS, 13(4), 786–797, 2007.

[7] Drewes, F.; du Toit, C.; Ewert, S.; van der Merwe, B.; van der Walt, A.P.: Bag context tree grammars. In Proceedings of the 10th International Conference, DLT 2006, Santa Barbrara, CA, 226–237. Springer, June 26-29, 2006.

[8] Ertelt, C.; Shea, K.: Shape grammar implementation for machining planning. In Proceedings of the 4th International Conference of Design Computing and Cognition, 2010.

[9] Ewert, S.; Jingili, N.; Mpota, L.; Sanders, I.: Bag context picture grammars. Journal of Computer Languages, 2019. doi:https://doi.org/10.1016/j.cola.2019.04.001.

[10] George, S.: Shape: Talking about seeing and doing. MIT Press, 2006.

[11] Herbert, T.; Sanders, I.; Mills, G.: African shape grammar: A language of linear Ndebele homesteads. Environment and Planning B: Planning and Design, 21(4), 453–476, 1994.

[12] Hoisl, F.; Shea, K.: An interactive, visual approach to developing and applying parametric three-dimensional spatial grammars. Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 25(04), 333–356, 2011.

[13] Iestyn, J.; Chris, E.; George, S.: Shapes, structures and shape grammar implementation. Computer-Aided Design, 111, 80–92, 2019. https://doi.org/10.1016/j.cad.2019.02.001.

[14] Jiang, T.; Li, M.; Ravikumar, B.; Regan, K.W.: Formal grammars and languages. In Algorithms and Theory of Computation Handbook, 20–26. Chapman & Hall/CRC, 2010.

[15] Jowers, I.; Earl, C.: Implementation of curved shape grammars. Environment and Planning B: Planning and Design, 38(4), 616–635, 2011.

[16] Jowers, I.; Hogg, D.C.; McKay, A.; Chau, H.H.; De Pennington, A.: Shape detection with vision: Implementing shape grammars in conceptual design. Research in Engineering Design, 21(4), 235–247, 2010.

[17] Krishnamurti, R.; Earl, C.: Shape recognition in three dimensions. Environment and Planning B: Planning and Design, 19(5), 585–603, 1992.

[18] Li, A.K.; Chau, H.; Chen, L.; Wang, Y.: A prototype system for developing two-and three-dimensional shape grammars. In Proceedings of the 14th International Conference of Computer-Aided Architectural Design Research in Asia, 717–726, 2009.

[19] McKay, A.; Chase, S.; Shea, K.; Chau, H.H.: Spatial grammar implementation: From theory to useable software. Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 26(02), 143–159, 2012.

[20] Minh, D.; Stefan, L.; Duygu, C.; Boris, N.; Peter, W.; Mark, P.: Interactive design of probability density functions for shape grammars. ACM Transactions on Graphics (TOG), 34(6), 01–13, 2015.

[21] Okundaye, B.; Ewert, S.; Sanders, I.: A novel approach to visual password schemes using tree picture grammars. In Proceedings of the 2014 PRASA, RobMech and AfLaT International Joint Symposium, 247–252, 2014.

[22] Santiago, B.; Gonzalo, B.; Gustavo, P.: Visual copy & paste for procedurally modeled buildings by ruleset rewriting. Computers & Graphics, 37, 238–246, 2013.

[23] Simon, H.; Peter, W.; Stefan, M.A.; Gool, L.V.; Pascal, M.: Grammar-based encoding of facades. Comput. Graph. Forum, 29, 1479–1487, 2010. https://DOI:10.1111/j.1467-8659.2010.01745.x.

[24] Speller, T.H.; Whitney, D.; Crawley, E.: Using shape grammar to derive cellular automata rule patterns. Complex Systems-Champaign, 17(1/2), 79, 2007.

[25] Stiny, G.: Introduction to shape and shape grammars. Environment and Planning B, 7(3), 343–351, 1980.

[26] Stiny, G.; Gips, J.: Shape grammars and the generative specification of painting and sculpture. In IFIP Congress (2), vol. 2, 125–135, 1971.

[27] Tapia, M.: A visual implementation of a shape grammar system. Environment and Planning B: Planning and Design, 26(1), 59–73, 1999.

[28] Trescak, T.; Esteva, M.; Rodriguez, I.: A shape grammar interpreter for rectilinear forms. Computer-Aided Design, 44(7), 657–670, 2012.