Design for Descent: What Makes a Shape Grammar Easy to Optimize?

MILIN KODNONGBUA*, University of Washington, USA
ZIHAN JACK ZHANG*, University of Washington, USA
NICHOLAS SHARP, NVIDIA, USA
ADRIANA SCHULZ, University of Washington, USA

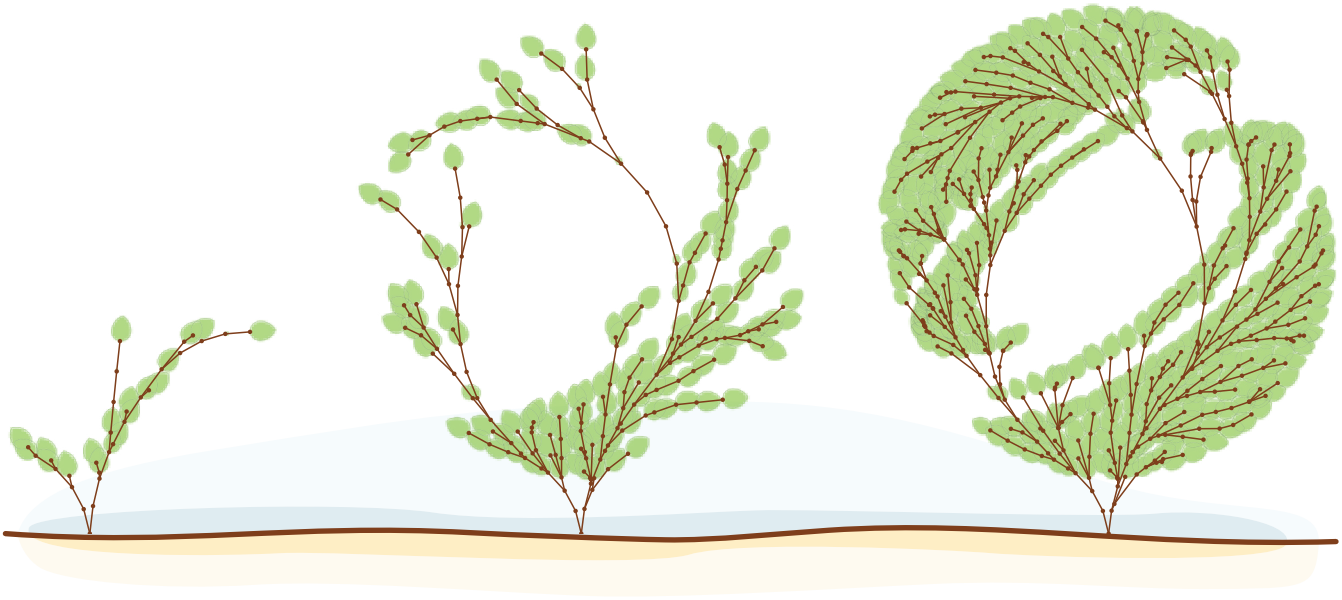

Fig. 1. Successive steps of a tree grammar designed for descent being optimized to match the SIGGRAPH logo. This work shows how careful consideration of the needs of optimization motivates the design of shape grammars well-suited for descent, turning difficult inverse tasks on structured representations into a straightforward gradient-based iterative optimization on the user's objectives.

Shape grammars offer a powerful framework for computational design, but synthesizing shape programs to achieve specific goals remains challenging. Inspired by the success of gradient-based optimization in high-dimensional, nonconvex spaces such as those in machine learning, we ask: what makes a shape grammar amenable to gradient-based optimization? To explore this, we introduce Stochastic Rewrite Descent (SRD), an algorithm that interleaves structural rewrites with continuous parameter updates, taking steps in both to optimize a given objective. We analyze the core challenges which have previously prevented optimizing shape programs via descent, and identify a set of desirable properties for grammars that support effective optimization, along with concrete grammar design recommendations to achieve them. We validate this approach across three shape grammars, demonstrating its effectiveness in diverse domains including image fitting, text-driven generation, and topology optimization. Through ablations and comparisons, we show that grammars satisfying our proposed properties lead to significantly better optimization performance. The goal of this work is to open the door to more general and flexible computational paradigms for inverse design with shape grammars.

CCS Concepts: • **Computing methodologies** → **Shape analysis**.

Additional Key Words and Phrases: optimization, shape grammar, procedural modeling

ACM Reference Format:
Milin Kodnongbua, Zihan Jack Zhang, Nicholas Sharp, and Adriana Schulz. 2025. Design for Descent: What Makes a Shape Grammar Easy to Optimize?. In *SIGGRAPH Asia 2025 Conference Papers (SA Conference Papers '25), December 15–18, 2025, Hong Kong, Hong Kong*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3757377.3764004

1 Introduction

Procedural modeling was introduced to the graphics community in the late 1980s [Prusinkiewicz 1986] and quickly gained popularity for its ability to generate a wide variety of shapes from simple shape grammars. However, a longstanding challenge has been control—finding an optimal sequence of grammar productions to achieve a

*Equal contribution

Authors' Contact Information: Milin Kodnongbua, milink@cs.washington.edu, University of Washington, Seattle, USA; Zihan Jack Zhang, jzhang18@cs.washington.edu, University of Washington, Seattle, USA; Nicholas Sharp, nsharp@nvidia.com, NVIDIA, Seattle, USA; Adriana Schulz, adriana@cs.washington.edu, University of Washington, Seattle, USA.

SA Conference Papers '25, Hong Kong, Hong Kong

ACM ISBN 979-8-4007-2137-3/2025/12
https://doi.org/10.1145/3757377.3764004

desired output. This inverse problem is difficult due to the complex discrete structure of the production space. As a result, despite the fact that many problems in computational design are naturally expressed as optimization of some objective, solutions are instead typically implemented through hand-crafted, domain-specific algorithms.

Meanwhile black-box gradient-based optimization has been widely successful for inverse design tasks in other domains like geometric optimization and reconstruction. In machine learning it is common to simply write down any desired set of objectives and iteratively descend to a solution, leveraging the effectiveness of stochastic gradient descent on high-dimensional nonconvex spaces. The same should be possible on structured shape grammar representations, but has not yet been realized.

This work revisits inverse procedural modeling through the lens of gradient descent. Our key insight is to reverse the conventional perspective: rather than starting with a fixed grammar and designing an algorithm to search over it, we instead ask—*what makes a general shape grammar amenable to optimization via gradient descent?* This reframing shifts the focus to the grammar design itself, allowing us to construct grammars that are intrinsically easier to control and optimize.

First, we must define how gradient descent operates over shape grammars. While parameter gradients are relatively straightforward to compute, the large combinatorial space of production rule applications—called rewrites—poses a challenge, as many rewrites change both the number and meaning of parameters. We introduce stochasticity to manage this complexity: at each step, we sample a finite, randomly selected set of rewrites and apply all promising ones in parallel. Our optimization scheme, SRD (Stochastic Rewrite Descent), interleaves this step with traditional gradient descent on continuous parameters, allowing coordinated progress across both the discrete and continuous components of the grammar.

However, even with a good optimization algorithm, descending on many grammars quickly gets trapped in bad minima, or explodes to nonsensical solutions. This motivates a more fundamental question: what makes a shape grammar amenable to descent-based optimization? Much like neural networks are parameterized to allow nonconvex optimization to succeed, how should grammars be designed for inverse optimization tasks?

We identify four properties of grammars that make gradient-based optimization practical. Formally stated later, they ask that: (1) any target shape be reachable from any starting point through some sequence of rewrites and parameter updates; (2) those rewrites behave smoothly enough that the space is as similar as possible to an ordinary continuous space; (3) a desired shape can be expressed in many different parameter configurations, giving the optimizer plenty of improving directions and reducing the risk of poor local minima; and (4) each update can be projected back onto the feasible set so hard design constraints are always respected.

We validate the importance of these properties across a range of applications on several grammars. Experiments show that classic rewrite systems like L-systems can be redesigned to express the shape space of trees while incorporating these properties to enable optimization, while ablations and comparisons to validate that the properties we identify indeed make gradient descent succeed. We demonstrate how other important problems such as parametric curve fitting, text-driven CSG construction, and topology optimization can be naturally expressed within our framework, allowing inverse procedural modeling to solve complex design challenges. We hope that this study will open the door to more widespread use of inverse optimization on shape grammars, integrating naturally with modern differentiable optimization. Code is available at https://github.com/milmillin/d4descent.

2 Related Work

Procedural Modeling and Control. Procedural models [Müller et al. 2006; Smelik et al. 2014; Wonka et al. 2003] generate complex structures by repeatedly applying production rules from a formal grammar, such as a shape grammar [Stiny 1975]. A key challenge is control: finding a sequence of rules—a program—that generates a structure satisfying a given specification [Aliaga et al. 2016]. This can be framed as a challenging mixed discrete-continuous optimization problem. A common strategy is to use global search algorithms to explore the discrete program structure while using local methods to optimize continuous parameters. Prominent approaches include Markov Chain Monte Carlo (MCMC) and its variants [Ritchie et al. 2015; Talton et al. 2011; Vanegas et al. 2012], which iteratively randomly walks over the production rules in search of better solutions. MCMC have also been applied to diverse domains like material graphs [Guo et al. 2020a], robots [Zhao et al. 2020], and sheet metal design [Barda et al. 2023]. Other search-based techniques include evolutionary algorithms [Krs et al. 2021] and analytical formal methods [Du et al. 2018; Ellis et al. 2018; Lau et al. 2011]. More recently, neural networks have been used to guide the selection of production rules [Plocharski et al. 2024; Ritchie et al. 2016]. Instead of proposing another sophisticated search algorithm, our work asks a more fundamental question: what properties must a grammar possess to make this search tractable for simple gradient descent?

Gradient-Based Optimization of Procedural Models. Recent work has explored making procedural models differentiable to enable gradient-based optimization. Several methods focus on optimizing the continuous parameters of a program, assuming a fixed discrete structure [Cascaval et al. 2022; Krs et al. 2021; Michel and Boubekeur 2021; Yuan et al. 2024]. While effective for local editing, these approaches cannot alter the program's topology. To address this, Liu et al. [2024] relax the discrete structure using fuzzy logic, allowing for global search over CSG trees. However, this relaxation means that intermediate shapes during optimization are not guaranteed to be valid. While gradient-informed MCMC methods exist [Wyse et al. 2012], they have not been applied to procedural modeling. Our Stochastic Rewrite Descent (SRD) algorithm, by contrast, exploits gradient information on both discrete and continuous components. While SRD is a straightforward adaptation of stochastic gradient descent (SGD), the challenge lies in making discrete changes descendable and avoiding local minima through careful grammar design.

Grammar Design and Generation. Grammar design plays a central role in program synthesis. For instance, the programming languages community often uses *grammar decomposition* to define concise subsets of programs and enable faster solvers; see [Cai et al. 2025] for a recent example in register allocation. There has also been significant

work on automating grammar design, including shape grammar inference from a single shape across several domains [Demir and Aliaga 2018; Guo et al. 2020b; Merrell 2023; Talton et al. 2012; Wu et al. 2013], and recent efforts focus on learning from sets of examples [Cao et al. 2023; Ellis et al. 2023; Jones et al. 2023].

Our work builds on this literature but introduces a key twist: while traditional approaches aim to minimize the number of rules to simplify the shape space, we show that enabling gradient descent often benefits from *more* rules—overparameterizing the space to make optimization easier. This may seem obvious in hindsight, but it marks a significant departure from standard grammar design principles.

Designing Optimization Landscapes. This work takes inspiration from optimization and machine learning to design grammars suitable for descent. In optimization, many properties have been identified for successful optimization of functions, from convexity and Lipschitz continuity to the Polyak-Łojasiewicz condition and many more [Karimi et al. 2016; Kurdyka 1998], which motivate our design of jump-continuous rewrites. Other work has focused on defining a continuous shape space with a well-behaved metric over the outputs of a grammar, enabling continuous analysis and optimization [Wang et al. 2018]. In the context of neural network optimization the role of overparameterization and stochastic descent for effective optimization in nonconvex landscapes has been widely studied [Jacot et al. 2018; Sankararaman et al. 2020], which inspires our redundant landscapes and stochastic optimization for grammars.

3 The Framework

In this section, we present our approach to procedural generation through gradient-based optimization. We first establish our design space and formalize the optimization problem we wish to solve. Then, we discuss the key grammar properties and our optimization strategy that effectively navigates the design space given some objective.

3.1 Problem Setup

Shape grammar. Distinct from string-based grammars, shape grammars [Stiny 1975] define their languages and rewrite rules directly on shapes. We define a parametric shape grammar as a tuple $G = (V, \Sigma, R, \omega)$ where

- V is a finite set of geometric primitives
- Σ is the set of formal parameters
- R is a finite set of rewrite rules $\rho : (x, p) \to (y, q)$ where $(x, p), (y, q) \in V^+ \times \Sigma^+$
- $\omega \in V^+ \times \Sigma^+$ is the initial shape (axiom)

Here, the set V^+ is formed by finite arrangements of primitives in V, where each primitive can appear multiple times. Each rewrite rule $\rho \in R$ may be guarded by a predicate ϕ_ρ which determines whether ρ can be applied.

Design Space. Let $S = V^+$ be the set of discrete structures generated by the parametric shape grammar G, where each structure $s \in S$ is associated with a vector of continuous parameters $p \in \mathbb{R}^{d(s)}$ where $d(s)$ is the dimensionality of structure s. The design space is the disjoint union of all (possibly different-dimensional) parameter spaces $X = \bigsqcup_{s \in S} \{s\} \times \mathbb{R}^{d(s)}$. A state $(s, p) \in X$ can be viewed as a particular derivation s from the grammar G together with a realized vector of parameters p.

Optimization Objective. We consider a differentiable *rendering* function $I : X \to \mathcal{D}$, which maps a state in the grammar to some representation of the shape, as well as a differentiable objective function $f : \mathcal{D} \to \mathbb{R}$ and a non-differentiable objective function $g : X \to \mathbb{R}$ (e.g., simplicity metric). For instance, I might yield an image for visual objectives or a mesh for a physics-based objective. Our problem can be formulated as finding (s^*, p^*) such that

$$(s^*, p^*) = \underset{s \in S, p \in \mathbb{R}^{d(s)}}{\arg\min} \; f(I(s, p)) + g(s, p). \tag{1}$$

This is a challenging optimization problem as we must search over both the discrete space of structures S and the space of continuous parameters whose dimensionality $d(s)$ varies with the structure.

3.2 Designing Optimization-Friendly Grammars

A typical descent-based optimizer utilizes gradient information to guide parameter changes. For shape grammars, the process also involves evaluating how rewrites affect the objective, hence the choice of rewrite rules impacts the optimization trajectory. In Section 3.3 we will describe one such optimization scheme, but first we consider an essential question common to any descent-like optimization scheme:

> What makes a shape grammar amenable to gradient-based optimization?

Indeed, even with a good optimization algorithm, naively performing descent on shape grammars for various tasks will typically explode into nonsensical shapes (e.g., when rewrites only increase the complexity) or get trapped in poor local minima. The issue is that the grammars themselves are poorly-suited as parameterizations for the optimization problem, but we argue that this can be greatly improved by designing grammars with optimization in mind. Some of these properties are simple and intuitive, while others are more subtle: we hope that systemically enumerating them will advance the use of descent-style procedures for optimizing shape grammars. The analysis is organized as a set of properties which grammars should satisfy (paragraph headings below), as well as a collection of concrete recommendations for designing grammars to achieve those properties (bolded inline, and summarized in Table 1).

Reachable Solutions. The most basic essential property is that it should be possible to reach a desired shape from any other shape in the space via some sequence of rewrites and parameter updates. For example, consider that many grammars are constructive, with rewrites to build-up a shape from an axiom, one primitive at a time. In such a constructive grammar, any shape is reachable from the axiom, but shapes are generally not reachable from other shapes, because there is no rewrite to remove a primitive—this is a huge impediment to gradient-based optimization, as mistakes may be impossible to correct. This is easily resolved via our first design principle: **Reversibility**. Any rewrite present in the grammar

Table 1. The suggested guidelines for designing grammars for descent. Section 3.2 systematical considers desirable properties, condensed here to a set of concrete recommendations. These are intended to be guidelines rather than hard requirements: they need not be strictly satisfied by all rewrites, it may even be contradictory to attempt to satisfy them all simultaneously for a given task. In Section 6 we experimentally validate the effect of these rules improving the efficacy of inverse optimization in several settings.

Design Guidelines	Description	Definition	Examples
Reversibility	If there is a rule $A \to B$, there should also be a rule $B \to A$	$\forall \rho \in R : \rho(x,p) = (y,q)$, $\exists \rho' \in R : \rho'(y,q) = (x,p)$	Split/Merge, Add/Remove-Loop
Jump Continuity	Applying rules incurs negligible instantaneous change in the shape	$\forall \rho \in R$ applicable to $(x,p) \in X$, $\left\lvert I(x,p) - I(\rho(x,p))\right\rvert < \epsilon$.	Local segment splitting
Local Geom. Control	There should exist rules which allow any local change to a shape without affecting distant parts	$\forall v \in V, \exists \rho \in R : \rho(v,p) = (w,q)$, where $\lvert w\rvert$ is small	Add-Anywhere
Repairability	If constraints exist, there should exist rules to repair a shape which violates the constraints	If $C \subset X$ is a feasible region, $\forall (x,p) \in X, \exists \rho \in R : \rho(x,p) \in C$	Resolve-Intersections

should have a complementary rewrite with the opposite effect. Contrasting with RJMCMC [Talton et al. 2011] where the ability to reach a desired shape from any other shape is granted by resampling derivation subtrees, reversibility becomes important for iterative gradient-descent-style optimization where the optimizer needs to incrementally correct small mistakes with fine-grained control.

Continuous-like Space. Shape grammars are an unusual space for optimization: a collection of local representations each with its own continuous parameters, linked by discrete rewrites which transform between structures. During optimization, these rewrites are potentially discontinuous jumps which can obstruct progress. We argue that for gradient-based optimization to be successful, the space should locally be as similar as possible to an ordinary continuous space. In fact, all of the principles considered here could be interpreted as aiming to make optimization in irregular, discretely-structured grammars more akin to optimization in a smooth, well-parameterized continuous space for which gradient-based optimization is already broadly effective.

In particular, this motivates the next grammar design principle which we call **Jump Continuity**. Rewrites should have negligible change on the shape itself, which in turn implies the objective (assuming it is a continuous function of the shape) also changes continuously. This allows optimization to proceed smoothly through the change of structure. This principle can also be understood through the lens of Lipschitz continuity, a key property in continuous optimization that the change in objective from any step should be bounded.

Redundant and Local Representations. Even if a grammar and a given target objective function $f(\cdot)$ behave as a nice continuous space under rewrites, this space is likely nonconvex, and riddled with local minima. Fortunately, in modern optimization it is increasingly common to optimize nonlinear, nonconvex objectives nonetheless, such as fitting neural networks via SGD. A key property for the practical success such optimization is overparameterization: that any desired solution should be representable by many possible configurations of parameters, and there should be many trajectories through the high-dimensional parameter space along which the objective improves. An analogous set of properties are beneficial for successful gradient-based descent on nonconvex shape grammar objectives. Ideally, there should be many structures which yield a desirable solution, and many possible chains of rewrites which would yield those structures. More precisely in the context of shape grammars, the grammar should offer **Local Geometric Control**: the ability to make localized changes anywhere on the shape without affecting distant parts. This allows the optimizer to explore the shape space effectively and helps avoid poor minima, particularly when the objective depends on local geometric features, creating an overparameterized space of many possible solutions.

Projection to Constraints. Many shape design problems include nontrivial constraints that must be satisfied, such as non-intersecting primitives. In continuous optimization, satisfying constraints is much easier if one has access to a projection operation to project onto the constraint set. Likewise, shape grammars subject to constraints benefit from **Repairability**: if constraints are present, rewrites should exist which project the design back to satisfying the constraint with minimal change. This frees the optimization process, and the grammar designer, to take steps which temporarily violate the constraints knowing that they will be re-enforced later, and further enhancing redundancy by permitting additional rewrite sequences which would otherwise be obstructed by constraints.

In general, we do not mean to imply that a grammar must always strictly satisfy each of these properties; rather they are guidelines that a grammar should follow as much as possible, to facilitate easy and effective optimization.

Remark. *One might attempt to demand even stronger guarantees from grammar design, where the objective becomes convex with respect to the rewrites, meaning that every local minimum is global and steepest descent through rewrites converges to global optimum in polynomial time. We show that such requirements are impossible for*

any grammar with an undecidable word problem (Thm. A.1 in supplemental). In general, shape grammars are Turing-complete [Stiny 1975], making its word problem undecidable. This negative result motivates us to design grammars for optimizers which have worked efficiently over highly nonconvex loss landscapes, such as SGD.

3.3 Optimization Over Grammar Space

We now present a particular optimization algorithm for shape programs which we call Stochastic Rewrite Descent, formalized in Algorithm 1 in supplemental. Recall that our goal is to optimize the composite loss $\mathcal{L}(s,p) = f(I(s,p)) + g(s,p)$ over the transdimensional design space $X = \bigsqcup_{s\in S}\{s\} \times \mathbb{R}^{d(s)}$. The key difficulty is that the *geometry* of X changes whenever a rewrite is applied: the parameter dimension jumps from $d(s)$ to $d(s')$. We therefore alternate *differential* updates in the current parameter block with *discrete* rewrites that switch between blocks.

Parameter Updates. Within a fixed structure s we perform gradient steps $p \leftarrow p - \eta\nabla_p f(s,p)$. This updates the continuous parameters only and does not change the structure s of the shape.

Discrete Rewrites. We randomly select K (64 in our implementation) valid rewrite rules, and for each rule $\rho \in R$, we compute an *improvement estimate*

$$\Delta\mathcal{L}_\rho \approx \big(\mathcal{L}(s,p) - \mathcal{L}(s',\hat{p})\big) \quad \text{with} \quad \hat{p} = \text{LocalOptimize}(s',p'),$$

where $(s',p') = \rho(s,p)$ and LocalOptimize performs one parameter update step under the new structure. While ideally we would apply all rewrites for which $\Delta\mathcal{L}_\rho$ is positive, this is not always feasible as applying one rewrite may invalidate others. To address this, we perform a greedy max-cover over the compatibility graph implicitly defined by the neighborhood $\mathcal{N}(s)$ (see Algorithm 2 in the supplemental for details).

4 Case Study Grammars

To validate our framework, we engineer three shape grammars, inspired by existing ones, to satisfy the recommended properties. We provide a brief description for each grammar. The rewrites for each grammar are summarized in Figure 2 and are listed in Appendix D in supplemental.

4.1 Tree Grammar

We design a shape grammar for trees inspired by traditional parametric L-systems for trees. Our Tree grammar is an upward growing binary tree where each node has a *rendered leaf*, the union of which defines the shape of the tree. Each non-root node stores the length l and turn angle θ relative to the parent. Every node stores the size of the rendered leaf r. We impose constraints on the maximum length, absolute turn angle, and rendered leaf size. The repair operation clamps length, turn angle, and rendered leaf size to their limits at the end of each continuous step, and removes leaves with small renderings.

4.2 Arc–Line (AL) Grammar

Circular arc and *straight line* primitives are important for creating 2D sketches in CAD systems. The Arc-Line Grammar can represent multiple, non-intersecting closed loops consisting of lines and

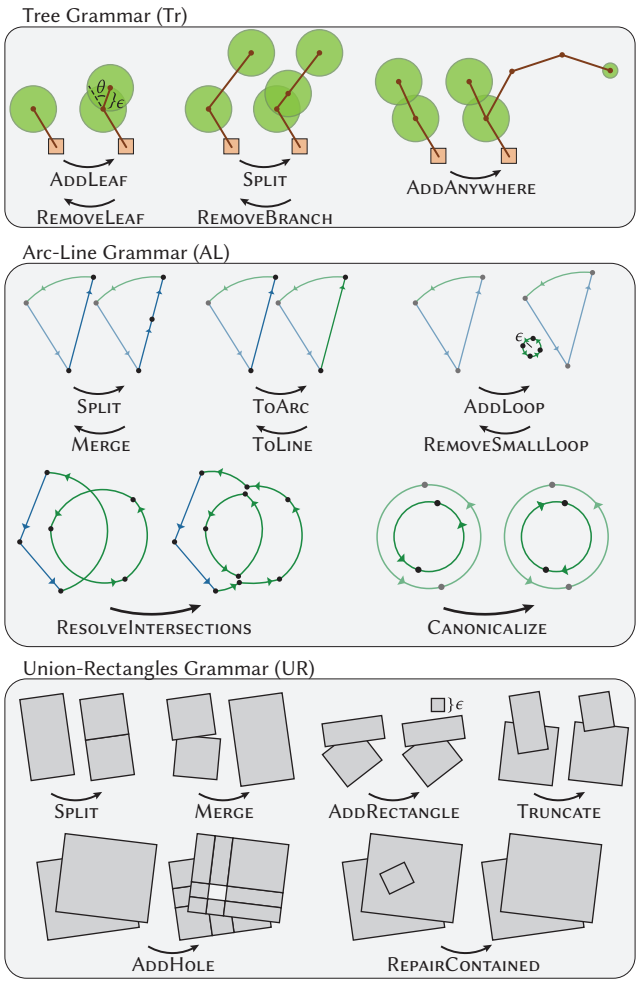

Fig. 2. The three representative shape grammars and their rewrites engineered to satisfy our guidelines. Tree grammar produces a binary tree where each node has a leaf (green circle). Arc-Line (AL) grammar produces shapes with multiple loops containing arcs and lines. The winding number determines the insideness. Union-Rectangles (UL) grammar produces rotatable and scalable rectangles.

circular arcs. Regions with winding number zero are considered outside and inside otherwise. A line and an arc are parameterized by the coordinates of their two endpoints (x_1,y_1) and (x_2,y_2) with an arc having an additional parameter k that controls the orthogonal deviation from the midpoint. Non-exact operations **Merge** and **ToLine** are permitted only if the deviation is less than ϵ. All other operations are exact up to numerical precision and do not change the shape. The repair operations compute intersections and reroutes intersecting loops so they no longer intersect, and also removes loops with small area.

4.3 Union-Rect (UR) Grammar

The Union-Rect Grammar represents a shape composed by unioning multiple rectangles. Each rectangle is parameterized by its midpoint

(c_x, c_y), its half-size s_x and s_y, and rotation angle θ. The repair operations shrink portions of rectangles that are covered by other rectangles in a way that does not change the union. This operation implies that fully enclosed rectangles get removed, as well as removing small rectangles.

5 Results and Applications

5.1 Optimization Towards A Target Image

We first evaluate our approach on an image matching objective. Specifically, given a shape (s, p), our objective is the L^2 distance between a soft rasterization $I(s, p)$ and a bitmap target I^*.

To obtain the target images for testing, we collect sketches from the SketchGraph dataset [Seff et al. 2020] and filter for only those that contain lines and arcs. We handpick the sketches and further group them to form three datasets based on their topology: (1) *OneComp* – a single closed loop, (2) *Donut* – two loops with one inside the other, and (3) *TwoComp* – two separated loops. The datasets contain 128, 25, and 23 shapes, respectively.

We show a representative set of results on all three grammars for this task in Fig. 4. While the grammars span fundamentally different design spaces of shapes, they all converge reliably under the same SRD optimizer. In the Tree grammar, gradient-guided rewrites insert, split, and occasionally retract branches until the canopies cover the target silhouettes, and the rewrites are able to distribute the branches organically. In the Union-Rect grammar, while the target image contain curved regions that are difficult to decompose into rectangular slabs, the rewrites spawns smaller rectangles that approximate these curved edges. Finally, for the Arc-Line grammar, despite the initial shape bounding a region with disk topology, the rewrites are able to alter the topology to create multiple connected components and extra boundaries.

In addition, we show results on morphing in Fig. 6 where we alter the target image during optimization. Due to the reversible, jump-continuous, and locally controllable rewrites (Section 3.2), the SRD optimizer is able to continue the optimization towards the new objective smoothly through deleting obsolete primitives, spawning new ones, and continuously updating the parameters.

5.2 Score Distillation Sampling (SDS) Loss

With the ability to directly optimize shapes based on gradient information, our algorithm can incorporate virtually any differentiable objective into our framework. To this end, we test our framework on Score Distillation Sampling (SDS) loss for text-to-shape generation. The SDS loss converts the denoising-score of a pretrained diffusion model into a gradient that pulls a parametrized geometry towards a given prompt.

Fig. 5 displays a gallery of shapes optimized using SDS loss on various text prompts. The optimized shapes are coherent with the text prompts. For instance the prompt "astronaut" yields a shape with a recognizable human and scattered rectangles that resemble stars, while the prompt "flower" produces an organic blossom with realistic petals and stem. The ability to add loops helps in materializing the eye in the "skull" and the horn in the "bull". For Union-Rect, the optimization utilizes the negative space to create the specular shading in "bottle" and to provide visual separation in "car". The difference in style between these two grammars also hint at the potential to control the characteristic of shapes via modifications of the grammar rules.

5.3 Topology Optimization

We explore the use of our framework in topology optimization, an engineering technique used to optimize structural integrity while reducing weight. Traditional methods are often density-based and, depending on the target fabrication technique, may require post-processing to convert results into CAD formats. Our approach allows direct optimization of this objective over a grammar-defined shape space, producing structures that are optimized for performance while remaining constrained to the primitives defined by the grammar.

Our approach to topology optimization is inspired by level-set methods—since the differentiable rasterizer produces a signed distance field from the shape grammar, they can be applied directly. We follow the implementation from [Wei et al. 2018], but propagate the gradient through our rasterizer instead of their level set parameters.

We evaluate two classic problems: the Cantilever beam, with the left edge fixed and a downward force applied at the right midpoint, and the MBB beam, with the bottom corners fixed in the y-direction and a downward force at the top midpoint. Figure 7 shows Arc-Line shape programs optimized for these objectives. These results demonstrate that our framework can synthesize shape programs for nontrivial objectives, enabling many applications.

6 Ablations and Comparisons

6.1 Properties of Shape Grammars

To study how different properties influences the optimization and convergence, we experiment with six variations of Tree grammar described in Table 2 (Tr-1–Tr-6). Notable differences from Tr-Full are that here the rendered leaf sizes are not optimizable and the root node is initialized at the bottom most white pixel of the target image. The AddLeaf* and RemoveLeaf* denote the same operations without ϵ-closeness check. We also refer readers to Fig. 8 for qualitative results.

Every additional capability either improves or maintains performance, corroborating the importance of the recommended properties. Our simplest grammar Tr-1, which models the L-system , do not satisfy any properties. The optimizer struggles to fill the lower right region as it is bounded by the angle constraint. In Tr-2, while the added **Reversibility** does not affect performance, it allows shapes to simplify and contain fewer nodes. This property is beneficial when doing morphing when targets have different area. We provide **Jump Continuity** in Tr-3. The optimizer succeeds in filling the region as taking small steps allows multiple nodes to bend at a higher angle. This also shown by the better loss across all datasets. In Tr-4, the ability to modify anywhere on the tree grants us **Local Geom. Control**. In Fig. 8 (Tr-4), we can see that branches are allowed to grow anywhere notably around the root where they previously cannot because those nodes already have two children. In Tr-5, the AddAnywhere operation allows the grammar to discover disconnected component and fill the entire target shape. This can also be seen in the significant improvements on Two

Table 2. Optimization quality (PSNR) and simplicity (number of primitives) using different grammar variations evaluated over OneComp (One), Donut (Dnt.), and TwoComp (Two) datasets.

		PSNR ↑			# Primitives ↓		
Gr.	Description	One	Dnt.	Two	One	Dnt.	Two
Tr-1	AddLeaf*	15.4	9.7	10.3	155	93	45
Tr-2	AddLeaf* + RemoveLeaf*	15.4	9.7	10.3	145	86	42
Tr-3	AddLeaf + RemoveLeaf	21.5	21.1	11.4	182	195	76
Tr-4	Tr-3 + RemoveBranch + Split	23.0	21.3	11.4	206	218	85
Tr-5	Tr-4 + AddAnywhere	23.2	21.0	14.6	197	208	121
Tr-F	Full	22.0	21.7	22.6	246	274	212
AL-1	no AddLoop	44.1	11.4	20.9	9	6	10
AL-F	Full	44.3	48.1	49.3	9	10	11
UR-1	no AddRect	26.3	26.7	20.3	11	13	9
UR-2	no AddHole	26.7	27.0	26.1	10	12	10
UR-F	Full	27.6	26.6	26.9	11	14	11

dataset. Note that we violate **Jump Continuity** because we fix the rendered leaf size. However, the optimization can still progress because the objective improvement from the new node inside new region exceeds the impediment from intermediate nodes.

We further show the effects of disabling **Local Geom. Control**-providing rewrites in Arc-Line (AL-1) and Union-Rect (UR-1 and UR-2). These grammars show similar performance on OneComp dataset as expected as they only contain a single component. While Arc-Line without AddLoop (AL-1) struggles with the Donut dataset since it cannot add an extra loop, removing AddHole from Union-Rect (UR-2) does not impact the performance on Donut because the splits and slight jittering causes the shape to break into multiple components. For both grammars, without the ability to add component (AL-1 and UR-1), the shape can be get stuck if the current shape does not intersect with the missing component.

6.2 Configurations of SRD

We now study the effects of the optimization when SRD only selects only the best rewrite in each step (denoted as OneRewrite in Table 2) and does not perform a continuous step to evaluate and rank rewrites (NoStep). Theoretically, OneRewrite should perform similarly but slower, and NoStep should prevent rewrites that exactly preserve or worsen the objective from being selected. For trees, the optimization progress is indeed much slower without much sacrifice on performance with OneRewrite, however it stops early with NoStep due to AddAnywhere being unable to land in the target region. For Arc-Line, with OneRewrite, the progress are drastically slower inducing our custom step size scheduler to reduce the step size even further. With NoStep, the AddLoop helps in making progress because it is not exact. For Union-Rect, these variations have little impact on performance and time because AddHole is not exact and Split introduce slight discrepancy around the cut edge. We refer readers to Table 7 in supplemental for results without AddLoop-like operations or step size scheduler.

6.3 MCMC comparison

We benchmark *reversible-jump MCMC* (RJMCMC) [Talton et al. 2011] against SRD for the Tree grammar. Since the method requires a context-free grammar to compute the jump probabilities efficiently, we use a simple branching context-free L-system which generates the same space of trees as the full version of our Tree grammar. While RJMCMC employs jump moves and diffusion moves that resamples derivation subtrees and their continuous parameters, it is unable to leverage gradient information from the objective. We note that the RJMCMC optimization maximizes the posterior $p(\delta \mid I^*) \propto L(I^* \mid \delta)\pi(\delta)$, where δ is the derivation tree, I^* is the target image, L is the likelihood function, and $\pi(\cdot)$ is the prior. Maximizing this term does not explicitly encourage simplicity of the derivation tree, thus we turn off the simplicity objective in our optimization for this comparison.

Table 3. Optimization quality (PSNR) and time (s) using different variations of SRD evaluated over OneComp (One), Donut (Dnt.), and TwoComp (Two) datasets.

		PSNR ↑			Time ↓		
Gr.	Description	One	Dnt.	Two	One	Dnt.	Two
Tr-F	NoStep	9.9	10.4	10.6	181	197	107
Tr-F	OneRewrite	19.4	17.5	18.4	1047	981	662
Tr-F	Full	22.0	21.7	22.6	385	437	291
AL-F	NoStep	30.7	24.3	31.8	118	154	107
AL-F	OneRewrite	40.0	19.1	24.0	90	126	102
AL-F	Full	44.3	48.1	49.3	80	106	107
UR-F	NoStep	25.1	24.8	24.7	85	89	57
UR-F	OneRewrite	26.9	27.0	26.4	85	109	80
UR-F	Full	27.6	26.6	26.9	88	106	75

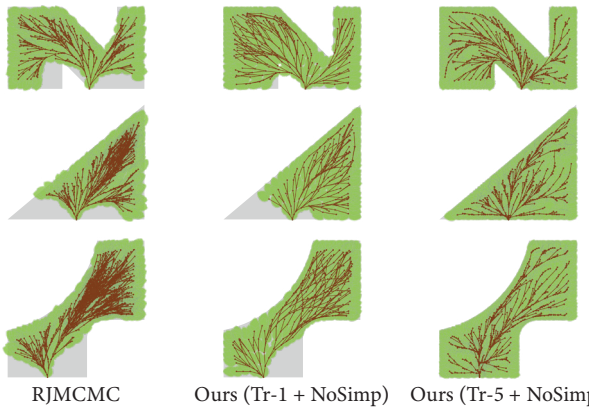

Fig. 3. Comparison with RJMCMC. RJMCMC and Tr-1 operate on the same grammar with only AddLeaf* and produce comparable results. Adding rewrites according to our guidelines significantly improves the results.

We compare RJMCMC against Tr-1 and Tr-5 grammar without simplicity weight. The PSNR are 15.3, 20.0, and 31.3; and the run times are 219, 19, and 99 minutes, respectively. As shown in Fig. 3, gradient guidance accelerates optimization and RJMCMC occasionally struggles to ensure full coverage of the target. In addition, despite turning off the simplicity objective, we are still able to find a more parsimonious representation than RJMCMC.

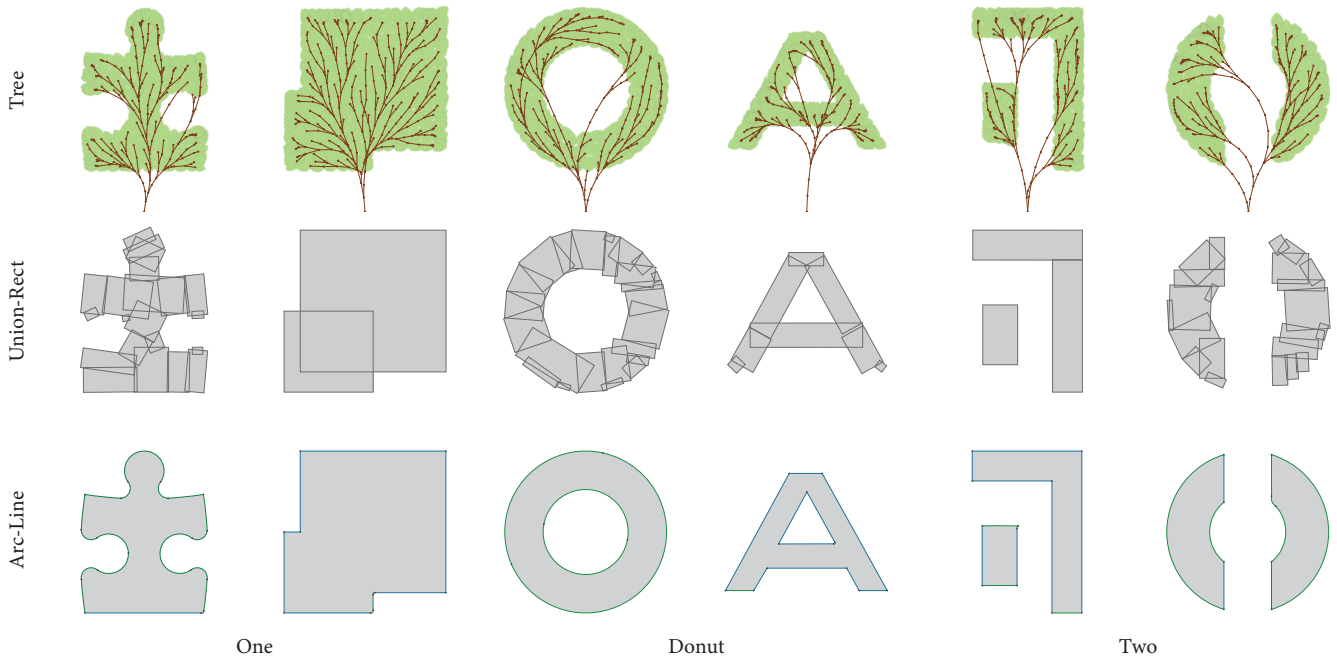

Fig. 4. Result of optimization towards a target image over tree grammar, grammar with rectangles, and grammar with connecting arcs and lines. The optimized shape matches the target image while also maintaining simplicity objective.

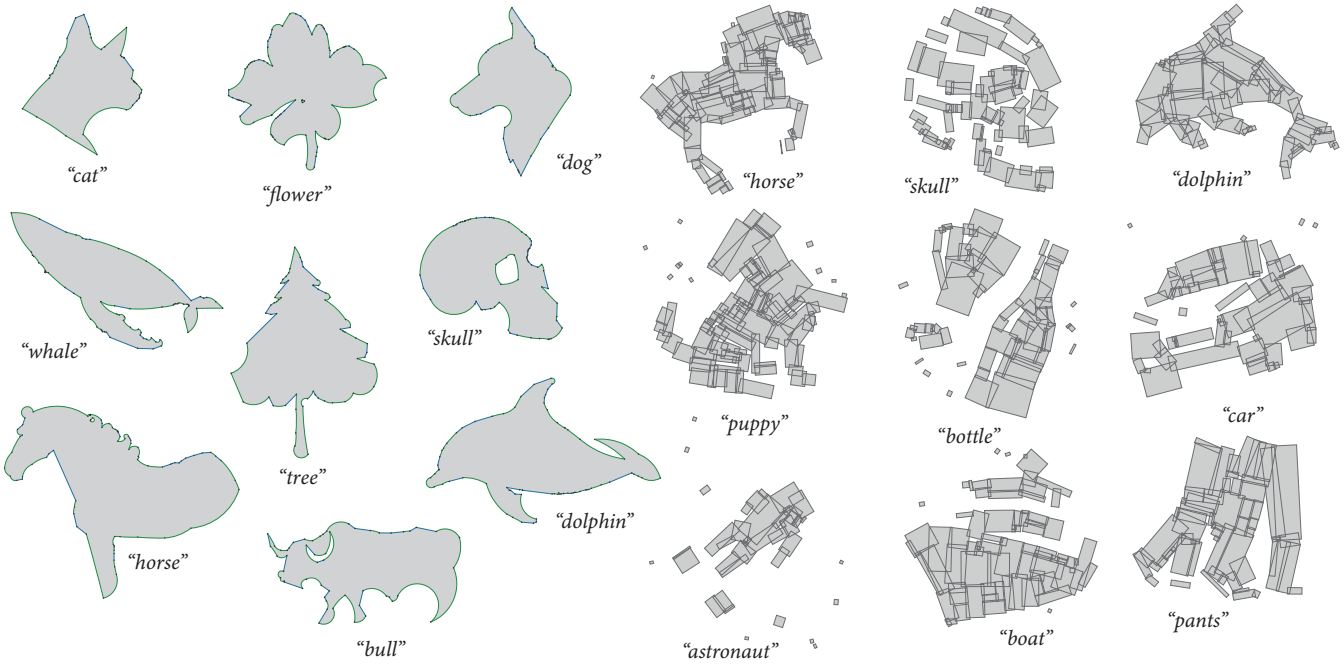

Fig. 5. Results of text-based optimization using Score Distillation Sampling (SDS) over grammars that (left) uses arcs and lines and (right) uses rectangles. The optimized shapes are coherent with the text prompts showing the versatility of our framework on different objectives. See Fig. 10 and 11 in the supplemental for additional results.

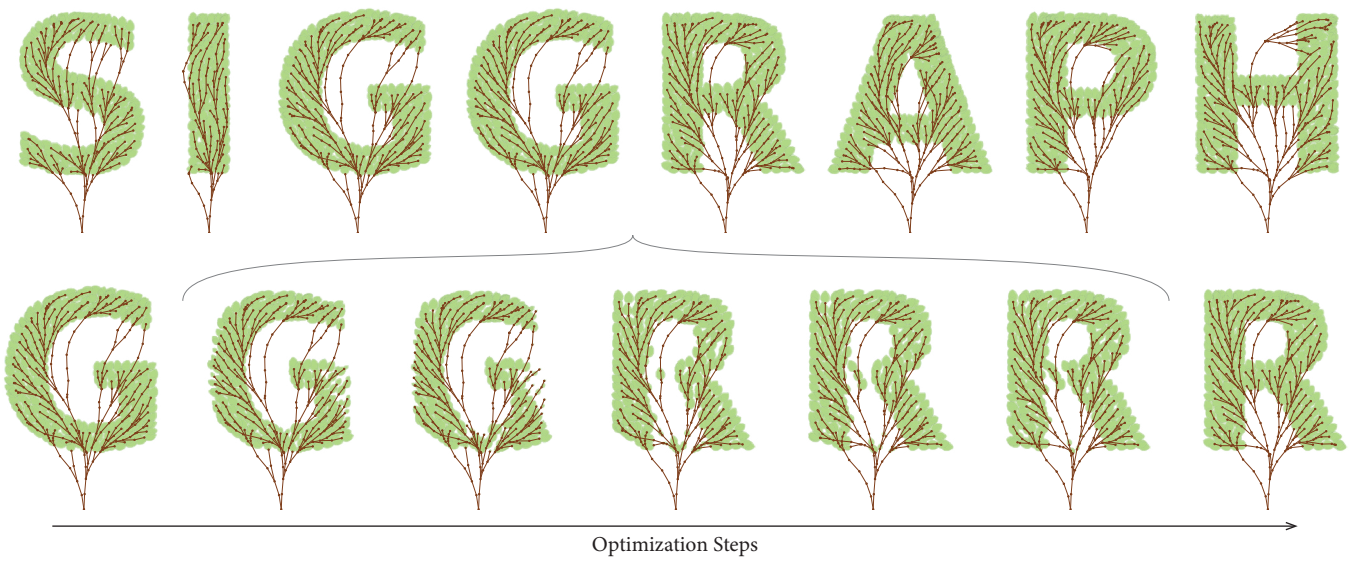

Fig. 6. Progression of trees optimized towards changing target images. The tree is first optimized for 'S'. The resulting tree is then optimized for 'I', and so on to spell SIGGRAPH. Bottom row shows sequential progression from 'G' to 'R'. The morphing ability shows our framework is less sensitive to initialization.

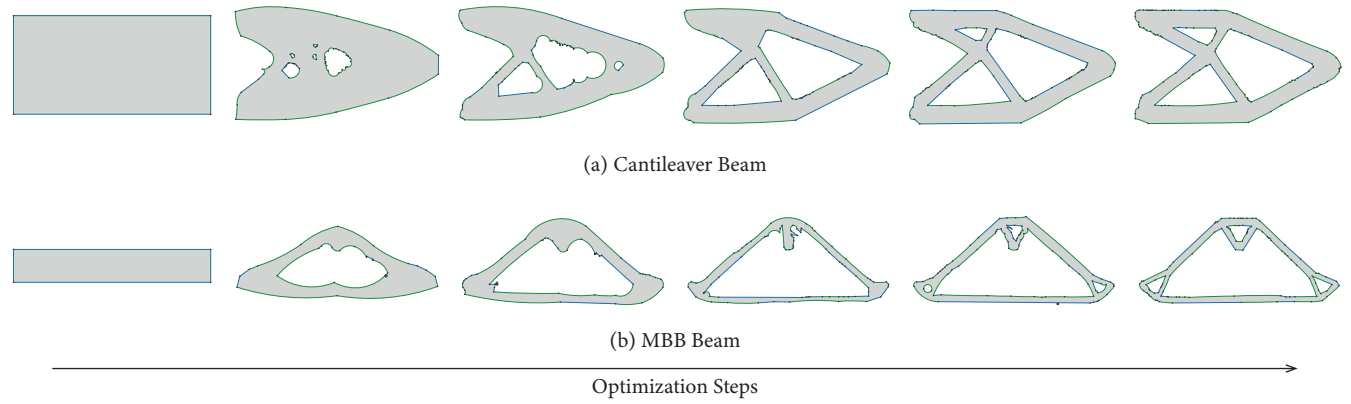

Fig. 7. Application of our framework for topology optimization over grammars that use arcs and lines. The Cantilever beam closely matches results from the literature. The MBB beam produces a structurally sound solution that, while different from typical published results, reflects the fact that we did not constrain the design to remain within the initial region.

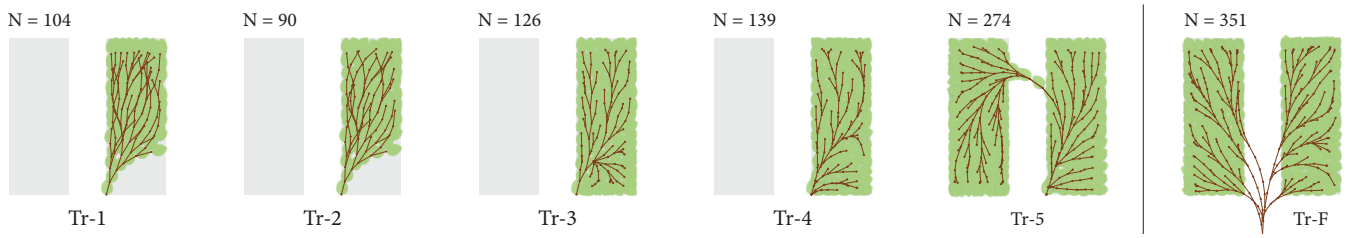

Fig. 8. Results using variation of tree grammars that are progressively conforming to our guidelines. Adding remove operation helps simplify the shape (Tr-2). Small rewrites help reach hard-to-reach regions (Tr-3). The ability to modify anywhere on the tree helps regularized the optimized structure (Tr-4). AddAnywhere helps discover disconnected regions (Tr-5). Tr-F is our full grammar with optimizable leaf sizes.

7 Limitations and Future work

Hyperparameters. Our descent approach inherits common limitations of gradient descent, such as sensitivity to step size and regularity of the gradients. For now, we address this using grammar-specific scaling factors and a step size scheduler. Future work could adapt advanced optimizers like Adam [Kingma 2014], but one must first determine how stateful optimizers (e.g., momentum) should be updated after a rewrite.

Distributions of rewrites. Our current algorithm assumes each rewrite type is sampled with roughly the same probability, and the continuous parameters of rewrites (e.g., positions to AddHole) are sampled uniformly randomly. Future work could tune this distribution based on the state of the shape and gradient information (e.g., through learning) to improve convergence speed.

Analysis. The fields of optimization and programming languages offer rich foundations that we have only begun to tap into. While our proposed properties are inspired by these theories and supported by experiments, there remains significant opportunity for formal analysis and deeper theoretical grounding.

Applications. On the application side, there are many domains where applying these ideas would be valuable. In particular, extending our method to richer 3D grammars—such as constructive solid geometry with boolean operations—is a promising direction. A key step toward enabling this broader applicability is automating grammar design. It would be interesting to build on work from the programming languages community, such as rewrite rule inference [Nandi et al. 2021], to automatically discover additional rewrite rules that help a grammar satisfy the properties needed for effective optimization.

Continuous Relaxations. An intriguing direction for future work is exploring fully continuous relaxations of discrete operations. Recent work has shown that in certain cases, discrete operations can indeed admit continuous relaxations—for example, Liu et al. [2024] demonstrates a unified differentiable boolean operator using fuzzy logic. While we have not explored this direction for general shape grammar rewrites, it is possible that some discrete operations could be similarly relaxed. This would align well with the spirit of our jump continuity property and could further bridge the discrete-continuous divide in shape grammars, though it is unlikely that discrete rewrites can be avoided entirely in general.

8 Conclusion

This work introduces a new perspective on shape program synthesis based on gradient descent. Through example applications and ablations, we show that gradient-based optimization can be effectively applied to shape grammars, and that grammar design plays a critical role in synthesis quality. It also opens a new direction for shape grammar research—bridging insights from optimization, programming languages, and graphics. From formalizing the theoretical foundations to developing better algorithms and extending to more complex domains, this space is rich with opportunity. Structure-aware shape processing is a cornerstone of computer graphics [Mitra et al. 2014], and grammars are among its most powerful representational tools. Enabling efficient optimization over them has the potential to unlock a wide range of applications in design, fabrication, and beyond.

Acknowledgments

This work is supported by the National Science Foundation under Grant No. 2212049 and 2219864 and the Sloan Research Fellowship. GPU compute is provided by NVIDIA Academic Grant Program.

References

Daniel G Aliaga, İlke Demir, Bedrich Benes, and Michael Wand. 2016. Inverse procedural modeling of 3d models for virtual worlds. In *ACM SIGGRAPH 2016 Courses*. 1–316.

Amir Barda, Guy Tevet, Adriana Schulz, and Amit Haim Bermano. 2023. Generative Design of Sheet Metal Structures. *ACM Transactions on Graphics (TOG)* 42, 4 (2023), 1–13.

Xuran Cai, Amir Kafshdar Goharshady, S Hitarth, and Chun Kit Lam. 2025. Faster Chaitin-like Register Allocation via Grammatical Decompositions of Control-Flow Graphs. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*. 463–477.

David Cao, Rose Kunkel, Chandrakana Nandi, Max Willsey, Zachary Tatlock, and Nadia Polikarpova. 2023. babble: Learning better abstractions with e-graphs and anti-unification. *Proceedings of the ACM on Programming Languages* 7, POPL (2023), 396–424.

D. Cascaval, M. Shalah, P. Quinn, R. Bodik, M. Agrawala, and A. Schulz. 2022. Differentiable 3D CAD Programs for Bidirectional Editing. *Computer Graphics Forum* 41, 2 (2022), 309–323. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14476 doi:10.1111/cgf.14476

Ilke Demir and Daniel G Aliaga. 2018. Guided proceduralization: Optimizing geometry processing and grammar extraction for architectural models. *Computers & Graphics* 74 (2018), 257–267.

Tao Du, Jeevana Priya Inala, Yewen Pu, Andrew Spielberg, Adriana Schulz, Daniela Rus, Armando Solar-Lezama, and Wojciech Matusik. 2018. Inversecsg: Automatic conversion of 3d models to csg trees. *ACM Transactions on Graphics (TOG)* 37, 6 (2018), 1–16.

Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Josh Tenenbaum. 2018. Learning to infer graphics programs from hand-drawn images. *Advances in neural information processing systems* 31 (2018).

Kevin Ellis, Lionel Wong, Maxwell Nye, Mathias Sable-Meyer, Luc Cary, Lore Anaya Pozo, Luke Hewitt, Armando Solar-Lezama, and Joshua B Tenenbaum. 2023. DreamCoder: growing generalizable, interpretable knowledge with wake–sleep Bayesian program learning. *Philosophical Transactions of the Royal Society A* 381, 2251 (2023), 20220050.

Jianwei Guo, Haiyong Jiang, Bedrich Benes, Oliver Deussen, Xiaopeng Zhang, Dani Lischinski, and Hui Huang. 2020b. Inverse procedural modeling of branching structures by inferring l-systems. *ACM Transactions on Graphics (TOG)* 39, 5 (2020), 1–13.

Yu Guo, Miloš Hašan, Lingqi Yan, and Shuang Zhao. 2020a. A bayesian inference framework for procedural material parameter estimation. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 255–266.

Arthur Jacot, Franck Gabriel, and Clément Hongler. 2018. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems* 31 (2018).

R Kenny Jones, Paul Guerrero, Niloy J Mitra, and Daniel Ritchie. 2023. Shapecoder: Discovering abstractions for visual programs from unstructured primitives. *ACM Transactions on Graphics (TOG)* 42, 4 (2023), 1–17.

Hamed Karimi, Julie Nutini, and Mark Schmidt. 2016. Linear convergence of gradient and proximal-gradient methods under the polyak-łojasiewicz condition. In *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 795–811.

Diederik P Kingma. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

Vojtěch Krs, Radomír Měch, Mathieu Gaillard, Nathan Carr, and Bedrich Benes. 2021. PICO: Procedural Iterative Constrained Optimizer for Geometric Modeling. *IEEE Transactions on Visualization and Computer Graphics* 27, 10 (Oct. 2021), 3968–3981. doi:10.1109/TVCG.2020.2995556

Krzysztof Kurdyka. 1998. On gradients of functions definable in o-minimal structures. In *Annales de l'institut Fourier*, Vol. 48. 769–783.

Manfred Lau, Akira Ohgawara, Jun Mitani, and Takeo Igarashi. 2011. Converting 3D furniture models to fabricatable parts and connectors. *ACM Transactions on Graphics (TOG)* 30, 4 (2011), 1–6.

Hsueh-Ti Derek Liu, Maneesh Agrawala, Cem Yuksel, Tim Omernick, Vinith Misra, Stefano Corazza, Morgan Mcguire, and Victor Zordan. 2024. A Unified Differentiable Boolean Operator with Fuzzy Logic. In *ACM SIGGRAPH 2024 Conference Papers* (Denver, CO, USA) *(SIGGRAPH '24)*. Association for Computing Machinery, New York, NY, USA, Article 109. doi:10.1145/3641519.3657484

Paul Merrell. 2023. Example-based procedural modeling using graph grammars. *ACM Transactions on Graphics (TOG)* 42, 4 (2023), 1–16.

Élie Michel and Tamy Boubekeur. 2021. DAG amendment for inverse control of parametric shapes. *ACM Trans. Graph.* 40, 4, Article 173 (July 2021), 14 pages. doi:10.1145/3450626.3459823

Niloy J Mitra, Michael Wand, Hao Zhang, Daniel Cohen-Or, Vladimir Kim, and Qi-Xing Huang. 2014. Structure-aware shape processing. In *ACM SIGGRAPH 2014 Courses*. 1–21.

Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. 2006. Procedural modeling of buildings. In *ACM SIGGRAPH 2006 Papers*. 614–623.

Chandrakana Nandi, Max Willsey, Amy Zhu, Yisu Remy Wang, Brett Saiki, Adam Anderson, Adriana Schulz, Dan Grossman, and Zachary Tatlock. 2021. Rewrite rule inference using equality saturation. *Proceedings of the ACM on Programming Languages* 5, OOPSLA (2021), 1–28.

Aleksander Plocharski, Jan Swidzinski, Joanna Porter-Sobieraj, and Przemyslaw Musialski. 2024. FaçAID: A Transformer Model for Neuro-Symbolic Facade Reconstruction. In *SIGGRAPH Asia 2024 Conference Papers* (Tokyo, Japan) *(SA '24)*. Association for Computing Machinery, New York, NY, USA, Article 123, 11 pages. doi:10.1145/3680528.3687657

Przemyslaw Prusinkiewicz. 1986. Graphical applications of L-systems. In *Proceedings of graphics interface*, Vol. 86. 247–253.

Daniel Ritchie, Ben Mildenhall, Noah D. Goodman, and Pat Hanrahan. 2015. Controlling procedural modeling programs with stochastically-ordered sequential Monte Carlo. *ACM Trans. Graph.* 34, 4, Article 105 (July 2015), 11 pages. doi:10.1145/2766895

Daniel Ritchie, Anna Thomas, Pat Hanrahan, and Noah Goodman. 2016. Neurally-Guided Procedural Models: Amortized Inference for Procedural Graphics Programs using Neural Networks. In *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (Eds.), Vol. 29. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2016/file/40008b9a5380fcacce3976bf7c08af5b-Paper.pdf

Karthik Abinav Sankararaman, Soham De, Zheng Xu, W Ronny Huang, and Tom Goldstein. 2020. The impact of neural network overparameterization on gradient confusion and stochastic gradient descent. In *International conference on machine learning*. PMLR, 8469–8479.

Ari Seff, Yaniv Ovadia, Wenda Zhou, and Ryan P. Adams. 2020. SketchGraphs: A Large-Scale Dataset for Modeling Relational Geometry in Computer-Aided Design. In *ICML 2020 Workshop on Object-Oriented Learning*.

Ruben M Smelik, Tim Tutenel, Rafael Bidarra, and Bedrich Benes. 2014. A survey on procedural modelling for virtual worlds. In *Computer graphics forum*, Vol. 33. Wiley Online Library, 31–50.

George Nicholas Stiny. 1975. *Pictorial and formal aspects of shape and shape grammars and aesthetic systems*. University of California, Los Angeles.

Jerry Talton, Lingfeng Yang, Ranjitha Kumar, Maxine Lim, Noah Goodman, and Radomír Měch. 2012. Learning design patterns with bayesian grammar induction. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*. 63–74.

Jerry O Talton, Yu Lou, Steve Lesser, Jared Duke, Radomír Mech, and Vladlen Koltun. 2011. Metropolis procedural modeling. *ACM Trans. Graph.* 30, 2 (2011), 11–1.

Carlos A. Vanegas, Ignacio Garcia-Dorado, Daniel G. Aliaga, Bedrich Benes, and Paul Waddell. 2012. Inverse design of urban procedural models. *ACM Trans. Graph.* 31, 6, Article 168 (Nov. 2012), 11 pages. doi:10.1145/2366145.2366187

Guan Wang, Hamid Laga, Ning Xie, Jinyuan Jia, and Hedi Tabia. 2018. The Shape Space of 3D Botanical Tree Models. *ACM Trans. Graph.* 37, 1, Article 7 (Jan. 2018), 18 pages. doi:10.1145/3144456

Peng Wei, Zuyu Li, Xueping Li, and Michael Yu Wang. 2018. An 88-line MATLAB code for the parameterized level set method based topology optimization using radial basis functions. *Structural and Multidisciplinary Optimization* 58, 2 (aug 2018), 831–849. doi:10.1007/s00158-018-1904-8

Peter Wonka, Michael Wimmer, François Sillion, and William Ribarsky. 2003. Instant architecture. *ACM Transactions on Graphics (TOG)* 22, 3 (2003), 669–677.

Fuzhang Wu, Dong-Ming Yan, Weiming Dong, Xiaopeng Zhang, and Peter Wonka. 2013. Inverse procedural modeling of facade layouts. *arXiv preprint arXiv:1308.0419* (2013).

J Wyse, N Friel, and M Girolami. 2012. Reversible jump Riemann Manifold Hamiltonian Monte Carlo. (2012).

Haocheng Yuan, Adrien Bousseau, Hao Pan, Chengquan Zhang, Niloy J. Mitra, and Changjian Li. 2024. DiffCSG: Differentiable CSG via Rasterization. In *Proceedings of ACM SIGGRAPH Asia (Conference track)*. ACM. http://www-sop.inria.fr/reves/Basilic/2024/YBPZML24

Allan Zhao, Jie Xu, Mina Konaković-Luković, Josephine Hughes, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. 2020. Robogrammar: graph grammar for terrain-optimized robot design. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–16.