Lattice-Based Generation of Euclidean Geometry Figures

Jonathan Henning, Hanna King, Sophie Ngo, Jake Shore, Alex Gardner, Chris Alvin, Grace Stadnyk
Furman University
Greensville, SC, USA
chris.alvin@furman.edu

Abstract

We present a user-guided method to generate geometry figures appropriate for high school Euclidean geometry courses: a useful starting point for an intelligent tutoring system to provide meaningful, realistic figures for study. We first establish that a two-dimensional geometry figure can be represented abstractly using a complete, lattice we call a *geometry figure lattice* (GFL). As input, we take a user-defined vector of primitive geometry shapes and convert each into a GFL. We then exhaustively combine each these 'primitive' GFLs into a set of complex GFLs using a process we call gluing. We mitigate redundancy in GFLs by introducing a polynomial-time algorithm for determining if two GFLs are isomorphic. These lattices act as a template for the second step: instantiating GFLs into a sequence of concrete geometry figures. To identify figures that are structurally similar to textbook problems, we use a discriminator model trained on a corpus of textbook geometry figures.

1 Introduction

A Euclidean geometry problem often consists of a narrative sequence of facts alongside a corresponding figure. It was shown in (Alvin et al. 2014) that, given a figure and a sequence of facts, many geometry problems can be simultaneously synthesized and solved using a directed hypergraph technique. In this work, we take one step back in this synthesis process and describe a *de novo* technique to synthesize two-dimensional geometry figures that are appropriate for Euclidean geometry problems.

Representation. Each 'primitive' geometry shape (e.g., square, right triangle, etc.) corresponds to a complete *geometry figure lattice* (GFL) encoding a topological abstraction: right triangle T corresponds to GFL L_T in Figure 1. In L_T we observe that the level 1 elements are constituent vertices (v_1, v_2, and v_3), rank 2 elements are sides of the triangle (s_1, s_2, and s_3), and triangle T is the only rank 3 element. More interesting geometry figures consist of several primitive shapes such as $L_{S,T}$ in Figure 1. The top element ($\top$) of $L_{S,T}$, represents the entire geometry figure while the bottom element of L_T, $\bot$, ensures a complete lattice. We now describe our user-guided process summarized in Figure 2.

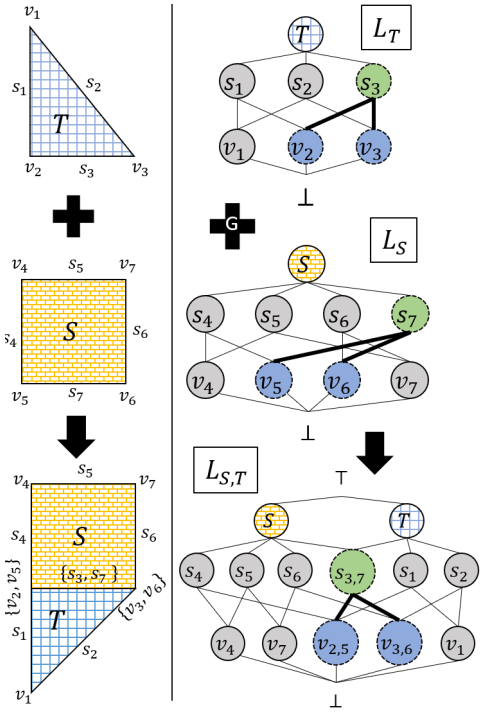


Figure 1: Segment gluing of two lattices: $L_T + L_S \to L_{S,T}$.

Lattice Generation. User-defined input consists of a number and type of primitive geometry shapes that will appear in the generated geometry figures. Generating a GFL is accomplished by combining a set of individual GFLs, each of which correspond directly to a shape indicated by the user. In Figure 1, a GFL L_T corresponding to right triangle T is combined with a GFL L_S corresponding to square S into $L_{S,T}$, a GFL acquired by 'gluing' s_3 in T to s_7 in S. $L_{S,T}$ is one way in which these two lattices may be combined; e.g., s_1 in T may be associated with any of the 'sides' in S. However, all associations of sides in T to sides in S lead to combinatorially equivalent geometry figures, and thus isomorphic GFLs. Hence, identifying isomorphic GFLs mitigates computational redundancy during figure generation. An important contribution of this paper is a polynomial-time

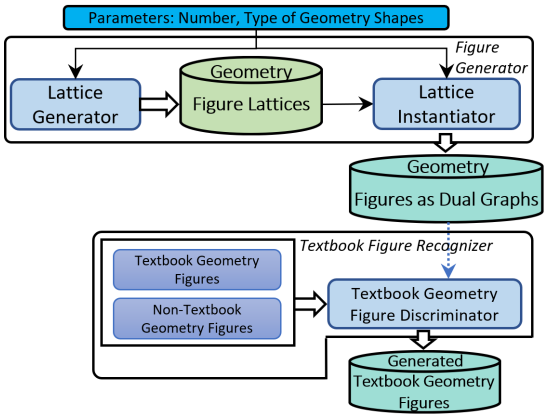


Figure 2: The geometry figure generation process.

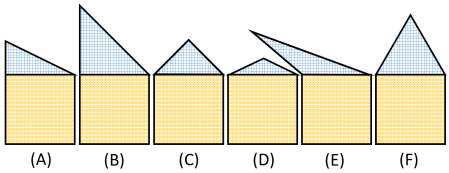


Figure 3: Generated figures combining a square and triangle: (A) non-isosceles right, (B) isosceles right, (C) isosceles right, (D) isosceles, (E) isosceles, and (F) equilateral.

algorithm to determine if two GFLs are isomorphic.

Textbook Figure Discriminator. Each GFL can be used to generate many figures. However, not every GFL serves as a template for geometry figures that would be found in a textbook. Using a corpus of geometry textbooks, we trained a graph neural network model to classify if a GFL is 'textbook'-like or not. As shown in Figure 2, we take each GFL, construct dual graphs corresponding to geometry figure templates, and pass each through the model to determine if it conforms to a typical textbook problem.

Figure Generation. Each GFL is a template: a unique structural abstraction for a set of geometry figures. For example, even though L_S in Figure 1 corresponds to a square, the lattice generally represents a sequence of 4 connected segments. Thus L_S may represent any standard concave or convex quadrilateral (e.g. trapezoid, rhombus, etc.); we restrict generated geometry shapes to be simple (not self-intersecting or crossing). If we fix S in $L_{S,T}$ to be a square, then any type of triangle can be 'glued' to create a valid geometry figure. Figure 3 depicts 6 unique geometry figures in using different types of triangles; each arise from our GFL abstraction technique and are topologically equivalent.

2 Lattice Representation

We assume standard definitions of *partially ordered set* (poset: $(P, \leq)$ with P a set and $\leq$ a relation on P), *meet* ($\wedge$), *join* ($\vee$), and *lattice* as defined in (Grätzer 1998). A *polyhedron* is a compact set of vertices in two dimensions.

A face of a polyhedron is a subset of vertices of the polyhedron that corresponds to a vertex, a segment, a polygon, or the empty set. The geometry figures we study are sets of polygons concatenated in particular ways. These sets are called polyhedral complexes.

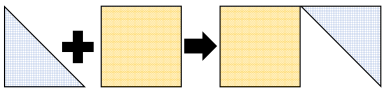


Figure 4: Point-based gluing of triangle and square.

Definition 1 (Polyhedral Complex)**.** A *polyhedral complex* Δ is a set of polyhedra that satisfies the following:

- If $D \in \Delta$ and σ is a face of D, then $\sigma \in \Delta$.
- If $D, D' \in \Delta$, then their intersection $D \cap D'$ is a face of both D and D'.

This definition ensures that we are not gluing, for example, a point from one primitive figure to the middle of a segment for another primitive figure. The final geometry figure in Figure 1 is an example of a polyhedral complex.

Definition 2 (Face Poset)**.** The *face poset* of a polyhedral complex Δ is the set of faces of Δ ordered by inclusion. If a unique top element does not exist, adding a top element, $\top$, yields a lattice called the *face lattice* of Δ.

For example, in Figure 1, L_S is the face lattice for the square S. We now have the mathematical tools to define a GFL.

Definition 3 (GFL)**.** A *geometry figure lattice* is the face lattice of a two-dimensional polyhedral complex.

To ensure we generate unique lattices, we define lattice isomorphism for use in our gluing operation.

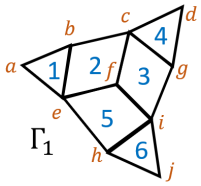
Definition 4 (Lattice Homomorphism and Lattice Isomorphism)**.** Let $(L, \leq)$ and $(K, \leq)$ be lattices, and $h : L \to K$. Then h is a *lattice homomorphism* if and only if for all $a, b \in L, h(a \vee b) = h(a) \vee h(b)$ and $h(a \wedge b) = h(a) \wedge h(b)$. A bijective lattice homomorphism is a *lattice isomorphism*.

3 GFL Generation with Gluing

We may construct a new polyhedral complex Δ^+ from two simpler polyhedral complexes Δ and Δ' by an operation we call *gluing*. This operation concatenates Δ and Δ' either along specified segments (as in Figure 1) or along specified vertices (as in Figure 4). The GFL for Δ^+ may be obtained directly from Δ^+, or it may be obtained from the GFLs for Δ and Δ' as we describe next.

Gluing GFLs. For an element e in poset $(P, \leq)$, we define the *downset* of e, notationally $\downarrow e$, to be the set of all elements in P less than e, that is, $\downarrow e = \{p \mid p \in P \text{ and } p \leq e\}$. As an example, for L_T in Figure 1, $\downarrow s_1 = \{s_1, v_1, v_2, \bot\}$.

Let $(L, \leq)$ and $(R, \leq)$ be GFLs. Suppose $\ell_i \in L$ and $r_i \in R$ with $\downarrow \ell_i$ isomorphic to $\downarrow r_i$ ($\downarrow \ell_i \cong \downarrow r_i$) for $1 \leq i \leq t$. The *gluing* operation $(\downarrow \ell_1, \ldots \downarrow \ell_t) \uplus (\downarrow r_1, \ldots \downarrow r_t) \to$ Res first combines each element in $\downarrow \ell_i$ with its isomorphic mapping in $\downarrow r_i$ into respective equivalence class structures. Let $[\ell_i, r_i]$ denote the set of these equivalence classes. For $1 \leq j \leq s$, consider elements m_j of $L \cup R$ that are not in any $\downarrow \ell_i$ or $\downarrow r_i$ for $1 \leq i \leq t$ as singleton elements in their own equivalence classes $[m_j]$. The gluing operation creates



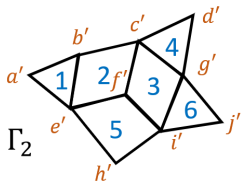


Figure 5: Γ_1 and Γ_2 are not combinatorially equivalent.

	1	2	3	4	5	6
a	1	0	0	0	0	0
b	1	1	0	0	0	0
c	0	1	1	1	0	0
d	0	0	0	1	0	0
e	1	1	0	0	1	0
f	0	1	1	0	1	0
g	0	0	1	1	0	0
h	0	0	0	0	1	1
i	0	0	1	0	1	1
j	0	0	0	0	0	1

Figure 6: $M(\Gamma_1)$.

	2	3	5	1	4	6
c	1	1	0	0	1	0
e	1	0	1	1	0	0
f	1	1	1	0	0	0
i	0	1	1	0	0	1
b	1	0	0	1	0	0
g	0	1	0	0	1	0
h	0	0	1	0	0	1
a	0	0	0	1	0	0
d	0	0	0	0	1	0
j	0	0	0	0	0	1

Figure 7: $M_2(\Gamma_1)$.

a poset consisting of the set of equivalence classes $\mathsf{Res} = \{[\ell_1, r_1], \ldots, [\ell_t, r_t], [m_1], \ldots, [m_s]\}$ with the order relation $x \leq y$ if and only if there exists $u \in x$ and $v \in y$ such that either $u \leq v$ in $(L, \leq)$ or $u \leq v$ in $(R, \leq)$.

The downset isomorphism requirement allows gluing only along like geometric constructs: point-to-point and segment-to-segment. For example, in Figure 1, since s_1 in L_T and s_4, s_5, s_6, s_7 in L_S each represent a segment, $\downarrow s_1$ is isomorphic to all the corresponding segment downsets in L_S: $\downarrow s_4, \downarrow s_5, \downarrow s_6$, and $\downarrow s_7$. We observe in $L_{S,T}$ 3 equivalence classes with more than one element from the operation $\downarrow s_1 \uplus \downarrow s_6 \to L_{S,T}$, namely $s_{1,6} = \{s_1, s_6\}$, $v_{2,6} = \{v_2, v_6\}$, and $v_{1,7} = \{v_1, v_7\}$. It is not necessary to create an equivalence class with multiple $\perp$ elements.

Theorem 1. *Suppose L is a GFL with $\ell_i \in L$, R is a GFL with $r_i \in R$, and $\downarrow \ell_i \cong \downarrow r_i$, where $1 \leq i \leq t$. Suppose $(\downarrow \ell_1, \ldots \downarrow \ell_t) \uplus (\downarrow r_1, \ldots \downarrow r_t) \to \mathsf{Res}$. The poset $(\mathsf{Res}, \leq)$ is a GFL.*

4 GFL Isomorphism

For a geometry figure Γ, the *vertex-facet incidence matrix* $M(\Gamma)$ is the matrix with columns corresponding to the polygons that are glued together to form Γ and rows corresponding to the vertices. The matrix $M(\Gamma)$ has a 1 in the ij entry if the vertex corresponding to the ith row is a vertex of the polygon corresponding to the jth column. For example, $M(\Gamma_1)$ (Figure 6) corresponds to figure Γ_1 (Figure 5).

Recall that two polyhedral complexes are *combinatorially equivalent* if they have isomorphic face lattices.

Lemma. *Two geometry figures, Γ_1 and Γ_2 are combinatorially equivalent if and only if $M(\Gamma_1)$ can be obtained from $M(\Gamma_2)$ by a sequence of row and column swaps.*

Example. Our goal is to use Algorithm 1 to transform $M(\Gamma_1)$ (Figure 6) into its *standard form*, $\widehat{M}(\Gamma_1)$ (Figure 11).

Let M be a matrix, r be a row, and c and d are columns.

- Let $rs(r)$ be the row sum of r and $cs(c)$ be the column sum of c.
- If $rs(r) = k$, let $u(r)$ be the $k-$tuple whose ith entry gives the column sum of the column containing the ith 1 in r. This is the *vector of column sums*.
- If $rs(r) = k$, let $p(r)$ be the $k-$tuple whose ith entry gives the index of the column containing the ith 1 in r. This is the *vector of column indices*.
- If $cs(c) = k$, let $v(c)$ be the $k-$tuple whose ith entry gives the row sum of the row containing the ith 1 in c. We will call this vector the *vector of row sums*.
- For any vector w, define a *group of tied columns*, as $A_w = \{c : v(c) = w\}$. For any A_w with $|A_w| = 1$, we will say A_w is a *trivial* group of tied columns.
- Let $s(c, d)$ be the number of rows that have a 1 in both column c and column d.
- For $A = \{a_0, a_1, \ldots\}$ an index-based ordered, proper subset of the columns of M and d a column of M not in A, we define the *similarity vector* as $q_A(d) = (\, s(a_i, d)\,)_{i=0}^{|A|-1}$. We define $q_{\{\}}(c) = (0)$ for any column c.
- S is an ordered set of columns induced by the relative order on the corresponding columns in the matrix.

Figure 8: Definitions for Algorithm 1.

	2	3	5	1	4	6
f	1	1	1	0	0	0
c	1	1	0	0	1	0
e	1	0	1	1	0	0
i	0	1	1	0	0	1
b	1	0	0	1	0	0
g	0	1	0	0	1	0
h	0	0	1	0	0	1
a	0	0	0	1	0	0
d	0	0	0	0	1	0
j	0	0	0	0	0	1

Figure 9: $M_3(\Gamma_1) = M_4(\Gamma_1)$.

	2	5	3	1	4	6
f	1	1	1	0	0	0
c	1	0	1	0	1	0
e	1	1	0	1	0	0
i	0	1	1	0	0	1
b	1	0	0	1	0	0
g	0	0	1	0	1	0
h	0	1	0	0	0	1
a	0	0	0	1	0	0
d	0	0	0	0	1	0
j	0	0	0	0	0	1

Figure 10: $M_5(\Gamma_1)$.

Steps 1 and 2. We sort rows by row sums and columns by column sums; order of these steps is arbitrary. We observe the result of $M(\Gamma_1) \to M_1(\Gamma_1) \to M_2(\Gamma_1)$ in Figure 7.

Step 3. We establish groups of columns each having the same column sum. For each group having column sum k, and each column within this group, construct the vector of row sums. For a column within this group, this is a vector of length k whose ith entry gives the row sum of the row containing the ith 1 in the column. We then sort the columns within this group by their vector of row sums in weakly decreasing lexicographic (dictionary) order. For example, consider $M_2(\Gamma_1)$ in Figure 7. The vector of row sums for the column '2' is given by $v(\text{'2'}) = (3, 3, 3, 2)$ since rows 'c', 'e', and 'f' have sum 3 and row 'b' has sum 2. Similarly, $v(\text{'3'}) = v(\text{'5'}) = (3, 3, 3, 2)$. No column swaps occur since each vector of row sums is equal.

Algorithm 1: Vertex-facet incidence matrix standardization: $M(\Gamma) \to \widehat{M}(\Gamma)$

$M(\Gamma)$: vertex-facet incidence matrix for geometry figure Γ.

1. Sort the rows of $M(\Gamma)$ by row sum in weakly decreasing order to yield $M_1(\Gamma)$.
2. Sort the columns of $M_1(\Gamma)$ by column sum in weakly decreasing order to yield $M_2(\Gamma)$.
3. In $M_2(\Gamma)$, for all possible k, sort the set of columns each having column sum k by $v(c)$ in weakly decreasing lexicographic (dictionary) order. This yields $M_3(\Gamma)$.
4. In $M_3(\Gamma)$, for all possible k, sort the set of rows each having row sum k by $u(r)$ in weakly decreasing lexicographic order. This yields $M_4(\Gamma)$.
5. Initialize $S = \{c : c \in A_w, |A_w| = 1 \text{ for some } w\}$.
   while not all columns of $M_4(\Gamma)$ are in S:
   let m be the set of columns in $M_4(\Gamma)$ not in S
   for each group of tied columns A_{w_i} in m (taken left-to-right):
   for all $c \in A_{w_i}$, establish a lexicographically decreasing order of all columns in A_{w_i} using $q_S(c)$; call this order O
   add to S, at the same indices of A_{w_i} in $M_4(\Gamma)$, all columns in O up to, but excluding, the first lexicographic equality of $q_S(c)$ for all $c \in O$
   If S was not changed by any tied column group, add the left-most column c of the first group of tied columns to S at the same index as c in $M_4(\Gamma)$
6. In $M_5(\Gamma)$, for all possible $k-$tuples v, sort the set of rows each having $u(r) = v$ by the vector $p(r)$ in increasing lexicographic order. This yields $\widehat{M}(\Gamma)$.

Step 4. Similar to Step 3, our goal is to sort rows within groups of rows having the same row sum. We do so by sorting by the vector of column sums in weakly decreasing lexicographic order. Consider $M_2(\Gamma_1)$ in Figure 7. The vector of column sums for the row labeled f is given by $u(\text{`}f\text{'}) = (4,4,4)$ since columns '2', '3', and '5' all have column sum 4. We thus move row $f$ to be first as all other rows with row sum 3 (rows '$c$', '$e$', '$i$') have $u(\text{`}c\text{'}) = u(\text{`}e\text{'}) = u(\text{`}i\text{'}) = (4,4,3)$ and $(4,4,4) > (4,4,3)$ in lexicographic order. The order of Steps 3 and 4 is arbitrary.

Step 5. This step reorders columns by iteratively constructing an ordered set S using similarity vectors. It is important to note that when a column is added to S, the column is in its final, indexed position. In this step of the algorithm, we compute similarity scores using S; in particular, we calculate $q_S(c)$ where c will be a column in $M_4(\Gamma_1)$ that is not in S.

S is initialized with trivial groups of tied columns. In Figure 9, there are no trivial groups of tied columns so we have $S = \{?,?,?,?,?,?\}$ (a '?' indicate columns to be populated in future iterations) although we consider S to be empty. There are two groups of (nontrivial) tied columns

	2	5	3	1	4	6
f	1	1	1	0	0	0
e	1	1	0	1	0	0
c	1	0	1	0	1	0
i	0	1	1	0	0	1
b	1	0	0	1	0	0
h	0	1	0	0	0	1
g	0	0	1	0	1	0
a	0	0	0	1	0	0
d	0	0	0	0	1	0
j	0	0	0	0	0	1

Figure 11: $M_6(\Gamma_1) = \widehat{M}(\Gamma_1)$.

	3	2	5	4	6	1
f'	1	1	1	0	0	0
c'	1	1	0	1	0	0
i'	1	0	1	0	1	0
g'	1	0	0	1	1	0
e'	0	1	1	0	0	1
b'	0	1	0	0	0	1
h'	0	0	1	0	0	0
d'	0	0	0	1	0	0
j'	0	0	0	0	1	0
a'	0	0	0	0	0	1

Figure 12: $\widehat{M}(\Gamma_2)$.

$A_{w_1} = \{\text{`}2\text{'}, \text{`}3\text{'}, \text{`}5\text{'}\}$ and $A_{w_2} = \{\text{`}1\text{'}, \text{`}4\text{'}, \text{`}6\text{'}\}$, where $w_1 = (3,3,3,2)$ and $w_2 = (3,2,1)$. For our first step, as $S = \emptyset$, $q_S(c) = (0)$ for all c in $M_4(\Gamma_1)$. We take, by default, the leftmost column not in S, '2', and add it to S in its same index position, yielding $S = \{\text{`}2\text{'}, ?, ?, ?, ?, ?\}$.

In the next pass, we modify A_{w_1} to remove column '2' since it was added to S, leaving us with $A_{w_1} = \{\text{`}3\text{'}, \text{`}5\text{'}\}$. Sequentially, with each group of tied columns, we compute similarity vectors for columns not in S. Thus, for our updated A_{w_1} we have $q_S(\text{`}3\text{'}) = (2) = q_S(\text{`}5\text{'})$. As these vectors are lexicographically equal, no changes are made to S. We then compute similarity vectors for the next group of tied columns (excluding those in S), $A_{w_2} = \{\text{`}1\text{'}, \text{`}4\text{'}, \text{`}6\text{'}\}$. The similarity vectors are $q_S(\text{`}1\text{'}) = (2)$, $q_S(\text{`}4\text{'}) = (1)$, and $q_S(\text{`}6\text{'}) = (0)$. As these similarity vectors are in lexicographically decreasing order (with no ties), we add each column to S at the same index at which they occur in Figure 9. We now have $S = \{\text{`}2\text{'}, ?, ?, \text{`}1\text{'}, \text{`}4\text{'}, \text{`}6\text{'}\}$

For our final pass, we have one group of tied columns not in S: $A_{w_1} = \{\text{`}3\text{'}, \text{`}5\text{'}\}$. We compute similarity scores for these columns: $q_S(\text{`}3\text{'}) = (2,0,2,1)$ and $q_S(\text{`}5\text{'}) = (2,1,0,2)$. Since $q_S(\text{`}3\text{'})$ is lexicographically less than $q_S(\text{`}5\text{'})$, we insert these columns into S so that '5' occurs first. We conclude this step of Algorithm 1 since all columns of $M_4(\Gamma_1)$ are in $S = \{\text{`}2\text{'}, \text{`}5\text{'}, \text{`}3\text{'}, \text{`}1\text{'}, \text{`}4\text{'}, \text{`}6\text{'}\}$. We reorder the columns of $M_4(\Gamma)$ so they coincide with the order given by S. This gives $M_5(\Gamma)$.

Step 6. The final step orders all rows with a given vector of column sums in increasing lexicographic order based on tuples of column indices. Consider Figure 10. With indexing starting at 0, row 'c' has index tuple $(0,2,4)$ while 'e' has index tuple $(0,1,3)$. Since $(0,1,3) < (0,2,4)$, we move 'e' before 'c' in $\widehat{M}(\Gamma_1)$. Similarly, row 'h' with index tuple $(1,5)$ is shifted before row 'g' with index tuple $(2,4)$. The result is $\widehat{M}(\Gamma_1)$ in Figure 11.

Conclusion of Example. We conclude that Γ_1 and Γ_2 in Figure 5 are not combinatorially equivalent as $\widehat{M}(\Gamma_1) \neq \widehat{M}(\Gamma_2)$ and thus their GFLs are distinct.

5 Filtering Toward Textbook Figures

Figure Representation. Each geometry figure is encoded as a *dual graph* where each node is labeled with the specific primitive shape, except a 'universe' node corresponding to

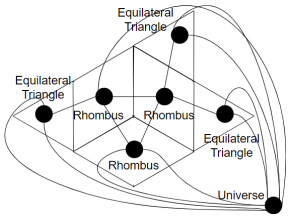


Figure 13: A dual graph for a figure generated using the GFL corresponding to Γ_2 from Figure 5.

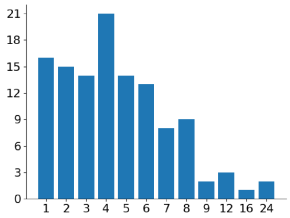


Figure 14: Number of primitive shapes (horiz. axis) by subset size (vertical axis) of corpus figures.

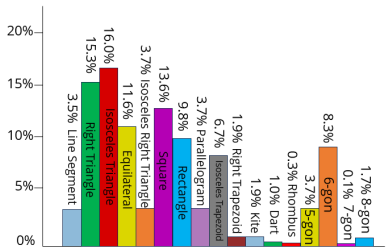


Figure 15: Occurrence rate of primitive shapes in our corpus.

the rest of the plane exterior to the figure. This representation can be observed in Figure 13: Γ_2 (Figure 5) is encoded with 6 labeled face nodes along with a universal face node.

Textbook Figures. Our corpus was created by an exhaustive analysis of 8 textbooks (Jurgensen, Brown, and Jurgensen 1951; Larson, Boswell, and Stiff 2004; Rhoad, Milauskas, and Whipple 1991; Gantert 2007; Michael 2002; Posamentier and Salkind 1996; Larson et al. 2009; Cord 2022). Each of the 118 figures has a unique dual graph and may contain any subset of the primitive shapes in Figure 15.

The most common textbook figure consists of a single primitive shape; many of these arise when introducing fundamental properties of primitive shapes (e.g., opposing parallel sides in a parallelogram). Comparing textbooks chapter by chapter (e.g., triangle chapters of all texts), we observe common figures. Such figures are so common when introducing similar content, we might refer to them as canonical, pedagogical figures. As an example, the most common figure among textbooks is two right triangles sharing a common side. This figure appeared 401 times compared to the next most repeated figure occurring 203 times.

We observe in Figure 14 many figures are built from 1, 2, or 4 primitive shapes. Triangles are also often used to construct figures: it is common for a quadrilateral to be constructed with 2 or 4 triangles. Figures with 4 or more primitive shapes are more likely to be unique to a textbook.

Discriminator model. We created a discriminator model[1] (Figure 17) based on our corpus of textbook figures to filter for textbook-like figures. For training, we used the 118

[1]https://github.com/hannaking/Summer-Research-2023

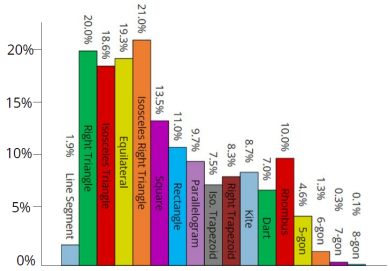


Figure 16: Occurrence rate of primitive shapes in figures selected as textbook-like over 100000 random dual graphs.

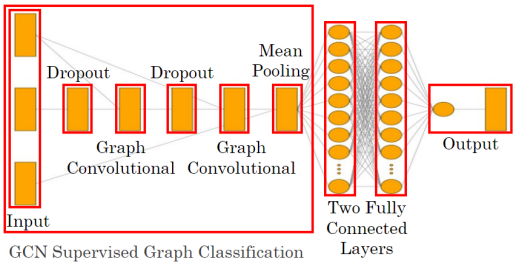


Figure 17: Architecture of the textbook figure discriminator.

textbook dual graphs as positive examples and a corresponding 118 unique dual graphs generated randomly as negative examples. The `random_search` function in the `keras_tuner` library was used to tune sizes and activation functions of each layer, dropout, and learning rate. The model achieved 62% accuracy on the textbook graphs, 83% accuracy on the random non-textbook graphs and 75% accuracy under 10-fold stratified cross-validation with 5 repetitions. We attribute our lower accuracies to small sample size and general lack of uniqueness in textbook figures (see Section 7). Using a quantile-quantile, we compared the distribution of frequencies of primitive shapes from textbook problems (Figure 15) to the distributions of primitive shapes from randomly generated figures (Figure 16). With $r^2 = 0.986$ we are confident the model can discriminate for figures found in a textbook.

6 Geometry Figure Generation

For a GFL L, we generate candidate geometry figures by taking each shape in sequence at rank 3 in L and projecting that shape onto the Cartesian plane according to the properties of the shape. All figures begin at the origin and, if a distance is not known, 1 is used. If the user requests 1 square and 1 triangle but does not specify the type of triangle, our algorithm generates all options as shown in Figure 3 using all possible triangles; similarly for quadrilaterals.

As an example of our process, we generate the geometry figure with square S and right triangle T from Figure 1. As input, a user indicates a square and an isosceles right triangle (with no other primitive shapes). We begin with the square. Since the projection is empty, the origin is assigned as a vertex of the square S with default side length 1. The remaining 3 vertices are determined according to the geometric prop-

Table 1: Results of generated textbook-like figures satisfying user stated queries.

Shape Queries	Lattices	Dual Graphs	Figures
4 × Segment	3	3	6,451
Triangle + Triangle	2	32	53,932
Triangle + Quadrilateral	3	96	122,389
Quadrilateral + Quadrilateral	3	3	222
2 × Equilateral Triangle + Dart	16	16	14,037
2 × Equilateral Triangle + Rhombus	16	16	4,003
3 × Equilateral Triangle	6	6	739
Triangle + Dart	3	12	24,327
Isosceles Right Triangle + Isosceles Trapezoid	3	3	933
2 × Triangles where either is Equilateral or Isosceles	2	6	1,568

erties of a square. We then glue the triangle T along a side of S. Since 2 vertices of T are defined, the final point is determined according to the rules of a $45-45-90$ triangle; there are 3 possible coordinate pairs for the last point of T since the 'glued' side may be the hypotenuse or a leg of T. To generate other primitive shapes, we use the following:

1. Default length of 1 when needed,
2. Rules of geometry for a shape (e.g., if two vertices are known for a right triangle a third point can be computed),
3. Standard angles of the unit circle (i.e., $30°$, $45°$, and $60°$) when a choice is required, and
4. The growing set of fixed vertices in the plane (i.e., the first shape has no fixed vertices, but as more shapes are projected onto the plane, more vertices become fixed).

7 Experimental Analyses

Analysis: Discriminating for numbers of primitive shapes. We compared the distributions of the number of primitive shapes per figure of the textbook versus textbook-like figures. Both distributions follow an exponential decay pattern as can be observed in Figure 14. The Mann-Whitney U Test returned a p-value of 0.0000021 indicating the distributions are likely the same. This serves as validation of the discriminator model's ability to recognize figures consistent with textbook problems in terms of the number of shapes.

Analysis: Shape-shape edges. Factoring out biases from shape counts and sizes of different shapes, we ran an experiment to determine the likelihood of a generated textbook-like figure having two shapes sharing a side. We found figures are more likely to have shared sides between shapes with different numbers of sides. For example, there is a 54.1% chance that an equilateral triangle shares a side with a regular pentagon, but only a 25.1% chance that an isosceles triangle shares a side with an equilateral triangle.

Experiment: Targeted figure generation. The strength of our proposed pipeline for figure generation is that any request from a user is satisfied. Table 1 serves as a sample of the efficacy of our pipeline. One of the 53932 figures generated from a query consisting of 'triangle, triangle' is shown as the top-middle figure in Figure 18. In general, our technique generates few GFLs constrained by isomorphism, but is still able to generate a significant number of dual graphs from GFLs, and even more textbook-like figures satisfying user constraints.

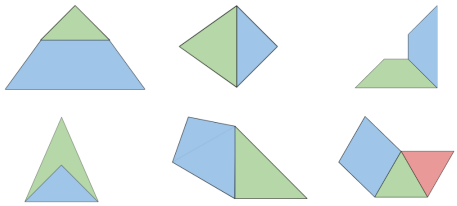


Figure 18: Sample generated, textbook-like figures.

8 Related Works

Many synthesis approaches attempt to generate structures and solids in three-dimensions. Scan2Mesh (Dai and Nießner 2019) is a generative neural network that takes 3D objects as input and outputs a mesh. By contrast our work seeks to generate geometry figures with specific properties: right angles, isosceles triangles, etc. By contrast, our approach is query-based (not random): if a user requests a particular set of polygons, they will receive a figure with those shapes. Another notable technique is proposed in (Simonovsky and Komodakis 2018) and implemented as GraphVAE. GraphVAE performs a topological generation approach for small graphs, a technique that was applied to 2D-based molecule generation. Our lattice-based approach guarantees mathematically sound geometry figures whereas GraphVAE is a probabilistic approach that does not, for example, generate valid molecules.

Geometrically, the closest works to ours are a sequence of works from Alvin, et al. that generate and solve geometry proof problems (Alvin et al. 2014) and shaded area problems (Alvin et al. 2017a; 2017b). In particular, (Alvin et al. 2017a) attempts to combine shapes with computable areas into an interesting geometry figure. This technique works within the confines of an area-based template: e.g., $\alpha - \beta$ refers to a shape β within a shape α. By contrast, our figure synthesis technique constructs a cohesive, abstract structure and instantiates that structure based on user query. The goal of (Alvin et al. 2017a) is for interesting figures for a particular class of problems (shaded area problems) whereas our technique is more general.

9 Conclusions

We have described a parameterized technique to generate Euclidean geometry figures consistent with textbooks. From a user-defined query stating constituent geometry shapes, we generate an abstract topological structure, *geometry figure lattice*, and ensuring isomorphism among GFLs with a polynomial time algorithm. We then use a graph neural network discriminator to filter toward textbook-quality figures. Last, we generate Euclidean geometry figures appropriate for high school geometry by instantiating a geometry figure lattice. We demonstrated the utility and efficacy of our approach with sample user queries and believe this technique can be leveraged to generate realistic figures for study in an intelligent tutoring system.

References

Alvin, C.; Gulwani, S.; Majumdar, R.; and Mukhopadhyay, S. 2014. Synthesis of geometry proof problems. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, 245–252.

Alvin, C.; Gulwani, S.; Majumdar, R.; and Mukhopadhyay, S. 2017a. Synthesis of problems for shaded area geometry reasoning. In *Artificial Intelligence in Education - 18th International Conference, AIED 2017, Wuhan, China, June 28 - July 1, 2017, Proceedings*, volume 10331 of *Lecture Notes in Computer Science*, 455–458. Springer.

Alvin, C.; Gulwani, S.; Majumdar, R.; and Mukhopadhyay, S. 2017b. Synthesis of solutions for shaded area geometry problems. In *Proceedings of the Thirtieth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2017, Marco Island, Florida, USA, May 22-24, 2017.*, 14–19.

Cord. 2022. *Geometry:Mathematics In Context*. Cord, third edition.

Dai, A., and Nießner, M. 2019. Scan2mesh: From unstructured range scans to 3d meshes. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, 5574–5583. Computer Vision Foundation / IEEE.

Gantert, A. X. 2007. *Amsco's Geometry*. Professional BookDistributors, Inc.

Grätzer, G. 1998. *General Lattice Theory*. Birkhauser.

Jurgensen, R. C.; Brown, R. G.; and Jurgensen, J. W. 1951. *Geometry*. Houghton Mifflin Company.

Larson, R.; Boswell, L.; Kanold, T. D.; and Stiff, L. 2009. *Holt McDougal Larson Geometry: Student Edition 2011*. HOLT MCDOUGAL, first edition.

Larson, R.; Boswell, L.; and Stiff, L. 2004. *Geometry, Grades 9-12: Mcdougal Littell High School Math (McDougal Littell High Geometry)*. McDougal Littell/Houghton Mifflin Company, tenth edition.

Michael, S. 2002. *Discovering Geometry: An Investigative Approach*. Key Curriculum Press, third edition.

Posamentier, A. S., and Salkind, C. T. 1996. *Challenging Problems in Geometry (Dover Books on Mathematics)*. Dover Publications, second edition.

Rhoad, R.; Milauskas, G.; and Whipple, R. 1991. *Geometry for Enjoyment and Challenge*. McDougal Littell.

Simonovsky, M., and Komodakis, N. 2018. Graphvae: Towards generation of small graphs using variational autoencoders. In Kurková, V.; Manolopoulos, Y.; Hammer, B.; Iliadis, L. S.; and Maglogiannis, I., eds., *Artificial Neural Networks and Machine Learning - ICANN 2018 - 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part I*, volume 11139 of *Lecture Notes in Computer Science*, 412–422. Springer.